Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Hermand Dieumo Kenfack

Designmigrationsstrategien von FlexRay nach Time-Triggered Ethernet

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Hermand Dieumo Kenfack

Designmigrationsstrategien von FlexRay nach Time-Triggered Ethernet

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Franz Korf
Zweitgutachter: Prof. Dr. Gunter Klemke

Eingereicht am: 12. April 2012

Hermand Dieumo Kenfack

Thema der Arbeit
Designmigrationsstrategien von FlexRay nach Time-Triggered Ethernet

Stichworte
Automobil-Netzwerke, Echtzeitsysteme, Migration, Modellierung, Task-Graphen, Scheduling, FlexRay, TTEthernet, event- und time-triggered Kommunikation, Framework

Kurzzusammenfassung
Ein Automobil-Netzwerk ist ein komplexes, verteiltes System, an das eine Vielfalt von Anforderungen (Determinismus, hohe Bandbreite, etc.) gestellt werden. Viele herkömmliche Protokolle wie LIN und CAN sind diesen Anforderungen nicht gewachsen. Daher werden alternative Lösungen untersucht. Momentan gibt es zwei viel versprechende Lösungen: FlexRay und TTEthernet. Obwohl FlexRay sich in der Automobil-Industrie bereits etabliert hat, weist TTEthernet zusätzliche Vorteile (preiswerter, höhere Bandbreite, etc.) auf. So ist der Wechsel von FlexRay nach TTEthernet lohnenswert. In dieser Arbeit werden Migrationsstrategien, welche demonstrieren, dass dieser Wechsel möglich ist, entwickelt. Dabei wird der Fokus auf das Kommunikationsmodell der verteilten Anwendungen gelegt. Die Migrationsstrategien werden anhand ausgewählter Automobil-Anwendungen validiert.

Title of the paper
Design migration strategies from FlexRay to time-triggered Ethernet

Keywords
automotive networks, real time systems, migration, modeling, scheduling, task-graphs, FlexRay, TTEthernet, event- and time-triggered communication, framework

Abstract
An automotive network is a complex, distributed system with a variety of requirements (determinism, high bandwidth, etc.). Many traditional communication protocols such as LIN and CAN are not up to these requirements. Therefore, alternative solutions are examined. There are currently two promising solutions: FlexRay and TTEthernet. Although FlexRay has already been established in the automotive industry, TTEthernet has additional advantages (low cost, higher bandwidth, etc.). Thus, the change from FlexRay to TTEthernet is worthwhile. In this work, migration strategies are developed which demonstrate that this change is possible. Thereby, the focus is laid on the communication model of the distributed applications. The migration strategies are validated by means of selected automotive applications.

Inhaltsverzeichnis

1 Einführung

Die Anzahl der eingesetzten elektrischen und elektronischen Systeme im Automobil steigt immer weiter an. Es gibt immer mehr Sensoren, Aktoren, Steuereinheiten, Sicherheits-, Hilfs- und Multimedia-Systeme. Ein modernes Automobil setzt sich aus mehr als 100 solcher Systeme zusammen, welche mit mehr als 2500 Signalen (d. h. elementare Informationen wie die Geschwindigkeit des Fahrzeugs) miteinander kommunizieren (vgl. Navet u. a., 2005; Lukasiewycz u. a., 2011). Da die Funktionen der verschiedenen Systeme systemübergreifend benötigt werden, ist ein umfangreicher Datenaustausch in Echtzeit zwischen diesen unabdingbar. So wird beispielsweise die Fahrtgeschwindigkeit im ESP (Elektronisches Stabilitätsprogramm) für die Fahrdynamikregelung, im Motormanagement für die automatische Geschwindigkeitsregelung und im Navigationssystem benötigt (vgl. Reif, 2011, S. 82).

Die Heterogenität der Anwendungen führt zu verschiedenen Anforderungen (Echtzeitanforderungen, Sicherheitsanforderungen, verschiedene Datenraten, etc.). Um diese zu erfüllen, wird momentan die Vernetzung von Kraftfahrzeugkomponenten in Domänen (Funktionsbereiche) unterteilt. Für jede Domäne werden ein oder mehrere spezielle Bussysteme eingesetzt, wie LIN (Local Interconnect Network (vgl. LIN-Administration)), CAN (Controller Area Network (vgl. Robert Bosch GmbH)) und MOST (Media Oriented Systems Transport (vgl. MOST Cooperation)). Eine Anwendung wird einer Domäne zugeordnet, die am besten ihren Anforderungen entspricht. Die verschiedenen Bussysteme machen das Fahrzeugnetzwerk kompliziert und damit in der Entwicklung teuer. Außerdem können sie die an zukünftige Automobilnetzwerke gestellten Anforderungen nicht vollständig erfüllen. Beispiele für diese Anforderungen sind: Zuverlässige Datenübertragung, Fehlertoleranz und hohe Bandbreite. Letztere wird z. B. für videobasierte Fahrassistenzsysteme benötigt. Aus diesen Gründen werden neue Lösungen für die Vernetzung von Kraftfahrzeug-Komponenten untersucht.

Momentan gibt es zwei vielversprechende Lösungen: einerseits FlexRay, andererseits TTEthernet (Time-Triggered Ethernet). FlexRay ist ein vom FlexRay-Konsortium (vgl. FlexRay Consortium, a) entwickeltes Kommunikationsprotokoll für Fahrzeugnetzwerke. Es soll für zeitkritische Anwendungen, wie die Motor- und Fahrwerk-Steuerung, verwendet werden. Zu den Eigenschaften von FlexRay gehören: eine deterministische Datenübertragung, Flexibilität und Fehlertoleranz (vgl. Rausch, 2008, S. 8). TTEthernet (vgl. Steiner, 2008) ist eine Echtzeit-Variante des Standard-Switched-Ethernet. Die Echtzeitfähigkeit wird wie, bei FlexRay, durch die Synchronisation aller Teilnehmer und das Aufstellen eines Zeitplans (Schedule) gewährleistet, welcher das simultane Senden zweier Echtzeit-Nachrichten über denselben Link ausschließt.

TTEthernet hat aufgrund dessen, dass es auf dem Standard-Ethernet basiert, einige Vorteile gegenüber FlexRay. Ethernet ist eine Standardtechnologie, die sich mit Computernetzen durchgesetzt und in der Automatisierungsindustrie erfolgreich etabliert hat (vgl. Schwager, 2010). Durch die weite Verbreitung sind die Preise für Ethernet-Komponenten gesunken (vgl. Greifeneder und Frey, 2007, S. 23-33). Weiterhin gibt es bereits eine große Palette an Entwicklungs- und Diagnosewerkzeugen. Die Zahl an fachkundigen Entwicklern ist beim Ethernet-Protokoll weit größer als bei automobilspezifischen Lösungen, wie z. B. MOST (Media Oriented System Transport) oder FlexRay. Darüber hinaus haben viele heutige Mikrocontroller bereits eine integrierte Ethernet-Schnittstelle. Schließlich hat TTEthernet eine deutlich höhere Bandbreite ($100/1000\frac{Mbit}{s}$).

FlexRay ist dabei, sich in der Automobilindustrie zu etablieren. Es gibt bereits Serienfahrzeuge mit FlexRay. Beispiele hierfür sind die BMW-7er-Reihe (vgl. BMW Deutschland) und der Audi A8 (vgl. AUDI AG, TTTech Automotive GmbH, 2009). Daraus kann man schließen, dass Ingenieure der Automobilindustrie bereits FlexRay-Projekte geführt haben und Designentscheidungen bezüglich der Vernetzung von Kraftfahrzeug-Komponenten über FlexRay getroffen haben. Aufgrund dessen, dass FlexRay und TTEthernet konzeptionell ähnlich aufgebaut sind (vgl. FlexRay Consortium, 2005b; Steiner, 2008; Steinbach u. a., 2010), können bei einer Migration von FlexRay nach TTEthernet oder bei der Entwicklung eines neuen TTEthernet-Systems viele konzeptionelle Arbeiten eines FlexRay-Systems übernommen und angepasst werden. Somit kristallisieren sich folgende Fragen heraus:

- Wie können Designentscheidungen eines FlexRay-Systems in einem TTEthernet-System wiederverwendet werden?
- Wie können Erfahrungen aus vergangenen FlexRay-Projekten für TTEthernet-Projekte genutzt werden?

- Wie können Beschränkungen von FlexRay durch TTEthernet überwunden werden?

Ziel dieser Arbeit

In dieser Arbeit wird ein analytisches Framework zur Migration von FlexRay-Systemmodellen nach TTEthernet-Systemmodellen entwickelt. Dabei wird der Fokus auf das Kommunikationsmodell des Systems gelegt. Folgende Punkte werden dabei erarbeitet:

- Festlegung eines Frameworks zur Modellierung bzw. Beschreibung von FlexRay- und TTEthernet-Systemen.
- Entwicklung von Methoden für die Migration eines FlexRay-Systemmodells nach einem TTEthernet-Systemmodell. Dabei soll sowohl das statische als auch das dynamische Segment von FlexRay betrachtet werden.
- Analyse und Validierung der Migrationsmethoden durch ausgewählte Fallstudien oder Beispiel-Anwendungen, wie die elektrische Servolenkung, die Antriebsschlupfregelung und die Adaptive Geschwindigkeitsregelung.

2 Architektur und Konzept der Migration

In diesem Kapitel werden die Architektur und das Konzept für die Entwicklung der Migrationsstrategien dargelegt. Bevor dies geschieht, wird die Abgrenzung der Migration festgelegt.

2.1 Abgrenzung der Migration

Die Abgrenzung der Migration wird anhand der in Abbildung 2.1 dargestellten Elektrischen/-Elektronischen-Architektur (E/E-Architektur) eines Kraftfahrzeugs beschrieben. Diese umfasst das Funktions-, Knoten-, Hard-, Soft-, Topologie- und Leitungssatz-Modell. Von oben betrachtet, werden zuerst die Anforderungen des Systems formuliert. Diese werden durch kooperierende Funktionen realisiert (Funktionsmodell). Der funktionale Aspekt einer E/E-Architektur beschreibt das gewünschte Verhalten des E/E-Systems in Form von Funktionen, welche in Form von Hard- oder Software realisiert und auf ein Netzwerk aus Steuergeräten (Netzwerk-Komponenten) abgebildet werden. Die Kooperation zwischen den Funktionen erfolgt über Nachrichten-Austausch. Die Nachrichten werden während der Erstellung des Funktionsmodells festgelegt. Das Hardwaremodell stellt die Struktur der elektronischen Hardware einer einzelnen elektronischen Komponente dar, während das Softwaremodell die in Software zu realisierenden Funktionen beschreibt. Das Topologiemodell stellt die physikalischen Verbindungen und Anordnungen aller Komponenten des Kraftfahrzeugs über ein Kommunikationsmedium dar. In den letzten Ebenen befindet sich das Bauraum- und Leitungssatzmodell, welches die definierten Komponenten einem bestimmten Einbauort im Fahrzeug zuordnet (vgl. Reif, 2011, S. 155). In dieser Arbeit werden die Anforderungen, das Funktionsmodell und das Topologiemodell betrachtet, da diese Ebenen für die Erstellung eines Kommunikationsmodells relevant sind. Die anderen Ebenen sind nicht Bestandteil dieser Arbeit.

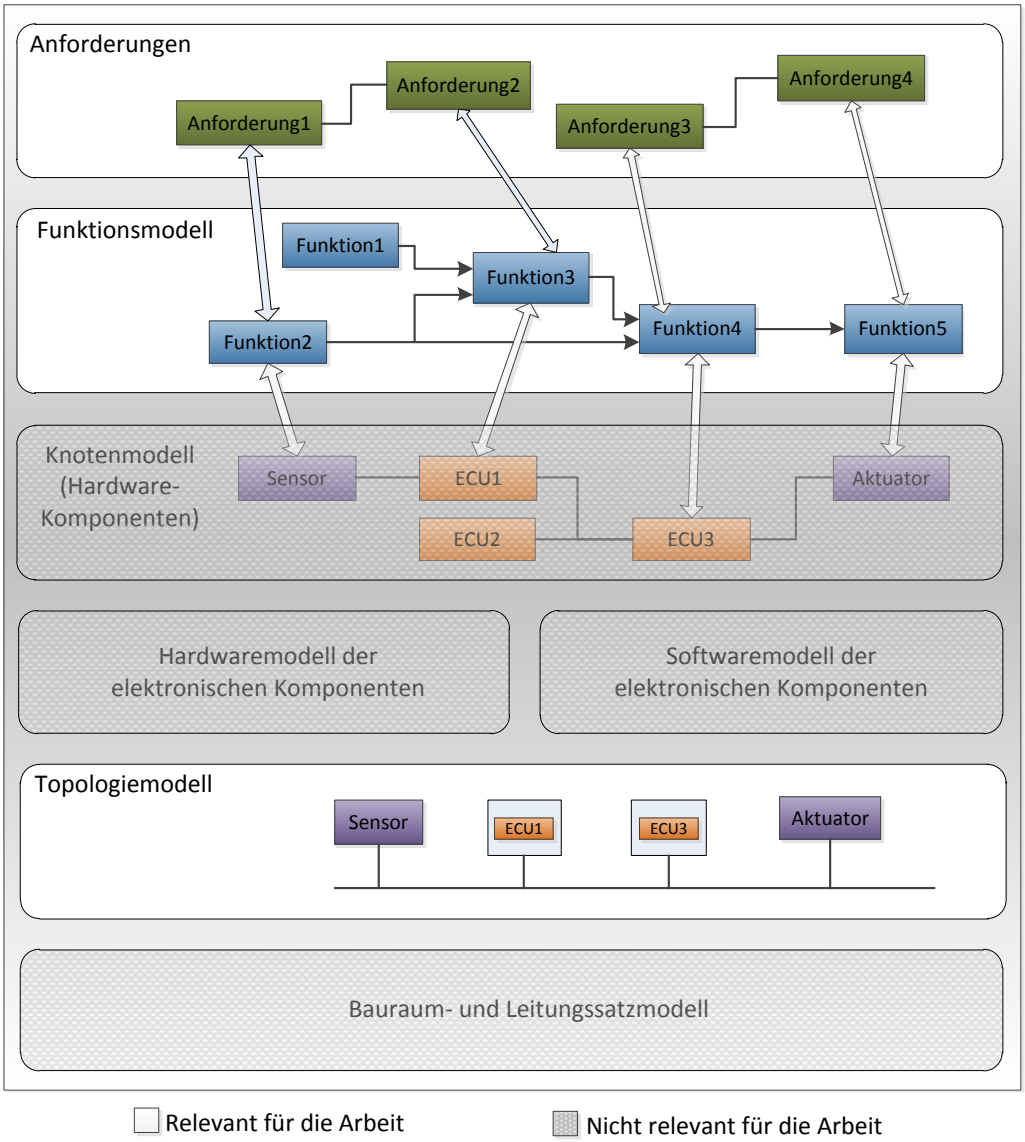


Abbildung 2.1: Die einzelnen Ebenen einer E/E-Architektur und die für die Migration relevanten Abschnitte (vgl. Reif, 2011, S. 155), (vgl. Traub, 2010, S. 20)

2.2 Komponenten der Migration und deren Zusammenspiel

Die an der Entwicklung der Migration beteiligten Komponenten sowie deren Zusammenspiel werden in Abbildung 2.2 dargestellt. Für die Migration wird eine Beschreibung der Funktionen des Systems (Systemspezifikation) sowie deren zeitliche Anforderungen vorausgesetzt. Weiterhin wird ein Systemmodell für FlexRay (FlexRay-Systemmodell) und eine zeitliche Planung der Ausführung der Funktionen und der Versendung der Nachrichten des Systems für FlexRay (FlexRay-Systemschedule) gefordert. Das Systemmodell ist eine abstrakte Darstellung der Systemspezifikation und setzt sich aus dem Funktions- und Topologie-Modell zusammen (siehe Abschnitt 2.1). Das Funktionsmodell stellt die logischen Verbindungen und das Topologie-Modell die physikalischen Verbindungen zwischen den Teilnehmern dar. Die logischen und physikalischen Verbindungen bilden zusammen das Kommunikationsmodell ab. Aus dem FlexRay-Systemmodell wird der FlexRay-Systemschedule, welcher die zeitliche Anforderung des Systems einhält, anhand eines FlexRay-Schedulers erzeugt. Das Migrations-Framework nimmt das FlexRay-Systemmodell und den FlexRay-Systemschedule als Eingabe und generiert daraus sowohl ein TTEthernet-Systemmodell als auch ein TTEthernet-Systemschedule, welche der Systemspezifikation entsprechen bzw. die zeitlichen Anforderungen einhalten.

Der FlexRay-Scheduler ist entweder Bestandteil eines FlexRay-Designers wie der FlexRay Network Designer (vgl. Vector Informatik) von Vector Informatik, der EB tresos Designer (vgl. Elektrobit) von Elektrobit oder ein FlexRay-Scheduling-Algorithmus wie aus den Arbeiten von Ma (2008), Zhao (2011) und Zeng u. a. (2009). Letztere wird im Rahmen dieser Arbeit aufgrund des analytischen Ansatzes verwendet. Die Aufgabe eines Schedulers ist es, basierend auf einem Systemmodell, einen ausführbaren Schedule des Systems (Systemschedule) zu generieren. Mit dem Systemschedule wird die zeitliche Planung der Ausführung der Funktionen des Systems sowie die Planung der Versendung und der Weiterleitung der Nachrichten bezeichnet. In FlexRay wird das Versenden/Weiterleiten/Empfangen von Nachrichten in einem Kommunikationszyklus ausgeführt, welcher aus einem statischen und einem dynamischen Segment besteht. Der FlexRay-Scheduler soll nicht im Rahmen dieser Arbeit entwickelt werden. Es soll jedoch auf vorhandene Lösungen (wie Zhao (2011)) zurückgegriffen werden.

Das Migrations-Framework stellt Werkzeuge und Methoden zur Verfügung, mit denen aus einem FlexRay-Systemmodell und einem FlexRay-Systemschedule ein TTEthernet-Systemmodell und ein TTEthernet-Systemschedule generiert, analysiert und validiert werden können. Wichtig ist, dass die generierten TTEthernet-Modelle die System-Spezifikation und die zeitlichen Anforderungen der Nachrichten, insbesondere deren Deadlines, einhalten. Das Migrations-Framework berücksichtigt sowohl das dynamische als auch das statische Segment von FlexRay.

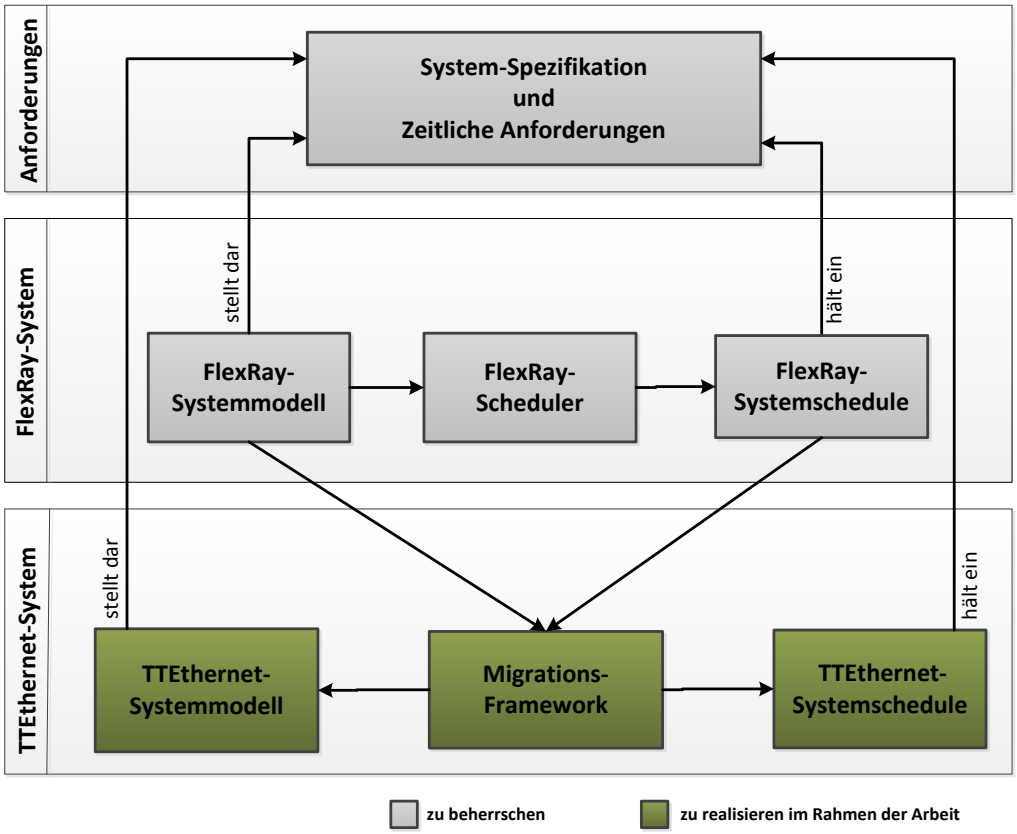


Abbildung 2.2: Komponenten der Migration

Zur Darstellung der Systemmodelle und Schedules (sowohl für FlexRay als auch für TTEthernet) wird ein Modellierungs- bzw. Beschreibungs-Framework im Rahmen dieser Arbeit definiert. Es wird angenommen, dass die FlexRay-Modelle in diesem Beschreibungsschema vorliegen, so dass keine Modell-Transformation nötig ist.

2.3 Detailliertes Vorgehen bei der Entwicklung der Migrationsstrategien

Dieser Abschnitt beschreibt das Vorgehen und die Aufgaben, die bei der Entwicklung der Migrationsstrategien durchgeführt werden sollen. Außerdem werden die Abschnitte bzw. Kapitel, in denen diese realisiert werden, angegeben.

Untersuchung von FlexRay

Die Beherrschung von FlexRay ist für die Entwicklung der Migrationsstrategien unbedingt notwendig. Hierfür sollen die FlexRay-Spezifikationen durchgearbeitet werden. Die für die Arbeit wichtigen FlexRay-Spezifikationen sind: die „FlexRay Requirements Specification“ (vgl. FlexRay Consortium, 2005c) „FlexRay Communications System Protocol Specification Version 2.1 Revision A“ (vgl. FlexRay Consortium, 2005b) und „FlexRay Communications System Electrical Physical Layer Specification, v2.1 Revision A“ (vgl. FlexRay Consortium, 2005a). Außerdem sollen Arbeiten, die sich mit dem Design von FlexRay-Netzwerken befasst haben, untersucht werden. Beispiel hierfür sind Jang u. a. (2011); Navet (2008); Pop u. a. (2007a); Zeng u. a. (2009). Die Beschreibung des FlexRay-Protokolls wird in Kapitel 3 gegeben.

Untersuchung von TTEthernet

Die Beherrschung von TTEthernet ist genauso wie die Beherrschung von FlexRay unabdingbar für eine erfolgreiche Entwicklung der Migrationsstrategien. Hierfür soll die TTEthernet-Spezifikation [1](vgl. Steiner, 2008) durchgearbeitet werden. Dadurch, dass TTEthernet eine neue Technologie ist, ergibt es Sinn, nicht nur die Spezifikation zu untersuchen, sondern auch Abschlussarbeiten (Steinbach (2011); Bartols (2010)) und Veröffentlichungen in diesem Kontext (Kopetz (2008); Kopetz u. a. (2005); Steiner u. a. (2009); Steiner (2011)). Die Beschreibung von TTEthernet wird in Kapitel 4 erfolgen.

Festlegung des Modellierungs- und Beschreibungs-Frameworks

Wie bereits erwähnt, wird ein Framework für die Beschreibung von Systemmodellen und Schedules für FlexRay und TTEthernet benötigt. Dies wird anhand von azyklischen Graphen und der Mengentheorie realisiert. Grund der Auswahl dieses Beschreibungs-Schemas ist sein verbreiteter Ansatz zur Modellierung und Analyse von verteilten Echtzeit-Systemen (vgl. Pop, 2003; Sinnen, 2007; Voss, 2010). In Kapitel 5 wird gezeigt, wie das Funktionsmodell (logische Verbindungen) und Topologiemodell (physikalische Verbindungen) sowie das Systemmodell von verteilten Anwendungen mit Graphen und der Mengentheorie beschrieben werden. Außerdem werden die Begriffe für die Beschreibung von zeitlichen Eigenschaften und Anforderungen von Funktionen und Nachrichten definiert.

[1] In dieser Arbeit wurde die TTEthernet-Spezifikation, die bei der Society of Automotive Engineers (SAE) in der Arbeitsgruppe AS-2D (vgl. SAE - AS-2D Time Triggered Systems and Architecture Committee, 2009) zur Standardisierung eingereicht wurde, verwendet, da die standardisierte Version erst veröffentlicht wurde, als diese Arbeit sich bereits in einer fortgeschrittenen Phase befand.

Untersuchung von verwandten und vergleichbaren Arbeiten

Die Untersuchung von verwandten und vergleichbaren Arbeiten soll dazu dienen, erste Erkenntnisse über vorhandene Ansätze und Vorgehensweisen für die Migration von Automobil-Netzwerk-Protokollen zu gewinnen. Weiterhin soll damit die eigene Arbeit von vergleichbaren Arbeiten abgegrenzt werden. Die mit dieser Arbeit verwandten und vergleichbaren Arbeiten werden in Kapitel 6 dargelegt.

Entwicklung der Migrationsstrategien

Vor der Entwicklung der Migrationsstrategien werden zunächst die Gemeinsamkeiten und Unterschiede zwischen TTEthernet und FlexRay herausarbeitet. In Steinbach u. a. (2010) wurde bereits auf dieses Thema eingegangen. Dieses soll in dieser Arbeit in Hinsicht auf das Designen komplettiert werden. Weiterhin sollen weitere Protokoll-Eigenschaften, wie Nachrichten-Klassen, Nachrichten-Formate und Adressierung, verglichen werden. Nach dem Vergleich von FlexRay und TTEthernet werden Methoden entwickelt, mit denen verteilte Anwendungen eines FlexRay statischen als auch dynamischen Segments nach TTEthernet migriert werden, so dass diese danach in einem TTEthernet-Kommunikationszyklus ausgeführt werden können. Die Grundlage für die Entwicklung der Strategien sind die Spezifikationen der beiden Protokolle und deren erarbeitete Gemeinsamkeiten und Unterschiede sowie das vorhandene FlexRay-Systemmodell und der Schedule. Jede Migrationsstrategie wird durch Beispiele und Fallstudien von Automobil-Anwendungen illustriert, analysiert und validiert. Die Entwicklung der Migrationsstrategien wird in Kapitel 7 durchgeführt.

Zusammenfassung und Ausblick

In Kapitel 8 wird die Zusammenfassung der Arbeit sowie ein Ausblick für zukünftige Arbeiten gegeben.

3 FlexRay

In diesem Kapitel wird das FlexRay-Protokoll detailliert dargelegt. Dies stützt sich vor allem auf die FlexRay-Protokollspezifikationen (vgl. FlexRay Consortium, 2005b,a) und das Buch „FlexRay: Grundlagen, Funktionsweise, Anwendung“ (vgl. Rausch, 2008).

3.1 Allgemeines

Das FlexRay-Kommunikationssystem ist ein robustes, skalierbares, deterministisches und fehlertolerantes, digitales, serielles Bussystem, welches für die Vernetzung von Automobil-Komponenten entwickelt wurde. Dieses wurde vom FlexRay-Konsortium entwickelt, welches im Jahr 2000 gegründet wurde, als BMW, DaimlerChrysler, Motorola und Philips sich zusammen schlossen, um ein neues Protokoll zu entwickeln, welches den zukünftigen Anforderungen von im Automobil verteilten Anwendungen gerecht werden kann. Weitere Firmen wie Bosch, General Motors und Volkswagen zögerten nicht, dem Konsortium beizutreten (vgl. FlexRay Consortium, b).

3.2 Eigenschaften von FlexRay

FlexRay hat unter anderem folgende Eigenschaften:

- **Datenrate von** $2 \times 10 \frac{Mbit}{s}$
  Ein FlexRay-Knoten kann auf maximal zwei Kommunikationskanälen, die beide je eine maximale Datenrate von $10 \frac{Mbit}{s}$ haben, angeschlossen werden.

- **Redundante Kommunikationskanäle**
  Wie bereits gesagt, kann ein FlexRay-Teilnehmer an zwei Kommunikationskanäle angeschlossen werden. Die beiden Kanäle können benutzt werden, um entweder eine Nachricht redundant zu übertragen und somit die Fehlertoleranz des Systems zu erhöhen oder um verschiedene Nachrichten zu senden und somit die Bandbreite zu erhöhen.

- **Unterstützung von time- und event-triggered Kommunikation**
  Der Austausch von Nachrichten kann im FlexRay time-triggered (zeitgesteuert) oder event-triggered (ereignisgesteuert) erfolgen. Je nach Anwendungen eignet sich die eine oder die andere Kommunikations-Art am besten. Für Regelungen wird meist die time-triggered Kommunikation verwendet, für Diagnose-Anwendungen hingegen die event-triggered Kommunikation.

- **Deterministisches Zeitverhalten bei time-triggered Nachrichten**
  Die Kommunikation über den FlexRay-Bus wird in einem Zyklus, welcher sich periodisch wiederholt, organisiert. Dabei werden alle time-triggered Nachrichten in dem Kommunikationszyklus einem festen Zeitrahmen, in dem diese gesendet werden sollen, zugeordnet. Dadurch werden die Ankunftszeiten dieser Nachrichten garantiert und deren zeitliche Schwankungen gering gehalten.

- **Synchronisierte Zeitbasis**
  Um das deterministische Nachrichten-Zeitverhalten von FlexRay zu garantieren, müssen die Uhren aller Teilnehmer miteinander synchronisiert werden. Das FlexRay-Protokoll definiert hierfür eine fehlertolerante synchronisierte Zeitbasis. Die Genauigkeit dieser Zeitbasis liegt zwischen $0,5\mu s$ bis $10\mu s$. Typische Werte liegen jedoch bei 1 bzw. $3\mu s$ (vgl. Rausch, 2008, S. 8).

3.3 Mögliche FlexRay-Topologien

In diesem Abschnitt werden die möglichen Topologien eines FlexRay-Systems vorgestellt. Es gibt verschiedene Möglichkeiten für die Konfiguration der Topologie eines FlexRay-Systems. Diese kann als Singlekanal oder Dualkanal-Bus, Singlekanal oder Dualkanal-Stern oder in verschiedenen Kombinationen von Hybrid-Bus- und Stern konfiguriert werden.

Bus-Topologien

In Abbildung 3.1 wird ein Beispiel einer Dualkanal-Bus-Topologie gezeigt. Ein Knoten kann auf die Kanäle A und B (Knoten A, C und E), nur auf Kanal A (Knote D) oder nur auf Kanal B (Knote B) angeschlossen werden. Ein FlexRay-Kommunikationssystem kann auch nur aus einem Single-Bus bestehen. Im diesem Fall werden alle Knoten auf diesem Bus angeschlossen (siehe Abbildung 3.2).

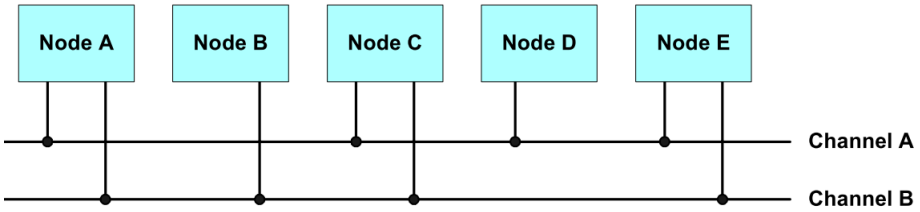


Abbildung 3.1: Beispiel einer FlexRay-Dualkanal-Bus-Topologie (vgl. FlexRay Consortium, 2005b)

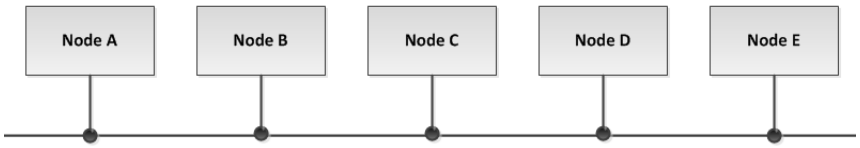


Abbildung 3.2: Beispiel einer FlexRay-Singlekanal-Bus-Topologie

Stern-Topologie

Abbildung 3.3 zeigt ein Beispiel einer FlexRay-Dualkanal-Singlestern-Topologie (aktiver Stern). In einer aktiven Stern-Verbindung werden eingehende Signale verstärkt und an allen an dem Stern angeschlossen Knoten bis auf den Sender weitergeleitet. Diese Funktionalität wird in einem Computer-Netzwerk durch Hubs realisiert. Neben aktiven Sternkopplern gibt es passive

Sternkoppler, welche keine aktiven Elemente besitzen, sodass in diesem Fall die Signale vor dem Weiterleiten nicht verstärkt werden können. In dieser Arbeit wird mit Sternkoppler (kurz Stern) ohne vorangestelltes Adjektiv immer der aktive Sternkoppler gemeint und bei den passiven Sternkopplern das Adjektiv vorangestellt. Die logische Struktur (d. h. die logischen Verbindungen zwischen den Knoten) der Dualkanal-Singlestern-Topologie aus Abbildung 3.3 ist identisch mit der Dualkanal-Bus-Topologie aus Abbildung 3.1. Es ist auch möglich, eine Singlekanal-Singlestern-Topologie zu konfigurieren. Die logische Struktur dieser entspricht dann der einer Singlekanal-Bus-Topologie (siehe Abbildung 3.2).

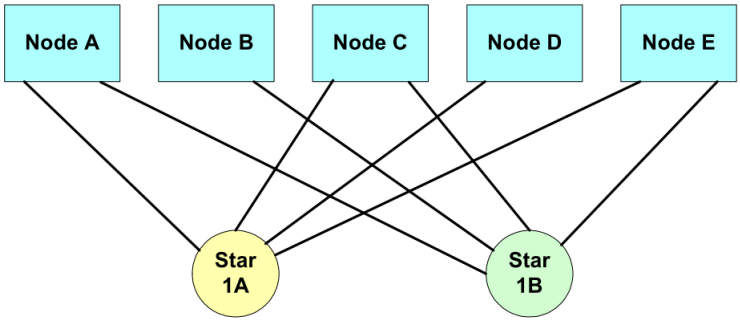


Abbildung 3.3: Beispiel einer FlexRay-Dualkanal-Singlestern-Topologie (vgl. FlexRay Consortium, 2005b)

In Abbildung 3.4 wird ein Beispiel einer FlexRay-Singlekanal-Stern-Topologie mit zwei Stern-Kopplern gezeigt. In dieser Konfiguration hat jeder Knoten eine Punkt-zu-Punkt-Verbindung mit einer der beiden Sterne. Die beiden Sterne sind direkt miteinander verbunden. Man spricht in diesem Fall von kaskadierten Sternkopplern. Erwähnenswert ist, dass nicht mehr als zwei Sternkoppler je Kanal in einem FlexRay-System kaskadiert werden können. Der Vorteil dieser Topologie ist die große Anzahl an Knoten, die angeschlossen werden können. Weiterhin ist es möglich, eine kaskadierte Dualkanal-Sternkoppler-Topologie einzurichten. Abbildung 3.5 zeigt ein Beispiel hierzu.

Hybrid-Topologien

Neben Topologien, die vollständig aus Bussen oder Sternen gebildet werden, können auch Hybrid-Topologien konfiguriert werden. Das heißt Topologien, die sowohl aus Sternen als auch Bussen bestehen. Es gibt viele Varianten von möglichen Hybrid-Topologien, es werden hier jedoch nur zwei dargestellt. Abbildung 3.6 zeigt das erste Beispiel. In dieser Topologie erfolgt die Verbindung einiger Knoten (Knoten 1, 2, und 3) über eine Punkt-zu-Punkt-Verbindung zu den aktiven Sternkopplern, während andere Knoten (Knoten 4, 5, 6, 7, 8, 9 und 10) über

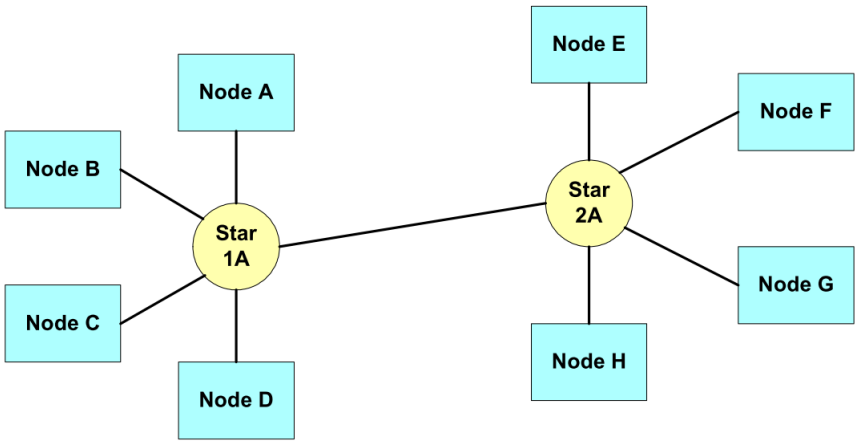


Abbildung 3.4: Beispiel einer FlexRay singlekanal kaskadierten Stern-Topologie (vgl. FlexRay Consortium, 2005b)

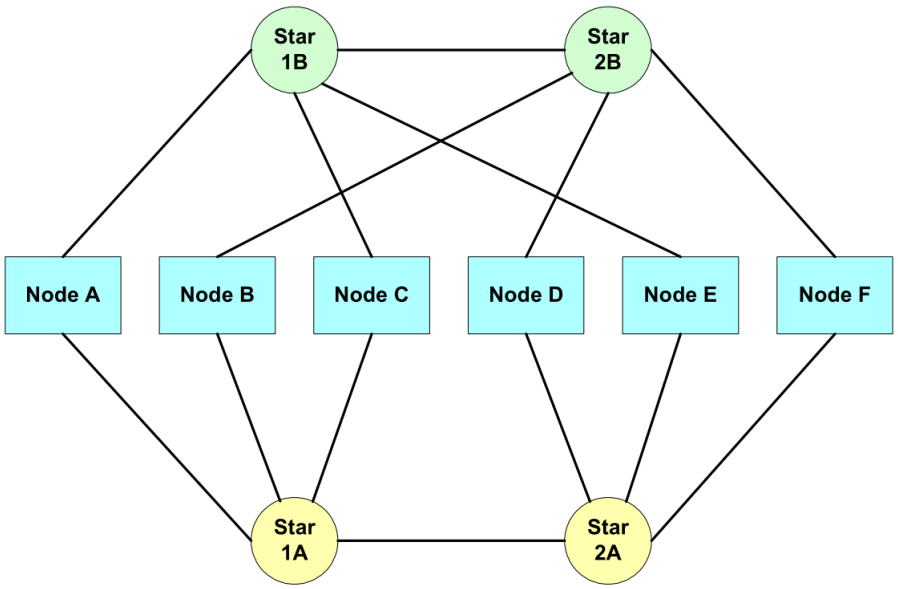


Abbildung 3.5: Beispiel einer FlexRay dualkanal kaskadierten Stern-Topologie (vgl. FlexRay Consortium, 2005b)

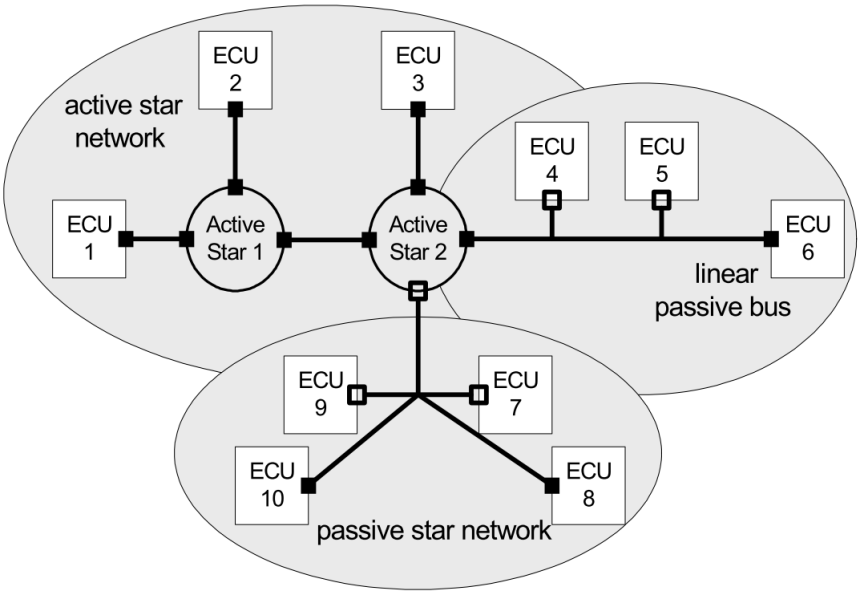


Abbildung 3.6: Beispiel einer FlexRay-Singlekanal-Hybrid-Topologie (vgl. FlexRay Consortium, 2005a)

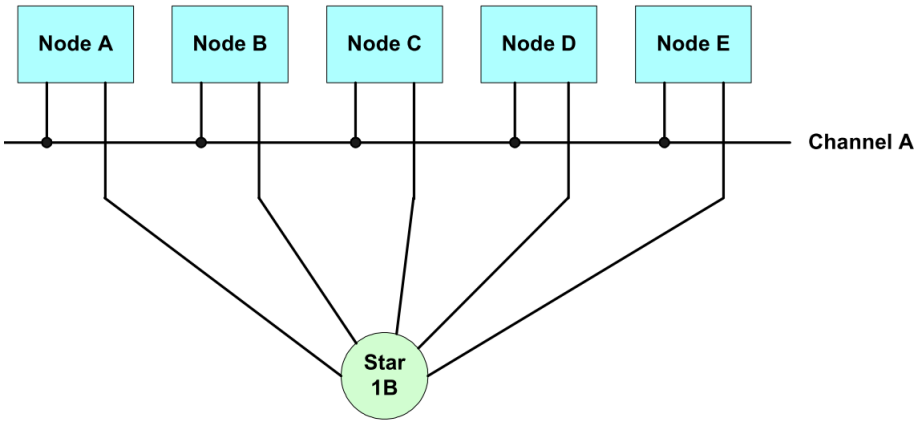


Abbildung 3.7: Beispiel einer FlexRay-Dualkanal-Hybrid-Topologie (vgl. FlexRay Consortium, 2005b)

einen Bus bzw. passiven Stern-Koppler miteinander verbunden sind. Bus und passiver Stern sind auch zu einem Sternkoppler verbunden, sodass deren Knoten mit den anderen Knoten kommunizieren können. Laut der FlexRay-Spezifikation sind Topologien, in denen Busse an aktiven Sternkopplern angeschlossen werden, noch nicht vollständig untersucht worden und demnach nicht empfohlen. Das zweite Beispiel wird in Abbildung 3.7 dargestellt. In diesem Beispiel wird eine Dualkanal-Hybrid-Topologie angezeigt. Hierbei sind alle Knoten sowohl mit dem Bus als auch mit dem Sternkoppler verbunden.

3.4 Architektur eines FlexRay-Kommunikationsknotens

Ein FlexRay-Kommunikationsknoten (Netzwerk-Teilnehmer) besteht aus dem Host-Mikrocontroller (auf dem die Anwendung läuft), dem eigentlichen FlexRay-Controller, auch Communication-Controller genannt, und jeweils einem Bus-Driver (BD) für jeden Kanal, an dem der Knoten angeschlossen werden soll (siehe Abbildung 3.8). Der Host und der Communication-Controller teilen sich eine erhebliche Menge an Informationen gegenseitig mit. Dieser Austausch erfolgt über eine Schnittstelle, das Controller-Host-Interface. Der Host teil dem FlexRay-Controller Steuerung und Konfigurationsinformationen mit. Außerdem fragt er die Sensor-Werte ab, die er an den Communication-Controller in Form von Nachrichten zur Übertragung weitergibt. Der Communication-Controller sendet dann die Sensoren-Werten nach dem Schedule an die Aktoren, die diese interpretieren und entsprechend darauf reagieren. Der FlexRay-Controller teil dem Host die Status-Informationen und die eingehenden Nachrichten mit. Der Communication-Controller realisiert weiterhin die Funktionalitäten eines FlexRay-Knoten, die sich auf das Protokoll beziehen. Diese sind: die Ausführung des Schedules, die Synchronisation des Knoten mit anderen Knoten, die Timer-Funktion (Generierung von Macrotick-Signalen), die Erzeugung eines Bitstreams aus den Host-Informationen (Nachrichten), die Steuerung des Bus-Zugriffes, etc. Der Bus-Driver stellt die Verbindung zum Kommnunikationskanal her. Er hat die Aufgabe eines Traceivers. Er wandelt den vom Communication-Controller erzeugten Bitstream in physikalische Spannungen um und umgekehrt. Es gibt eine Schnittstelle zwischen dem Bus-Driver und dem Host. Diese ermöglicht dem Host, die Betriebsmodi des Bus-Drivers zu steuern und die Statusinformationen des Bus-Drivers auszulesen. Außerdem kann der Bus-Driver optional die Wakeup-Signale verarbeiten und somit den Power-Supply ansteuern.

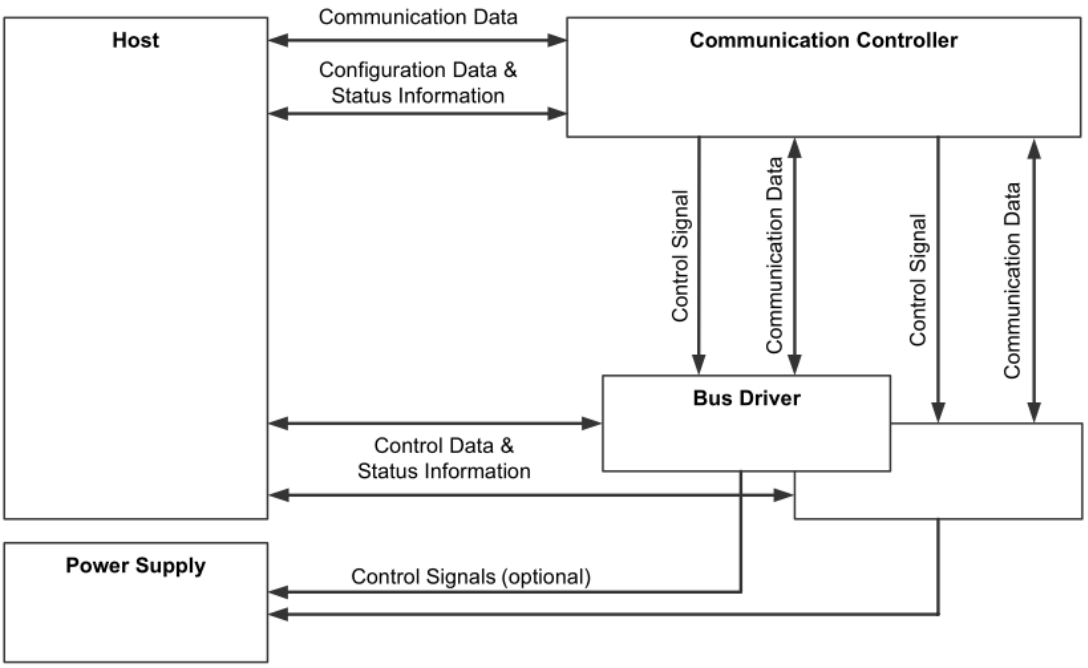


Abbildung 3.8: Struktur eines FlexRay Kommunikationsknotens (vgl. FlexRay Consortium, 2005b)

3.5 Kommunikationsmedium Zugriffsverfahren

Der Medienzugriff in einem FlexRay-System erfolgt innerhalb eines periodischen Kommunikations-Zyklus. Innerhalb dieses Zyklus gibt es zwei Zugriffsverfahren: das Coordinated-Time-Division-Multiple-Access-Verfahren (CTDMA) und das dynamische Mini-Sloting-Verfahren. Diese werden in diesem Abschnitt detailliert beschrieben. Es wird aber zunächst der FlexRay-Kommunikationszyklus vorgestellt.

3.5.1 Der Kommunikationszyklus

Ein FlexRay-Kommunikationszyklus besteht aus einem obligatorischen statischen und einem optionalen dynamischen Segment sowie einem oder zwei Protokoll-Segmenten, nämlich dem obligatorischen Network Idle Time (NIT) und dem optionalen Symbol-Window (siehe Abbildung 3.9). Das statische Segment besteht aus einer bestimmten Anzahl von statischen Slots, die alle eine gleiche feste Dauer haben. Diese werden für die Übertragung von Frames [1] mit

[1] Frame und Nachricht werden sinngleich in dieser Arbeit verwendet.

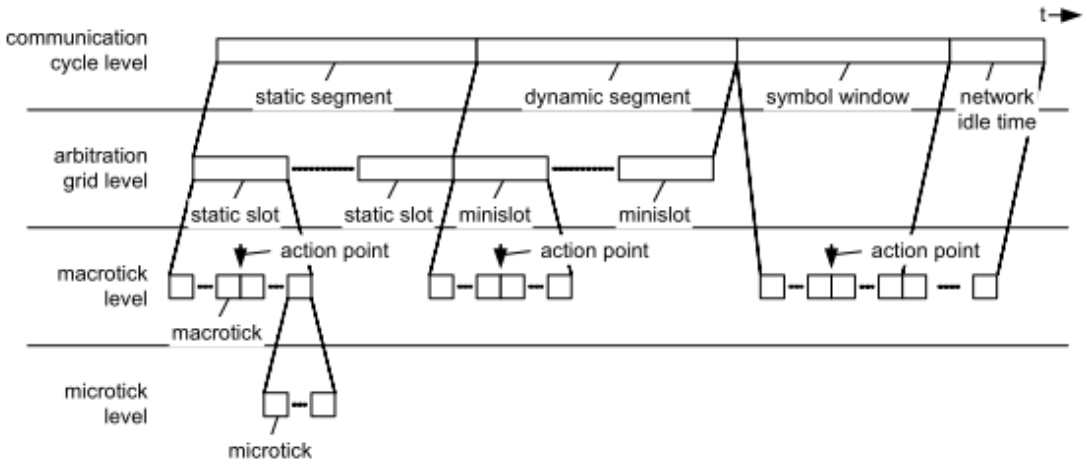


Abbildung 3.9: Der FlexRay-Kommunikationszyklus (vgl. FlexRay Consortium, 2005b)

deterministischen Zeitverhalten verwendet. Das dynamische Segment besteht aus sogenannten Mini-Slots, die durch den Host für die Übertragung von event-triggered Nachrichten mit variablen Payload-Längen verwendet werden. In diesem Segment werden außerdem Nachrichten mit weichen oder ohne Echtzeit-Anforderungen übertragen. Die Slots der beiden Segmente werden fortlaufend nummeriert. Die Nummerierung beginnt mit eins und im statischen Segment (vgl. Navet und Simonot-Lion, 2009).

Das FlexRay-Protokoll unterstützt 64 verschiedene Kommunikationszyklen, die sich über einen Zyklus-Zähler identifizieren lassen. Durch diese Differenzierung ist es möglich, für einen Knoten verschiedene Nachrichten in dem gleichen Slot in verschiedene Zyklen zu übertragen.

Ein FlexRay-Kommunikationszyklus setzt sich aus einer definierten Anzahl von Makroticks zusammen (Der Makrotick ist die kleinste Einheit für die gemeinsame, synchronisierte Zeit in einem Cluster). Diese werden aus einer Anzahl von Mikroticks gebildet, welche die kleinste Zeiteinheit lokaler Uhren darstellen. Die Mikroticks werden aus der Quarzfrequenz des jeweiligen FlexRay-Knoten abgeleitet. Da aber die Quarzfrequenzen unterschiedlich sein können, können sich die Makroticks verschiedener FlexRay-Knoten aus unterschiedlich vielen Mikroticks zusammensetzen.

3.5.2 Das statische Segment

Im statischen Segment (ST-Segment) basiert der Zugriff auf das Kommunikationsmedium auf dem CTDMA-Verfahren. Dafür wird jeder Nachricht, die im statischen Segment gesendet

wird, ein statischer Zeit-Slot zugewiesen, in dem dieser gesendet werden soll. Im FlexRay sind alle statischen Slots mit der gleichen Länge konfiguriert. Die Kommunikation innerhalb eines statischen Segments erfordert gewisse Randbedingungen, nämlich:

- Ein Knoten soll seine Sync-Frames (Synchronisation-Frames) auf alle Kanäle übertragen, an denen er angeschlossen ist.
- Nicht-Sync-Frames, also normale Kommunikations-Frames, können entweder über einen oder beide Kanäle übertragen werden.
- In einem Cluster soll nur ein Knoten als Sender einer Frame-ID pro Kanal konfiguriert werden. Das heißt, dass die gleiche Frame-ID zwei Knoten, die auf verschiedene Kanäle senden, zugewiesen werden kann (siehe Abbildung 3.11).

Um das Senden der Nachrichten im statischen Segment zu planen, verwaltet jeder Knoten eine Slotzähler-Variable (*vSlotCounter*) [2] für Kanal A und eine für Kanal B. Beide Zähler werden am Anfang jedes Kommunikationszyklus mit eins initialisiert und am Ende eines statischen Slots um eins erhört. Die Anzahl der statischen Slots (*gNumberOfStaticSlots*) ist gleich für alle Knoten eines Clusters. Es können Maximal 1023 statische Slots definiert werden. Weil zur Generierung der globalen Zeitbasis mindestens zwei FlexRay-Knoten erforderlich sind, muss sich das statische Segment mindestens aus zwei statischen Slots zusammensetzen.

Die Länge eines statischen Slots (*gdStaticSlot*) setzt sich aus einer gewissen Anzahl an Makroticks zusammen. Alle statischen Slots eines Clusters haben die gleiche Anzahl von Makroticks.

Für Sync-Knoten soll ein statischer Slot zum Senden von Sync-Frames festgelegt werden. Dieser Slot wird durch die Variable *pKeySlotUsedForSync* identifiziert. Spezifische Sync-Frames können als Startup-Frames konfiguriert werden. Diese werden mit der Variable *pKeySlotUsedForStartup* identifiziert.

Jeder statische Slot hat einen Action-Point, welcher einen Offset (*gdActionPointOffset*) ausgehend vom Anfang des statischen Slot ist. Dieser Offset ist ein Clusterparameter und wird in Macrotick konfiguriert. Die Übertragung eines Frames, eines statischen Slots, startet immer am Action-Point. Die Timing-Struktur eines statischen Slots wird durch die Abbildung 3.10 illustriert.

Wenn ein Cluster aus zwei Kanälen besteht, kann ein Controller einen statischen Slot entweder für die redundante Übertragung eines Frames oder für die Übertragung von zwei

[2] Die in diesem Kapitel im Klammer und kursiv geschriebenen Wörter, sind die in der FlexRay-Protokollspezifikation festgelegten Variablen- bzw. Parameter-Namen.

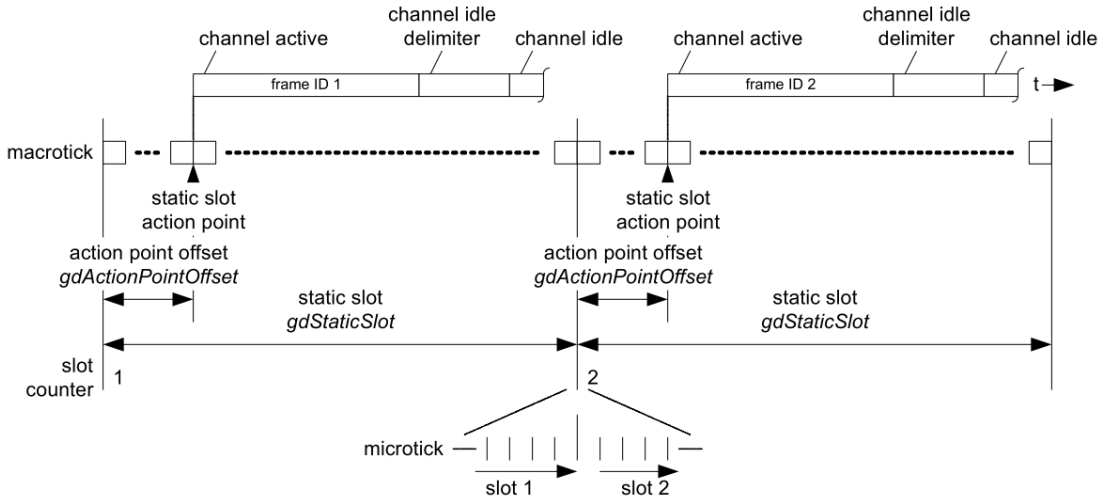


Abbildung 3.10: Timing-Struktur eines FlexRay statischen Slots (vgl. FlexRay Consortium, 2005b)

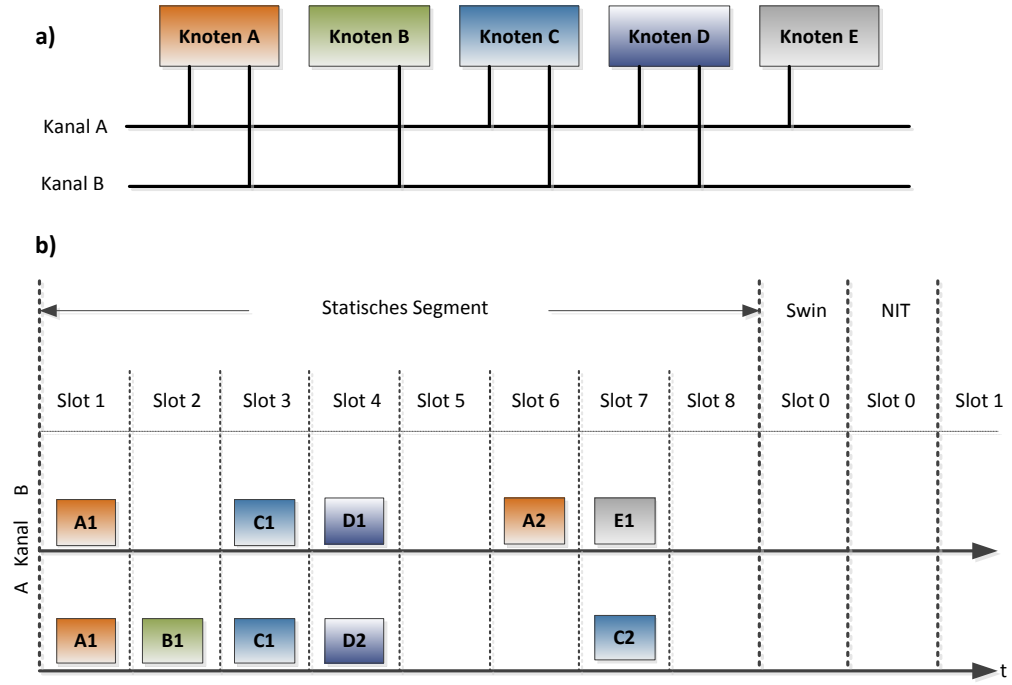


Abbildung 3.11: Beispiel-Konfiguration eines FlexRay statischen Segments

verschiedenen Frames verwenden. Die dritte bereits erwähnte Möglichkeit wäre, dass sich zwei Communication Controller einen Slot teilen. Voraussetzung hierfür ist, dass diese auch in dem Slot auf verschiedenen Kanäle senden. Abbildung 3.11 a) illustriert eine FlexRay-Cluster-Topologie während 3.11 b) eine Beispiel-Konfiguration des statischen Segments angibt. Knoten A und C benutzen beide Kanäle für die redundante Übertragung ihrer ersten Frames während Knoten D die Bandbreite ausnutzt und auf beiden Kanälen zwei verschiedene Frames übertragt. Die Knoten B und E sind jeweils nur an den Kanälen B und A angeschlossen und senden auch nur auf diese Kanäle. Die Knoten C und E teilen sich den Slot 7, übertragen jedoch auf verschiedenen Kanälen. Die Slots 5 und 8 sind unbenutzt. Es ergibt auch Sinn, einige Slots frei zu lassen, falls beabsichtigt wird, das System in der Zukunft zu erweitern. Wenn ein statischer Slot frei gelassen wird, wird auch in ihm nichts gesendet und die Bandbreite bleibt ungenutzt.

Durch die feste Zuweisung von Time-Slots an den Communication-Controller werden im statischen Segment deterministische Nachrichten-Latenzen garantiert. Es ist genau bekannt, wann ein bestimmter Frame auf einem Kanal übertragen wird. Da die Kommunikation kollisionsfrei ist, kann die Worst-Case-Latenz der Nachrichten berechnet werden.

3.5.3 Das dynamische Segment

Im dynamischen Segment eines FlexRay-Kommunikationszyklus kommt ein flexibles Mediumzugriffsverfahren zum Einsatz, das sogenannte Flexibel-TDMA-Verfahren (FTDMA). Dieses Verfahren ist ereignis- und prioritätsbasiert. Das dynamische Segment setzt sich aus mehreren sogenannten Minislots zusammen. Ein Minislot hat nur eine kleine Zeitdauer, welche clusterweit in Makroticks konfiguriert wird. Ähnlich wie im statischen Segment, bekommt jede Nachricht, die im dynamischen Segment übertragen werden soll, einen Minislot zugewiesen. Je kleiner der Minislot einer Nachricht ist, desto höher ihre Priorität. Für die Arbitrierung des Mediumzugriffs wird ein Zähl-Verfahren angewendet, in dem die Minislots fortlaufend beginnend von der letzten statischen Slot-Nummer plus ein gezählt werden. Alle Knoten haben die gleiche Sicht auf den aktuellen Minislot. Wenn der Minislot-Zähler dem Minislot einer Nachricht entspricht, kann der Knoten, welcher diese Nachricht sendet, auf den Bus zugreifen. Wenn ein Knoten in seinem Minislot übertragen will, greift er auf das Medium zu und fängt mit der Übertragung des Frames an. Dieses wird von allen anderen Knoten erkannt und das Zählen der Minislots unterbrochen. Somit erweitert der sendende Controller seine Minislots auf die Länge des zu übertragenen Frames. Es kann nur weitergezählt werden, wenn die Übertragung des Frames abgeschlossen wurde. Mit der Erweiterung eines Minislots wird die Anzahl der zur Verfügung stehenden Minislots reduziert. Aus Abbildung 3.12(b) kann man aus Kanal B sehen, dass die Anzahl der verfügbaren Minislots auf 15 reduziert wurde, da der

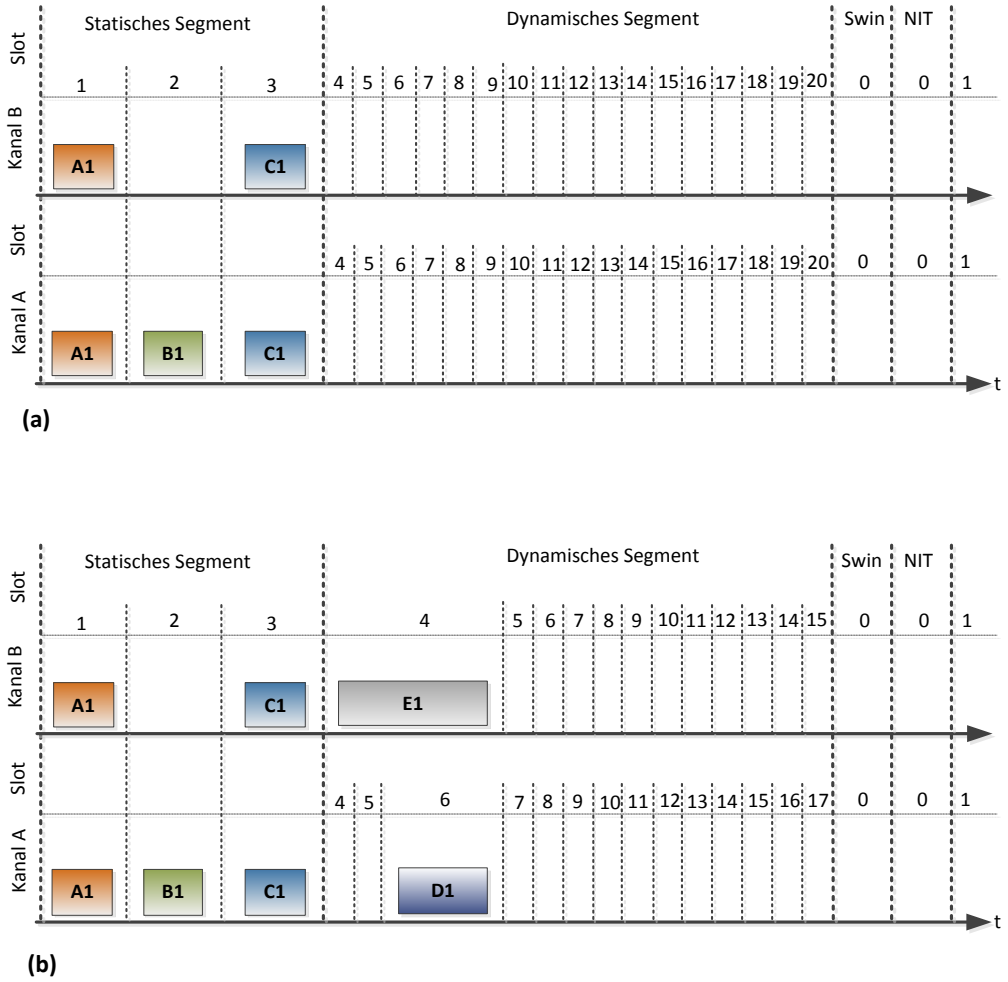


Abbildung 3.12: Beispiel-Konfiguration eines FlexRay dynamischen Segments (vgl. Millinger, 2011)

Knoten E für die Übertragung seines Frames 4 zusätzliche Minislots benötigt. Wenn ein Knoten in seinem Minislot nicht übertragen will, bleibt er ruhig, also greift nicht auf den Bus zu. So wird der Minislot nicht erweitert und das Zählen mit dem nächsten Minislot fortgesetzt. Wenn ein Minislot nicht erweitert wird, wird nur so viel Bandbreite wie die Länge eines Minislots „verschwendet“. Erwähnenswert ist, dass das Verfahren unabhängig vom Kanal durchgeführt wird. Das heißt, dass es für beide Kanäle zwei verschiedene Minislots-Zähler gibt, die nicht simultan erhört werden müssen.

Die Übertragung der Frames im dynamischen Segment ist prioritätsbasiert. Frames mit kleinere Slotnummer haben eine höhere Priorität als Frames mit größerer Nummer. Je mehr Frames am Anfang des Segments gesendet werden, desto niedriger ist die Chance, dass Frames, die sich am Ende des Segments befinden, übertragen werden. Ein Frame mit niedriger Priorität, welcher in einem Kommunikationszyklus nicht gesendet werden konnte, wird im nächsten Kommunikationszyklus, wenn möglich in seiner Slotnummer, gesendet. Das heißt, dass es möglich ist, dass Frames mit niedrigeren Prioritäten nie oder mit größerer Verzögerung gesendet werden.

Die Payload-Länge von Frames, die im dynamischen Segment übertragen werden, sind nicht fest, sondern können zur Laufzeit von der Anwendung geändert werden.

3.5.4 Das Symbol-Window (SWin)

Das Symbol-Window dient zur Übertragung von Symbolen. Über das Collision Avoidance Symbol zeigt ein FlexRay-Knoten den Start des ersten Kommunikationszyklus an. Das Media Test Symbol kommt beim Testen eines Busguardians und das WakeUp Symbol zum Wecken des FlexRay Clusters zum Einsatz. Es wurde keine Arbitrierung für das Symbol-Window vom Protokoll definiert. Wenn eine Arbitrierung in diesem Segment benötigt wird, soll diese von höheren Schichten realisiert werden.

3.5.5 Die Network Idle Time (NIT)

Während der Network Idle Time findet keine Kommunikation statt. Diese Zeit wird stattdessen vom Communication-Controller benutzt, um den Zeit-Sychronisationsalgorithmus durchzuführen. Die NIT wird zur Design-Zeit konfiguriert. Es ist wichtig, dass die NIT lang genug ist, damit alle FlexRay-Controller ihre Berechnungen innerhalb dieser Zeit durchführen können.

3.6 Das FlexRay-Frameformat

Das FlexRay-Frameformat wird in Abbildung 3.13 gezeigt. Ein FlexRay-Frame besteht aus drei Segmenten: Header, Payload und Trailer-Segment.

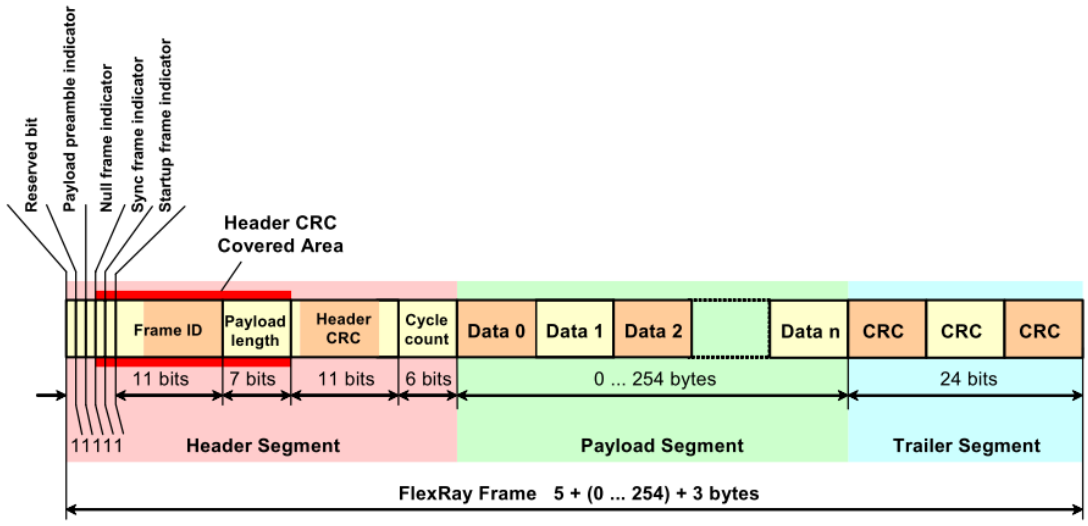


Abbildung 3.13: Das FlexRay-Frameformat (vgl. FlexRay Consortium, 2005b)

3.6.1 Das Header-Segment (5 Bytes)

Im Header-Segment werden Steuerinformationen übertragen, die zur reibungslosen Funktion des Protokolls notwendig sind. Das Header-Segment umfasst insgesamt 5 Bytes, welche ein Reserved-Bit, einen Payload Preamble Indicator(PPI), einen Null Frame Indicator (NFI), einen Sync Frame Indicator, einen Startup Frame Indicator, eine Frame-ID, eine Payload-Länge, einen Header-CRC und einen Zykluszähler (Cycle count) beinhalten.

Das **Reserved-Bit** soll für zukünftige Protokoll-Erweiterungen zur Verfügung stehen und von der Anwendung als logisches „0“ übertragen werden.

Der **Payload Preamble Indicator (PPI)** gibt an, ob bestimmte Daten im Payload als Steuer-Informationen übertragen werden. Ist der PPI auf eins gesetzt und der Payload im statischen Segment übertragen, dann befindet sich am Anfang der Payload ein Network Management Vector. Ist der PPI auf eins gesetzt und wird der Frame im dynamischen Segment gesendet, dann befindet sich am Anfang der Payload eine Nachricht-ID.

Der **Null Frame Indicator (NFI)** gibt an, ob ein Frame ein Null-Frame ist, das heißt ein Frame, der keine nutzbaren Daten in der Payload enthält. Die Payload wird stattdessen mit Nullen gefüllt.

Der **Sync Frame Indicator** signalisiert Frames, die für die Synchronisation des Systems verwendet werden sollen. Dieser Indikator darf nur von Sync-Knoten gesetzt werden.

Der **Startup Frame Indicator** signalisiert Frames, die in der Start-Phase des Netzwerks verwendet werden. Diese dürfen ausschließlich von speziellen Knoten gesendet werden, den Coldstart-Knoten.

Die **Frame-ID** hat eine Länge von 11 Bit und entspricht der Nummer des Slots, in dem der Frame übertragen wird. Eine Frame-ID darf nur einmal je Kanal in einem Zyklus benutzt werden. Eine Frame-ID mit dem Wert null ist reserviert, um ungültige Frames zu Kennzeichen.

Die **Payload-Länge** gibt die Anzahl der Bytes im Payload-Segment, also die Anzahl der Nutzdaten an. Die Nutzdaten eines FlexRay-Frames werden immer in 2-Byte-Wörtern übertragen. D. h., wenn zum Beispiel 20 im Payload-Länge-Feld eingetragen wird, dann ist die Payload 40 Bytes groß. Mit der maximal 7 Bit Payload-Länge können Längeangaben zwischen 0 und 127 definiert werden, was einer Payload zwischen 0 und 254 entspricht. Die Länge der Payload ist im statischen Segment innerhalb eines Clusters immer gleich groß, im dynamischen Segment hingegen variabel.

Der **Header-CRC** ist eine Summe, die über den Sync Frame Indicator, den Startup Frame Indicator, die Frame-ID und die Payload-Länge berechnet wird. Durch die CRC können während der Übertragung aufgetretene Veränderungen im Header-Segment vom Empfänger erkannt werden.

Der **Zykluszähler** ist die Zyklus-Nummer, in dem sich der sendende Knote befindet. Es sind Zyklus-Nummern von 0 bis 63 möglich.

3.6.2 Das Payload-Segment

Das Payload-Segment enthält die tatsächlichen Daten (Nutzdaten), welche die Netzwerk-Knoten miteinander austauschen. Diese werden in der Regel vom Host geschrieben und interpretiert.

3.6.3 Das Trailer-Segment

Im Trailer-Segment befindet sich der über den gesamten Frame (Header und Payload) berechnete CRC. Dieser dient zur Überprüfung der empfangenen Daten auf Übertragungsfehler.

4 Time-Triggered Ethernet

In diesem Kapitel wird das TTEthernet-Protokoll detailliert beschrieben. Dies stützt sich auf die zur Standardisierung eingereichte TTEthernet-Protokollspezifikation [1](vgl. Steiner, 2008) und die Literaturen Steinbach (2011) und Bartols (2010).

4.1 Allgemeines

TTEthernet ist eine Echtzeit-Erweiterung des Standard-Ethernet-Protokolls. Die Echtzeitfähigkeit beruht auf einem Verfahren, welches auf Zeitslots aufbaut (CTDMA -Coordinated Time Division Multiple Access-). Dabei wird ein Zeitplan vordefiniert, nach dem alle Teilnehmer operieren müssen. Hierfür wird für jeden Zyklus ein Sende-Zeitintervall für jede time-triggered Nachricht festgelegt. Erwähnenswert ist, dass diese Zeitintervalle sich nicht überschneiden dürfen für Nachrichten, die über den selben physikalischen Link übertragen werden. Somit wird die Kollision der Übertragung von zwei Echtzeit-Nachrichten verhindert. Bei einem CTDMA-Verfahren ist es wichtig, dass alle Teilnehmer miteinander synchronisiert sind. Daher werden diese im TTEthernet über ein ausfallsicheres Synchronisations-Protokoll mit einer globalen Zeit synchronisiert. TTEthernet ermöglicht nicht nur die Übertragung von Daten mit harten Echtzeitanforderungen (über die time-triggered Kommunikation), sondern auch mit weichen und ohne Echtzeit-Anforderungen (über die event-triggered Kommunikation). Dies wird anhand verschiedener Traffic-Klassen realisiert. Weiterhin ist es im TTEthernet wie bei FlexRay möglich, die Nachrichten über redundante Kommunikationskanäle zu übertragen.

TTEthernet entstand aus einem Projekt der Real-Time Systems Group (vgl. Real Time Systems Group (RTS)) der TU Wien aus dem Jahre 2004 und wird heute von der österreichischen Firma TTTech (vgl. TTTech Computertechnik AG) in modifizierter Form weiterentwickelt und kommerziell angeboten.

4.2 Mögliche TTEthernet-Topologien

TTEthernet basiert auf dem Standard Switched-Ethernet. Hierbei werden Endsysteme mit Switches über bidirektionale Links verbunden. Die Kommunikation zwischen Endsystemen

erfolgt, indem ein Sender-Endsystem die zu übermittelnde Nachricht an den Switch, an dem er angeschlossen ist, sendet und dieser leitet die Nachricht an alle Empfänger-Endsysteme weiter. Switches können auch miteinander über bidirektionale Links verbunden werden. In diesem Fall spricht man von einer Multi-Hop-Architektur oder kaskadierten Switches. Kommunikationslinks und Switches bilden zusammen Kommunikationskanäle zwischen den Endsystemen. Abbildung 4.1 zeigt eine Beispiel-Topologie mit einem Kommunikationskanal, drei kaskadierten Switches und sechs Endsystemen.

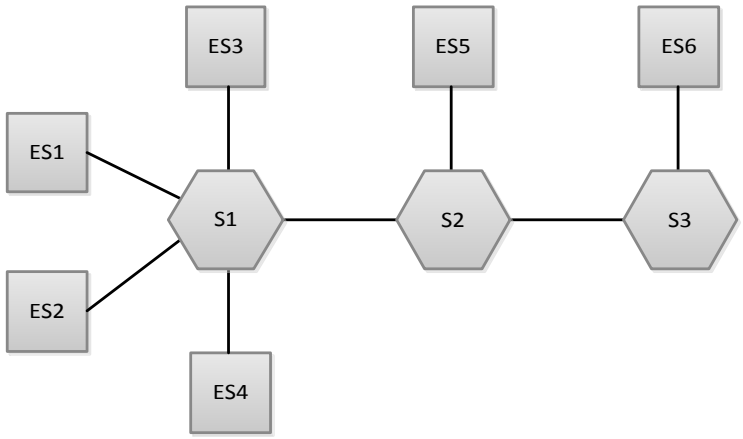


Abbildung 4.1: Singlekanal-TTEthernet-Topologie mit drei kaskadierten Switches (s1, s2 und s3) und sechs Endsystemen (ES1 bis ES6)

Eine redundante Vernetzung kann durch redundante Kommunikationskanäle zwischen den Endsystemen erreicht werden. Dabei besteht jedes Endsystem aus mindestens zwei physikalischen Schnittstellen, die jeweils an einem der redundanten Switches über einen Kommunikationslink verbunden sind. Ein Beispiel einer Dualkanal-TTEthernet-Topologie mit zwei kaskadierten Switches und drei Endsystemen wird in Abbildung 4.2 gezeigt.

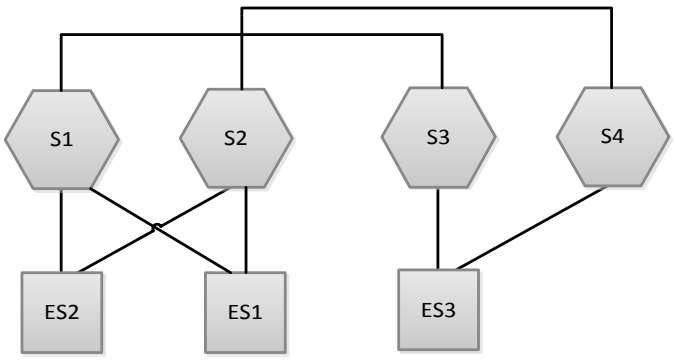


Abbildung 4.2: Dualkanal-TTEthernet-Topologie mit zwei kaskadierten Switches und drei Endsystemen

4.3 TTEthernet-Traffic-Klassen

TTEthernet definiert die drei folgenden Traffic-Klassen, die unterschiedliche Prioritäten aufweisen.

(i) **TT-Traffic (Time-Triggered Traffic)**: Dient zur Übertragung von Nachrichten (TT-Nachrichten) mit harten Echtzeit-Anforderungen. Diese unterliegen, wie bereits gesagt, einem festen Zeitplan (Schedule) und haben Vorrang vor allen anderen Nachrichten. Der Zeitplan wird zur Design-Zeit konfiguriert und definiert die Zeiten für das Senden, Weiterleiten und Empfangen von time-triggered Nachrichten. Dieser Zeitplan ist für alle TTEthernet-Geräte, also Endsystem und Switches, gültig. Das Routing von TT-Nachrichten ist statisch, was eine vollständig deterministische Nachrichten-Übertragung ermöglicht.

(ii) **RC-Traffic (Rate-Constrained Traffic)**: Dient zur Übertragung von Daten mit wenigen harten Echtzeit-Anforderungen (RC-Nachrichten). Für den RC-Traffic wird sichergestellt, dass ausreichend Bandbreite für die Übertragung zur Verfügung steht. Dafür wird für jede RC-Nachricht eine feste Bandbreite definiert, indem sogenannte BAGs (Bandwidth Allocation Gap) festgelegt werden. Diese definieren die minimale Zeit zwischen zwei RC-Nachrichten sowie die maximal Framelänge. Somit wird sichergestellt, dass Nachrichten mit einem „begrenzten Jitter“ übermittelt werden. Die sendende Applikation muss die Beschränkungen der BAG-Accounts einhalten. Andernfalls werden die Nachrichten als ungültig betrachtet und im Switch verworfen. Dieses Protokoll entspricht dem

AFDX (Avionics Full DupleX Switched Ethernet)-Protokoll (vgl. Aeronautical Radio Incorporated, 2009) und wird mit der zweiten Priorität behandelt.

Da die Weiterleitung von RC-Nachrichten nicht synchronisiert und von niedrigerer Priorität als die Weiterleitung von TT-Nachrichten ist, kann die exakte Sende- oder Weiterleitungszeit nicht vorhergesagt werden. Damit haben RC-Nachrichten eine unvorhersagbare Latenz und einen höheren Jitter als TT-Nachrichten. Dennoch kann für RC-Nachrichten anhand der festlegten BAGs eine obere Grenze für die Verzögerung berechnet werden. Damit sind die maximale Latenz und der maximale Jitter berechenbar. RC-Nachrichten eignen sich insbesondere für die Übertragung von asynchronen Echtzeitdaten (vgl. Steinbach, 2011).

(iii) **BE-Traffic (Best-Effort Traffic):** Dient zur Übertragung von Nachrichten ohne Echtzeit-Anforderungen (BE-Nachrichten). Dies entspricht dem Standard-Ethernet-Traffic und wird mit niedrigster Priorität weitergeleitet. Die Übertragung von BE-Nachrichten bietet keinerlei Garantie für die Laufzeit (Latenz) oder sogar die generelle Auslieferung von Nachrichten. BE-Nachrichten werden nur übertragen, nachdem alle bestehenden TT- und RC-Nachrichten übertragen wurden. Die Unterstützung des BE-Traffics ermöglicht es jedoch, Teilnehmer im TTEthernet-Netzwerken zu integrieren, die das time-triggered Protokoll nicht unterstützen. Diese Teilnehmer bleiben unsynchronisiert und kommunizieren ausschließlich über BE-Nachrichten.

Abbildung 4.3 zeigt ein Beispiel einer Automotive-Anwendung, in der die drei Nachrichten-Klassen (TT-, RC- und BE-Nachrichten) auf demselben physikalischen Link übertragen werden. Während TT-Nachrichten (beispielsweise für die Übertragung von Fahrwerksinformationen) unter strikter Einhaltung des Übertragungs-Zeitpunktes und mit der höchsten Priorität weitergeleitet werden, werden RC-Nachrichten in freien Zeitslots übertragen. Die danach noch freie Bandbreite wird für die Übertragung von BE-Nachrichten verwendet.

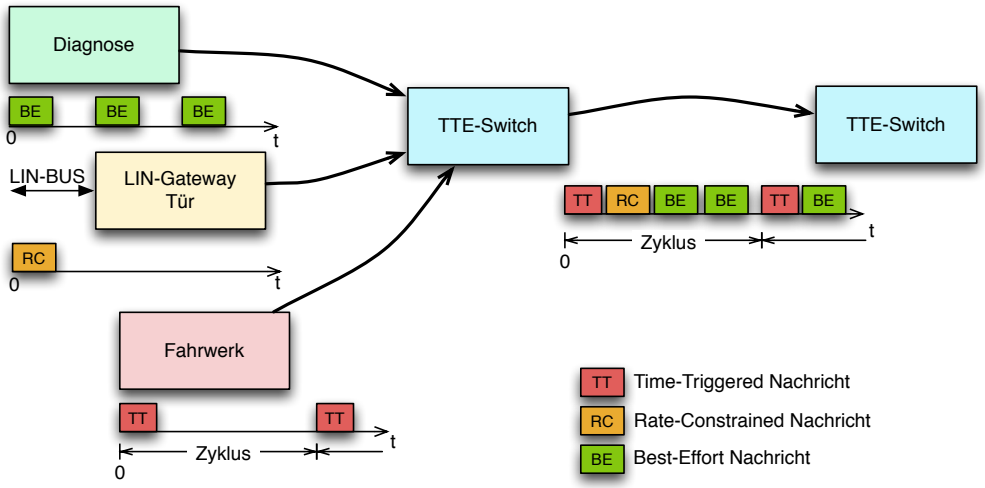


Abbildung 4.3: Beispiel einer TTEthernet-Anwendung – Fahrzeugkommunikation mit drei unterschiedlichen Trafficklassen (vgl. Steinbach, 2011, S. 23)

4.4 TTEthernet-Protokollstack

Der TTEthernet-Protokollstack [2] entspricht dem Standard-Ethernet-Protokollstack mit Erweiterungen der Echtzeiteigenschaften des TTEthernet-Protokolls. Der TTEthernet-Protokollstack besteht aus drei Schichten: die physikalische, die Data-Link- und die Applikationsschicht. Wobei die beiden letzten sich in weiteren Subschichten unterteilen, und zwar die MAC- (Media Access Control), TTEthernet-Protokoll-, API- und Applikations-Subschicht (siehe Abbildung 4.4).

Die physikalische Schicht und die MAC-Subschicht korrespondieren mit denen des Standard-Ethernet-Protokollstacks.

Die TTEthernet-Protokollschicht bzw. MAC-Relay-Unit gewährleisten die Echtzeitfähigkeit des TTEthernet-Protokolls. Dieses besteht aus einer Konfiguration, einem Scheduler und einem Klassifikations-Modul. Die Konfiguration wird offline durchgeführt und legt unter anderem fest, zu welchem Zeitpunkt Nachrichten gesendet/erwartet werden. Der Scheduler hat die Aufgabe, das Versenden/Empfangen der Nachrichten zu triggern. Dafür verwendet er die Nachrichtenplanung, welche Teil der Konfiguration ist. Das Klassifikations-Modul ordnet die Nachrichten beim Eintreffen ein und leitet diese bei der Anordnung des Schedulers weiter. Diese Module stellen den wesentlichen Unterschied zum Standard-Ethernet dar.

[2] Die hier angegebene Beschreibung des TTEthernet-Protokollstacks basiert auf der INET-Implementierung aus Steinbach u. a. (2011).

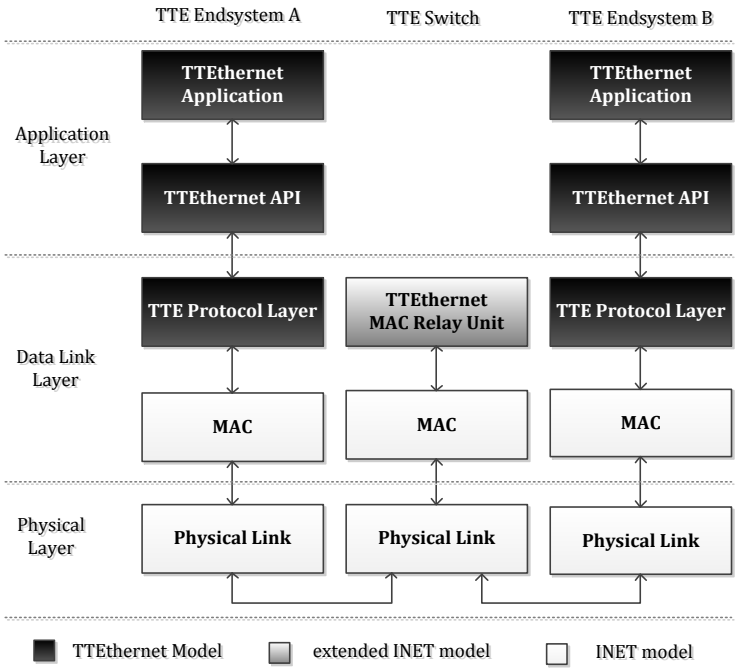


Abbildung 4.4: TTEthernet-Protokoll-Stack (vgl. Steinbach u. a., 2011)

Die TTEthernet-API ist eine von TTTech festgelegte Schnittstelle zum Initialisieren des Protokollstacks sowie Versenden/Empfangen von Nachrichten auf der Applikationsebene (vgl. TTTech Computertechnik AG, 2008).

4.5 Erkennung vom Echtzeit-Nachrichten im TTEthernet

Um die verschiedenen Nachrichten-Klassen zu unterscheiden, wird im TTEthernet die Ziel-Adresse des Standard-Ethernetframes, welcher normalerweise den Empfänger repräsentiert, anders interpretiert (siehe Abbildung 4.5). Das Zieladresse-Feld setzt sich aus einem 32-Bit-CT-Marker (Critical Traffic) und einer 16-Bit CT-ID zusammen. Der CT-Marker (z. B. *„0x03 0x04 0x05 0x06“*) dient zur Kennzeichnung von zeitkritischen Nachrichten (TT- oder RC-Nachrichten). Diese werden somit von BE-Nachrichten unterschieden. Durch Die CT-ID werden alle zeitkritischen Nachrichten eindeutig und systemweit identifiziert. Durch das 16-Bit große CT-ID-Feld wird es möglich, innerhalb eines TTEthernet-Systems bis zu 4096 verschiedene zeitkritische Nachrichten zu definieren. Der CT-Marker muss auf allen sich im System befindlichen Teilnehmern gleich sein.

Bei eingehenden Nachrichten im TTEthernet-Switch wird die Ziel-Adresse mit der Bitmaske *„0xFFFFFFFF0000“* maskiert, um den CT-Marker auszulesen. Entspricht das Ergebnis dem konfigurierten CT-Marker, handelt es sich um eine zeitkritische Nachricht. Anhand der CT-ID wird dann im Switch entschieden, ob es sich um einen TT-Frame oder RC-Frame handelt. Zuordnung von CT-IDs zu einer Zeitkritischen Klassen (TT oder RC) erfolgt zur Design-Zeit. Zeitkritische Nachrichten werden außerdem vom Standard-Ethernet-Nachrichten anhand des Typ-Felds unterschieden. IEEE hat für zeitkritischen Traffic ein spezielles Typ-Feld *0x88D7* reserviert.

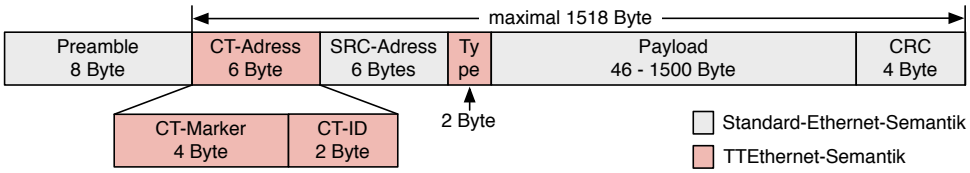


Abbildung 4.5: Format eines TTEthernet-Frames (vgl. Bartols, 2010, S. 32)

4.6 Scheduler-Verhalten im TTEthernet

Ein Scheduler kann preemptiv oder nicht preemptiv operieren.

Beim preemptiven Verhalten werden niederpriorisierte Nachrichten (z.B. RC- oder BE-Nachrichten) verdrängt, wenn der Zeitpunkt zur Übertragung einer höher priorisierten Nachricht (z. B. TT-Nachricht) eingetroffen ist. Somit wird sichergestellt, dass der zeitkritische Datenverkehr ohne Verzögerungen übermittelt wird. Die unterbrochene, niederpriorisierte Nachricht muss jedoch erneuert übertragen werden.

Der TTEthernet-Scheduler operiert nicht preemptiv. Dabei wird hingegen die Übertragung einer nicht priorisierten Nachricht (RC- oder BE-Nachrichten) beim Eintreffen einer TT-Nachricht nicht unterbrochen. Es wird jedoch sichergestellt, dass die Übertragung von TT-Nachrichten sich nicht mit der Übertragung von RC- oder BE-Nachrichten überschneidet. Der Scheduler weiß, wann als nächstes eine TT-Nachricht gesendet werden soll und erkennt an der Länge der zu übertragenden RC-/BE-Nachricht, dass eine Übertragung dieser nicht mehr möglich ist, ohne dass eine Überschneidung mit einer TT-Nachricht zu Stande kommt. Diese Nachricht wird solange blockiert, bis genug Zeit vorhanden ist, sie ohne Probleme zu senden.

5 Modellierung von verteilten Echtzeitsystemen mit Task-Graphen

In einer verteilten Embedded-System-Umgebung interagieren Tasks untereinander über Nachrichten, um bestimmte Funktionen zu realisieren. Jeder Task läuft auf einem Prozessor. Auf einem Prozessor können mehrere Tasks ausgeführt werden, wobei nur ein Task für eine bestimmte Zeit aktiviert werden kann. Die Prozessoren werden miteinander über ein oder mehrere Kommunikationsmedien verbunden. Der Zugriff auf das Kommunikationsmedium ist exklusiv, das heißt, dass nur ein Task zu einem bestimmten Zeitpunkt auf dieses zum Senden einer Nachricht zugreifen darf. So muss der Zugriff von Tasks auf das Kommunikationsmedium koordiniert werden. Dies kann erreicht werden, indem die verteilten Anwendungen als eine Menge von Task-Graphen modelliert werden und daraus der Schedule, welcher festlegt, wann welcher Task aktiviert werden kann, abgeleitet wird. Durch Task-Graphen kann außerdem das Zeitverhalten der verteilten Anwendung analysiert werden. In diesem Kapitel wird beschrieben, wie man verteilte Anwendungen mit Task-Graphen designen kann. Die verwendeten Methoden beruhen auf den Arbeiten Pop (2003), Nagarajan u. a. (2004), Sinnen (2007) und Buttazzo (2005), wobei Definitionen hauptsächlich auf Sinnen (2007) basieren.

5.1 Grundlegende Graphen-Konzepte

Für die Modellierung von Anwendungen mit Task-Graphen ist es zunächst wichtig, die grundlegenden Graphen-Konzepte zu beschreiben, auf denen die Task-Graphen-Konzepte aufgebaut werden. Die in diesem Abschnitt und im weiteren Verlauf des Kapitels verwendete Notation basiert auf der aus Cormen u. a. (2003).

Definition 5.1 (*Graph*)

Ein Graph G ist ein Tupel $(\mathbf{V}, \mathbf{E})$, wobei

(i) $\mathbf{V}$ *eine endliche Menge von Knoten (engl. vertices) ist.*

(ii) $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ *eine endliche Menge von Kanten (engl. edges) ist.*

Ein Element $v \in \mathbf{V}$ wird Knoten und ein Element $e \in \mathbf{E}$ Kante genannt. Eine Kante $e = (u, v)$ verbindet die Knoten u und $v \in \mathbf{V}$. Per Konvention wird die Notation $e_{[u,v]}$ für eine Kante zwischen den Knoten u und v verwendet.

*In einem **gerichteten Graph** hat eine Kante $e_{[u,v]}$ die gegebene Richtung von u nach v, also $e_{[u,v]} \neq e_{[v,u]}$. Solch eine Kante wird gerichtete Kante genannt.*

*In einem **ungerichteten Graph** hat eine Kante $e_{[u,v]}$ keine Richtung, also $e_{[u,v]} = e_{[v,u]}$. Solch eine Kante wird ungerichtete Kante genannt.*

Abbildung 5.1 zeigt ein Beispiel von zwei einfachen Graphen: Abbildung 5.1(a) einen ungerichteten und Abbildung 5.1(b) einen gerichteten. Die beiden Graphen haben die gleichen Mengen von Knoten $\mathbf{V} = \{a, b, c, d\}$ und Kanten
$\mathbf{E} = \{e_{[a,c]}, e_{[b,c]}, e_{[c,d]}, e_{[c,e]}, e_{[d,a]}, e_{[e,b]}\}$.

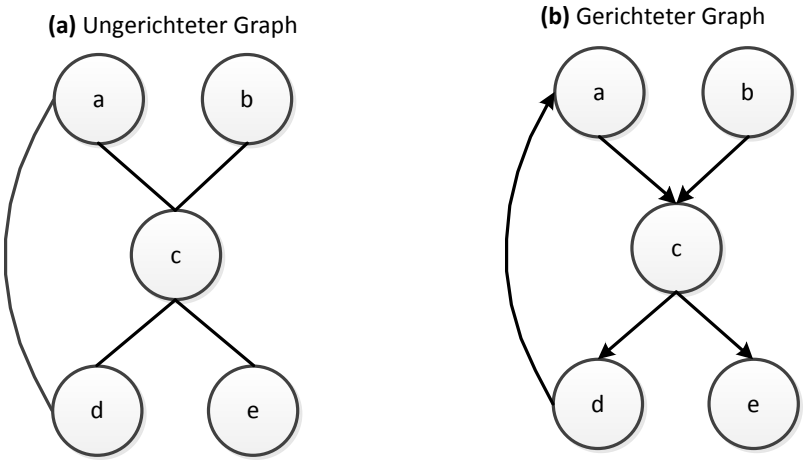


Abbildung 5.1: Beispiel von zwei einfachen Graphen: (a) ungerichtet und (b) gerichtet

Definition 5.2 (*Pfad*)

Sei $G = (\mathbf{V}, \mathbf{E})$ ein gerichteter Graph. Ein Pfad $path = \langle v_0, v_1, v_2, \ldots, v_k \rangle \in \mathbf{V}^{k+1}, k \in \mathbf{N}$ vom Knoten v_0 nach Knoten v_k ist eine Folge von Knoten, so dass diese mit den Kanten $e_{[v_{i-1},v_i]} \in \mathbf{E}$, für $i = 1, 2, \ldots, k$, verbunden sind. Ein Pfad vom Knoten v_0 nach Knoten v_k kann auch mit der Folge $\langle e_{[0,1]}, e_{[1,2]}, \ldots, e_{[(k-1),k]} \rangle$ der Kanten beschrieben werden. Die Länge eines Pfades ist die Anzahl der Kanten dieses Pfades.

Aus der Abbildung 5.1(b) ist z. B. $\langle a, c, d\rangle$ ein Pfad mit der Länge 2.

Definition 5.3 (*Knoten Beziehungen*)

In einem gerichteten Graph $G = (\mathbf{V}, \mathbf{E})$ *werden folgende Beziehungen definiert:*

(i) $\mathbf{pred}(v) = \{x \in \mathbf{V} : e_{[x,v]} \in \mathbf{E}\}$ *ist die Menge aller Vorgänger des Knotens* v*.*

(ii) $\mathbf{succ}(v) = \{x \in \mathbf{V} : e_{[v,x]} \in \mathbf{E}\}$ *ist die Menge aller Nachfolger des Knotens* v*.*

Aus der Abbildung 5.1(b) ist z. B. $\mathbf{prev}(c) = \{a, b\}$ und $\mathbf{succ}(c) = \{d, e\}$.

Definition 5.4 (*Quell- und Senke-Knoten*)

Sie $G = (\mathbf{V}, \mathbf{E})$ *ein gerichteter Graph. Die Menge der Quell- und Senke-Knoten wird wie folgt definiert.*

(i) $\mathbf{source}(G) = \{v \in \mathbf{V} : \mathbf{pred}(v) = \emptyset\}$ *ist die Menge der Quell-Knoten von* G*.*

(ii) $\mathbf{sink}(G) = \{v \in \mathbf{V} : \mathbf{succ}(v) = \emptyset\}$ *ist die Menge der Senke-Knoten von* G*.*

Aus der Abbildung 5.1(b) ist $\mathbf{source}(G) = b$ und $\mathbf{sink}(G) = e$.

5.2 Graph als Anwendungs-Modell

Eine Anwendung $\mathcal{A}$ kann mit einem Abhängigkeits-Graphen DG (Definition 5.5) abstrahiert werden. Dabei können zwei Arten von Aktivitäten von $\mathcal{A}$ dargestellt werden — Berechnungen und Kommunikationen. Die Berechnungen werden mit den Knoten/Tasks [1] und die Kommunikationen mit den Kanten des Graphen dargestellt. Der Inhalt einer Kommunikation wird mit einer Nachricht übertragen. Ein Task kann aus einer atomaren Operation bis komplexen zusammengesetzten Operationen wie Schleifen bestehen. Alle Operationen eines Tasks werden sequenziell durchgeführt. Für eine gegebene Task-Kante-Kombination kann zu einem bestimmten Zeitpunkt nur eine Aktivität durchgeführt werden — entweder die Berechnung oder die Kommunikation, wobei die Kommunikation immer nachdem die Berechnung abgeschlossen wurde durchgeführt werden kann. Eine Kante stellt Außerdem eine Abhängigkeits-Beziehung zwischen zwei Tasks dar.

[1] Im weiteren Verlauf der Arbeit wird nur noch der Begriff Task verwendet, um die Knoten eines Graphen zu Kennzeichen, da dieser zur Beschreibung einer Berechnung/Aufgabe geläufig und besser geeignet ist.

Man unterscheidet zwei Arten von Abhängigkeiten zwischen Tasks einer Anwendung — Nachrichten- und Steuerungs-Abhängigkeiten (vgl. Pop, 2003). Die letztere stellt die Steuerungsstruktur einer Anwendung dar, welche durch festgelegte Bedingungen etabliert wird, die zum Beispiel mit „if ... else"-ähnlichen Anweisungen abgefragt werden können, um den Fluss der Anwendung zu bestimmen. In dieser Arbeit werden jedoch ausschließlich Nachrichten-Abhängigkeiten in Betracht gezogen. Diese spiegeln die Abhängigkeiten zwischen Teilen einer Anwendung (Tasks), welche durch einen Nachrichten-Austausch verursacht wird, wider. Der Nachrichten-Abhängigkeits-Fluss wird durch die Kanten eines gerichteten Graphen dargestellt. Diese Abhängigkeiten legen außerdem das Kommunikations-Verhalten der Anwendung fest, welches für die Erzeugung eines effizienten Schedules gut verstanden werden soll. Die formale Definition eines Abhängigkeits-Graphen wird nachfolgend angegeben.

Definition 5.5 (*Abhängigkeits-Graph*)

Ein Abhängigkeits-Graph $DG = (\mathbf{T}, \mathbf{E}, \mathbf{M}, s_m)$ *ist ein Graph, welcher eine Anwendung* $\mathcal{A}$ *darstellt. Wobei*

(i) $(\mathbf{T}, \mathbf{E})$ *ein azyklisch gerichteter Graph ist.*

(ii) $\mathbf{T}$ *eine endliche Menge von Tasks von* $\mathcal{A}$ *ist* [2]*.*

(iii) $\mathbf{E}$ *eine endliche Menge der Nachrichten-Abhängigkeits-Beziehungen zwischen den Tasks ist.*

(iv) $\mathbf{M}$ *eine endliche Menge der Nachrichten von* $\mathcal{A}$ *ist.*

(v) $s_m : \mathbf{M} \rightarrow \mathbf{T}$ *die Funktion ist, welche eine Nachricht* $m \in \mathbf{M}$ *zu einem Task* $\tau \in \mathbf{T}$ *zuweist.*

Eine Kante $e_{[i,j]} \in \mathbf{E}$ vom Task τ_i nach τ_j, legt fest, dass die Berechnung des Tasks τ_j von der Nachricht des Tasks τ_i abhängt. Wenn eine Nachricht $m \in \mathbf{M}$ einem Task $\tau \in \mathbf{T}$ zugewiesen wird, dann sendet dieser Task diese Nachricht an alle seine Nachfolger $\mathbf{succ}(\tau)$. Alle Kanten, über die m gesendet wird, werden mit der Relation 5.1 und alle Empfänger von m mit der Relation 5.2 definiert.

$$e_m : \mathbf{M} \rightarrow 2^{\mathbf{E}} \tag{5.1}$$

[2] Im weiteren Verlauf der Arbeit wird nur noch $\mathbf{T}$ verwendet, um die Menge der Knoten/Tasks zu kennzeichnen. Alle Definition aus Abschnitt 5.1 sind weiterhin gültig, wobei $\mathbf{V}$ mit $\mathbf{T}$ ersetzt werden soll.

$$r_m : \mathbf{M} \rightarrow 2^{\mathbf{T}} \tag{5.2}$$

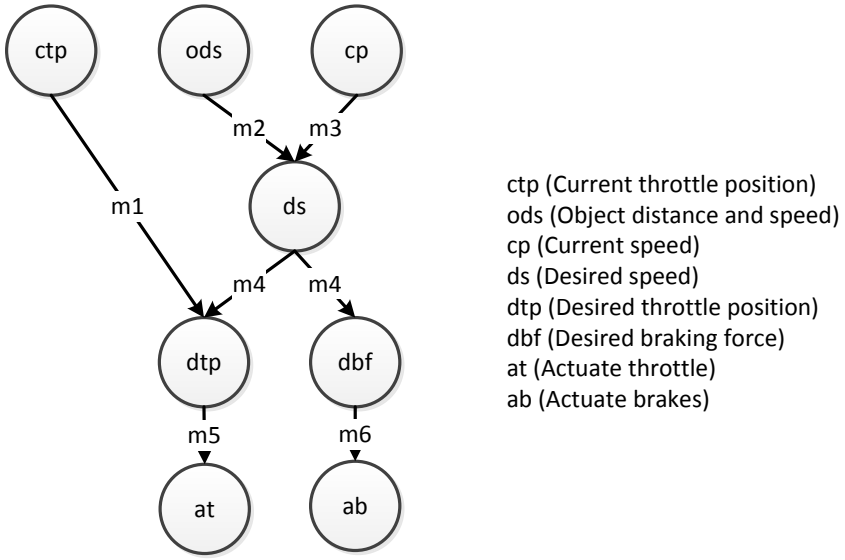


Abbildung 5.2: Adaptive Cruise Control (ACC) als Abhängigkeits-Graph (vgl. Nagarajan u. a., 2004)

In Abbildung 5.2 wird die Adaptive-Cruise-Control-Anwendung (ACC) als Abhängigkeits-Graph $DG = (\mathbf{T}, \mathbf{E}, \mathbf{M}, s_m)$ dargestellt. Wobei $\mathbf{T} = \{ctp, ods, cp, ds, dtp, dbf, at, ab\}$, $\mathbf{E} = \{e_{[ctp,dtp]}, e_{[ods,ds]}, e_{[cp,ds]}, e_{[ds,dtp]}, e_{[ds,dbf]}, e_{[dtp,at]}, e_{[dbf,ab]}\}$, $\mathbf{M} = \{m_1, m_2, m_3, m_4, m_5, m_6\}$ und z. B. $s_m(m_4) = ds, edg_m(m_4) = \{e_{[ds,dbf]}, e_{[ds,dtp]}\}, r_m(m_4) = \{dtp, dbf\}$. Die ACC-Anwendung hat als Aufgabe, eine sichere Distanz zwischen zwei Fahrzeugen einzuhalten. Dabei haben z. B. die Tasks ods und cp als Aufgabe, jeweils die Distanz und Geschwindigkeit des vorausfahrenden Fahrzeugs und die aktuelle Geschwindigkeit des Fahrzeugs zu berechnen bzw. auszulesen – Berechnungs-Aktivität – und diese an den Task ds über die Nachrichten m_2 und m_3 zu senden – Kommunikations-Aktivität –. Das heißt, dass die Berechnung von Task ds von den Nachrichten m_2 und m_3 abhängt. Der Task ds berechnet dann die gewünschte Geschwindigkeit und sendet diese sowohl an den Task dtp als auch den Task dbf mit der gleichen Nachricht m_4. Der Task dbf berechnet daraufhin die gewünschte Bremskraft und aktualisiert dann die Bremse (Task ab) mit der Nachricht m_6, während der Task dtp die gewünschte Gaspedal-Position berechnet, abhängig von der

aktuellen Gaspedal-Position, welche er vom Task ctp mit der Nachricht m_1 bekommt, und aktualisiert dann die Gaspedal-Position (Task at) mit der Nachricht m_5.

5.3 Topologie-Modell

Für die Task-Graph-Definition und das Knoten- bzw. Nachrichten-Scheduling soll die physikalische Struktur (Topologie-Modell) der Prozessoren bekannt sein. Es werden in dieser Arbeit zwei Arten von Topologien in Betracht gezogen — Bus- und Switch-Topologie (siehe hierfür auch die Abschnitte 3.3 und 4.2).

5.3.1 Bus-Topologie

Bei einer Bus-Topologie sind alle Prozessoren direkt mit demselben Übertragungsmedium (dem Bus) verbunden. Ein Bus ist eine ungerichtete Kante, welche mehrere Prozessoren miteinander verbindet. Abbildung 5.3 zeigt ein einfaches Beispiel eines Busses.

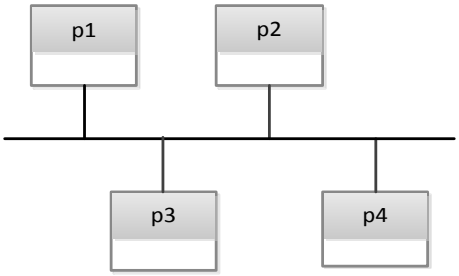


Abbildung 5.3: Beispiel einer Bus-Topologie mit vier Prozessoren (p_1 bis p_4)

5.3.2 Switch-Topologie

Ein Switch ist ein Netzwerk-Knoten, welcher Prozessoren miteinander verbindet. Um Switches in den Graphen darzustellen, wird ein neuer Typ von Knoten eingefügt — der Switch-Knoten. Abbildung 5.4 zeigt ein Beispiel von ungerichteten Graphen mit Switch-Knoten.

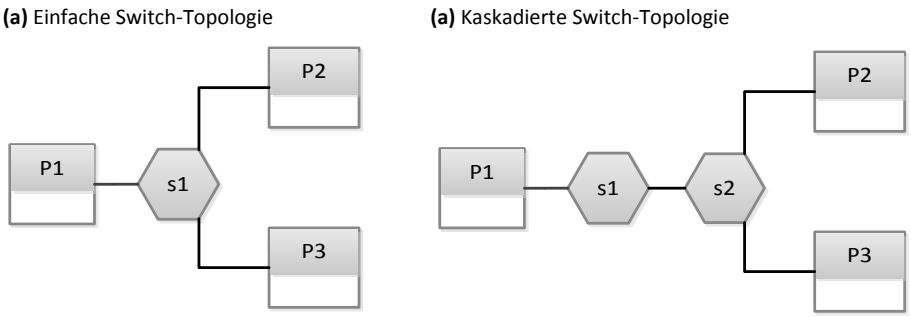


Abbildung 5.4: Beispiel von Netzwerk-Graphen mit Switch-Knoten. (a) Einfache Switch-Topologie mit einem Switch s_1 und drei Prozessoren (p_1, p_2 und p_3). (b) Kaskadierte Switch-Topologie mit zwei Switches und drei Prozessoren (p_1, p_2 und p_3)

5.3.3 Topologie-Graph

Die Definition eines Topologie-Graphen wird nachfolgend gegeben.

Definition 5.6 (*Topologie-Graph*)

Die Topologie eines Kommunikations-Netzwerks wird als ein Graph $TPG = (\mathbf{P}, \mathbf{S}, \mathbf{L}, \mathbf{B}, b)$ *modelliert, wobei*

(i) $(\mathbf{P} \cup \mathbf{S}, \mathbf{L} \cup \mathbf{B})$ *ein ungerichteter Graph ist. Mit* $\mathbf{P} \cap \mathbf{S} = \emptyset$, $\mathbf{L} \cap \mathbf{B} = \emptyset$.

(ii) $\mathbf{P}$ *eine endliche Menge von Prozessoren ist.*

(iii) $\mathbf{S}$ *eine endliche Menge von Switches ist.*

(iv) $\mathbf{L}$ *die Menge der physikalischen Kommunikations-Links zwischen Prozessoren aus* $\mathbf{P}$ *und Switches aus* $\mathbf{S}$ *oder zwischen Switches und Switches ist. Eine ungerichtete Kante* $l_{ij} \in \mathbf{L}$ *stellt einen bidirektionalen physikalischen Kommunikations-Link vom Netzwerk-Knoten* nn_i *nach Netzwerk-Knoten* nn_j, $nn_i, nn_i \in (\mathbf{P} \cup \mathbf{S})$, *(*$nn_i \in \mathbf{P} \wedge nn_j \in \mathbf{S}$*)* $\vee$ *(*$nn_i \in \mathbf{S} \wedge nn_j \in \mathbf{P}$*)* $\vee$ *(*$nn_i \in \mathbf{S} \wedge nn_j \in \mathbf{S}$*), dar.*

(v) $\mathbf{B}$ *eine endliche Menge von Bussen ist. Ein Bus kann mehrere Netzwerk-Knoten aus* $\mathbf{P}$ *miteinander verbinden.*

(vi) $b : (\mathbf{L} \cup \mathbf{B}) \rightarrow \mathbb{Q}^+$ *die Funktion ist, welche die relative Datenrate der Links und Busse festlegt.*

Definition 5.6 spiegelt alle Arten von Netzwerk-Topologien wider — direkte Links, Busse und Switches. Handelt es sich um eine reine Bus-Topologie, dann sind die Mengen $\mathbf{L}$ und $\mathbf{S}$ leer, während die Menge $\mathbf{B}$ bei reinem Switch-Netzwerk leer ist.

5.4 Task-Graph

Ein Task-Graph [3] ist eine Erweiterung eines Abhängigkeits-Graphen auf die Berechnungs- und Kommunikations-Kosten. Task-Graphen werden für das Scheduling von Tasks und Nachrichten verwendet. Die formale Definition eines Task-Graphen wird nachfolgend gegeben.

Definition 5.7 (*Task-Graph*)

Ein Task-Graph ist ein Graph $TG = (\mathbf{T}, \mathbf{E}, \mathbf{M}, s_m, t_W, c)$ *, welcher eine Anwendung* $\mathcal{A}$ *darstellt. Wobei:*

(i) $(\mathbf{T}, \mathbf{E}, \mathbf{M}, s_m)$ *ein Abhängigkeits-Graph ist.*

(ii) $t_W : \mathbf{T} \to \mathbb{Q}^+$ *die Berechnung-Kosten-Funktion der Tasks* $\tau \in \mathbf{T}$ *ist.*

(iii) $c : \mathbf{M} \to \mathbb{N}$ *die Kommunikations-Kosten-Funktion der Nachrichten* $m \in \mathbf{M}$ *ist.*

Die Berechnungs-Kosten $t_W(\tau)$ des Tasks τ ist die Zeit, welche der Task τ für seine Ausführung auf einem Prozessor aus $\mathbf{P}$ benötigt.

Die Kommunikations-Kosten $c(m)$ der Nachricht m ist die Anzahl der Datenmenge, die mit dieser Nachricht übertragen werden soll.

Abbildung 5.5 zeigt den Task-Graph der ACC-Anwendung der Abbildung 5.2. Die Tasks werden durch Kreise dargestellt und mit der Kurz-Beschreibung der dargestellten Funktion genannt. Die Berechnungs- und Kommunikations-Kosten werden in Klammer neben bzw. in dem entsprechenden Graph-Element jeweils in μs und $Byte$ angegeben. Man hat z. B. $t_W(ctp) = 175\mu s$ und $c(m_1) = 1Byte$.

[3] Stellt das in Kapitel 2 angesprochene Funktionsmodell dar. Die beiden Begriffe Task-Graph und Funktionsmodell werden in dieser Arbeit synonym verwendet.

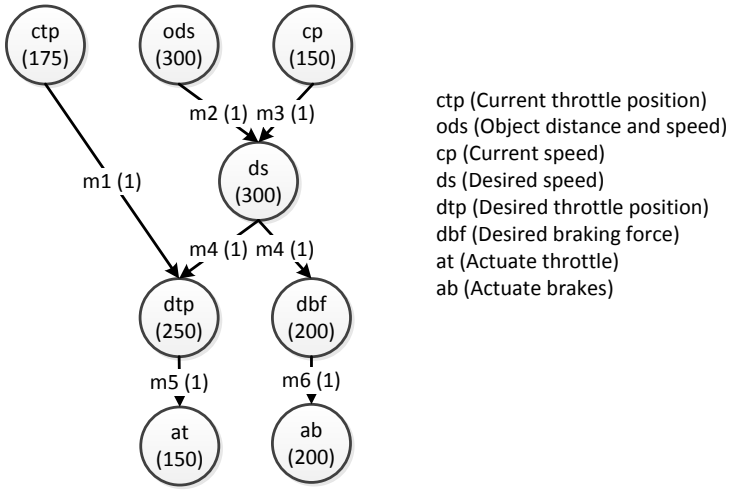


Abbildung 5.5: Task-Graph der ACC-Anwendung

5.5 System-Modell

Da die Definitionen des Task-Graphen und Topologie-Modells feststehen, kann nun die Definition des System-Modells darauf basierend gegeben werden.

Definition 5.8 (*Verteiltes System*)

Ein verteiltes System $\Gamma = (\mathbf{TG}, TPG, T_{CC})$ *ist ein Tripel, wobei*

(i) $\mathbf{TG}$ *eine endliche Menge von Task-Graphen ist.*

(ii) TPG *ein Topologie-Modell der Task-Graphen aus* $\mathbf{TG}$ *ist.*

(iii) $T_{CC} \in \mathbb{R}^+$ *die Dauer des Kommunikationszyklus ist.*

Der Kommunikationszyklus ist das Zeitintervall, in dem jeder Task-Graph $TG \in \mathbf{TG}$ mindestens einmal ausgeführt wird.

Ein verteiltes System hat in diesem Kontext folgende Eigenschaften:

(i) **Dediziertes System**. Die Tasks aus den Task-Graphen $\mathbf{TG}$ werden auf die Prozessoren aus dem Topologie-Modell TPG gemappt. Diese Prozessoren stehen dann ausschließlich für die Ausführung der ihnen zugewiesenen Tasks zur Verfügung. Kein anderes

Programm oder keine anderen Tasks sollen auf dem System ausgeführt werden während die geplanten Task-Graphen ausgeführt werden.

(ii) **Zyklische Ausführung der Task-Graphen**. Die Task-Graphen aus $\mathbf{TG}$ werden zyklisch innerhalb eines Kommunikationszyklus ausgeführt.

(iii) **Exklusive Prozessor-Zuweisung**. Ein Prozessor $p \in \mathbf{P}$ kann nur einen Task gleichzeitig ausführen und die Ausführung dieses Tasks ist nicht preemptiv.

(iv) **Keine Übertragungszeit bei lokaler Kommunikation**. Bei einer lokalen Kommunikation, also eine Kommunikation zwischen Tasks, die auf dem gleichen Prozessor ausgeführt werden, wird die Übertragungszeit der Nachrichten vernachlässigt.

(v) **Statisches Routing**. Die Route für eine Nachricht von einem Quell- zu einem Ziel-Prozessor aus $\mathbf{P}$ wird zur Design-Zeit festgelegt und ändert sich nicht zur Laufzeit.

5.6 Task-Scheduling

Das Framework für das Task-Scheduling wird in diesem Abschnitt beschrieben. Die Grundlage hierfür sind die Task-Graphen- und Topologie-Modell-Festlegungen aus den vorherigen Abschnitten. Die Aufgabe des Task-Schedulings ist es, eine mögliche räumliche und zeitliche Zuordnung der Tasks zu finden. Bei der räumlichen Zuordnung wird jeder Task einem Prozessor zugewiesen (siehe Definition 5.9) und bei der zeitlichen Zuordnung wird jedem Task eine Startzeit zugewiesen (siehe Definition 5.10). Hier wird lediglich auf das statische Task-Scheduling eingegangen. Dieses wird zur Design-Zeit festgelegt, im Gegensatz zum dynamischen Scheduling, welches zur Laufzeit bestimmt wird.

Definition 5.9 (*Prozessor-Zuweisung*)

Seien $TG = (\mathbf{T}, \mathbf{E}, \mathbf{M}, s_m, t_W, c)$ ein Task-Graph und $TPG = (\mathbf{P}, \mathbf{S}, \mathbf{L}, \mathbf{B}, b)$ eine Topologie. Die Zuweisung eines Tasks $\tau \in \mathbf{T}$ zu einem Prozessor $p \in \mathbf{P}$ wird durch die Prozessor-Zuweisungs-Funktion $proc : \mathbf{T} \to \mathbf{P}$ definiert. $proc(\tau) = p, \tau \in \mathbf{T}$ ist der Prozessor, auf dem der Task τ ausgeführt wird.

Definition 5.10 (*Task-Startzeit*)

Die Startzeit eines Tasks ist der Zeitpunkt, an dem seine Ausführung auf seinem Prozessor startet. Diese wird mit der folgenden Funktion definiert.

- $t_{s\tau} : \mathbf{T} \rightarrow \mathbb{R}_0^+$.

Damit das Senden einer Nachricht geplant werden kann, soll bekannt sein, wann die Ausführung ihres Tasks abgeschlossen wird. Das Ende der Ausführung eines Tasks wird nachfolgend definiert.

Definition 5.11 (*Task-Endzeit*)

Die Endzeit eines Tasks ist der Zeitpunkt, an dem seine Ausführung auf seinem Prozessor abgeschlossen wird. Diese kann durch die folgende Gleichung berechnet werden.

$$t_{f\tau}(\tau) = t_{s\tau}(\tau) + t_W(\tau). \tag{5.3}$$

Aufgrund der Eigenschaft (iii) des Systems (siehe Abschnitt 5.5) soll sichergestellt werden, dass zwei Tasks den gleichen Prozessor nicht zur gleichen Zeit belegen. Dies wird durch die folgende Bedingung garantiert.

Bedingung 5.1 (*Exklusive Prozessor-Zuweisung*)

Sei $TG = (\mathbf{T}, \mathbf{E}, \mathbf{M}, s_m, t_W, c)$ Task-Graph und $TPG = (\mathbf{P}, \mathbf{S}, \mathbf{L}, \mathbf{B}, b)$ eine Topologie. Für jede zwei Tasks $\tau_i, \tau_j \in \mathbf{T}$, gilt

$$proc(\tau_i) = proc(\tau_j) \Rightarrow \begin{cases} & t_{s\tau}(\tau_i) < t_{f\tau}(\tau_i) \leq t_{s\tau}(\tau_j) < t_{f\tau}(\tau_j) \\ \text{oder} & t_{s\tau}(\tau_j) < t_{f\tau}(\tau_j) \leq t_{s\tau}(\tau_i) < t_{f\tau}(\tau_i). \end{cases} \tag{5.4}$$

5.7 Nachrichten-Scheduling

Eine Nachricht m_i wird vom Task, zu dem diese zugewiesen wurde, gesendet. Der Sendezeitpunkt der Nachricht soll jedoch beim statischen Scheduling vorab festgelegt werden (Definition 5.12). Man soll jedoch bedenken, dass eine Nachricht erst dann gesendet werden kann, wenn ihr Task seine Berechnung abgeschlossen hat (siehe Bedingung 5.2).

Definition 5.12 (*Nachricht-Startzeit*)

Die Startzeit einer Nachricht ist der Zeitpunkt, an dem ihre Übertragung gestartet wird. Diese wird mit der folgenden Funktion definiert.

- $t_{sm} : \mathbf{M} \rightarrow \mathbb{R}^+$.

Bedingung 5.2 (*Nachricht-Startzeit-Einschränkung*)

Die Übertragung einer Nachricht m_i kann nur gestartet werden, wenn ihr Sender-Task seine Berechnung abgeschlossen hat. Dies wird durch die folgende Bedingung gesichert.

$$t_{sm}(m_i) \geq t_{f\tau}(s_m(m_i)) \tag{5.5}$$

Der Empfänger einer Nachricht kann seine Berechnung nur dann starten, wenn die Übertragung der Nachricht abgeschlossen ist. Daher ist als nächstes die Endzeit einer Nachricht zu beschreiben. Diese wird durch die Definition 5.13 gegeben.

Definition 5.13 (*Nachricht-Endzeit*)

Die Endzeit einer Nachricht ist der Zeitpunkt, an dem ihre Übertragung abgeschlossen wird. Also der Zeitpunkt, zu dem diese bei allen Empfängern angekommen ist. Diese kann mit der folgenden Gleichung ermittelt werden:

$$t_{fm}(m_i) = t_{sm}(m_i) + \zeta_B(m_i) \tag{5.6}$$

wobei:

- $\zeta_B(m_i) \in \mathbb{R}^+$ *die maximale Übertragungszeit der Nachricht* m_i *vom Sender-Task* s_m *zu allen Empfänger-Tasks* r_m *ist.*

Die Übertragungszeit einer Nachricht kann bei einer Bus-Topologie mit der Gleichung 5.7 und bei einer Switch-Topologie [4]mit der Gleichung 5.8 ermittelt werden. Bei Switch-Topologien sollen außerdem die Routen der Nachricht bekannt sein, damit die Verzögerungen aller Links und Switches, welche die Nachricht traversiert, bei der Berechnung der Übertragungszeit berücksichtigt werden. Daher wird die Definition einer Route zunächst gegeben.

[4]Es wird angenommen, dass die Switches nach dem Store-and-Forward-Prinzip operieren

Definition 5.14 (*Route*)

Sei $TPG = (\mathbf{P}, \mathbf{S}, \mathbf{L}, \mathbf{B}, b)$ eine Topologie. Eine Route von einem Quell-Prozessor p_{src} zu einem Ziel-Prozessor p_{dst}, $p_{src}, p_{dst} \in \mathbf{P} \wedge p_{src} \neq p_{dst}$, ist eine Liste von gerichteten physikalischen Kommunikationslinks $R = \{l_{[p_{src},s_1]}, l_{[s_1,s_2]}, \ldots, l_{[s_n,p_{dst}]}\}, s_i \in \mathbf{S}, p_{src}, p_{dst} \in \mathbf{P}$. Die Route einer Nachricht kann auch durch die Netzwerk-Knoten, welche diese traversiert, beschrieben werden $R = \{p_{src}, s_1, s_2, \ldots, s_n, p_{dst}\}, s_i \in \mathbf{S}, p_{src}, p_{dst} \in \mathbf{P}$.

Die Übertragungszeit $\zeta_B(m_i)$ in μs einer Nachricht m_i bei einer Bus-Topologie kann mit der folgenden Gleichung ermittelt werden:

$$\zeta_B(m_i) = (d(m_i) \times 8) \times t_b(B_i) \tag{5.7}$$

wobei:

- $d(m_i) = c(m_i) + O_p$ in $Byte$ die gesamte Datenmenge ist, die mit der Nachricht m_i übertragen wird, inklusive dem Protokoll-Overhead O_p. Das Protokoll-Overhead ist die Anzahl der im Header und Trailer übertragenen Datenmenge.
- $t_b(B_i) = \dfrac{1}{b(B_i)}$ in $\dfrac{\mu s}{bit}$ die Bitzeit des jeweiligen Busses B_i ist. Mit $b(B_i)$ die Bandbreite in $\frac{Mbit}{s}$ von B_i. Nimmt man z. B. eine Bandbreite von $b(B_i) = 10\frac{Mbit}{s}$ an, dann ist $t_b = \dfrac{1}{10\frac{Mbit}{s}} = 0,1\dfrac{\mu s}{bit}$.

Die Übertragungszeit $\zeta_{SWh}(m_i)$ in μs einer Nachricht m_i bei einer heterogenen Switch-Topologie kann mit der folgenden Gleichung ermittelt werden:

$$\zeta_{SWh}(m_i) = \max_{\forall R_{m_i} \in \mathbf{R_{m_i}}} \{(d(m_i) \times 8) \times \sum_{\forall l_i \in R_{m_i}} t_b(l_i) + \sum_{\forall s_i \in R_{m_i}} t_{DS}(s_i)\} \tag{5.8}$$

wobei:

- $\mathbf{R_{m_i}}$ die Menge der Routen der Nachricht m_i ist.
- $t_b(l_i) = \dfrac{1}{b(l_i)}$ in $\dfrac{\mu s}{bit}$ die Bitzeit des jeweiligen Links l_i ist (vgl. Gleichung 5.7).
- $t_{DS} : \mathbf{S} \rightarrow \mathbb{Q}^+$ die Funktion ist, welche die Zeit festlegt, die der jeweilige Switch für die Bearbeitung der Nachricht m_i vom Ein- zum Ausgangsport benötigt.

Handelt es sich jedoch um eine homogene Switch-Topologie, also wenn alle Links die gleiche Datenrate $b(l)$ und alle Switches die gleiche Verarbeitungszeit $t_{DS}(s)$ haben, dann

reduziert sich die Gleichung 5.8 auf die folgende Gleichung. Wobei R_{m_i} die Menge der von m_i traversierten Links ist.

$$\zeta_{SW}(m_i) = \max_{\forall R_{m_i} \in \mathbf{R_{m_i}}} \{(\zeta_B(m_i) \times |R_{m_i}|) + (t_{DS}(s) \times (|R_{m_i}| - 1))\} \tag{5.9}$$

5.8 Abhängigkeits-Einschränkung

Nach der Abhängigkeits-Beziehung zwischen Tasks eines Abhängigkeits-Graphen (Definition 5.5) wird von einem Task τ_j verlangt, auf die Ankunft aller eingehenden Nachrichten $m_i \in \mathbf{M}, s_m(m_i) \in \mathbf{pred}(\tau_j)$ zu warten. Der früheste Zeitpunkt, an dem ein Task τ_j seine Ausführung starten kann, wird DRT (Data Ready Time) genannt. Diese kann anhand der folgenden Gleichung ermittelt werden.

$$t_{dr}(\tau_j) = \max_{\forall s_m(m_i) \in \mathbf{pred}(\tau_j)} \{t_{fm}(m_i)\} \tag{5.10}$$

Wenn $\mathbf{pred}(\tau_j) = \emptyset$, τ_j, also ein Quell-Task (siehe Definition 5.4) ist, dann ist $t_{dr}(\tau_j) = 0$.

Nun kann die Bedingung angegeben werden, welche garantiert, dass ein Schedule eines Task-Graphen die Abhängigkeits-Einschränkung des Task-Graphen erfüllt.

Bedingung 5.3 (*Abhängigkeits-Einschränkung*)

Sei $TG = (\mathbf{T}, \mathbf{E}, \mathbf{M}, s_m, t_W, c)$ *ein Task-Graph. Für alle* $\tau_j \in \mathbf{T}$*, gilt*

$$t_{s\tau}(\tau_j) \geq t_{dr}(\tau_j). \tag{5.11}$$

Die Bedingung 5.3 verlangt es, den Start von Task τ_j zu verzögern, bis alle eingehenden Nachrichten vollständig bei seinem Prozessor $proc(\tau_j)$ angekommen sind. Außerdem garantiert diese Bedingung die Ausführung der Tasks in deren Abhängigkeits-Beziehung.

5.9 Release-Zeiten und Deadlines eines Task-Graphen und deren Elementen

In diesem Abschnitt werden die Startzeit und Deadline eines Task-Graphen, die Release-Zeit und Deadline eines Tasks und einer Nachricht definiert. Diese sind neben den in den vorherigen Abschnitten besprochenen Start- und Endzeiten unabdingbar für das Scheduling. Aufgrund der Konzentration auf das Kommunikationsmodell der Anwendungen wird im Rahmen dieser Arbeit meistens nur auf die Deadlines des Task-Graphen und der Nachrichten eingegangen und die der Tasks unterlassen.

Definition 5.15 (*Task-Graph-Startzeit*)

Sei $\Gamma = (\mathbf{TG}, TPG, T_{CC})$ *ein verteiltes System. Die Startzeit eines Task-Graphen* $TG \in \mathbf{TG}$ *ist der Zeitpunkt, an dem seine Ausführung startet. Das heißt auch, dass ein Task* $\tau \in \mathbf{T}$ *seine Ausführung erst ab diesem Zeitpunkt starten kann. Die Startzeit eines Task-Graphen wird durch die folgende Funktion definiert.*

- $t_{sg} : \mathbf{TG} \rightarrow \mathbb{R}^+$.

Definition 5.16 (*Task-Graph-Deadline*)

Die Deadline eines Task-Graphen ist der Zeitpunkt, an dem die Ausführungen aller Senke-Tasks (siehe 5.4) des Task-Graphen $\mathbf{sink}(TG)$ *abgeschlossen sein müssen. Die Deadline eines Task-Graphen wird durch die folgende Funktion definiert.*

- $t_{dg} : \mathbf{TG} \rightarrow \mathbb{Q}^+$.

Die zeitlichen Eigenschaften eines Task-Graphen werden in der Abbildung 5.6 gezeigt.

Definition 5.17 (*Task-Release-Zeit*)

Die Release-Zeit (Aktivierungszeitpunkt) eines Tasks $\tau \in \mathbf{T}$ *ist der Zeitpunkt, an dem dieser zur Ausführung bereit ist. Diese wird durch die folgende Funktion definiert.*

- $t_{r\tau} : \mathbf{T} \rightarrow \mathbb{R}^+$.

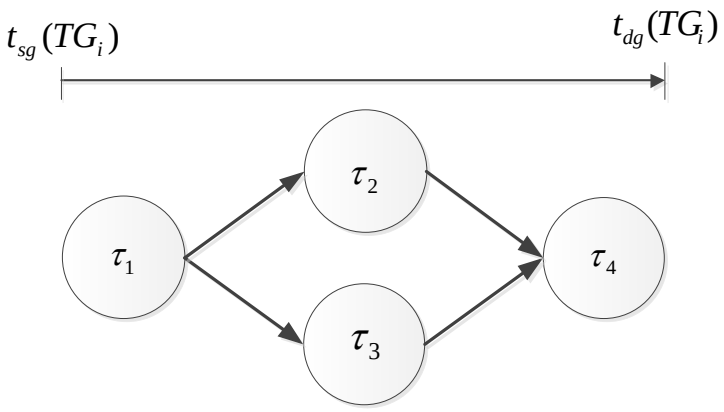


Abbildung 5.6: Zeitliche Eigenschaften eines Task-Graphen

Definition 5.18 (*Task-Deadline*)

Die Deadline eines Tasks $\tau \in \mathbf{T}$ ist der Zeitpunkt, an dem seine Ausführung abgeschlossen sein muss. Diese wird durch die folgende Funktion definiert.

- $t_{d\tau} : \mathbf{T} \rightarrow \mathbb{R}^+$.

Definition 5.19 (*Nachricht-Release-zeit*)

Die Release-Zeit (Aktivierungszeitpunkt) einer Nachricht $m \in \mathbf{M}$ ist der Zeitpunkt, an dem diese zur Übertragung bereit steht. Diese wird durch die folgende Funktion definiert.

- $t_{rm} : \mathbf{M} \rightarrow \mathbb{R}^+$.

Definition 5.20 (*Nachricht-Deadline*)

Die Deadline einer Nachricht ist der Zeitpunkt, an dem ihre Übertragung bei allen Empfängern angekommen sein muss. Diese wird durch die folgende Funktion definiert.

- $t_{dm} : \mathbf{M} \rightarrow \mathbb{R}^+$.

Die zeitlichen Eigenschaften eines Tasks und einer Nachricht werden in der Abbildung 5.7 zusammengefasst.

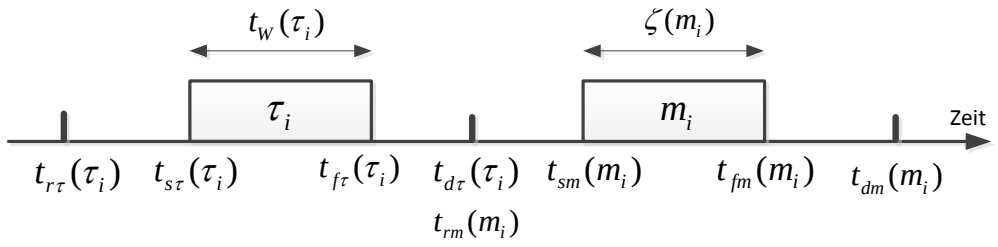


Abbildung 5.7: Zeitliche Eigenschaften eines Tasks und einer Nachricht

5.10 Deadline-Einhaltung

Die Deadlines eines Task-Graphen werden eingehalten, wenn folgende Bedingung erfüllt ist.

Bedingung 5.4 (*Deadlines-Einhaltung*)

Seien $TG = (\mathbf{T}, \mathbf{E}, \mathbf{M}, s_m, t_W, c)$ *ein Task-Graph. Für alle Nachrichten* $m_i \in \mathbf{M}$ *und alle Tasks* $\tau_i \in \mathbf{sink}(G)$*, gilt:*

$$t_{fm}(m_i) \leq t_{dm}(m_i), \tag{5.12}$$

$$\textit{und} \quad \max_{\forall \tau_i \in \mathbf{sink}(G)} \{t_{f\tau}(\tau_i)\} \leq t_{dg}(G). \tag{5.13}$$

5.11 Ausführbarer Schedule

Die drei Bedingungen 5.1 (exklusive Prozessor-Zuweisung), 5.3 (Abhängigkeits-Einschränkung) und 5.4 (Deadlines-Einhaltung) sind die essenziellen Einschränkungen, welche einem Schedule $\mathcal{S}$ eines Task-Graphen $TG = (\mathbf{T}, \mathbf{E}, \mathbf{M}, s_m, t_W, c)$ unterliegen. Ein Schedule eines Task-Graphen, welcher diese Bedingungen erfüllt, hält die Abhängigkeits-Einschränkung und die Deadlines des Task-Graphen sowie die Eigenschaften des System-Modells ein. Solch ein Schedule ist ein ausführbarer Schedule.

Definition 5.21 (*Ausführbarer Schedule*)

Sei $\mathcal{S}$ ein Schedule eines Task-Graphen $TG = (\mathbf{T}, \mathbf{E}, \mathbf{M}, s_m, t_W, c)$. $\mathcal{S}$ ist ausführbar nur dann, wenn alle Tasks $\tau \in \mathbf{T}$ die Bedingungen 5.1 und 5.3 und alle Nachrichten $m \in \mathbf{M}$ und der Task-Graph die Bedingung 5.4 erfüllen. Wenn die Abhängigkeits-Beziehungen zwischen den Tasks nicht berücksichtigt werden, dann sind die Bedingungen 5.1 und 5.4 ausreichend.

5.12 Kritischer Pfad eines Task-Graphen und Schlupf einer Nachricht

In diesem Abschnitt werden Begriffe wie Länge eines Pfades, kritischer Pfad und Schlupf eingeführt, welche für den Deadlineverteilungs-Algorithmus (nächster Abschnitt) benötigt werden.

Der kritische Pfad eines Task-Graphen $TG = (\mathbf{T}, \mathbf{E}, \mathbf{M}, s_m, t_W, c)$ ist der Pfad mit der längsten Länge. Mit der Gleichung 5.14 kann die Länge eines Pfades und der Gleichung 5.15 der kritische Pfad eines Task-Graphen bestimmt werden.

$$len(path) = \sum_{\tau \in path, \mathbf{T}} t_W(\tau) + \sum_{m \in path, \mathbf{M}} \zeta_B(m) \tag{5.14}$$

$$cpath = \{path \in TG \mid len(path) = \max_{path \in TG} \{len(path)\}\} \tag{5.15}$$

Der Schlupf einer Nachricht (eng. slack) ist die maximale Zeit, um die die Übertragung dieser verzögert werden kann, ohne dass die Nachricht ihre Deadline verpasst. Der Schlupf der Nachrichten eines Pfades $path$ kann anhand der Gleichung 5.16 ermittelt werden. Dabei wird der Schlupf des Pfades zwischen allen Nachrichten gleichmäßig verteilt, indem die Ausführungszeiten der Tasks und die Übertragungszeiten der Nachrichten aufaddiert werden und durch die Anzahl der Kanten des Pfades $size_{path}$ geteilt werden.

$$slack = \left\lfloor \frac{t_{dg}(TG_i) - (\sum_{\tau_i \in path} t_W(\tau_i) + \sum_{m_i \in path} \zeta_B(m_i))}{size_{path}} \right\rfloor \tag{5.16}$$

5.13 Deadline-Verteilung

Die Definition 5.21 legt unter anderem fest, dass ein Schedule $\mathcal{S}$ eines Task-Graphen nur dann ausführbar ist, wenn die Deadlines aller Nachrichten eingehalten werden. An dieser Stelle kann man sich die Frage stellen: Wie werden denn diese Deadlines bestimmt? Diese werden in der Praxis aus erprobten Systemen, Erfahrungen des System-Designers oder intensiven Simulationen bestimmt (Schmidt, 2011). Aufgrund dessen, dass die Deadlines aus der Praxis nicht immer zur Verfügung stehen und aufgrund der analytischen Neigung dieser Arbeit werden hier die Deadlines teilweise anhand des Deadline-Verteilungs-Algorithmus (siehe Algorithmus 1) bestimmt (vgl. Jonsson und Shin, 1997; Natale und Stankovic, 1994). Dieser nimmt als Eingabe einen Task-Graphen an und gibt einen Task-Graphen mit festgelegten Release-Zeiten und Deadlines der Nachrichten zurück. Hierfür werden zunächst die zeitlichen Eigenschaften (Startzeit und Deadline) des Task-Graphen festgelegt. Die Release-Zeiten der Quell-Tasks bekommen dann die Startzeit des Task-Graphen und die Deadlines der Senke-Tasks die Deadline des Task-Graphen zugewiesen. Als nächstes wird der Prozess zur Verteilung der Deadlines gestartet und der erste Schritt dabei ist es, den kritischen Pfad $cpath$ des Task-Graphen zu suchen und den Schlupf ($slack$) der Nachrichten dieses Pfades zu berechnen. Anschließend werden die Deadlines und Release-Zeiten aller Tasks und Nachrichten des kritischen Pfades festgelegt. Nachdem die zeitlichen Eigenschaften der Elemente des kritischen Pfades festgelegt wurden, sollen die Deadlines bzw. Release-Zeiten der Tasks bzw. Nachrichten, die von den Elementen des kritischen Pfades abhängen bzw. von denen die Elemente des kritischen Pfades abhängen, bestimmt werden. Abschließend werden die Releases und Deadlines der restlichen Tasks und Nachrichten aus den bereits festgelegten abgeleitet.

Die Release-Zeiten und Deadlines (in μs) der Nachrichten und Tasks der ACC-Anwendung (siehe Abbildung 5.8) werden berechnet, um den Algorithmus zu erläutern. Dabei wird zunächst der kritische Pfad ermittelt, dieser ist $cpath =< ods, ds, dbf, ab >$. Für die Berechnung der Übertragungszeiten der Nachrichten wurde ein Bussystem mit einer Bandbreite von $10\frac{Mbit}{s}$ ausgewählt. Alle Nachrichten der Anwendung haben die gleiche Payload ($1Byte$). So ergibt sich eine Übertragungszeit von $\zeta_B(m) = ((1Byte + 10Byte) \times 8) \times 0,1\frac{\mu s}{bit} = 9\mu s$ für alle Nachrichten. Bei der Berechnung wurde das Protokoll-Overhead auf $10Byte$ geschätzt.

Nun, da der kritische Pfad und die Übertragungszeiten der Nachrichten bekannt sind, kann der Schlupf der Nachrichten des kritischen Pfades berechnet werden (siehe Gleichung 5.17).

Algorithm 1 Deadline-Verteilung

procedure DEADLINEVERTEILUNG
Require: TG * *Mit festgelegten Start-Zeiten und Deadlines des Task-Graphen* *\
Ensure: TG * *Mit festgelegten Release-Zeiten und Deadlines der Nachrichten und Tasks* *\
  Zeiteigenschaften $t_{sg}(TG)$ und $t_{dg}(TG)$ des Task-Graphen festlegen
  $t_{r\tau}(\tau_i) \leftarrow t_{sg}(TG), \forall \tau_i \in \mathbf{source}(TG)$
  $t_{d\tau}(\tau_i) \leftarrow t_{dg}(TG), \forall \tau_i \in \mathbf{sink}(TG)$
  Finde den kritischen Pfad $cpath \in \mathbf{T}$
  Berechne den Schlupf ($slack$) der Nachrichten des kritischen Pfades $cpath$ (Gleichung 5.16)
  for alle Tasks $\tau_i \in cpath, \tau_i \notin \mathbf{sink}(TG)$ **do**
    $m_i \leftarrow s_m^{-1}(\tau_i)$ * *Die vom τ_i gesendete Nachricht* *\
    $t_{d\tau}(\tau_i) \leftarrow t_{r\tau}(\tau_i) + t_W(\tau_i)$
    $t_{rm}(m_i) \leftarrow t_{d\tau}(\tau_i)$
    $t_{dm}(m_i) \leftarrow t_{d\tau}(\tau_i) + \zeta_B(m_i) + slack$
    $t_{r\tau}(\tau_j) \leftarrow t_{dm}(m_i), \mathbf{succ}(\tau_i) = \tau_j$
  end for
  for alle Tasks $\tau_j \in cpath$ **do**
    for alle Task $\tau_k \in \mathbf{pred}(\tau_j), \tau_k \notin cpath$ **do**
      $m_k \leftarrow s_m^{-1}(\tau_k)$ * *Die vom τ_k gesendete Nachricht* *\
      $t_{dm}(m_k) \leftarrow t_{r\tau}(\tau_j)$
    end for
    for alle Task $\tau_k \in \mathbf{succ}(\tau_j), \tau_k \notin cpath$ **do**
      $m_j \leftarrow s_m^{-1}(\tau_j)$ * *Die vom τ_j gesendete Nachricht* *\
      $t_{r\tau}(\tau_k) \leftarrow t_{dm}(m_j)$
    end for
  end for
  Leite die Releases und Deadlines der restlichen Tasks und Nachrichten aus den bereits festgelegten ab (siehe Beispiel).
end procedure

$$
\begin{aligned}
slack &= \frac{t_{dg}(TG_{ACC}) - (t_W(ods) + t_W(ds) + t_W(dbf) + t_W(ab) + \zeta_B(m) \times 3)}{3} \\
&= \frac{3000\mu s - ((300\mu s + 300\mu s + 200\mu s + 200\mu s) + (9\mu s \times 3))}{3} = 657\mu s
\end{aligned} \tag{5.17}
$$

Nun, da der Schlupf bekannt ist, können die Releases und Deadlines der Elementen des kritischen Pfades berechnet werden (siehe Tabelle 5.1). Wenn die zeitlichen Eigenschaften der Elemente des kritischen Pfades bekannt sind, können daraus die der restlichen Nachrichten und Tasks abgeleitet werden. Nehmen wir zu Illustration den Task dtp. Dieser kann nicht

gestartet werden, bevor die Übertragung der Nachricht m_4 abgeschlossen wurde. Somit ergibt sich $t_{r\tau}(dtp) = t_{dm}(m_4) = 1932\mu s$. Die Deadlines der Nachricht m_1 wird durch die eben ermittelte Release-Zeit von dtp festgelegt, also $t_{dm}(m_1) = t_{r\tau}(dtp) = 1932\mu s$. Dieses Verfahren wird analog für die anderen rechtlichen Tasks und Nachrichten angewandt. Die zeitlichen Anforderungen der Elemente der ACC-Anwendung werden in Tabelle 5.1 gezeigt.

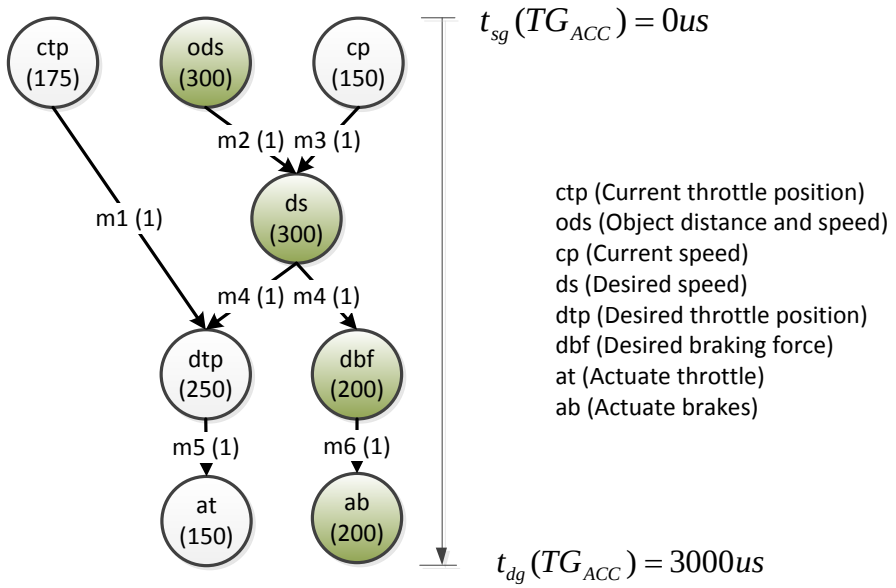


Abbildung 5.8: Task-Graph der ACC-Anwendung, mit $cpath =< ods, ds, dbf, ab >$

Tabelle 5.1: Releases und Deadlines der Tasks und Nachrichten der ACC-Anwendung.

t_i	$t_W(t_i)$	$t_{r\tau}(t_i)$	$t_{d\tau}(t_i)$	m_i	$t_{rm}(m_i)$	$t_{dm}(m_i)$
ctp	175	0	175	m_1	175	1932
ods	300	0	300	m_2	300	966
cp	150	0	150	m_3	150	966
ds	300	966	1266	m_4	1266	1932
dtp	250	1932	2182	m_5	2182	2850
dbf	200	1932	2132	m_6	2132	2798
at	150	2850	3000	–	–	–
ab	200	2798	3000	–	–	–

6 Verwandte und vergleichbare Arbeiten

In diesem Kapitel werden zwei Ansätze, mit denen die Migration eines Automobil-Netzwerk-Protokolls durchgeführt werden kann, beschrieben. Diese sind der Gateway-Ansatz und die vollständige Migration.

Bei der Migration eines Automobil-Netzwerks-Protokolls mit einer Gateway-Architektur geht es darum, ein Gerät (das Gateway) zu entwickeln, welches ein Quell-Protokoll in ein Ziel-Protokoll übersetzt. Das Gateway wird dann vor jedem Netzwerk-Knoten angeschlossen und ist seine Schnittstelle zum Netzwerk. Mit diesem Verfahren werden die Anwendungen nicht geändert. Deren zeitliche Anforderungen, wie Deadlines und Jitter, sollen nach der Migration weiterhin erfüllt werden. Es gibt bereits viele Autoren, die sich mit der Übersetzung von Automobil-Netzwerk-Protokollen anhand von Gateways befasst haben. Man kann unter anderem nennen: Seo u. a. (2006); Rui u. a. (2010) –Gateway zwischen CAN und FlexRay–; Kim u. a. (2008) –Gateway zwischen LIN, CAN und FlexRay–; Albert u. a. (2003) –Migration von CAN nach TTCAN mit Gateway–. Auf das letztere wird im Abschnitt 6.1 etwas detaillierter eingegangen.

Die vollständige Migration von einem Quell- nach einem Ziel-Protokoll erfolgt in der Regel auf der Applikations-Ebene. Dabei geht es darum, die Anwendungen mit dem Ziel-Protokoll auszuführen, ohne dass die Funktionalitäten und die Performance dieser gefährdet werden. Bei solch einer Migration wird der Fokus auf die Analyse und Optimierung des Zeitverhaltens der von den Anwendungen ausgeführten Tasks und gesendeten/empfangenen Nachrichten gelegt. Es existieren bereits viele Arbeiten, die sich einer vollständigen Migration eines Automobil-Netzwerk-Protokolls gewidmet haben. Man kann unter anderem erwähnen: Armengaud u. a. (2009) –Migrations-Herausforderungen von einem event triggered nach einem time-triggered Kommunikationsschema–; Kern u. a. (2011) –Migration von CAN nach Ethernet/IP–; Richard u. a. (2008) –Migration von CAN nach FlexRay–. Auf die beiden letzteren wird in den Abschnitten 6.2 und 6.3 etwas detaillierter eingegangen.

Diese Arbeit verwendet den vollständigen Migrations-Ansatz mit dem Fokus auf dem Design des Systems.

Erwähnenswert ist, dass zur Erarbeitungszeit dieser Arbeit keine veröffentlichten Arbeiten bezüglich der Migration von FlexRay nach TTEthernet herausgefunden werden konnten. Aus diesem Grund wurde bei der Untersuchung von verwandten Arbeiten auf andere Automobil-Netzwerk-Protokolle (CAN, TTCAN) eingegangen.

6.1 Migration von CAN nach TTCAN mit Gateways

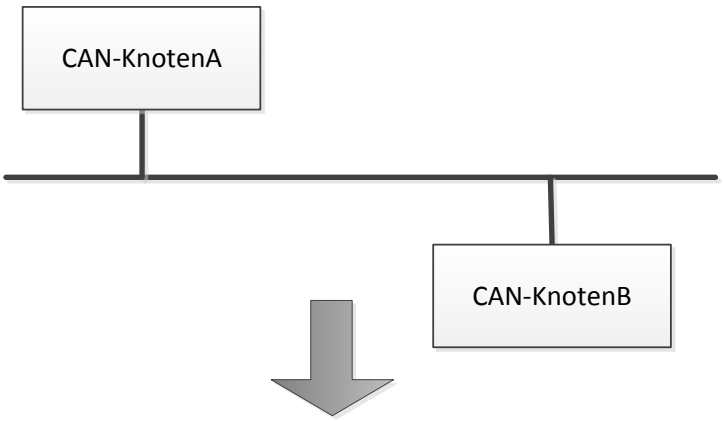
In Albert u. a. (2003) wurde die Migration von einer bestehenden event-triggered CAN-Vernetzung nach einer time-triggered CAN-Vernetzung (TTCAN) am Beispiel des von Bosch entwickelten Vehicle Dynamics Management (VDM) (vgl. Bosch, 2011) durchgeführt. In dieser Arbeit wurde zuerst die auf CAN basierende Kommunikationsstruktur skizziert und dann die Konzepte des TTCAN-Systems vorgestellt. Da zur Zeit der Migration keine ECUs, welche das TTCAN-Protokoll unterstützten, zur Verfügung standen, wurde für die Migration ein CAN-TTCAN-Gateway entwickelt.

Um dieses Gateway-Konzept zu erklären, wird die Abbildung 6.1 verwendet. In der Abbildung 6.1(a) wird ein Ausschnitt der Struktur des CAN-Busses mit zwei Knoten gezeigt und in der Abbildung 6.1(b) die Gateway-Architektur. In dieser Abbildung werden die CAN-Knoten nicht mehr direkt an dem CAN-Bus angeschlossen, sondern indirekt über die CAN-TTCAN-Gateways. Das Gateway hat als Aufgabe, die vom Knoten ausgehenden Nachrichten nach einem festgelegten Zeitplan (time-triggered) weiterzusenden. Außerdem leitet es die eingehenden Nachrichten an den Knoten weiter.

In einem zeitgesteuerten System ist es nicht nur wichtig, dass die Uhren aller Teilnehmer miteinander synchronisiert werden, sondern auch, dass die Applikationen mit dem Kommunikationszyklusschedule synchronisiert werden. Nur so kann garantiert werden, dass die Daten (z. B. neue Sensor-Werte) zum Zeitpunkt der Weiterleitung auch aktuell sind. Dieses wird hier gewährleistet, indem das CAN-TTCAN-Gateway die Applikation auf dem CAN-Knoten kurz von dem geplanten Zeitpunkt auffordert, die Daten zu senden. Wichtig hierbei ist, dass die Verzögerungszeit zwischen der Aufforderung des Gateways und dem Senden der Daten unter einer garantierten Grenze liegt.

Durch die Replikation der Nachrichten durch die Gateways entstehen zusätzliche Latenzen. Da aber jeder Knoten eine Punkt-zu-Punkt-Verbindung mit einem Gateway hat und der Zugriff auf den Bus zeitgesteuert erfolgt, sind die Latenzen bekannt und konstant. Für die Test-Szenarien ergaben sich Latenzen von $325\mu s$ bei $250\frac{Kbit}{s}$ und $200\mu s$ bei einer Übertragungsrate von $1\frac{Mbit}{s}$ für Nachrichten mit einer Nutzlast-Länge von $8Byte$.

(a) CAN-Bus mit zwei Knoten



(b) Gateware-Architektur der Abbildung a

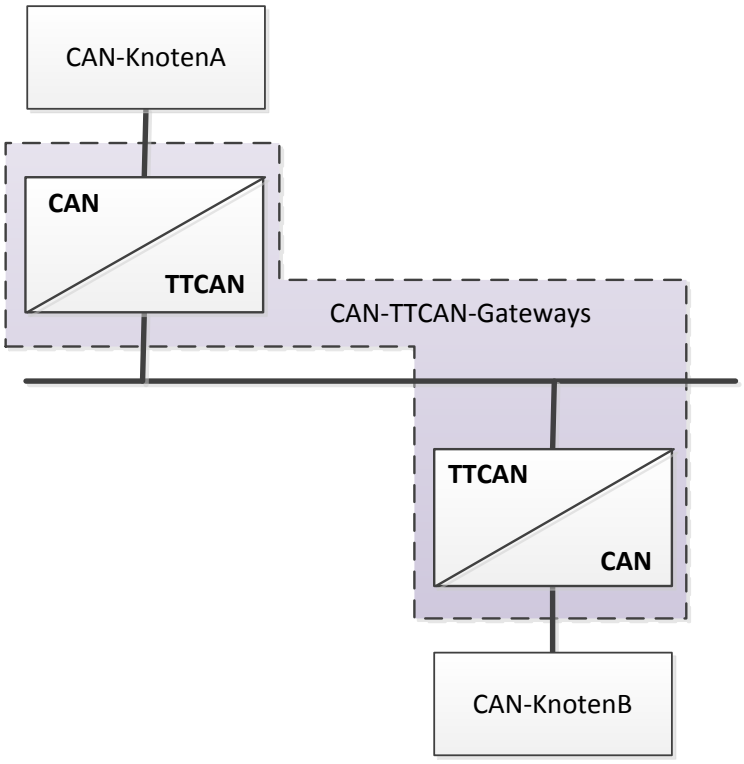


Abbildung 6.1: Migration von CAN nach TTCAN mit Gateways

Obwohl der Gateway-Ansatz einfach zu realisieren ist, ist dieser teuer, macht das Netzwerk komplex und bringt höhere Verzögerungen der Nachrichten mit sich. Aus diesen Gründen wird der vollständige Ansatz empfohlen.

6.2 Vollständige Migration von CAN nach FlexRay

In Richard u. a. (2008) wurde ein Migrations-Framework für eine komplette Migration von einem bestehenden CAN-Netzwerk zu einem FlexRay-Netzwerk entwickelt. Dabei wurde der Fokus auf die Software-Migration gelegt. Das Framework berücksichtigt sowohl das statische als auch das dynamische Segment von FlexRay.

Die Vorarbeit bei der Migration war, einen Vergleich zwischen CAN und FlexRay durchzuführen. Dabei konnten die Autoren feststellen, dass FlexRay viele Vorteile gegenüber CAN hat. Diese sind: eine höhere Bandbreite von $10\frac{Mbit}{s}$ gegenüber $1\frac{Mbit}{s}$, ein redundanter Kommunikationskanal, die Unterstützung von event und time triggered Kommunikation, Fehlertoleranz immer möglich gegenüber Fehlertoleranz nur bei „low-speed“ CAN. Die beiden Vorteile von CAN gegenüber FlexRay sind, dass CAN kostengünstiger und weniger komplex ist und somit von Netzwerk-Designern bevorzugt wird.

Im ersten Schritt der Migration wurden zuerst alle CAN-Anwendungen mit Taskgraphen abstrahiert, um die Ein- und Ausgabe-Parameter für die Migration zu extrahieren. Eine CAN-Nachricht M_{CAN} wurde z. B. durch das Tripel $(size(m_i), ID(m_i), period(m_i))$ beschrieben: mit $size(m_i)$ die Größe der Nachricht, $ID(m_i)$ die CAN-ID der Nachricht und $period(m_i)$ die Periode der Nachricht. In dem Graph repräsentiert jeder Knoten eine CAN-Task τ_i und jede Kante e_{ij} einen Kommunikationslink zwischen den Knoten. Darüber hinaus wurden auch die Task-Graphen-Eigenschaften t_{sg} und t_{dg} betrachtet (siehe Kapitel 5).

Im zweiten Schritt wurde für jeden Task aus den Task-Graphen festgelegt, ob dieser im statischen oder dynamischen Segment von FlexRay ausgeführt werden soll. Dafür wurden zuerst die Tasks als kritisch oder nicht kritisch eingeordnet. Danach wurden alle kritischen Tasks (z. B. Bracke-by-wire) dem statischen Segment und alle nicht kritischen Tasks (z. B. Klimaanlagen) dem dynamischen Segment zugewiesen. Die Basis für die Klassifizierung waren Erfahrungen aus vergangenen CAN-Projekten.

Bei der Migration wurden die Funktionsmodelle (Task-Graphen) der Anwendungen beibehalten. Also wurden alle Tasks und Nachrichten aus dem CAN-Netzwerk nicht geändert. Außerdem wurde die Bus-Topologie von CAN beibehalten, da FlexRay neben Stern- auch Bus-Topologien unterstützt.

Anhand der CAN-Nachrichten-Parameter wurde die Größe der Nachrichten in FlexRay festgelegt und optimiert. Hiermit wurde die Länge eines statischen Slots festgelegt und somit auch die Länge des statischen Segments. Die Länge des dynamischen Segments wurde während der Entwicklung des statischen Segments und anhand der Zyklus-Länge festgelegt. Die Response-Time-Analyse-Technik wurde bei der Entwicklung des dynamischen Segments angewendet, um zu überprüfen, ob seine Länge für die festgelegten Anforderungen (z. B. Deadlines der Nachrichten) geeignet ist. Weiterhin wurden Algorithmen für die Migration des dynamischen und auch des statischen Segments entwickelt. Diese werden Aufgrund deren Komplexität nicht weiter erläutert (siehe Richard u. a. (2008), S.163-167).

Die Ergebnisse der Fallstudie (Migration einer Adaptive Cruise Control (ACC) Anwendung) zeigten, dass alle Tasks und Nachrichten aus dem statischen und dem dynamischen Segment ihre Deadlines in FlexRay mit konsistenten Ergebnissen einhalten.

6.3 Vollständige Migration von CAN nach Ethernet/IP

In Kern u. a. (2011) wurde eine vollständige Migrationsstrategie für die Migration von CAN nach Ethernet/IP entwickelt. Die Autoren haben sich hierbei auf das Design des Netzwerks, insbesondere der Datenstruktur, fokussiert. Der Migrations-Prozess besteht aus fünf Schritten.

Im ersten Schritt werden die Konfigurationsdaten des zu migrierenden CAN-Busses aus einer Konfigurationsdatei importiert. Das Ergebnis des Imports ist eine Menge von ECUs E und eine Menge von Nachrichten M. Jede Nachricht $m \in M$ wird durch das folgende 4-Tupel definiert:

$$m = (s_m, l_m, e_m^t, E_m^r) \tag{6.1}$$

wobei:

- $s_m \in \{cyclic, cyclicIfActive, cyclicIfActiveFast, cyclicAndSpontanWithDelay, cyclicWithRepeatOnDemand, spontan\}$ der Sende-Typ der Nachricht ist.
- $l_m \in [1 - 8]$ die Payload-Länge der Nachricht m in Byte ist.
- $e_m^t \in E$ der ECU ist, welcher m sendet.
- $E_m^r \subseteq E$ die Menge der ECUs sind, welche m empfangen.

Im zweiten Schritt wird der Sende-Typ der Nachrichten angepasst. Hierbei werden alle Nachrichten-Sende-Typen auf einen Typ reduziert — der zyklische Typ, also $s_m = cyclic$.

Gründe der Reduzierung auf den zyklischen Typ sind die Verringerung der Komplexität der Handhabung der Nachrichten bei der Migration und die Tatsache, dass bereits 80 % der Nachrichten zyklisch gesendet werden.

Im dritten Schritt werden die Adressen aufgelöst. Hierbei wird die nachricht-basierte Adressierung von CAN auf die knoten-basierte Adressierung von Ethernet angepasst. Es wird außerdem in diesem Schritt die CAN-Bus-Topologie auf eine Full-Duplex-Switched-Ethernet-Topologie gemappt, wobei alle ECUs an einem Switch angeschlossen werden.

Im vierten Schritt werden die CAN-Nachrichten in einem UDP-Paket (User Datagram Protocol) verpackt. Es gibt zwei Möglichkeiten für die Verpackung der CAN-Nachrichten. Entweder wird eine CAN-Nachricht in einem UDP-Paket verpackt oder es werden mehrere CAN-Nachrichten in einem UDP-Paket verpackt. Beim letzteren wird versucht, Nachrichten gleicher Art zu verwenden. Diese können z. B. Nachrichten, die den gleichen Sender und/oder die gleiche Periode haben, sein.

Im fünften Schritt wird das Ergebnis evaluiert. Als Evaluierungskriterium wurde unter anderem die maximale Pakete-Länge, die Anzahl der Pakete und die benutzte Bandbreite ausgewählt. Abschließend wurde ein CAN-BODY-Netzwerk für verschiedene Automobilklassen (Kompakt Mittel und Premium) migriert. Die Anzahl der extrahierten Nachrichten variiert von 230 bis 406. Die Ergebnisse zeigten, dass eine Bandbreite von $14\frac{Mbit}{s}$ bis $25\frac{Mbit}{s}$ je nach Automobilklassen benutzt wurde. Bei einer optimalen Verpackung der Nachrichten, also wenn mehrere CAN-Nachrichten mit einem UDP-Paket übertragen wurden, konnte die Ausnutzung der Bandbreite auf $11\frac{Mbit}{s}$ bis $8\frac{Mbit}{s}$ reduziert werden.

7 Entwicklung der Migrationsstrategien

In diesem Kapitel wird das Framework für die Migration von einem TTEthernet-Systemmodell zu einem FlexRay-Systemmodell entwickelt. Die Entwicklung wird in zwei Bereichen durchgeführt: Migration des statischen und Migration des dynamischen Segments. Bevor dies geschieht, wird zunächst ein Vergleich zwischen FlexRay und TTEthernet gegeben. Dieser ist für die Festlegung der Schwerpunkte der Migration unabdingbar.

7.1 Vergleich von FlexRay und TTEthernet

Der Vergleich von FlexRay und TTEthernet mit dem Fokus auf der time-triggered Kommunikation wurde bereits in Steinbach u. a. (2010) durchgeführt. Die Arbeit demonstriert, dass es möglich ist, mindestens die gleiche Menge an Daten mit TTEthernet übertragen zu können, die ein voll ausgelastetes FlexRay-System bewältigen kann. Weiterhin wurden Gemeinsamkeiten und Unterschiede der beiden Protokolle herausgearbeitet. Es wurden insbesondere Metriken wie Latenz und Jitter anhand eines mathematischen Modells berechnet und Beispiel-Netzwerke verglichen. Die Tabelle 7.1 gibt einen Überblick über den Vergleich von FlexRay und TTEthernet. Diese Tabelle wurde auf weitere für diese Arbeit relevante Eigenschaften wie Max. Zyklen und Adressierung erweitert.

FlexRay und TTEthernet unterstützen sowohl den time- als auch den event-triggered Traffic. Der time-triggered Traffic findet jedoch in FlexRay im statischen Segment statt, während dieser im TTEthernet während des gesamten Kommunikationszyklus stattfinden kann. Weiterhin ist die Länge aller time-triggered Kommunikationsslots im FlexRay statisch, während diese im TTEthernet je nach Nachrichten-Länge und -Übertragungszeit festgelegt werden können. Somit wird im TTEthernet nur so viel Bandbreite wie nötig reserviert. Der event-triggered Traffic von FlexRay findet im dynamischen Segment statt, während dieser im TTEthenet, wie der time-triggered Traffic, während des gesamten Zyklus stattfinden kann. So können alle Lücken, die zwischen dem Senden von time-triggered Nachrichten entstehen, für die Übertragung von event-triggered Nachrichten ausgenutzt werden. Die Realisierung des event-triggered Traffic erfolgt im FlexRay durch das Minisloting-Verfahren, in dem jede Nachricht einen Minislot bekommt, welcher nach Bedarf erweitert werden kann. Im TTEthernet wird

Tabelle 7.1: Übersicht Vergleich von TTEthernet und FlexRay

Eigenschaft	**FlexRay**	**TTEthernet**
Traffic-Art	Time- und Event-Triggered	Time- und Event-Triggered
Time-Triggered-Slot	Statische Länge	Dynamische Länge
Max. Zyklen	64	1
Protokoll-Overhead	$8Byte$	$26Byte$
Min. Payload	$1Byte$	$46Byte$
Max. Payload	$254Byte$	$1500Byte$
Adressierung	$11bit$ Nachricht-ID	$16bit$ CT-Nachricht-ID
Topologie	Bus, Stern und Mix	Switch
Redundante Kanäle?	ja	ja
Kommunikation	Broadcast	Unicast-, Multi und Broadcast
Bandbreite	$10/20\frac{Mbit}{s}$	$100/1000\frac{Mbit}{s}$
Kommunikations-Richtung	Halb-Duplex	Full-Duplex
Latenz (TT-Traffic)	Abhängig von Signallaufzeit	Abhängig von Signallaufzeit + Verzögerungen im Switch
Jitter (TT-Traffic)	Fast Konstant	$< 10\mu s$

der event-triggered Traffic weiter unterteilt in best-effort und rate-constraint Traffic. Das letzte entspricht dem AFDX-Protokoll, welches für jede Nachricht ein gewisses Kontingent an Bandbreite verspricht. In Rahmen dieser Arbeit wird lediglich der rate-constraint Traffic in Betracht gezogen.

Die zyklische Kommunikation von TTEthernet erfolgt nicht in mehreren Zyklen (max. 64) wie bei FlexRay, sondern nur in einem Zyklus. Durch die mehreren Zyklen von FlexRay ist es möglich, dass sich mehrere Nachrichten den gleichen Slot teilen, jedoch in verschieden Zyklen gesendet werden. Die Teilung der Slots kann auch in TTEthernet aufgrund seines Full-Duplex-Switchings angewandt werden. Man soll jedoch erwähnen, dass die Nachrichten, die sich denselben Slot im TTEthernet teilen, nicht in verschieden Zyklen gesendet werden, sondern in demselben Zyklus. Aus diesem Grund sollen TT-Nachrichten nur gleichzeitig übertragen werden, wenn diese auch disjunkte Kommunikationspfade haben. Dadurch, dass das Senden von mehreren Nachrichten im TTEthernet gleichzeitig erfolgen kann, wird nicht nur die Bandbreite besser ausgenutzt, sondern auch die Antwortzeiten der Nachrichten reduziert. Letzteres ist ein sehr großer Vorteil für ein Echtzeit-System.

Der Nachrichten-Overhead ist wesentlich höher Bei TTEthernet ($26Byte$) als bei FlexRay ($8Byte$). Dies ist ein Nachteil für Nachrichten mit sehr kleiner Länge. Darüber hinaus ist die minimale Payload-Länge wesentlich kürzer bei FlexRay ($1Byte$) als bei TTEthernet ($46Byte$). Dies führt dazu, dass bei der Migration von kleinen FlexRay-Nachrichten die Bandbreite verschwendet wird. Um dies zu vermeiden, sollen, wenn möglich, mehrere Nachrichten in einer zusammengefasst werden. TTEthernet kompensiert sein großes Overhead durch eine längere Payload von max. $1500Byte$. Dies ist gut geeignet für Anwendungen (kamerabasierte Fahrerassistenzsysteme), die hohe Datenmengen übertragen.

Die Latenzzeit der TT-Nachrichten in FlexRay hängt im wesentlichen von der Signallaufzeit ab, weil abgesehen von aktiven Sternen FlexRay-Teilnehmer über einen Bus miteinander verbunden werden. Aufgrund der internen Verarbeitung verursacht ein aktiver Stern zusätzliche Verzögerung. Die maximale Verzögerung eines aktiven Sternes wurde auf $250ns$ festgelegt (vgl. FlexRay Consortium, 2005b). TTEthernet-Teilnehmer werden ausschließlich über Switches miteinander verbunden. Somit hängt die Latenzzeit der TT-Nachrichten in TTEthernet hauptsächlich von den Signallaufzeiten und Verzögerungen in den Switches ab. TTTech gibt eine TTEthernet-Switch-Verarbeitungszeit von $t_{DS}(s) = 1\mu s$ bis $t_{DS}(s) = 2.4\mu s$ für deren Implementierung an (vgl. TTTech Computertechnik AG), (vgl. Steinbach u. a., 2010, S. 2).

In FlexRay können die Nachrichten mit maximal $10\frac{Mbit}{s} \times 2$ übertragen werden, während in TTEthernet die Nachrichten mit $100/1000\frac{Mbit}{s}$ übertragen werden können. Also ist die Bandbreite im TTEthernet wesentlich höher.

Zusammenfassung

Die wesentlichen Unterschiede zwischen FlexRay und TTEthernet, die durch den Vergleich festgestellt werden konnte, werden nachfolgend aufgelistet. Diese werden die Migrationsstrategien prägen.

- Anzahl der Kommunikationszyklen
- Aufbau der Kommunikationszyklen
- Länge der time-triggered Kommunikationsslots
- Verschiedene Ansätze für die event-triggered Kommunikation
- Maximale und minimale Payload sowie Protokoll-Overhead
- Bandbreite
- Topologie

7.2 Migration des statischen Segments

In diesem Abschnitt werden Methoden entwickelt, mit deren Hilfe man time-triggered FlexRay-Anwendungen (TT-Anwendungen), also im statischen Segment ausgeführte Anwendungen, in TTEthernet-TT-Anwendungen umwandeln kann. Dafür werden zuerst die Voraussetzungen für die Migration festgelegt (Abschnitt 7.2.1). Dann wird gezeigt, wie der Schedule eines statischen FlexRay-Segments beschrieben werden kann (Abschnitt 7.2.2). Anschließend wird dargelegt, wie ein FlexRay-Systemmodell zu einem TTEthernet-Systemmodell werden kann (Abschnitt 7.2.3). Mit dem TTEthernet-Systemmodell und dem vorhandenen FlexRay-Schedule wird versucht, einen möglichen Schedule des Systems in TTEthernet zu finden (Abschnitt 7.2.4). Wird ein ausführbarer Schedule für die zu migrierenden TT-Anwendungen in TTEthernet gefunden, dann sind diese migrierbar. Die Migrationsmethoden werden anhand einer Fallstudie validiert.

7.2.1 Voraussetzungen für die Migration des statischen Segments

Für die Migration des statischen Segments wird folgendes vorausgesetzt:

- Die Funktionsmodelle (Task-Graphen) der TT-Anwendungen müssen vorhanden sein. Diese inkludieren alle innerhalb des statischen Segments gesendeten Nachrichten und deren zeitliche Anforderungen wie Release-Zeiten und Deadlines (siehe Kapitel 5).
- Ein FlexRay-Topologiemodell der Anwendungen.
- Ein Mapping der Funktionen (Tasks) auf die jeweiligen Prozessoren (siehe Definition 5.9).
- Ein ausführbarer Schedule der Anwendungen für das statische Segment.

Es wurde bis hier lediglich gezeigt, wie Funktions- und Topologie-Modelle beschrieben werden (siehe Kapitel 5) und nicht, wie der Schedule eines FlexRay-Systems definiert wird. Dies wird im folgenden Abschnitt nachgeholt.

7.2.2 Beschreibung des Schedules eines FlexRay-Systems

Die Kommunikation in einem FlexRay-System erfolgt zyklisch in einer Menge von Kommunikationszyklen, welche aus maximal 64 Zyklen bestehen kann. Jeder Kommunikationszyklus setzt sich aus 4 Teilen zusammen, nämlich das statische Segment, das dynamische Segment, das Symbol Window (SWin) und die Network Idle Time (NIT) (Siehe Abschnitt 3.5.1). Die

Aktivitäten der beiden letzteren werden in dieser Arbeit vernachlässigt, deren Längen jedoch nicht, da diese auch zur Verzögerung einer Nachricht beitragen können. Der Schedule eines FlexRay-Systems lässt sich mit dem folgenden 4-Tupel beschreiben:

$$\mathcal{S}_{FR} = (T_{CC}, N_{CC}, \mathcal{S}_{STS}, \mathcal{S}_{DYNS}) \tag{7.1}$$

wobei:

- $T_{CC}[\mu s]$[1] die Länge eines Kommunikationszyklus ist, $T_{CC} \in [1, 16000], T_{CC} \in \mathbb{N}$ (siehe Gleichung 7.2).
- N_{CC} die Anzahl der konfigurierten Kommunikationszyklen ist, $N_{CC} \in [0, 63], N_{CC} \in \mathbb{N}$.
- $\mathcal{S}_{STS}$ der Schedule des statischen Segments ist (siehe Gleichung 7.3).
- $\mathcal{S}_{DYNS}$ der Schedule des dynamischen Segments ist (siehe Gleichung 7.9 in Abschnitt 7.3.1).

Die Länge T_{CC} eines FlexRay-Kommunikationszyklus kann durch die folgende Gleichung berechnet werden:

$$T_{CC} = T_{STS} + T_{DYNS} + T_{SWin} + T_{NIT} \tag{7.2}$$

wobei:

- $T_{STS}[\mu s]$ die Länge des statischen Segments ist.
- $T_{DYNS}[\mu s]$ die Länge des dynamischen Segments ist.
- $T_{SWin}[\mu s]$ die Länge des Symbol Window ist.
- $T_{NIT}[\mu s]$ die Länge der Network Idle Time ist.

Der Schedule des statischen Segments wird im nächsten Abschnitt beschrieben, der des dynamischen Segments jedoch erst im Abschnitt 7.3.1.

[1] In einem FlexRay-System wird die kleinste gemeinsame Zeit, welche durch das Synchronisationsprotokoll korrigiert wird, zwischen den Teilnehmern in Macrotick angegeben. In dieser Arbeit entspricht ein Macrotick einer Mikrosekunde.

Beschreibung des Schedules eines FlexRay statischen Segments

Der Schedule eines FlexRay statischen Segments wird durch das folgende 4-Tupel dargestellt:

$$\mathcal{S}_{STS} = (T_{STS}, T_{STSlot}, N_{STSlot}, \mathcal{S}_{\mathbf{ST}}) \tag{7.3}$$

wobei:

- $T_{STS}[\mu s]$ die Länge des statischen Segments ist.
- $T_{STSlot}[\mu s]$ die Länge eines statischen Slots ist, $T_{STSlot} \in [1, 63], T_{STSlot} \in \mathbb{N}$ (siehe Gleichung 7.4).
- N_{STSlot} die Anzahl der verfügbaren Slots im statischen Segment ist, $N_{STSlot}[1, 1023] \in \mathbb{N}$ (siehe Gleichung 7.5).
- $\mathcal{S}_{\mathbf{ST}} \subseteq \mathbb{N}^4$ die Menge der Schedules aller ST-Nachrichten ist, die im statischen Segment übertragen werden sollen, (siehe Gleichung 7.6 für die Beschreibung der Schedule $\mathcal{S}_{ST}(m) \in \mathcal{S}_{\mathbf{ST}}$ einer einzelnen ST-Nachricht).

In einem FlexRay-Kommunikationszyklus haben alle statischen Slots die gleiche Länge T_{STSlot}. Das heißt, dass sichergestellt werden soll, dass die Übertragung jeder ST-Nachricht in solch ein Slot vollständig durchgeführt werden kann. Daher wird die Berechnung der Länge der statischen Slots abhängig von der Nachricht mit der größten Länge ermittelt . Die Länge der statischen Slots kann mit der Gleichung 7.4 ausgerechnet werden. Wobei $T_{cp}[\mu s]$ die Präzision der Uhren und $\mathbf{TG_{TT}} \subseteq \mathbf{TG}$ die Menge der time-triggered Task-Graphen ist.

$$T_{STSlot} = \max_{\forall m_i \in \mathbf{TG_{TT}}} \{\zeta_B(m_i)\} + 2 \times T_{cp} \tag{7.4}$$

Die Anzahl der statischen Slots kann mit der folgenden Gleichung berechnet werden:

$$N_{STSlot} = \left\lfloor \frac{T_{STS}}{T_{STSlot}} \right\rfloor \tag{7.5}$$

Der Schedule $\mathcal{S}_{ST}(m)$ einer statischen Nachricht (ST-Nachricht) wird durch das folgende 4-Tupel beschrieben (vgl. Zhao, 2011, S. 48):

$$\mathcal{S}_{ST}(m) = (bSlot(m), slotRep(m), bCC(m), CCRep(m))^2 \tag{7.6}$$

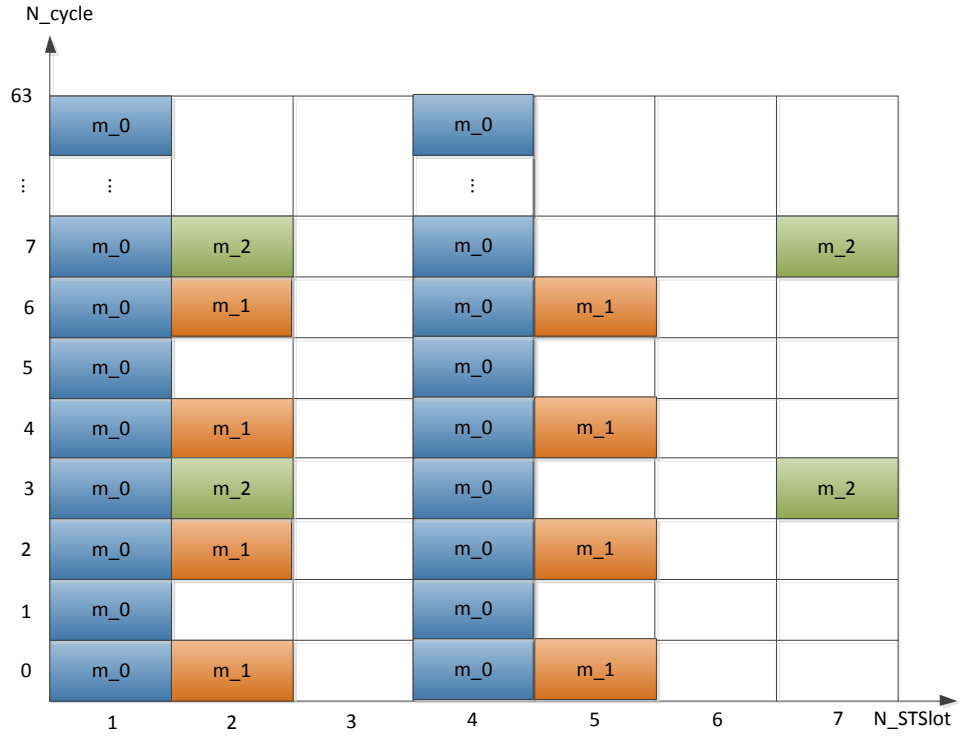


Abbildung 7.1: Beispiel-Schedule von drei statischen Nachrichten

wobei:

- $bSlot(m)$ – Basis-Slot – der erste Slot innerhalb eines statischen Segments ist, in dem die Nachricht m gesendet wird, $bSlot(m) \in [1, 1023], bSlot(m) \in \mathbb{N}$.
- $slotRep(m)$ die Anzahl der Slots ist, nach der die Übertragung der Nachricht m innerhalb eines Kommunikationszyklus wiederholt wird, $slotRep(m) \in [1, 1022], slotRep(m) \in \mathbb{N}$.
- $bCC(m)$ – Basis-CC – der erste Kommunikationszyklus ist, in dem die Nachricht m gesendet wird, $bCC(m) \in [0, 63], bCC(m) \in \mathbb{N}$.
- $CCRep(m)$ die Anzahl der Kommunikationszyklen ist, nachdem die Übertragung der Nachricht m wiederholt wird, $CCRep(m) = 2^n, n \in [0, 6], n \in \mathbb{N}$.

[2] Die Wertebereiche dieser Eigenschaften wurden aus der FlexRay-Spezifikation (vgl. FlexRay Consortium, 2005b) bzw. der Arbeit Zhao (2011) entnommen.

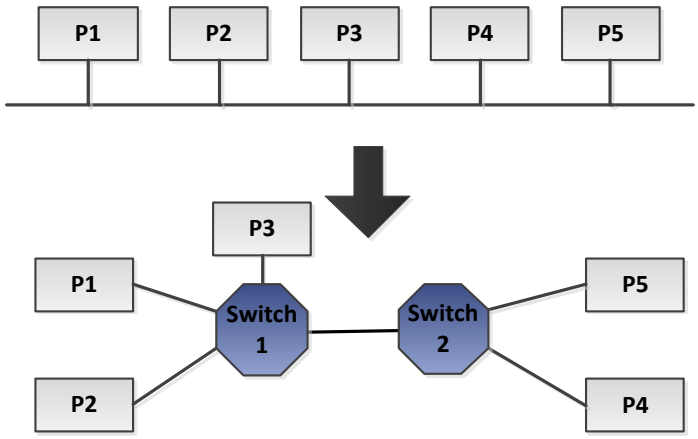
Abbildung 7.1 zeigt ein Beispiel-Schedule von 3 ST-Nachrichten (m_0, m_1 und m_2). Die Schedules der einzelnen Nachrichten sehen folgendermaßen aus: $\mathcal{S}_{ST}(m_0) = (1, 3, 0, 1)$, $\mathcal{S}_{ST}(m_1) = (2, 3, 0, 2)$ und $\mathcal{S}_{ST}(m_2) = (2, 5, 3, 4)$.

7.2.3 Umwandlung eines FlexRay-Systemmodells in ein TTEthernet-Systemmodell

Ein Systemmodell Γ besteht aus einer Menge von Funktionsmodellen $\mathbf{TG}$ und einer Topologie $TPG = (\mathbf{P}, \mathbf{S}, \mathbf{L}, \mathbf{B}, b)$ (siehe Definition 5.8). Dadurch, dass sich die Funktionen und Anforderung des Systems bei der Migration nicht ändern, werden die FlexRay-Funktionsmodelle im TTEthernet beibehalten. Das heißt, dass alle Tasks aus $\mathbf{T}$ (Funktionen des Systems), Kanten aus $\mathbf{E}$ (Abhängigkeits-Beziehungen und Logische-Links zwischen den Tasks) und Nachrichten aus $\mathbf{M}$ sich in TTEthernet-Funktionsmodellen nicht ändern. Somit findet eine Eins-zu-Eins-Funktionsmodell-Abbildung statt. Die Topologie kann jedoch nicht beibehalten werden, da ein TTEthernet-System nur mit Switch-Topologie ausgestattet werden kann (siehe Abschnitt 4.2). Bei der Migration der Topologie können passive bzw. aktive Stern-Koppler eins zu eins mit Switches ersetzt werden (siehe Abbildung 7.2(b)). Die über den Sternen verbundenen Prozessoren [3] sowie die redundanten Kanäle aus einer Stern-Topologie können beibehalten werden (siehe Abbildung 7.2(b)). Die Lokalität eines Prozessors kann jedoch beliebig verändert werden, falls diese nicht durch die Funktionalität des Prozessors gebunden ist. Eine Bus-Topologie kann mit einem Switch oder kaskadierten Switches ersetzt werden (siehe Abbildung 7.2(a)), wobei durch kaskadierte Switches die Last auf die Switches verteilt werden kann und die Skalierbarkeit erhört wird. Die hier angegebenen Möglichkeiten zur Migration der Topologien sind nur Vorschläge, letztendlich ist es möglich, zu einer beliebigen TTEthernet-Topologie zu migrieren, sofern die Anforderungen eingehalten werden. Man sollte jedenfalls bei der Migration der Topologie sowohl das statische als auch das dynamische Segment in Betracht ziehen, denn ein nachträgliches Ändern dieser kann dazu führen, dass der Schedule von TT-Nachrichten erneut generiert werden muss. Um die Umwandlung eines FlexRay-Systemmodells in einem TTEthernet-Systemmodell abzuschließen, soll jeder Task auf einen Prozessor gemappt werden. Auch hier können die während des Designs des FlexRay-Systems festgelegten Mappings behalten werden. Dies ist jedoch nicht zwingend, wenn ein anderes Mapping aufgrund der Switch-Topologie zu einem besseren Endergebnis führen kann. Man soll aber bedenken, dass Funktionen meistens auf bestimmte Prozessoren ausgeführt werden müssen. Aus diesem Grund werden im weiteren Verlauf dieser Arbeit die FlexRay-Mappings übernommen.

[3] Ein Prozessor wird in FlexRay auch Netzwerk-Knoten und in TTEthernet Endsystem genannt

(a) Beispiel FlexRay-Singlekanal-Bus- zu
TTEthernet-Singlekanal-Switch-Topologie



(b) Beispiel FlexRay-Dualkanal-Stern- zu
TTEthernet-Dualkanal-Switch-Topologie

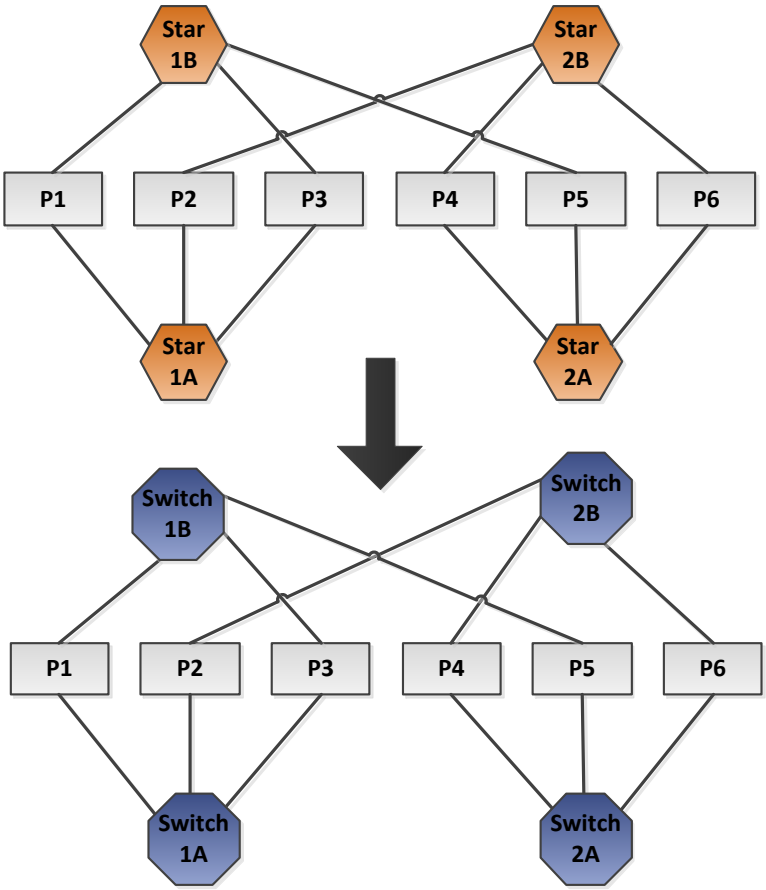


Abbildung 7.2: Beispiel-FlexRay-Topologien zu TTEthernet-Topologien

Das festgelegte TTEthernet-Systemmodell ist nur dann valid, wenn es einen ausführbaren Schedule für jede time-triggered und event-triggered Nachricht gibt. Um diese zu generieren, werden neue Verfahren bzw. Scheduling-Algorithmen benötigt. Denn die Nachrichten werden im TTEthernet über mehrere Switches geroutet. Außerdem wird ein anderes Frame-Format und eine andere Übertragungsgeschwindigkeit verwendet als bei FlexRay. Weiterhin wird im TTEthernet nur ein Kommunikationszyklus verwendet und nicht 64 (siehe Abschnitt 7.1). Der Scheduling-Algorithmus für time-triggered TTEthernet Nachrichten wird im folgenden Abschnitt entwickelt, der für event-triggered Nachrichten erst während der Entwicklung des dynamischen Segments in Abschnitt 7.3.

7.2.4 Erstellung des Schedules für TTEthernet time-triggered Nachrichten

Um die Migration des statischen Segments abzuschließen, benötigen wir noch einen ausführbaren Schedule der TTEthernet time-triggered Nachrichten. Dieser wird mit $\mathcal{S}_{\mathbf{TT}}$ gekennzeichnet. Um diesen zu berechnen, wird ein Algorithmus entwickelt, der folgende Eingaben benötigt (siehe auch Abbildung 7.3):

- Die Menge der time-triggered TTEthernet-Funktionsmodelle $\mathbf{TG_{TT}} \subseteq \mathbf{TG}$.
- Eine Topologie TPG des TTEthernet-Systems inklusive Mappings der TT-Tasks auf die Prozessoren.
- Die festgelegten Routen aller time-triggered Nachrichten $\mathbf{R_M}$. Die Routen $\mathbf{R_m} \in \mathbf{R_M}$ einer Nachricht m werden also nicht vom Algorithmus berechnet. Es wird von statischen, gegebenen Routen ausgegangen (siehe Eigenschaft (v) des Systemmodells im Abschnitt 5.5).
- Die Liste $\mathcal{ML}$ der Nachrichten, nach der die Planung erfolgen soll.
- Der FlexRay-Schedule $\mathcal{S}_{FR}$, welcher nicht nur die Eigenschaften des FlexRay-Kommunikationszyklus sondern auch den Schedule des statischen Segments $\mathcal{S}_{STS}$ beinhaltet (siehe Abschnitt 7.2.2).

Bis jetzt wurden der TTEthernet-Schedule und die Generierung der Liste $\mathcal{ML}$ nicht dargelegt. Dies werden jeweils im folgenden und darauf folgenden Abschnitt nachgeholt.

Beschreibung eines TTEthernet-Schedules

Ein TTEthernet-Schedule kann mit dem folgenden 3-Tupel beschrieben werden:

$$\mathcal{S}_{TTE} = (T_{CC}, \mathcal{S}_{\mathbf{TT}}, \mathcal{S}_{\mathbf{RC}}) \tag{7.7}$$

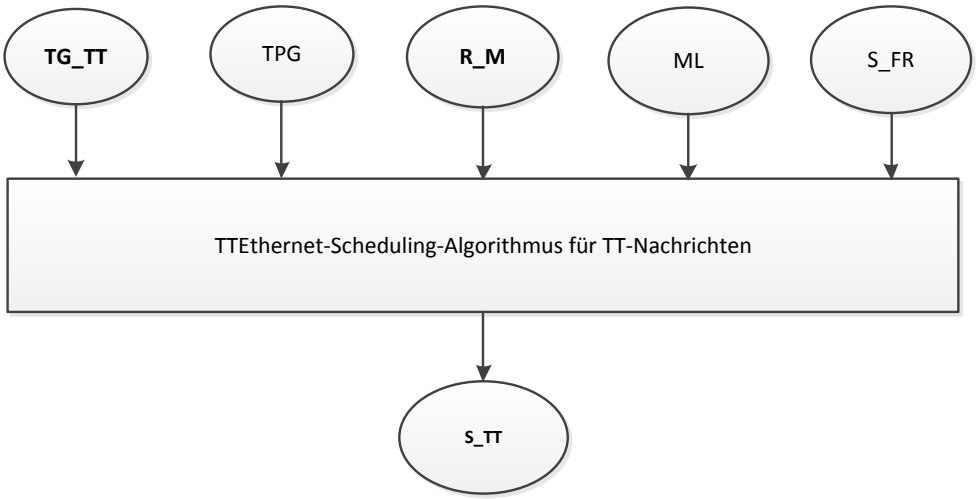


Abbildung 7.3: Input und Output des TTEthernet-Scheduling-Algorithmus für time-triggered Nachrichten

wobei:

- $T_{CC}[\mu s]$ die Länge eines TTEthernet-Kommunikationszyklus ist.
- $\mathcal{S}_{\mathbf{TT}}$ die Menge der Schedules aller time-triggered Nachrichten ist, $\mathcal{S}_{TT}(m) \in \mathcal{S}_{\mathbf{TT}}$.
- $\mathcal{S}_{\mathbf{RC}}$ die Menge der Schedules aller rate-constraint Nachrichten ist, $\mathcal{S}_{RC}(m) \in \mathcal{S}_{\mathbf{RC}}$.

Der Schedule einer TT-Nachricht lässt sich durch das folgende 5-Tupel beschreiben:

$$\mathcal{S}_{TT}(m) = (\mathbf{conf}(m), t_{sm}(m), t_{fm}(m), t_{pm}(m), N_{rep}(m)) \tag{7.8}$$

wobei:

- $\mathbf{conf}(m) \subset \mathbf{M}$ die Menge der Nachrichten ist, deren Routen sich mit der von m überschneiden; $\mathbf{conf}(m) = \{m_i \in \mathbf{M} | \mathbf{R_m} \cap \mathbf{R_{m_i}} \neq \emptyset\}$.
- $t_{sm}(m)[\mu s]$ die Startzeit der Nachricht m ist (siehe Definition 5.12).
- $t_{fm}(m)[\mu s]$ die Endzeit der Nachricht m ist (siehe Definition 5.13).
- $t_{pm}(m)[\mu s]$ die Zeit ist, nach der das Senden der Nachricht m innerhalb eines Kommunikationszyklus wiederholt wird.

- $N_{rep}(m) \in \mathbb{N}$ die Anzahl der wiederholten Übertragung der Nachricht m innerhalb eines Kommunikationszyklus mit Einhaltung der Periode $t_{pm}(m)$ ist.

Der Schedule einer RC-Nachricht $\mathcal{S}_{RC}(m)$ wird erst in Abschnitt 7.3.2 während der Migration des dynamischen Segments beschrieben.

Erstellung der time-triggered Nachrichten-Liste

Zu einer Scheduling-Strategie gehört auch eine passende Reihenfolge der Elemente (in diesem Fall TT-Nachrichten), mit der die Planung durchgeführt wird. Die Reihenfolge, mit der die TT-Nachrichten geplant werden, wird mit dem Algorithmus 2 festgelegt. Dieser nimmt die Menge der Task-Graphen $\mathbf{TG_{TT}}$ auf und gibt eine Liste $\mathcal{ML}$ aller Nachrichten $m \in \mathbf{TG_{TT}}$ sortiert nach aufsteigenden Deadlines und Release-Zeiten zurück. Diese Sortierung hält die Abhängigkeits-Beziehungen der Tasks ein (siehe Abschnitt 5.2).

Algorithm 2 Nachrichten-Liste erzeugen

```
1: procedure NachrichtenListeErzeugen
Require: TG_TT   \* Menge der Task-Graphen, die geplant werden sollten *\
Ensure: ML   \* Nachrichten-Liste nach Deadlines sortiert *\
2:     Erzeuge eine leere Liste ML   \* Nachrichten-Liste *\
3:     Füge alle Nachrichten aus TG_TT zu ML hinzu
4:     Sortiere die Nachrichten von ML nach aufsteigenden Deadlines
5:     Sortiere die Nachrichten von ML, die die gleiche Deadline haben, nach aufsteigenden
       Release-Zeiten
6:     return ML
7: end procedure
```

TTEthernet-Time-Triggered-Nachrichten Scheduling-Algorithmus

Der Schedule von TTEthernet TT-Nachrichten kann anhand des heuristischen Algorithmus 3 generiert werden. Dieser benötigt die Menge der Task-Graphen der TT-Anwendungen $\mathbf{TG_{TT}}$, eine TTEthernet-Topologie TPG, die Menge der Routen $\mathbf{R_M}$ aller Nachrichten, die Liste $\mathcal{ML}$, nach der die Nachrichten geplant werden sollen, und den FlexRay-Schedule $\mathcal{S}_{FR}$. Anhand dieser Elemente liefert er, wenn möglich, eine Menge von TT-Nachrichten-Schedules $\mathcal{S}_{\mathbf{TT}}$, welche einen ausführbaren Schedule für jede TT-Nachricht enthält. Dafür wird zuerst die Länge des TTEthernet-Kommunikationszyklus T_{CC} berechnet. Diese ist gleich der Länge des FlexRay-Kommunikationszyklus mal der Anzahl der FlexRay-Kommunikationszyklen (siehe Algorithmus 3 Zeile 2 und Abbildung 7.4). Danach wird versucht, die Zeitslots für die

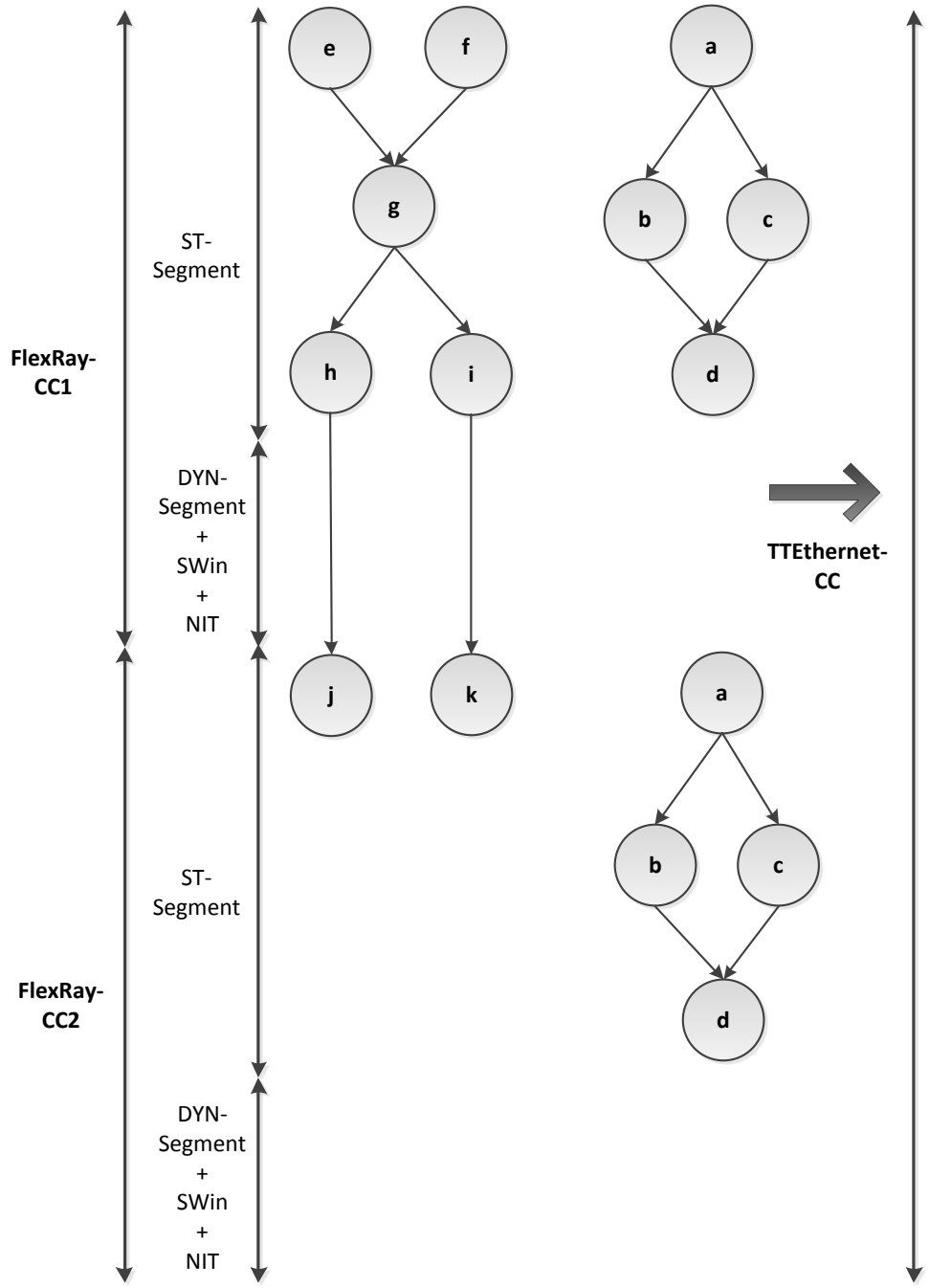


Abbildung 7.4: Umwandlung von mehreren FlexRay-Kommunikationszyklen in einem TTEthernet-Kommunikationszyklus

Algorithm 3 TTEthernet Time-Triggered-Schedule erzeugen

1: **procedure** TTSchldErzeugen
Require: $\mathbf{TG_{TT}} \subseteq \mathbf{TG}$ * *Task-Graphen der TT-Anwendungen* *\
Require: TPG * *Topologie inklusive Mapping der Task, auf die Prozessoren* *\
Require: $\mathbf{R_M}$ * *Menge der festgelegten Routen aller TT-Nachrichten* *\
Require: $\mathcal{ML}$ * *Liste, nach der die Planung der Nachrichten erfolgen soll* *\
Require: $\mathcal{S}_{FR}$ * *FlexRay-Schedule* *\
Ensure: $\mathcal{S}_{\mathbf{TT}}$ * *Menge der Schedules der TT-Nachrichten* *\
2: $T_{CC} \leftarrow \mathcal{S}_{FR}.T_{CC} \times \mathcal{S}_{FR}.N_{CC}$
3: Erstelle eine leere Menge $\mathcal{S}_{\mathbf{TT}}$ der TT-Nachrichten-Schedules
4: **while** $\mathcal{ML} \neq \emptyset$ **do**
5: sei m_i die erste Nachricht aus $\mathcal{ML}$
6: $t_{pm}(m_i) \leftarrow t_{dg}(TG_{m_i}), TG_{m_i} \in \mathbf{TG_{TT}} \wedge m_i \in TG_{m_i}$
7: $N_{rep}(m_i) \leftarrow \left\lfloor \frac{T_{CC}}{t_{pm}(m_i)} \right\rfloor$
8: Ermittle die Konflikt-Menge $\mathbf{conf}(m_i)$ der Nachricht m_i
9: $\Delta_{Slot} \leftarrow \zeta_{SW}(m_i) + (2 \times T_{cp})$ * *Slot-Länge berechnen* *\
10: **if** Existiert ein Zeitslot $[t_a, t_b = t_a + \Delta_{Slot})] : (t_{rm}(m_i) \leq t_a) \wedge (t_b \leq t_{dm}(m_i)) \wedge (\forall m_k \in \mathbf{conf}(m_i), (t_b \leq t_{sm}(m_k)) \vee (t_a \geq t_{fm}(m_k)))$ **then**
11: $t_{sm}(m_i) \leftarrow t_a$
12: $t_{fm}(m_i) \leftarrow t_b$
13: **else**
14: **return** $\emptyset$ * *Schedule nicht ausführbar* *\
15: **end if**
16: * *Die Intervalle für die Wiederholungen sollen auch frei sein.* *\
17: **if** $N_{rep}(m_i) > 0$ **then**
18: $k \leftarrow 1$
19: **while** $k \leq N_{rep}(m_i)$ **do**
20: $T_\theta = t_{pm}(m_i) \times k$
21: **if** Existiert kein Zeitslot $[t_a, t_b = t_a + \Delta_{Slot})] : (t_{sm}(m_i) + T_\theta \leq t_a) \wedge (t_b \leq t_{fm}(m_i) + T_\theta)$ **then**
22: **return** $\emptyset$ * *Schedule nicht ausführbar* *\
23: **end if**
24: $k \leftarrow k + 1$
25: **end while**
26: **end if**
27: $\mathcal{S}_{\mathbf{TT}} \leftarrow \mathcal{S}_{\mathbf{TT}} \cup \{\mathbf{conf}(m_i), t_{sm}(m_i), t_{fm}(m_i), t_{pm}(m_i), N_{rep}(m_i)\}$
28: $\mathcal{ML} \leftarrow \mathcal{ML} - m_i$ * *Lösche m_i von $\mathcal{ML}$* *\
29: **end while**
30: **return** $\mathcal{S}_{\mathbf{TT}}$
31: **end procedure**

Übertragung jeder TT-Nachricht m_i innerhalb des Kommunikationszyklus nach der Liste-Reihenfolge zu finden. Dafür wird für jede Nachricht $m_i \in \mathcal{ML}$ zunächst die Periode $t_{pm}(m_i)$, welche nichts anderes als die Deadline des Task-Graphen TG_{m_i} der Nachricht ist, berechnet. Der Kommunikationszyklus wird dann durch die Periode geteilt und abgerundet, um die Anzahl der Wiederholungen $N_{rep}(m_i)$ der Nachricht innerhalb des Kommunikationszyklus zu ermitteln. Abschließend wird die Menge der Konflikt-Nachrichten $\mathbf{conf}(m_i)$ von m_i ermittelt. Damit sind die Voraussetzungen erfüllt, um die Zeitslots der Nachricht m_i festzulegen. Es wird zunächst versucht, den ersten freien Zeitslot $[t_a, t_b]$ für die Übertragung der Nachricht zu finden. Dabei wird überprüft, ob dieser Zeitslot eine größere untere Grenze t_a als die Release-Zeit der Nachricht hat ($t_{rm}(m_i) \leq t_a$), für die Übertragung der Nachricht groß genug ist ($(t_b = t_a + \zeta_{SW}(m_i) + (2 \times T_{cp})) \leq t_{dm}(m_i)$) und sich mit den Zeitslots der Nachrichten $\mathbf{conf}(m_i)$, die mit m_i einen physikalischen Link teilen, nicht überschneidet. Für die Überprüfung ob, der Zeitslot ausreicht, wird die maximale Übertragungszeit der Nachricht $\zeta_{SW}(m_i)$ über alle ihre Routen (siehe Gleichung 5.9) und die Uhren-Genauigkeit T_{cp} in Betracht gezogen. Wenn der erste Zeitslot gefunden werden konnte, wird überprüft, ob solch ein Zeitslot für alle Wiederholungen $N_{rep}(m_i)$ zur Verfügung steht. Dabei wird die Periode $t_{pm}(m_i)$ der Nachricht eingehalten.

Wenn mit dem Algorithmus 3 ein Zeitslot $[t_{sm}(m_i), t_{fm}(m_i)]$ für jede TT-Nachricht $m_i \in \mathbf{TG_{TT}}$ (inklusive Wiederholung) gefunden werden kann, dann existiert ein ausführbarer Schedule der TT-Anwendungen. Daraus kann man schließen, dass die Anwendungen nach TTEthernet migriert werden können.

Der Prozess für die Migration des statischen Segments wird in Abbildung 7.5 zusammengefasst.

Der Algorithmus 3 ist eine Heuristik, weil das Ergebnis von den Eingaben abhängt. Wenn z. B. ein anderes Verfahren als jener des Algorithmus 2 für die Sortierung der Nachrichten angewandt wird, kann mit derselben Topologie und den gleichen Routen ein anderes Ergebnis geliefert werden. Analog gilt es auch, wenn andere Topologien oder Routen für eine gegebene Anwendung angewandt werden. Für eine gegebene Liste, Topologie und Routen operiert der Algorithmus 3 wie ein Suchalgorithmus. Dieser sucht für jede TT-Nachricht einen oder mehrere Übertragungsslots und würde auch eine Lösung finden, wenn es eine gibt.

Der Algorithmus 3 ist kein optimaler Algorithmus zum Scheduling von TT-Nachrichten, denn hier wurde für die simultane Übertragung von mehreren TT-Nachrichten nur überprüft, ob diese disjunkte Pfade haben. Besser wäre es, diese Überprüfung auf den Link-Ebenen durchzuführen. Dies mathematisch darzustellen kann sehr komplexer werden und würde den Rahmen dieser Arbeit sprengen. Solche weiter ausbauenden Gedanken für optimale Scheduling-Algorithmen sind Bestandteile zukünftiger Arbeiten.

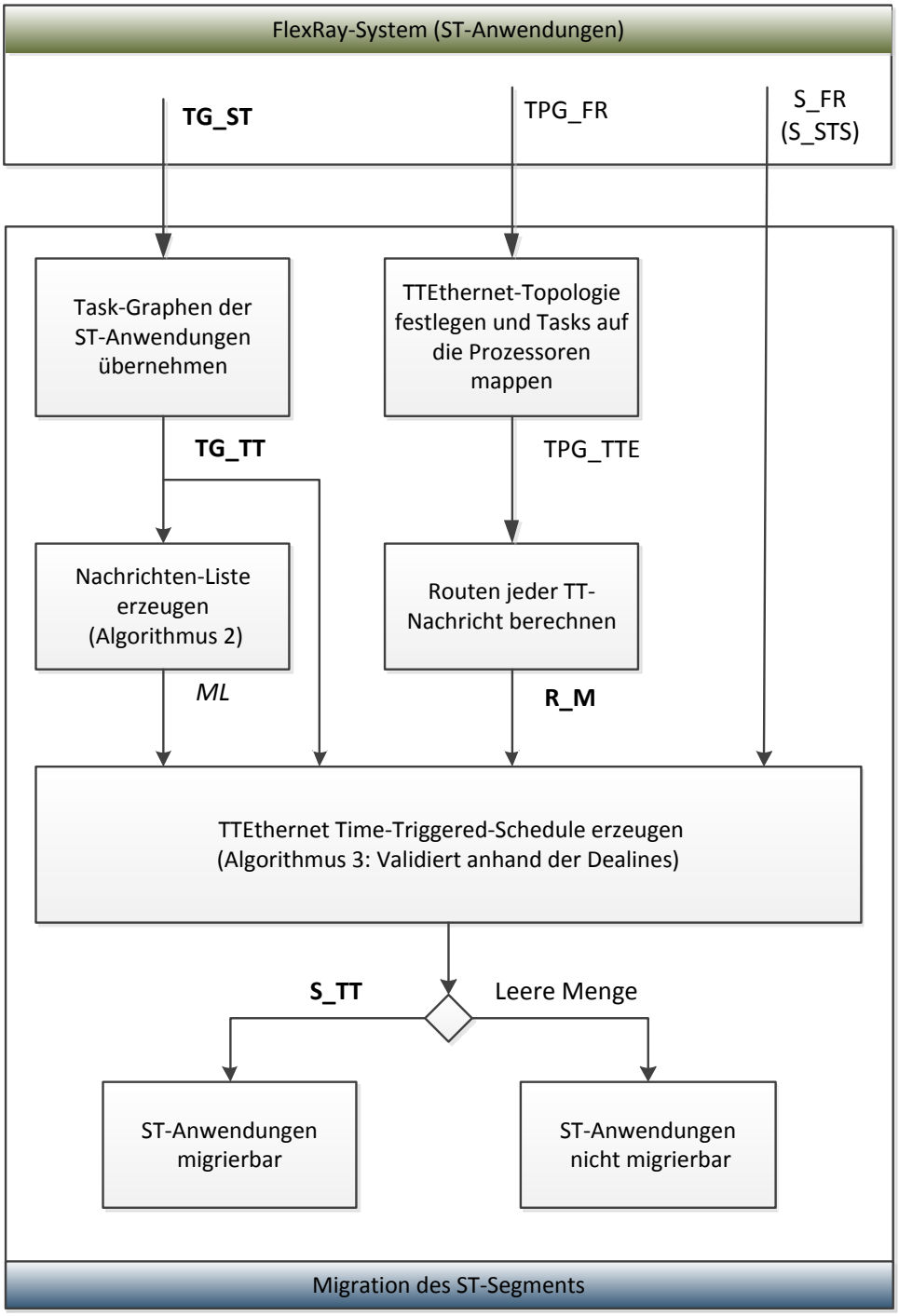


Abbildung 7.5: Prozess für die Migration des statischen Segments

7.2.5 Fallstudie: Migration des statischen Segments

In diesem Abschnitt wird ein time-triggered Automobil-System, welches für FlexRay designed wurde, nach TTEthernet migriert. Dieses wurde aus Nagarajan u. a. (2004) entnommen und besteht aus zwei Anwendungen, nämlich das EPS (Electric Power Steering) und das TC (Traction Control). Das EPS benutzt einen Elektromotor, um dem Fahrer die nötige Lenkunterstützung zu geben. Das TC, auch als ASR (Antriebsschlupfregelung) bekannt, sorgt dafür, dass die Räder beim Beschleunigen (auch bei rutschiger Fahrbahn) nicht durchdrehen. Diese Anwendungen beruhen auf mehreren Sensoren/Prozessoren/Aktuatoren, die untereinander interagieren, um Echtzeit- und Sicherheits-Funktionen zu realisieren, die festgelegt Deadlines einhalten müssen. Diese Deadlines sollen bei der Migration der verteilten Echtzeit-Anwendungen weiterhin erfüllt werden.

Die beiden Task-Graphen TG_{EPS} und TG_{TC} der Anwendungen werden in der Abbildung 7.6 gezeigt. Ihre Deadlines betragen jeweils $t_{dg}(TG_{EPS}) = 1500\mu s$ und $t_{dg}(TG_{TC}) = 3000\mu s$. Eine Bus-Topologie des Systems sowie die Zuweisung der Tasks zu den Prozessoren werden in der Abbildung 7.7 gezeigt, wobei hier Sensoren und Aktoren auch als Prozessor-Elemente betrachtet werden. Die Tabellen 7.3 und 7.4 fassen alle Daten (Releases, Deadlines, Ausführungszeiten, etc.) der beiden Anwendungen zusammen. Diese werden zum Erzeugen des Schedules und zum Prüfen dieser benötigt. Die Parameter des statischen Segments werden in der Tabelle 7.2 zusammengefasst. Dieses hat eine Länge von $3000\mu s$. Die maximale Payload beträgt $2Byte$. Das macht eine statische Slot-Länge von $T_{STSlot} = 118bit/10\frac{Mbit}{s} + 2 * 10\mu s = 31,8\mu s$, welche mit zusätzlichem Puffer und auf $35\mu s$ aufgerundet wurde. Somit ergibt sich eine maximale Anzahl von Slots von $N_{STSlot} = 3000\mu s/35\mu s = 85$. Für die Uhren-Abweichung T_{cp} wurde der maximal bekannte Wert von $10\mu s$ verwendet (vgl. Rausch, 2008, S. 10). Der FlexRay-Kommunikationszyklus besteht in diesem Fall aus einem Zyklus und somit beträgt die $CCRep$ für alle Nachrichten eins. Tabelle 7.5 zeigt einen ausführbaren Schedule der Nachrichten der Anwendungen für das statische Segment. Alle Nachrichten aus dieser Tabelle sind bereits nach dem Algorithmus 2 sortiert. Diese Reihenfolge wird während der gesamten Fallstudie beibehalten.

Tabelle 7.2: Fallstudie: Migration des statischen Segments: Parameter des statischen Segments.

T_{STS}	T_{STSlot}	N_{STSlot}
$\mu s3000$	$35\mu s$	85

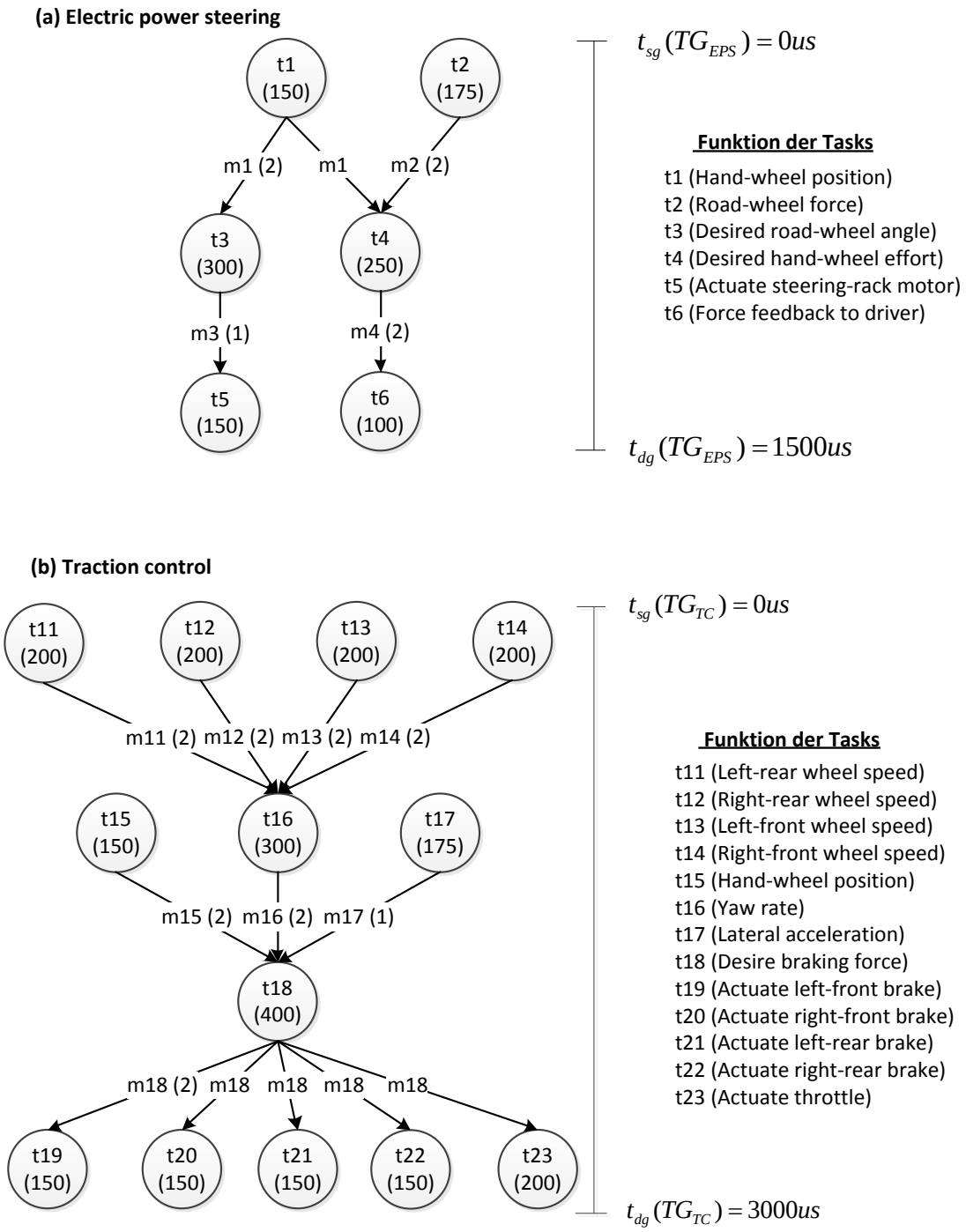


Abbildung 7.6: Fallstudie: Migration des statischen Segments: Task-Graphen

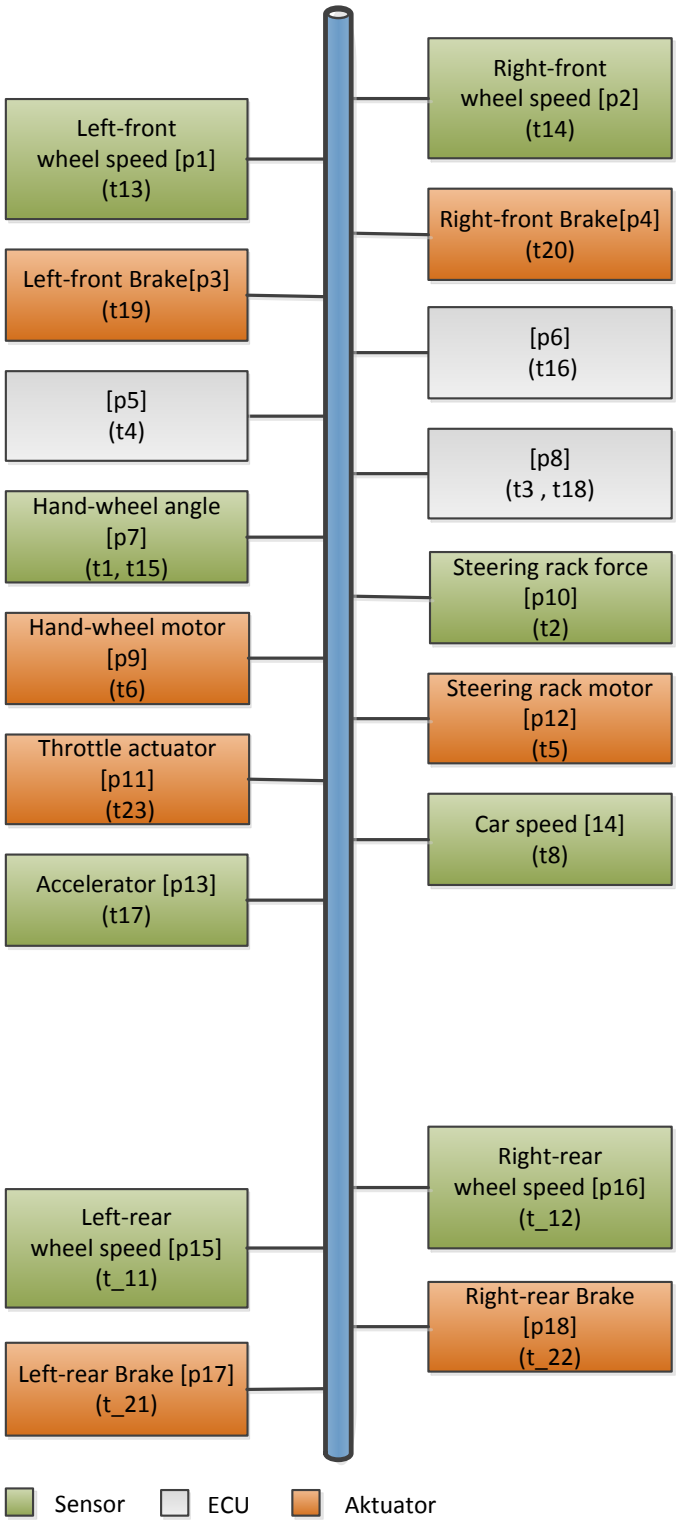


Abbildung 7.7: Bus-Topologie und Mapping der Tasks auf den Prozessoren– Fallstudie: Migration des statischen Segments

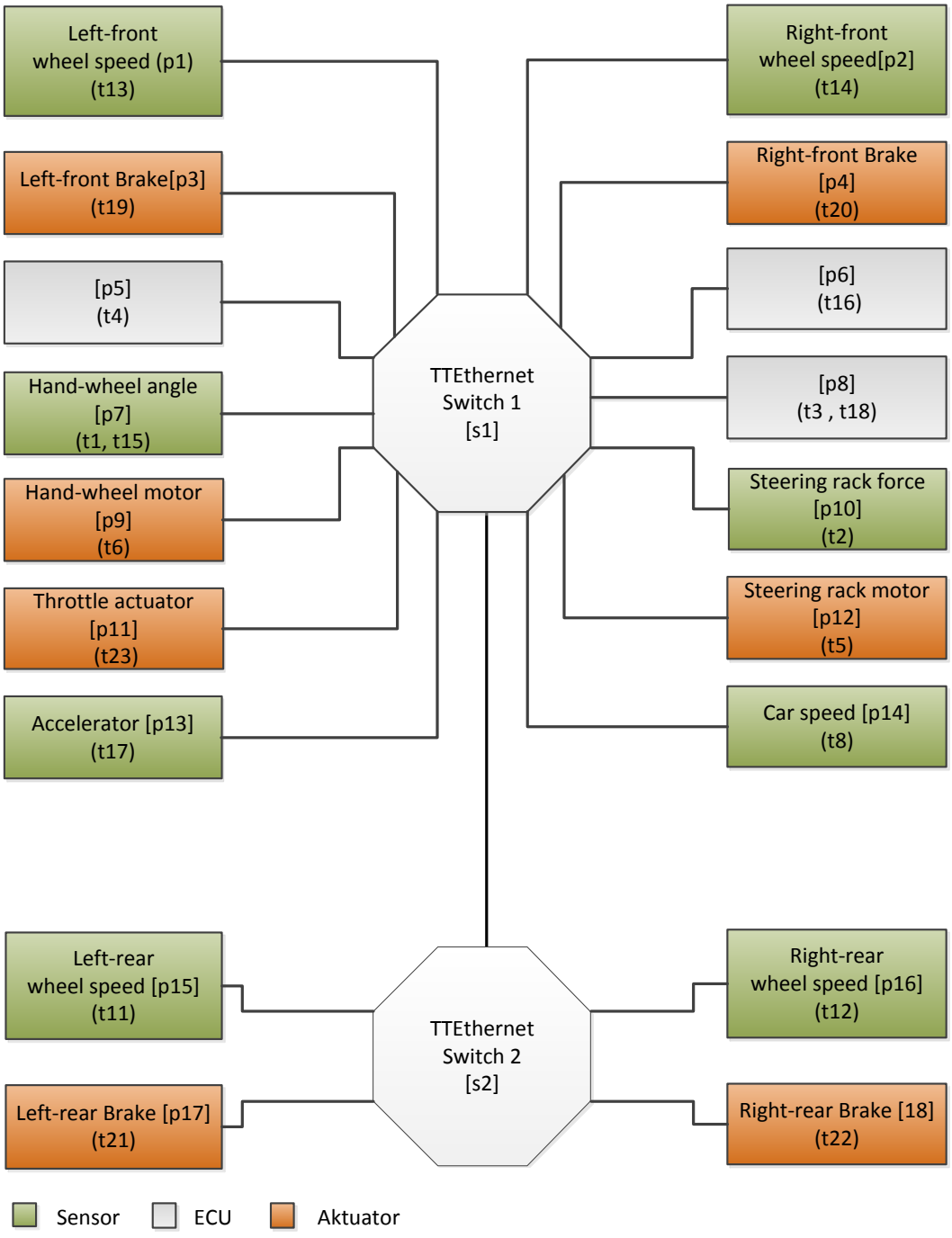


Abbildung 7.8: Switch-Topologie und Mapping der Tasks auf den Prozessoren – Fallstudie: Migration des statischen Segments

Tabelle 7.3: Fallstudie: Migration des statischen Segments: Releases und Deadlines der Tasks und Nachrichten der EPS-Anwendung. Außerdem werden in dieser Tabelle die Übertragungszeiten der Nachrichten $\zeta_B(m_i)$ gezeigt. Diese wurden mit einer Bandbreite von $10\frac{Mbit}{s}$ berechnet.

t_i	$t_W(t_i)$	$t_{r\tau}(t_i)$	$t_{d\tau}(t_i)$	m_i	$c(m_i)$	$\zeta_B(m_i)$	$t_{rm}(m_i)$	$t_{dm}(m_i)$
t_1	150	0	150	m_1	2	12	150	600
t_2	175	0	175	m_2	2	12	175	600
t_3	300	600	900	m_3	1	11	900	1350
t_4	250	600	850	m_4	2	12	850	1400
t_5	150	1350	1500	–	–	–	–	–
t_6	100	1400	1500	–	–	–	–	–

Tabelle 7.4: Fallstudie: Migration des statischen Segments: Releases und Deadlines der Tasks und Nachrichten der TC-Anwendung.

t_i	$t_W(t_i)$	$t_{r\tau}(t_i)$	$t_{d\tau}(t_i)$	m_i	$c(m_i)$	$\zeta_B(m_i)$	$t_{rm}(m_i)$	$t_{dm}(m_i)$
t_{11}	200	0	200	m_{11}	2	12	200	833
t_{12}	200	0	200	m_{12}	2	12	200	833
t_{13}	200	0	200	m_{13}	2	12	200	833
t_{14}	200	0	200	m_{14}	2	12	200	833
t_{15}	150	0	150	m_{15}	2	12	150	1766
t_{16}	300	833	1133	m_{16}	2	12	1133	1766
t_{17}	175	0	175	m_{17}	1	11	175	1766
t_{18}	400	1766	2166	m_{18}	2	12	2166	2799
t_{19}	150	2850	3000	–	–	–	–	–
t_{20}	150	2850	3000	–	–	–	–	–
t_{21}	150	2850	3000	–	–	–	–	–
t_{22}	150	2850	3000	–	–	–	–	–
t_{23}	200	2799	3000	–	–	–	–	–

Aus den Abbildungen 7.6, 7.7 und der Tabelle 7.5 kann man entnehmen, dass alle Voraussetzungen (FlexRay-Funktionsmodell, -Topologie, -Schedule) für die Migration erfüllt sind. Zur Erinnerung, es können die Task-Graphen (Funktionsmodell) bei der Migration beibehalten werden, da sich die Anforderungen des Systems auch nicht ändern. Die Topologie kann jedoch nicht beibehalten werden, da im TTEthernet nur Switch-Topologien in Frage kommen. Die Bus-Topologie aus der Abbildung 7.7 kann entweder mit einem Switch- oder durch zwei kaskadierte Switches ersetzt werden. Das letztere wurde ausgewählt (siehe Abbildung 7.8), da dadurch Worst-Case Szenarien untersucht werden können. Die Prozessoren und die Zuweisung der Tasks, auf diese wurden auch beibehalten. Mit den Task-Graphen, der TTEthernet-Topologie

Tabelle 7.5: Fallstudie: Migration des statischen Segments: Schedule der statischen Nachrichten.

m_i	$t_{rm}(m_i)$	$t_{dm}(m_i)$	$t_{sm}(m_i)$	$t_{fm}(m_i)$	$bSlot$	$slotRep$	bCC	$CCRep$
m_1	150	600	175	210	6(49)	43	1	1
m_2	175	600	210	245	7(50)	43	1	1
m_{11}	200	833	245	280	8		1	1
m_{12}	200	833	280	315	9		1	1
m_{13}	200	833	315	350	10		1	1
m_{14}	200	833	350	385	11		1	1
m_3	900	1350	910	945	27(70)	43	1	1
m_4	850	1400	945	980	28(71)	43	1	1
m_{15}	150	1766	385	420	12		1	1
m_{16}	1133	1766	1155	1190	34		1	1
m_{17}	175	1766	420	455	13		1	1
m_{18}	2166	2799	2170	2205	63		1	1

Tabelle 7.6: Fallstudie: Migration des statischen Segments: Routen und Konflikt-Nachrichten.

m_i	$\mathbf{R_{m_i}}$	$\mathbf{conf}(m_i)$	$\zeta_{SW}(m_i)$
m_1	$\{\{l_{[p_7,s_1]}, l_{[s_1,p_5]}\}, \{l_{[p_7,s_1]}, l_{[s_1,p_8]}\}\}$	$\{m_2, m_{15}\}$	15
m_2	$\{l_{[p_{10},s_1]}, l_{[s_1,p_5]}\}$	$\{m_1\}$	15
m_{11}	$\{l_{[p_{15},s_2]}, l_{[s_2,s_1]}, l_{[s_1,p_6]}\}$	$\{m_{12}, m_{13}, m_{14}\}$	25
m_{12}	$\{l_{[p_{16},s_2]}, l_{[s_2,s_1]}, l_{[s_1,p_6]}\}$	$\{m_{11}, m_{13}, m_{14}\}$	25
m_{13}	$\{l_{[p_1,s_1]}, l_{[s_1,p_6]}\}$	$\{m_{12}, m_{11}, m_{14}\}$	15
m_{14}	$\{l_{[p_2,s_1]}, l_{[s_1,p_6]}\}$		
m_3	$\{l_{[p_8,s_1]}, l_{[s_1,p_{12}]}\}$	$\{m_{18}\}$	15
m_4	$\{l_{[p_5,s_1]}, l_{[s_1,p_9]}\}$		
m_{15}	$\{l_{[p_7,s_1]}, l_{[s_1,p_8]}\}$	$\{m_1, m_{16}, m_{17}\}$	15
m_{16}	$\{l_{[p_6,s_1]}, l_{[s_1,p_8]}\}$	$\{m_1, m_{15}, m_{17}\}$	15
m_{17}	$\{l_{[p_{13},s_1]}, l_{[s_1,p_8]}\}$	$\{m_1, m_{15}, m_{16}\}$	15
m_{18}	$\{\{l_{[p_8,s_1]}, l_{[s_1,p_3]}\}, \{l_{[p_8,s_1]}, l_{[s_1,p_4]}\}, \{l_{[p_8,s_1]}, l_{[s_1,s_2]}, l_{[s_2,p_{18}]}\}, \{l_{[p_8,s_1]}, l_{[s_1,p_{11}]}\}\}$	$\{m_3\}$	25

und dem FlexRay-Schedule wurde ein Schedule für TTEthernet time-triggered Nachrichten anhand des Algorithmus 3 generiert. Dabei wurden die Länge des Kommunikationszyklus ($T_{CC} = 3000\mu s \times 1 = 3000\mu s$), die Routen, die Konflikt-Menge, die Übertragungszeit bei $100Mbi/s$ (Tabelle 7.6) und ein statischer Zeitslot $[t_{sm}(m), t_{fm}(m)]$ jeder Nachricht ermittelt.

Das letztere kann aus der Tabelle 7.7 entnommen werden. Man kann daraus erkennen, dass alle Nachrichten ihre festgelegten Deadlines einhalten. Somit kann man schließen, dass die Anwendungen migrierbar sind.

Tabelle 7.7: Fallstudie: Migration des statischen Segments: TTEthernet-TT-Nachrichten-Schedule.

m_i	$t_{rm}(m_i)$	$t_{dm}(m_i)$	$t_{pm}(m_i)$	$N_{rep}(m_i)$	$t_{ms}(m_i)$	$t_{mf}(m_i)$
m_1	150	600	1500	1	150(1650)	185(1685)
m_2	175	600	1500	1	185(1685)	220(1720)
m_{11}	200	833	0	0	200	245
m_{12}	200	833	0	0	245	290
m_{13}	200	833	0	0	290	325
m_{14}	200	833	0	0	200	235
m_3	900	1350	1500	1	900(2400)	935(2435)
m_4	850	1400	1500	1	850(2350)	885(2385)
m_{15}	150	1766	0	0	185	220
m_{16}	1133	1766	0	0	1133	1168
m_{17}	175	1766	0	0	220	255
m_{18}	2166	2799	0	0	2166	2211

7.3 Migration des dynamischen Segments

In diesem Abschnitt werden Methoden für die Migration eines FlexRay dynamischen Segments dargelegt. Dafür ist es wichtig zu wissen, wie die Schedules von DYN- und RC-Nachrichten beschrieben werden. Es werden zwei Strategien für die Migration eines FlexRay dynamischen Segments vorgestellt. Bei der ersten Strategie werden die DYN-Nachrichten auf TT-Nachrichten abgebildet und bei der zweiten auf RC-Nachrichten. Die Validierung des letzteren wird durch den Vergleich der Worst-Case-Response-Times (WCRT) der Nachrichten erfolgen. So werden außerdem in diesem Abschnitt Methoden zur Berechnung von WCRT von DYN- und RC-Nachrichten beschrieben.

7.3.1 Beschreibung des Schedules eines dynamischen FlexRay-Segments

Der Schedule eines FlexRay dynamischen Segments $\mathcal{S}_{DYNS}$ kann durch das folgende 4-Tupel beschrieben werden:

$$\mathcal{S}_{DYNS} = (T_{DYNS}, T_{MSlot}, N_{MSlot}, \mathcal{S}_{\mathbf{DYN}}) \tag{7.9}$$

wobei:

- $T_{DYNS}[\mu s]$ die Länge des dynamischen Segments ist.
- $T_{MSlot}[\mu s]$ die Länge eines Minislots ist, $T_{MSlot} \in [2, 63]$, $T_{MSlot} \in \mathbb{N}$.
- N_{MSlot} die Anzahl der Minislots des dynamischen Segments ist, $N_{MSlot} \in [0, 7988]$, $N_{MSlot} \in \mathbb{N}$.
- $\mathcal{S}_{\mathbf{DYN}} \in \mathbb{N}^4$ die Menge der Schedules aller DYN-Nachrichten ist, $\mathcal{S}_{DYN}(m) \in \mathcal{S}_{\mathbf{DYN}}$.

Die Anzahl der Minislots eines dynamischen Segments kann mit der folgenden Gleichung berechnet werden:

$$N_{MSlot} = \left\lfloor \frac{T_{DYNS}}{T_{MSlot}} \right\rfloor \tag{7.10}$$

Der Schedule einer DYN-Nachricht lässt sich durch das folgende 4-Tupel beschreiben:

$$\mathcal{S}_{DYN}(m) = (mSlot(m), bCC(m), CCRep(m), ltMSlot(m)) \tag{7.11}$$

wobei:

- $mSlot(m)$ die Nummer des Minislots ist, in dem die Übertragung der Nachricht m gestartet werden soll, $mSlot(m) \in [0, 7988]$, $mSlot(m) \in \mathbb{N}$.
- $bCC(m)$ der erste Kommunikationszyklus ist, in dem die Nachricht m gesendet wird, $bCC(m) \in [0, 63], bCC(m) \in \mathbb{N}$.
- $CCRep(m)$ die Anzahl der Kommunikationszyklen ist, nachdem die Übertragung der Nachricht m wiederholt wird, $CCRep(m) = 2^n, n \in [0, 6], n \in \mathbb{N}$.
- $ltMSlot(m)$ der letzte Minislot ist, in dem die Übertragung der DYN-Nachricht m noch starten kann, $ltMSlot(m) \in [0, 7988]$, $ltMSlot(m) \in \mathbb{N}$.

Der Wert von $ltMSlot(m)$ kann mit der folgenden Gleichung bestimmt werden:

$$ltMSlot(m) = N_{MSlot} - \left\lceil \frac{\zeta_B(m)}{T_{MSlot}} \right\rceil \tag{7.12}$$

Abbildung 7.9 zeigt ein Beispiel-Schedule von 3 DYN-Nachrichten (m_{10}, m_{11} und m_{12}). Hier haben alle Nachrichten die gleiche Länge und können innerhalb eines Minislots übertragen werden. Die Schedules der einzelnen Nachrichten sehen folgendermaßen aus: $\mathcal{S}_{DYN}(m_{10}) = (10, 0, 1, 11)$, $\mathcal{S}_{DYN}(m_{11}) = (11, 0, 2, 11)$ und $\mathcal{S}_{DYN}(m_{12}) = (11, 3, 4, 11)$.

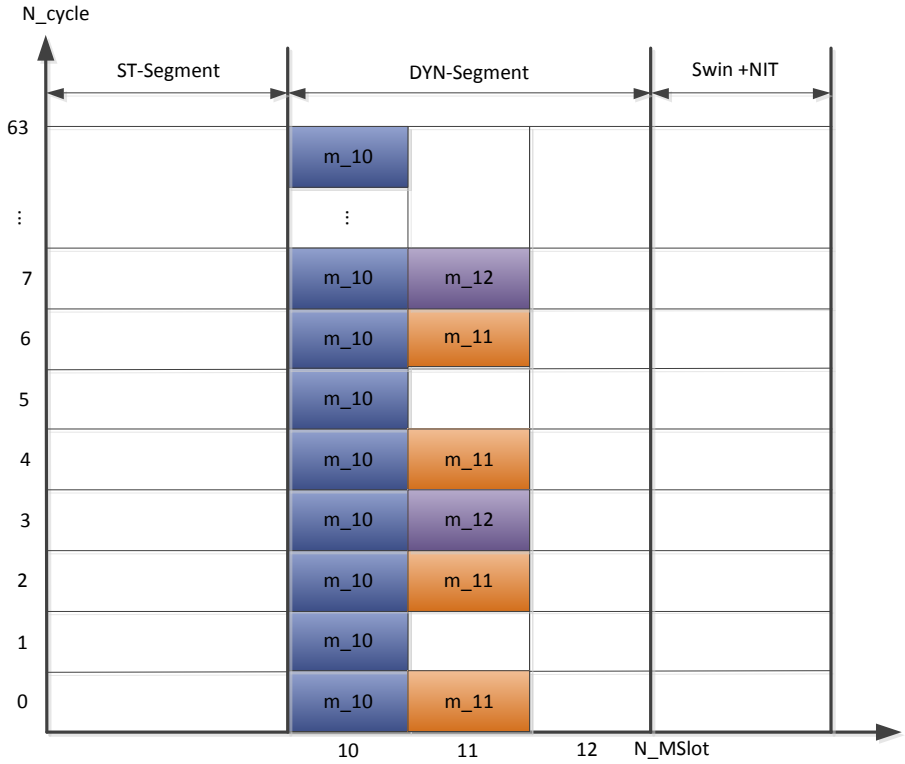


Abbildung 7.9: Beispiel-Schedules von DYN-Nachrichten

7.3.2 Beschreibung des Schedules einer RC-Nachricht

Der Schedule einer rate-constraint Nachricht (RC-Nachricht) kann durch folgendes 3-Tupel beschrieben werden:

$$\mathcal{S}_{RC}(m) = (CtId, T_{BAG}, D_{max}) \tag{7.13}$$

wobei:

- $CtId$ die Critical-Traffic-ID der Nachricht m ist, $CtId \in [1, 2^{16}], CtId \in \mathbb{N}$.
- $T_{BAG}[us]$ der minimale Zeit-Abstand ist, mit dem die Nachricht m gesendet werden soll, $T_{BAG} \in \mathbb{N}$.
- $D_{max}[Byte]$ die maximale Datenmenge ist, die mit der Nachricht m übertragen werden kann.

7.3.3 Worst-Case-Response-Time-Analyse des dynamischen FlexRay-Segments

In Pop u. a. (2007b) wurde ein Verfahren dargelegt, womit man die Worst-Case-Response-Time (WCRT) [4] von dynamischen Nachrichten (DYN-Nachrichten) untersuchen kann. Die Übertragung einer DYN-Nachricht wird hauptsächlich verzögert durch die Übertragung von statischen Nachrichten und von anderen dynamischen Nachrichten mit höheren Prioritäten. Die WCRT $\Re_{DYN}(m)$ einer DYN-Nachricht m kann mit der folgenden Gleichung ermittelt werden:

$$\Re_{DYN}(m) = t_\sigma(m) + t_\psi(m) + \zeta_B(m) \tag{7.14}$$

wobei:

- $\zeta_B(m)$ die Übertragungszeit der Nachricht m ist (siehe Gleichung 5.7).
- $t_\sigma(m)$ die längste Verzögerung der Nachricht m innerhalb eines Kommunikationszyklus ist, wenn diese gerade ihren Minislot verpasst hat (siehe Gleichung 7.15).
- $t_\psi(m)$ die Verzögerung der Nachricht m ist, welche durch das statische Segment und die Übertragung von DYN-Nachrichten mit höheren Prioritäten verursacht wird (siehe Gleichung 7.16).

Ein Knoten, der eine DYN-Nachricht senden will, wartet, bis der Minislot-Zähler gleich dem Minislot der Nachricht ist und greift dann auf den Bus zu, um die Nachricht zu senden. Es kann aber sein, dass der Knoten einen Übertragungsversuch erst startet, wenn der Minislot-Zähler bereits über den Minislot der Nachricht ist. Im diesen Fall muss der Knoten auf den nächsten Kommunikationszyklus warten, in der Hoffnung, dann übertragen zu können. Die Verzögerung einer DYN-Nachricht während eines Zyklus, wenn diese ihren Minislot knapp verpasst hat, wird mit der Gleichung 7.15 bestimmt.

$$t_\sigma(m) = T_{CC} - (T_{STS} + (MSlot_m - 1) \times T_{MSlot}) \tag{7.15}$$

Konnte eine DYN-Nachricht in einem Kommunikationszyklus nicht übertragen werden, wird versucht, diese im folgenden Zyklus in seinem Minislot zu übertragen. Diese muss jedoch neben dem statischen Segment (T_{STS}) noch auf die Übertragung von Nachrichten mit höherer Prioritäten warten. Diese werden mit $hp(m)$ gekennzeichnet. Außerdem muss diese eine

[4] Oder auch Worst-Case-Übertragungszeit.

Minislot-Länge für jede Nachricht mit einer höheren Priorität, die nicht übertragen werden, verzögert werden. Diese werden mit $ms(m)$ gekennzeichnet. Es kann auch passieren, dass eine DYN-Nachricht über mehrere Zyklen verzögert wird, wenn Nachrichten mit höheren Prioritäten wieder übertragen werden. Wenn diese immer wieder übertragen werden, kann es sogar dazu führen, dass die jeweiligen DYN-Nachrichten mit niedrigeren Prioritäten verhungern, also nie zur Übertragung kommen. Die Anzahl der Zyklen, die eine DYN-Nachricht verpasst hat, wird mit $N_{MissedCC}, N_{MissedCC} \in \mathbb{N}$, gekennzeichnet. Alle genannten Verzögerungen können mit der folgenden Gleichung berechnet werden:

$$\begin{aligned} t_\psi(m) = T_{STS} \\ + \sum_{\forall m_i \in hp(m)} \zeta_B(m_i) \\ + |ms(m)| \times T_{MSlot} \\ + N_{MissedCC} \times T_{CC} \end{aligned} \tag{7.16}$$

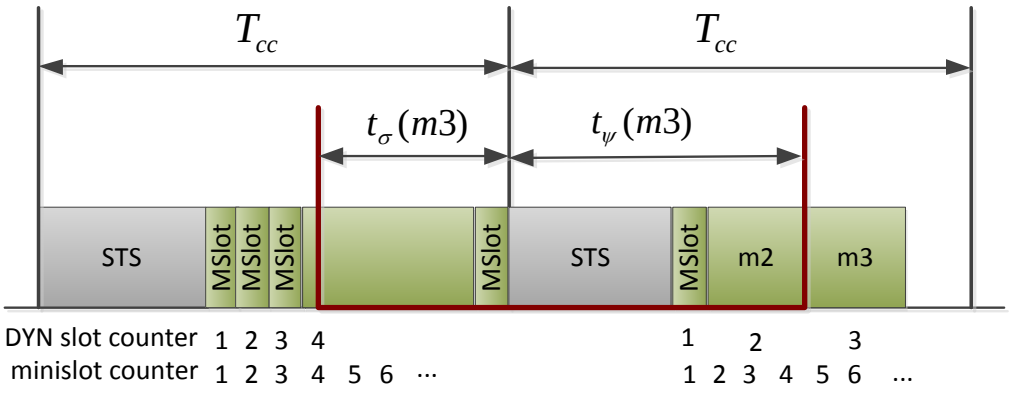


Abbildung 7.10: Beispiel-Worst-Case-Response-Time einer DYN-Nachricht (vgl. Pop u. a., 2007b)

Dadurch, dass die Gleichung 7.14 hier lediglich verwendet und nicht entwickelt wurde, wird für ihre Diskussionen und Einschränkungen auf Pop u. a. (2007b) verwiesen.

Abbildung 7.10 illustriert eine mögliche WCRT der Nachricht m_3. Diese soll im dritten Minislot übertragen werden. Im ersten Zyklus kann man sehen, dass die Nachricht knapp

ihren Slot verpasst hat und deswegen im aktuellen Zyklus um $t_\sigma(m_3)$ verzögert werden muss. Im nächsten Zyklus muss die Nachricht noch auf das statische Segment und die beiden höher priorisierten Nachrichten mit den Minislots 1 und 2 warten. Im Minislot 1 wurde keine Nachricht übertragen, im Minislot 2 jedoch die Nachricht m_2, sodass die Nachricht m_3 im zweiten Zyklus länger als 2 Minislots zusätzlich verzögert werden muss. Im diesem Beispiel kommt die Nachricht m_3 bereits im zweiten Zyklus zur Übertragung. Es hätte jedoch auch sein können, dass die Nachrichten mit den Minislots 1 und 2 mehrmals hintereinander übertragen werden und dabei alle Minislots belegen, so dass die Nachricht m_3 über mehrere Zyklen verzögert wird oder nicht zur Übertragung kommt.

7.3.4 Worst-Case-Response-Time-Analyse von rate-constraint Nachrichten

Es gibt bereits mehrere Arbeiten, die sich der numerischen Evaluierung von Worst-Case-Response-Time (WCRT) von Nachrichten in AFDX-Netzwerken gewidmet haben. Hierfür gibt es zwei Berechunsmethoden – das Network-Calculus (vgl. Zeng und Song, 2009; Scharbarg u. a., 2009; Boyer und Fraboul, 2008) und den Trajectory-Ansatz (vgl. Hu u. a., 2011; Bauer u. a., 2009, 2010). Es wurde jedoch bewiesen, dass der Trajectory-Ansatz genauere Ergebnisse liefert als das Network-Calculus (vgl. Bauer u. a., 2009, s. 5). Daher wird hier der Trajectory-Ansatz verwendet. Dieser wurde von Bauer u. a. (2011) auf die Priorisierung der Nachrichten erweitert. Aufgrund dessen, dass die Priorisierung der Nachrichten eine große Bedeutung für die Migration der DYN-Nachrichten nach RC-Nachrichten hat, wird Bauer u. a. (2011) bei der Berechnung der WCRTs als Referenz in Betracht gezogen.

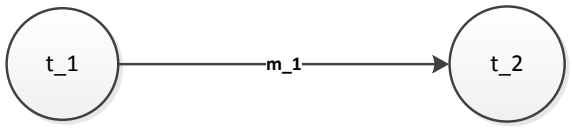
Die zusätzliche Verzögerung einer RC-Nachricht m, also die Verzögerung, die zu der Übertragungszeit $\zeta_{SW}(m)$ hinzukommt, wird verursacht durch andere RC-Nachrichten mit niedrigeren Prioritäten, deren Übertragung während der Übertragung von m bereits gestartet wurde, andere RC-Nachrichten mit höheren Prioritäten und alle TT-Nachrichten. Die Nachricht m kann jedoch nur dann von den eben genannten Nachrichten verzögert werden, wenn sich ihre Routen mit deren Routen überschneiden, also wenn Kollisionen am Ausgangsport des Sender-Endsystems oder des weiterleitenden Switch auftreten. Die WCRT einer RC-Nachricht hängt stark von der Priorität der Nachricht, Konfiguration des Netzwerks und von der vorhandenen Netzwerk-Last, ab. Diese kann mit der folgenden Gleichung ermittelt werden:

$$
\begin{aligned}
\Re_{RC}(m) = {} & \zeta_{SW}(m) \\
& + \max_{\forall R_m \in \mathbf{R_m}} \{ \\
& + \underbrace{\zeta_B(m_k) \times |R_m|, (m_k \in lp(m)) \wedge (R_m \cap \mathbf{R_{m_k}} \neq \emptyset)}_{\text{Verzögerung durch eine nieder-priorisierte Nachricht}} \\
& + \underbrace{\sum_{\forall m_h \in hp(m) \wedge R_m \cap \mathbf{R_{m_h}} \neq \emptyset} \zeta_B(m_h) \times |R_m \cap \mathbf{R_{m_h}}|}_{\text{Verzögerung durch höher-priorisierte Nachrichten}} \\
& \}
\end{aligned}
\tag{7.17}
$$

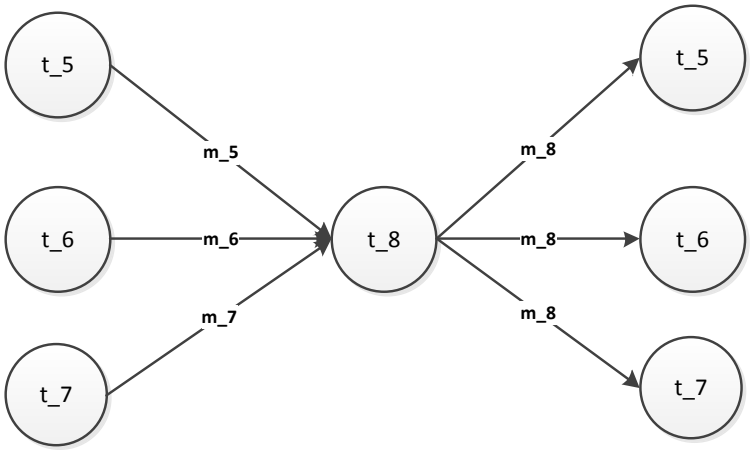
wobei:

- $\Re_{RC}(m)[\mu s]$ die WCRT der Nachricht m ist.
- $\zeta_{SW}(m)[\mu s]$ die Übertragungszeit der Nachricht m ohne Verzögerungen von TT-Nachrichten und RC-Nachrichten mit hoher Priorität ist (siehe Gleichung 5.8).
- $\mathbf{R_m}$ die Menge der Routen der Nachricht m ist. Die WCRT wird für alle Routen von m berechnet und das Maximum in Betracht gezogen.
- $d(m_i)[Byte]$ die gesamte Datenmenge inklusive Protokoll-Overhead ist, welche mit m_i übertragen wird.
- m_k der Repräsentant einer Nachricht mit niedrigerer Priorität als m ist, und dessen Routen sich mit der von m überschneiden.
- $\zeta_B(m_k)[\mu s]$ die Übertragungszeit der Nachricht m_k auf einem Link aus R_m ist (siehe Gleichung 5.7).
- $lp(m)$ die Menge der Nachrichten ist, die eine niedrigere Priorität als m haben. Die Verzögerung einer höher priorisierten Nachricht durch eine niedriger priorisierte Nachricht soll auch berücksichtigt werden, da die Übertragung einer Nachricht im TTEthernet wegen Store-And-Forward nicht unterbrochen werden kann.
- $hp(m)$ die Menge der Nachrichten ist, die eine höhere Priorität als m haben. Man soll bedenken, dass alle TT-Nachrichten eine höher Priorität als jede RC-Nachricht haben.

(a) Time-Triggered Anwendung



(a) Rate-Constraint Anwendung



(c) Topologie und Mapping der Tasks auf den Prozessoren

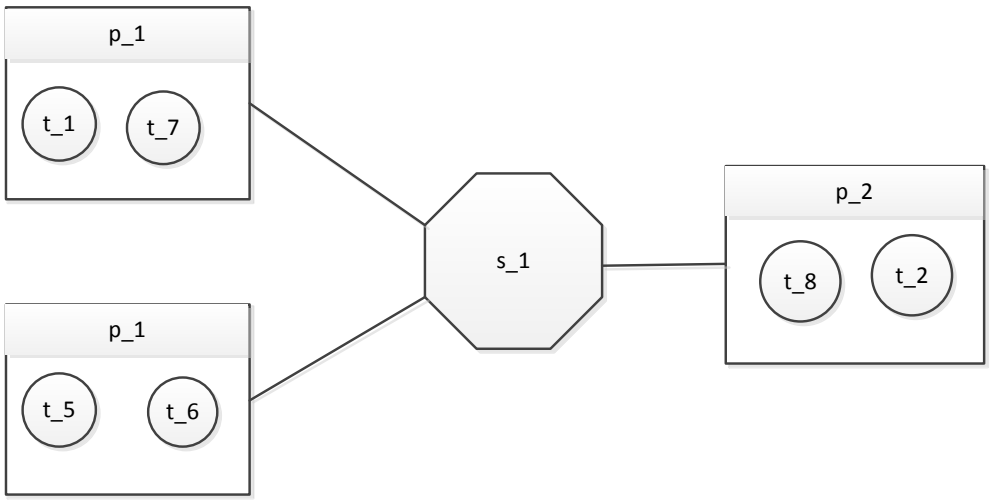


Abbildung 7.11: Beispiel-Worst-Case-Response-Time von RC-Nachrichten

Die Gleichung 7.17 wurde vereinfacht und im Bezug auf diesen Kontext aufgestellt. Es wurde dabei angenommen, dass alle RC-Nachrichten die Gleiche BAG haben, jedoch verschiedene Prioritäten. Deswegen kann auch eine RC-Nachricht durch alle höher priorisierten RC-Nachrichten, sofern die Routen sich überschneiden, verzögert werden. Außerdem wurde die serialisierte Übertragung der Nachrichten nicht berücksichtigt. Dabei wird eine nieder priorisierte Nachricht m_2 durch eine höher priorisierte Nachricht m_1 oder eine sich in der Übertragung befindene nieder priorisierte Nachricht m_3 nicht über die gesamte Route (wie dies hier der Fall ist), sondern lediglich bei der ersten Kollision verzögert, denn bei der nächsten möglichen Kollision soll die Übertragung der Nachricht m_1 bzw. m_3 bereits abgeschlossen sein, falls diese eine Länge kleiner oder gleich der von m_2 hat und nicht von anderen Nachrichten zusätzlich verzögert wurde. Solches Verhalten mathematisch darzustellen ist komplex und würde den Rahmen dieser Arbeit sprängen. Siehe hierfür Bauer u. a. (2009, 2011).

Die Berechnung der WCRT einer RC-Nachricht wird durch die Abbildung 7.11 illustriert. Diese besteht aus zwei fiktiven Anwendungen – eine TT-Anwendung (Abbildung 7.11(a)) und eine RC-Anwendung (Abbildung 7.11(b)). Die Topologie der Anwendungen und das Mapping der Tasks auf den Prozessoren werden in der Abbildung 7.11(c) gezeigt. Nehmen wir die Nachricht m_6 als Beispiel für die Untersuchung an. Wenn man annimmt, dass der Index der Nachricht die Prioritäten festlegt, dann hat man $lp(m_6) = \{m_5\}$, $hp(m_6) = \{m_1, m_7, m_8\}$. Weiterhin hat man $|R_{m_6} \cap \mathbf{R_{m_1}}| = |\{l_{[s_1,p_2]}\}| = 1$, $|R_{m_6} \cap \mathbf{R_{m_7}}| = |\{l_{[s_1,p_2]}\}| = 1$, $|R_{m_6} \cap \mathbf{R_{m_8}}| = |\{\emptyset\}| = 0$. Nun, da die höher und niedriger priorisierten Nachrichten als m_6 und die Routen-Überschneidungen bekannt sind, kann die WCRT berechnet werden. Dabei wird angenommen, dass alle Nachrichten die gleiche Datenmenge $d(m_i) = 100Byte$ haben und mit der gleichen Datenrate $b(l) = 100\frac{Mbit}{s}$ übertragen werden.

$$\begin{aligned} \Re_{RC}(m_6) &= \zeta_{SW}(m_6) + \zeta_B(m_5) \times 2 + \zeta_B(m_1) \times 1 + \zeta_B(m_7) \times 1 \\ &= 18,4 + 16 + 8 + 8 = 50,4\mu s \end{aligned} \tag{7.18}$$

7.3.5 Migrationsstrategie 1: Abbildung von DYN-Nachrichten auf TT-Nachrichten

Bei der Migration des dynamischen Segments können die DYN-Nachrichten auf TT-Nachrichten abgebildet werden. Die Idee hierbei ist es, für jede DYN-Nachricht einen TT-Slot zu reservieren. Hierfür wird der Algorithmus 5 verwendet. Dieser benötigt die Menge der zu planenden Task-Graphen $\mathbf{TG_{TT}}$, die Menge der Routen $\mathbf{R_M}$ aller Nachrichten aus $\mathbf{TG_{TT}}$, die Liste $\mathcal{ML}$, nach der die Reservierung der Slots erfolgen soll, der FlexRay-Schedule $\mathcal{S}_{FR}$ inklusive des

Schedules des dynamischen Segments $\mathcal{S}_{DYNS}$ und des Schedules $\mathcal{S}_{\mathbf{TT}}$ der TT-Nachrichten aus der Migration des statischen Segments. Er generiert anhand dieser Elemente, wenn möglich, einen Schedule für jede TT-Nachricht aus $\mathbf{TG_{TT}}$ und inkludiert diesen in die Menge der während der Migration des dynamischen Segments erzeugten Schedules $\mathcal{S}_{\mathbf{TT}}$. Also besteht der Kommunikationszyklus schließlich nur noch aus TT-Nachrichten. Die Task-Graphen $\mathbf{TG_{TT}}$ korrespondieren mit den ursprünglichen Task-Graphen der DYN-Anwendungen $\mathbf{TG_{DYN}}$. Die Menge $\mathbf{R_M}$ der Routen wird für die Berechnung der Übertragungszeiten $\zeta_{SW}(m)$ benötigt.

Die Nachrichten-Liste $\mathcal{ML}$ wird hier nicht mit dem Algorithmus 2 bestimmt, sondern mit dem Algorithmus 4. Dieser sortiert die DYN-Nachrichten anhand deren Schedules nach aufsteigender Minislots. Nachrichten, die den selben Minislot haben, werden anschließend nach aufsteigendem Basis-Zyklus $bCC(m)$ sortiert. Somit werden die Prioritäten der Nachrichten eingehalten. Ein neuer Algorithmus für die Festlegung der Nachrichten-Liste ist hier nötig, weil die Start-Zeiten der event-triggered Nachrichten nicht festgelegt werden können, da nicht immer bekannt ist, wann diese gesendet werden. Außerdem werden auch in der Regel die Deadlines der event-triggered Nachrichten nicht einzeln festgelegt, sondern es wird eine Deadline für alle Nachrichten einer Anwendung festgelegt.

Algorithm 4 Liste der DYN-Nachrichten für das Scheduling erzeugen

1: **procedure** ListeDYNNachrichten
Require: $\mathcal{S}_{\mathbf{DYN}}$ * *Schedule aller DYN-Nachrichten* *\
Ensure: $\mathcal{ML}$ * *Liste der DYN-Nachrichten sortiert nach Priorität* *\
2: Erzeuge eine leere Liste $\mathcal{ML}$ * *Nachrichten-Liste* *\
3: Füge alle Nachrichten aus $\mathcal{S}_{\mathbf{DYN}}$ zu $\mathcal{ML}$ hinzu
4: Sortiere die Nachrichten von $\mathcal{ML}$ nach aufsteigenden Minislots
5: Sortiere die Nachrichten von $\mathcal{ML}$, die den gleichen Minislot haben nach aufsteigendem Basis-Zyklus $bCC(m)$
6: **return** $\mathcal{ML}$
7: **end procedure**

Der Algorithmus 5 funktioniert im Grunde genommen wie der Algorithmus 3 mit den Unterschieden, dass die Wiederholungen $N_{rep}(m)$ und Perioden $t_{pm}(m_i)$ anhand des bestehenden dynamischen Segment-Schedules und nicht anhand des Task-Graphen bestimmt werden; die Reservierung der TT-Slots mit den DYN-Nachrichten mit den höchsten Prioritäten (kleinere Minislots zuerst) angefangen wird (siehe Algorithmus 4) und die Releasezeiten der Nachrichten nicht überprüft werden, da nicht immer vorab bekannt ist, wann event-triggered Nachrichten eintreffen. Außerdem wird für die Validierung des gefundenen TT-Slots nicht überprüft, ob bei der Übertragung der Nachricht in diesem Slot die Deadline eingehalten wird, wie dies der Fall bei „normalen“ TT-Nachrichten ist. Es wird eher überprüft, ob die Nachricht ihre Deadline

Algorithm 5 Abbildung von DYN-Nachrichten auf TT-Nachrichten

procedure DYNMsgToTTMsg
Require: $\mathbf{TG_{TT}} \subset \mathbf{TG}$ * *Task-Graphen, welche mit den DYN-Anwendungen-Task-Graphen korrespondieren* *\
Require: $\mathbf{R_M}$ * *Menge der Routen aller Nachrichten aus* $\mathbf{TG_{TT}}$ *\
Require: $\mathcal{S}_{FR}$ * *FlexRay-Schedule inklusive Schedule des dynamischen Segments* $\mathcal{S}_{DYNS}$ *\
Require: $\mathcal{ML}$ * *Liste der DYN-Nachrichten sortiert nach Priorität* *\
Require: $\mathcal{S}_{\mathbf{TT}}$ * *Schedule der TT-Nachrichten aus der Migration des statischen Segments* *\
Ensure: $\mathcal{S}_{\mathbf{TT}}$ * *Schedule des Kommunikationszyklus (Es gibt nur noch TT-Nachrichten)* *\
    Erzeuge einen leeren Schedule $\mathcal{S}_{TT}(m_i)$ für jede Nachricht $m_i \in \mathbf{TG_{TT}}$ und füge ihn zu der Menge $\mathcal{S}_{\mathbf{TT}}$ hinzu
    while $\mathcal{ML} \neq \emptyset$ **do**
        sei m_i die erste Nachricht aus $\mathcal{ML}$
        if $CCRep(m_i) > 0$ **then**
            $N_{rep}(m_i) \leftarrow \left\lceil \frac{(\mathcal{S}_{FR}.N_{CC}+1) - bCC(m_i)}{CCRep(m_i)} \right\rceil$
            $N_{rep}(m_i) \leftarrow N_{rep}(m_i) - 1$
            $t_{pm}(m_i) \leftarrow 0$
            if $N_{rep}(m_i) > 0$ **then**
                $t_{pm}(m_i) \leftarrow \left\lfloor \frac{T_{CC}}{N_{rep}(m_i)+1} \right\rfloor$
            end if
        end if
        Ermittele die Konflikt-Menge $\mathbf{conf}(m_i)$ der Nachricht m_i
        $\Delta_{Slot} \leftarrow \zeta_{SW}(m_i) + (2 \times T_{cp})$ * *Slot-Länge berechnen* *\
        if Existiert ein Zeitslot $[t_a, t_b = t_a + \Delta_{Slot})] : (t_b \leq t_{dm}(m_i) + t_{pm}(m_i) + \Delta_{Slot}) \wedge (\forall m_k \in \mathbf{conf}(m_i), (t_b \leq t_{sm}(m_k)) \vee (t_a \geq t_{fm}(m_k)))$ **then**
            $t_{sm}(m_i) \leftarrow t_a$
            $t_{fm}(m_i) \leftarrow t_b$
        else
            return $\emptyset$ * *Schedule nicht ausführbar* *\
        end if
        * *Die Intervalle für die Wiederholungen sollen auch frei sein.* *\
        if $N_{rep}(m_i) > 0$ **then**
            $k \leftarrow 1$
            while $k \leq N_{rep}(m_i)$ **do**
                $T_\theta = t_{pm}(m_i) \times k$
                if Existiert kein Zeitslot $[t_a, t_b = t_a + \Delta_{Slot})] : (t_{sm}(m_i) + T_\theta \leq t_a) \wedge (t_b \leq t_{fm}(m_i) + T_\theta)$ **then**
                    return $\emptyset$ * *Schedule nicht ausführbar* *\
                end if
                $k \leftarrow k + 1$
            end while
        end if
        $\mathcal{S}_{\mathbf{TT}} \leftarrow \mathcal{S}_{\mathbf{TT}} \cup \{\mathbf{conf}(m_i), t_{sm}(m_i), t_{fm}(m_i), t_{pm}(m_i), N_{rep}(m_i)\}$
        $\mathcal{ML} \leftarrow \mathcal{ML} - m_i$ * *Lösche* m_i *von* $\mathcal{ML}$ *\
    end while
    return $\mathcal{S}_{\mathbf{TT}}$
end procedure

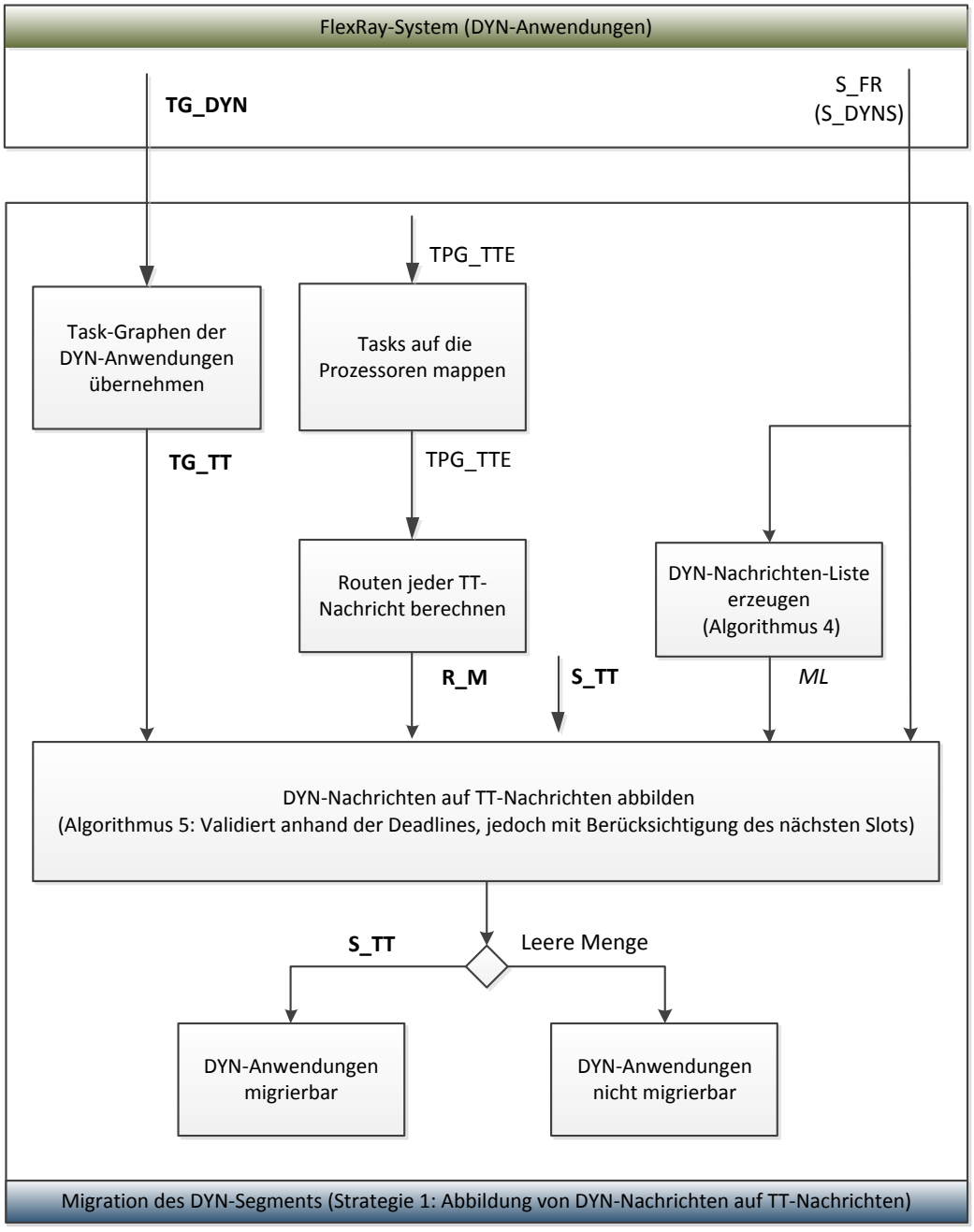


Abbildung 7.12: Prozess für die Migration des dynamischen Segments mit der ersten Strategie

noch einhalten kann, wenn diese in dem nachfolgenden Slot Übertragen wird. Grund hierfür ist, dass eine event-triggered Nachricht nicht mit dem time-triggered Schedule synchronisiert wird, und deswegen den reservierten Slot verpassen kann, und erst im nachfolgendem Slot gesendet wird. Wenn ein TT-Slot für jede DYN-Nachricht gefunden werden kann, dann ist die Migration des dynamischen Segments möglich.

Der Prozess für die Migration des dynamischen Segments mit der ersten Strategie wird in der Abbildung 7.12 zusammengefasst.

Mit dieser Strategie ist es jedoch nicht immer möglich, eine Lösung, also einen freien Slot, für jede DYN-Nachricht zu finden. Denn je kleiner die Länge des Minislots und je mehr davon konfiguriert wurden, desto schwieriger wird es, einen TT-Slot für jede DYN-Nachricht zu finden. Außerdem entfällt die Priorisierung der DYN-Nachrichten durch die time-triggered Konfiguration. Eine Möglichkeit, diese Priorisierung beizubehalten ist es, die DYN-Nachrichten mit priorisierten RC-Nachrichten zu ersetzten. Hierfür wird eine zweite Strategie im folgenden Anschnitt entwickelt.

7.3.6 Migrationsstrategie 2: Abbildung von DYN-Nachrichten auf RC-Nachrichten

Bei der zweiten Strategie für die Migration des dynamischen Segments geht es darum, jede DYN-Nachricht auf eine RC-Nachricht abzubilden. Hierfür wird der Algorithmus 6 verwendet. Dieser nimmt als Eingabe die Menge der zu planenden Task-Graphen $\mathbf{TG_{RC}} \subset \mathbf{TG}$, die Routen R_M aller Nachrichten, die Nachrichten-liste $\mathcal{ML}$ (nach Algorithmus 4) und der FlexRay-Schedule $\mathcal{S}_{FR}$, welcher auch den Schedule des zu migrierenden DYN-Segments $\mathcal{S}_{DYNS}$ beinhaltet. Basierend auf diesen Eingaben wird, wenn möglich, ein Schedule der DYN-Nachrichten nach der RC-Technik festgelegt. Dafür wird jeder DYN-Nachricht eine $CtId$ zugewiesen. Dabei wird die Priorität der ursprünglichen Nachrichten beibehalten. Das heißt je höher der Minislot der Nachricht, desto höher ihre $CtId$. Wenn man eine Konfiguration aus mehreren Zyklen hat, dann sollte die Vergabe der $CtIds$ zunächst für alle DYN-Nachrichten mit dem gleichen Minislots angefangen werden. Also alle Nachrichten, die z. B. in den Minislots mit der Nummer 1 übertragen werden, bekommen eine höhere Priorität als alle Nachrichten, die in den Minislots mit der Nummer 2 übertragen werden (entspricht dem Algorithmus 4). Nachdem die $CtId$ einer Nachricht festgelegt wurde, wird ihre BAG berechnet. Hierbei wird die Anzahl der Wiederholungen der Nachricht eingehalten. Das heißt, wenn eine DYN-Nachricht in der FlexRay Kommunikations-Matrix „n-mal“ übertragen wurde, dann wird die BAG dieser Nachricht so festgelegt, dass diese auch „n-mal“ in dem TTEthernet-Kommunikationszyklus übertragen werden kann. Damit wird versucht,

das Verhalten der ursprünglichen Anwendung annähend beizubehalten. Abschließend wird die Worst-Case-Response-Time (WCRT) jeder RC-Nachricht berechnet und überprüft, ob diese kleiner oder gleich der ursprünglichen DYN-Nachricht ist. Wenn jede RC-Nachricht aus $\mathbf{TG_{RC}}$ eine kleinere oder die gleiche WCRT als die ursprüngliche DYN-Nachricht hat, dann sind die Anwendungen migrierbar.

Algorithm 6 Abbildung von DYN-Nachrichten auf RC-Nachrichten

1: **procedure** DYNMsgZuRCMsg
Require: $\mathbf{TG_{RC}} \subset \mathbf{TG}$
Require: $\mathbf{R_M}$
Require: $\mathcal{ML}$
Require: $\mathcal{S}_{FR}$
Ensure: $\mathcal{S}_{\mathbf{RC}}$
2: $CtIdCnt \leftarrow 1024$
3: **while** $\mathcal{ML} \neq \emptyset$ **do**
4: sei m_i die erste Nachricht aus $\mathcal{ML}$
5: $CtId \leftarrow CtIdCnt++$
6: $D_{max} \leftarrow d(m_i)$
7: $N_{Rep} \leftarrow \left\lceil \frac{(\mathcal{S}_{FR}.N_{CC}+1) - bCC(m_i)}{CCRep(m_i)} \right\rceil$
8: $T_{BAG} \leftarrow \left\lfloor \frac{T_{CC}}{N_{Rep}} \right\rfloor$
9: $\mathcal{S}_{RC}(m_i) = (CtId, T_{BAG}, D_{max})$
10: $\mathcal{S}_{\mathbf{RC}} \leftarrow \mathcal{S}_{\mathbf{RC}} \cup \{\mathcal{S}_{RC}(m_i)\}$
11: $\mathcal{ML} \leftarrow \mathcal{ML} - m_i$ * *Lösche m_i von $\mathcal{ML}$* *\
12: **end while**
13: Berechne die Worst-Case-Antwortzeit $\Re_{DYN}(m_i)$ jeder DYN-Nachricht * *Siehe Abschnitt 7.3.3* *\
14: Berechne die Worst-Case-Antwortzeit $\Re_{RC}(m_i)$ jeder RC-Nachricht * *Siehe Abschnitt 7.3.4* *\
15: **for all** $m_i \in \mathbf{TG_{RC}}$ **do**
16: **if** $\Re_{RC}(m_i) > \Re_{DYN}(m_i)$ **then**
17: **return** $\emptyset$ * *Schedule nicht ausführbar* *\
18: **end if**
19: **end for**
20: **return** $\mathcal{S}_{\mathbf{RC}}$
21: **end procedure**

Der Prozess für die Migration des dynamischen Segments mit der zweiten Strategie wird in der Abbildung 7.13 zusammengefasst.

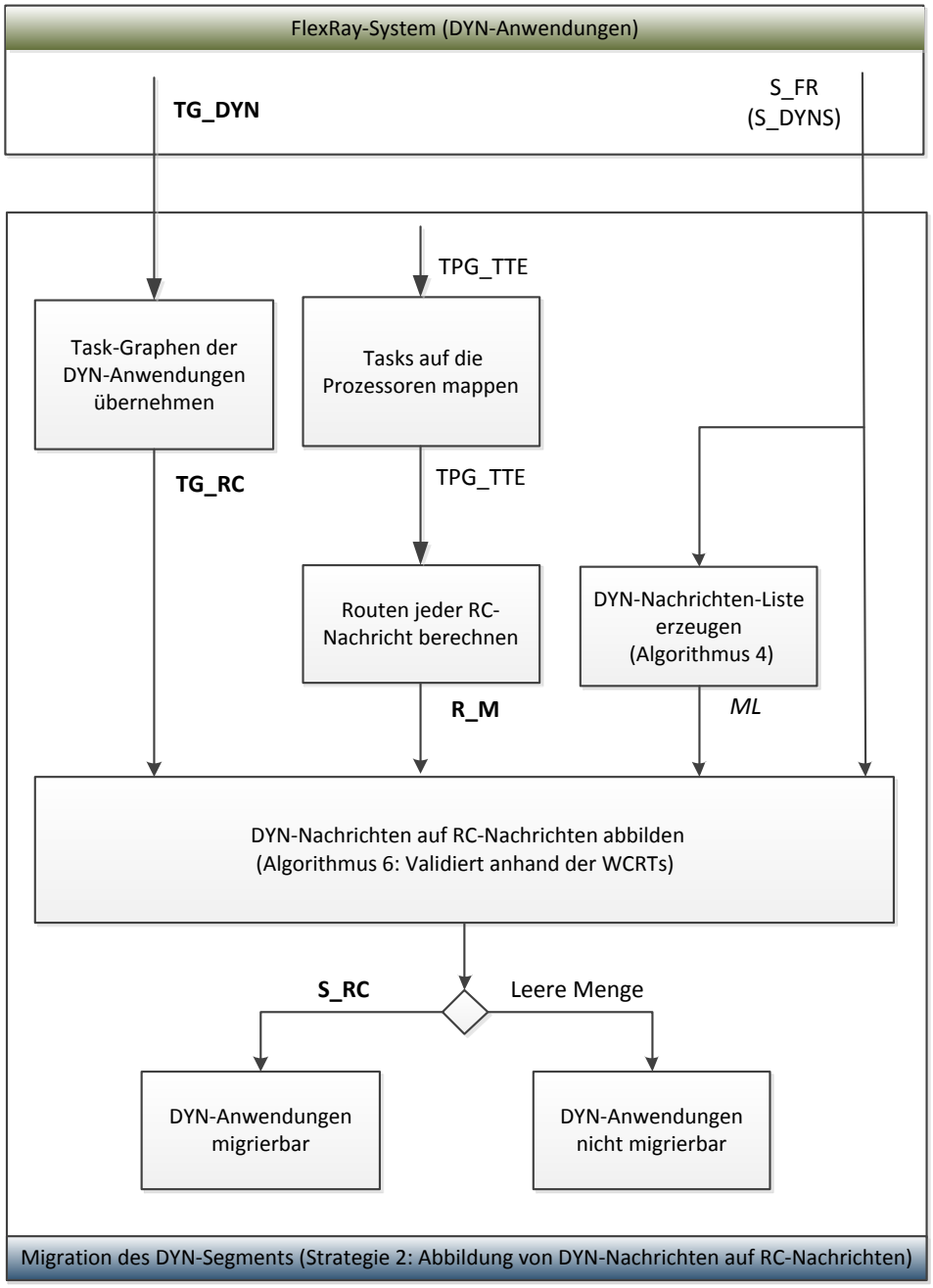


Abbildung 7.13: Prozess für die Migration des dynamischen Segments mit der zweiten Strategie

7.3.7 Diskussion der beiden Strategien

Mit der ersten Strategie werden event-triggered Anwendungen zu time-triggered Anwendungen migriert. Dadurch werden die Anwendungen deterministisch, da ein Zeitslot für jede Nachricht reserviert wird. Dadurch, dass die Zeitslots vorab festgelegt werden müssen, kann es passieren, dass nicht ein Zeitslot für jede event-triggered Nachricht gefunden wird. Außerdem fallen die Prioritäten der Nachrichten aus, wenn zu einem time-triggered System migriert wird. Daher kann es passieren, dass eine vorher höher priorisierte Nachricht nach der Migration schlechtere Response-times als eine nieder priorisierte Nachricht hat, da diese jetzt immer auf ihre Zeitslots warten muss. Weiterhin fällt die Flexibilität der Vergrößerung der Nachrichten-Länge zur Laufzeit von der Applikation aus, da vorab bekannt sein muss, wie groß eine Nachricht ist, damit der Zeitslot festgelegt werden kann. Aus diesen Gründen soll die zweite Strategie in Abwägung herangezogen werden. Hierbei werden die event-triggered Anwendungen nach event-triggered Anwendungen migriert. Mit dieser Strategie können die Eigenschaften der Anwendungen (Prioritäten der Nachrichten, Änderung der Nachrichten-Größe zur Laufzeit, etc.) beibehalten werden. Die Validierung dieser Strategie ist jedoch schwerer aufgrund des Nicht-Determinismus des event-triggered Mechanismus. Man wird hierbei gezwungen, auf die heuristische Worst-Case-Response-Time-Analyse zurückzugreifen. Wenn für jede Nachricht ein Zeitslot zu Verfügung steht, die Priorisierung der Nachricht keine große Rolle spielt und eine gewisse Flexibilität nicht gefordert ist, dann ist sicherlich die erste Strategie die beste Wahl, ansonsten sollte man eher für die zweite Strategie optieren.

7.3.8 Migration von zwei ET-Anwendungen

In diesem Abschnitt werden zwei in Rahmen dieser Arbeit kreierte event triggered Anwendungen (ET-Anwendungen) nach der ersten und zweiten Strategie für die Migration des dynamischen Segments migriert. Diese Anwendungen sind: ein Driver-Assistant und ein Diagnose-Tool. Der Driver-Assistant aktiviert bestimmte Aktuatoren (z. B. Bremse) anhand erkannter Bilder-Muster (z. B. Hindernisse, Personen) (siehe Task-Graph in der Abbildung 7.14(a)). Das Diagnose-Tool fragt Diagnose-Daten (z. B. Events, Fehler-Protokoll, etc) bestimmter ECUs zur Laufzeit ab und protokolliert diese (siehe Task-Graph in der Abbildung 7.14(b)).

Die Festlegung der Längen der einzelnen FlexRay-Kommunikations-Segmente wird in der Tabelle 7.8 und die Parameter des dynamischen Segments in der Tabelle 7.9 gezeigt. Bei der Festlegung der Länge eines Minislots wurde der in der Literatur meist benutzte Wert ($T_{MSlot} = 6\mu s$) verwendet (vgl. Schmidt u. a., 2010; Schmidt und Schmidt, 2009).

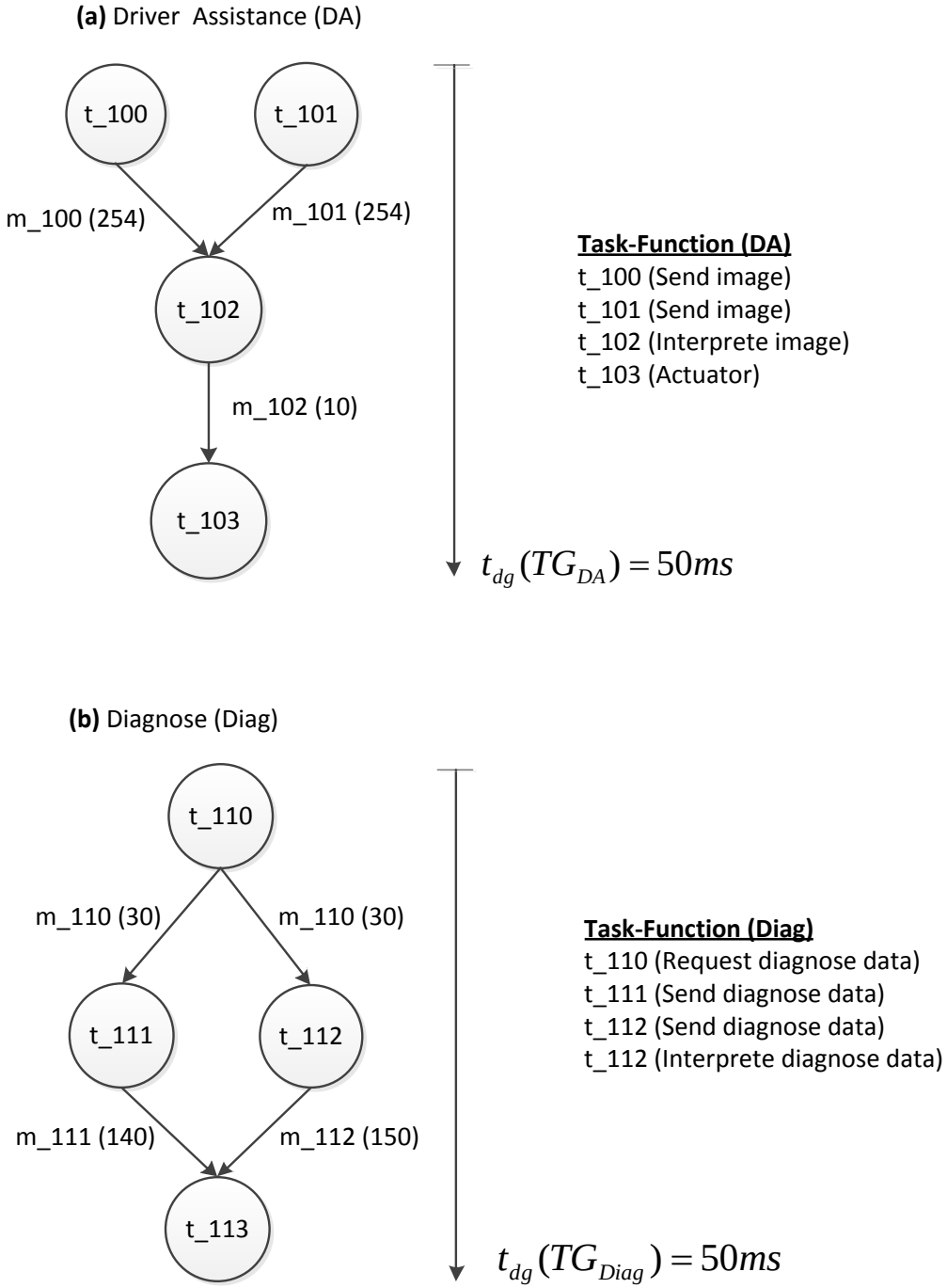


Abbildung 7.14: Task-Graphen der ET-Anwendungen — (a) Fahrerassistenz und (b) Diagnose

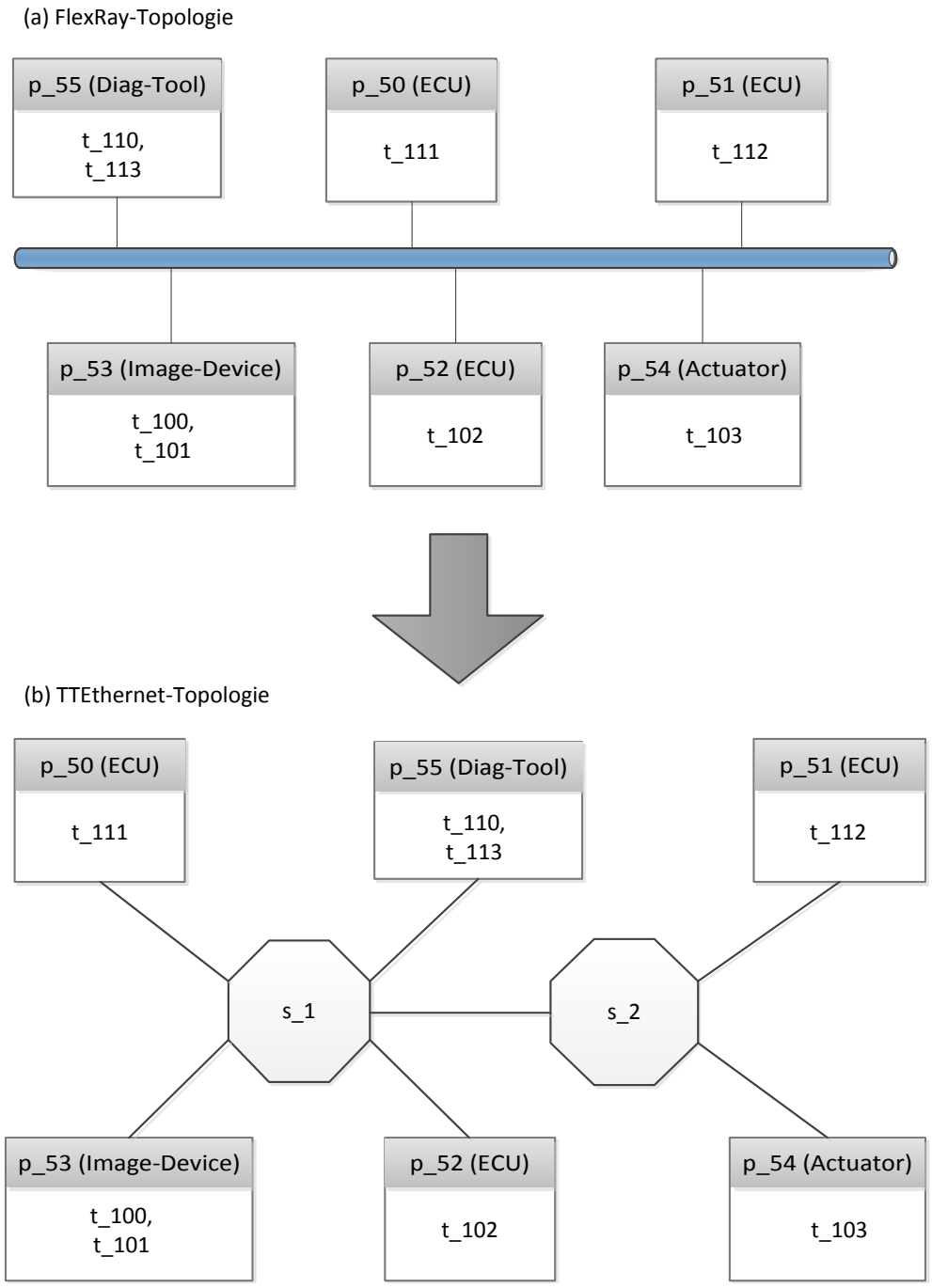


Abbildung 7.15: Topologien der ET-Anwendungen inklusive Mapping der Tasks auf die Prozessoren — (a) FlexRay-Topologie und (b) ausgewählte TTEthernet-Topologie

Tabelle 7.8: Migration-ET-Anwendungen: Länge der einzelnen Segmente und des Kommunikationszyklus. Die Länge des Kommunikationszyklus T_{CC} wurde nach der Formel 7.2 berechnet.

T_{STS}	T_{DYNS}	T_{SWin}	T_{NIT}	T_{CC}
$1000\mu s$	$600\mu s$	$100\mu s$	$300\mu s$	$2000\mu s$

Tabelle 7.9: Migration-ET-Anwendungen: Konfiguration der Länge eines Minislots und der Anzahl der Minislots.

T_{DYNS}	T_{MSlot}	N_{MSlot}
$600\mu s$	$6\mu s$	100

Während Tabelle 7.10 die zu übertragende Datenmenge aller DYN-Nachrichten und deren Übertragungszeiten ohne Verzögerung zeigt, wird der Schedule dieser Nachrichten in der Tabelle 7.11 dargestellt. Man kann aus der Tabelle 7.11 entnehmen, dass die Nachrichten m_{100}, m_{101} und m_{102} der Driver-Assistant-Anwendung eine höhere Priorität als die der Diagnose-Anwendung haben. Aufgrund der niedrigen Priorität der Diagnose-Anwendung kann es passieren, dass deren Nachrichten über mehrere Zyklen verzögert werden und somit die Deadline $t_{dg}(TG_{Diag}) = 50ms$ verpassen. Nehmen wir zum Beispiel an, dass das vom Driver-Assistant zu übertragende Bild $30000Bytes$ groß ist. Die Übertragung eines Bildes, welches von den Tasks t_{100} und t_{101} übertragen wird, würde ca. $(\frac{30000Byte}{262Byte+262Byte}) \times 600\mu s = 34352\mu s = 35ms$ dauern. Somit muss die Driver-Assistant-Anwendung nur zwei Bilder nacheinander senden, damit die Deadline der Diagnose-Anwendung verletzt wird. Denn wenn die Driver-Assistant-Anwendung alle ihre Nachrichten innerhalb eines Kommunikationszyklus überträgt, werden alle Nachrichten der Diagnose-Anwendung um einen Kommunikationszy-

Tabelle 7.10: Migration-ET-Anwendungen: Zu übertragende Datenmenge und Übertragungszeiten der DYN-Nachrichten. Für die Berechnungen der $\zeta_B(m_i)$ wurde eine Bandbreite von $10\frac{Mbit}{s}$ angewendet.

m_i	$d(m_i)[Byte]$	$\zeta_B(m_i)[\mu s]$
m_{100}	262	262
m_{101}	262	262
m_{102}	18	18
m_{110}	38	38
m_{111}	148	148
m_{112}	158	158

Tabelle 7.11: Migration-ET-Anwendungen: Schedule der DYN-Nachrichten.

m_i	$mSlot(m_i)$	$bCC(m_i)$	$CCRep(m_i)$	$ltMSlot(m_i)$
m_{100}	1	1	1	56
m_{101}	2	1	1	56
m_{102}	3	1	1	97
m_{110}	4	1	1	93
m_{111}	5	1	1	75
m_{112}	6	1	1	73

klus verzögert. Dies führt dazu, dass die Deadline dieser Anwendung nicht immer eingehalten wird. Dieses Problem kann durch die Migration gelöst werden.

Migration der ET-Anwendungen mit der ersten Strategie

Bei der Migration mit der ersten Strategie werden DYN-Nachrichten auf TT-Nachrichten anhand des Algorithmus 5 abgebildet. Die Voraussetzungen für die Migration werden durch die Funktionsmodelle der Anwendungen (Abbildung 7.14), das FlexRay-Topologie-Modell (Abbildung 7.15(a)) und den Schedule der DYN-Nachrichten (Tabelle 7.11) gegeben. Der erste Schritt bei der Migration ist es, eine TTEthenert-Topologie der Anwendungen festzulegen. Hier wurde der FlexRay-Bus mit zwei kaskadierten TTEthernet-Switches ersetzt (Abbildung 7.15(b)). Dies ist nur eine Möglichkeit von vielen.

Tabelle 7.12: Migration-ET-Anwendungen: Zu übertragende Datenmenge und Übertragungszeiten der Nachrichten in TTEthernet. Für die Berechnungen der $\zeta_{SW}(m_i)$ wurde eine Bandbreite von $100\frac{Mbit}{s}$ angewendet.

m_i	$d(m_i)[Byte]$	$\mathbf{R_{m_i}}$	$\zeta_{SW}(m_i)[\mu s]$
m_{100}	280	$\{l_{[p_{53},s_1]}, l_{[s_1,p_{52}]}\}$	48
m_{101}	280	$\{l_{[p_{53},s_1]}, l_{[s_1,p_{52}]}\}$	48
m_{102}	72	$\{l_{[p_{52},s_1]}, l_{[s_1,s_2]}, l_{[s_2,p_{54}]}\}$	23
m_{110}	72	$\{l_{[p_{55},s_1]}, l_{[s_1,s_2]}, l_{[s_2,p_{51}]}\}$	23
m_{111}	166	$\{l_{[p_{50},s_1]}, l_{[s_1,p_{55},\}]}$	30
m_{112}	176	$\{l_{[p_{51},s_2]}, l_{[s_2,s_1]}, l_{[s_1,p_{55}]}\}$	48

Steht die Topologie fest, dann können die Routen $\mathbf{R_m}$ jeder Nachricht und die Übertragungszeit dieser ohne Verzögerung $\zeta_{SW}(m)$ ermittelt werden (siehe Tabelle 7.12). Somit sind die Voraussetzungen gegeben, um den Schedule der Nachrichten im TTEthernet zu berechnen. Dies wird in der Tabelle 7.13 dargestellt. Man kann daraus erkennen, dass ein TT-Slot für jede DYN-Nachricht gefunden werden konnte. Somit sind diese Anwendungen nach der ersten

Tabelle 7.13: Migration-ET-Anwendungen: TT-Schedule der Nachrichten.

m_i	$\mathbf{conf}(m_i)$	$t_{sm}(m_i)[\mu s]$	$t_{fm}(m_i)[\mu s]$	$N_{rep}(m_i)$
m_{100}	$\{m_{101}\}$	0	48	1
m_{101}	$\{m_{100}\}$	48	96	1
m_{102}	$\{m_{110}\}$	0	23	1
m_{110}	$\{m_{102}\}$	23	46	1
m_{111}	$\{m_{112}\}$	0	30	1
m_{112}	$\{m_{111}\}$	30	78	1

Strategie migrierbar. Außerdem kann man aus der Tabelle erkennen, dass die Übertragungen aller Nachrichten in TTEthernet zum Zeitpunkt $96\mu s$ abgeschlossen werden. Das macht nur eine 16%-ige Nutzung des Kommunikationszyklus aus und lässt somit viel Bandbreite für zukünftige Nachrichten zur Verfügung. Man kann daraus schließen, dass die Chance, dass die Applikationen migriert werden können, umso höher ist, je mehr Nachrichten parallel übertragen werden.

Migration der ET-Anwendungen mit der zweiten Strategie

Bei der Migration mit der zweiten Strategie werden DYN-Nachrichten auf RC-Nachrichten abgebildet. Hierfür wird der Algorithmus 6 verwendet. Das Funktions-Modell, TTEthernet-Topologie-Modell und die Routen der Nachrichten aus der Migration mit der ersten Strategie sind auch hier gültig. Das Ergebnis der Abbildung von DYN-Nachrichten auf RC-Nachrichten wird in Tabelle 7.14 gezeigt. Eine Aussage darüber, ob die Anwendungen mit dieser Strategie migriert werden können, kann nur getroffen werden, wenn die WCRT der Nachrichten in FlexRay und TTEthernet bekannt sind. Diese werden in der Tabelle 7.15 dargestellt. Daraus kann man erkennen, dass die Response-Times der Nachrichten im TTEthernet eindeutig niedriger sind als bei FlexRay. Außerdem sind die Response-Times von niederpriorisierten Nachrichten bei TTEthernet gebunden, im Gegensatz zu FlexRay, wo diese immer weiter steigen können.

Für die Berechnung der Response-Times aus der Tabelle 7.15 wurde bei FlexRay angenommen, dass die drei Nachrichten ($m_{100}, m_{101}, m_{102}$) mit hohen Prioritäten deren Slots knapp verpasst haben und erst im nächsten Zyklus übertragen werden. Weiterhin wird die Übertragung der anderen niederpriorisierten Nachrichten ($m_{110}, m_{111}, m_{112}$) erst versucht, wenn die Übertragung der höherpriorisierten Nachrichten gestartet worden ist. Wenn man wieder annimmt, dass die Driver-Assistant-Anwendung ein $30000Byte$ großes Bild zu übertragen hat, dann können die Nachrichten der Diagnose-Anwendung über $35ms$ verzögert werden

Tabelle 7.14: Migration-ET-Anwendungen: RC-Schedule.

m_i	$CtID$	$T_{BAG}[\mu s]$	D_{max}
m_{100}	100	2000	280
m_{101}	101	2000	280
m_{102}	102	2000	72
m_{110}	110	2000	72
m_{111}	111	2000	166
m_{112}	112	2000	176

Tabelle 7.15: Migration-ET-Anwendungen: Worst-Case-Response-Times.

m_i	$\Re_{DYN}(m)[\mu s]$	$\Re_{RC}(m)[\mu s]$
m_{100}	2262	1136
m_{101}	2518	1136
m_{102}	2530	1035
m_{110}	> 35000	1035
m_{111}	> 35000	1045
m_{112}	> 35000	1062

und verpassen somit 58 Kommunikationszyklen. Im TTEthernet wird angenommen, dass die niederpriorisierten RC-Nachrichten immer durch höherpriorisierte RC-Nachrichten verzögert werden. Man soll bedenken, dass RC-Nachrichten im TTEthernet nur durch TT-Nachrichten, die die gleichen Routen wie diese haben, verzögert werden können. In diesem Beispiel wurde jedoch angenommen, dass die RC-Nachrichten durch alle TT-Nachrichten verzögert werden, da die Konfiguration der TT-Nachrichten hier nicht bekannt ist. Wenn dies jedoch der Fall ist, kann man für dieses Beispiel sogar noch bessere Response-Times für RC-Nachrichten bekommen, als jene aus der Tabelle 7.15. Denn eine RC-Nachricht, die eine disjunkte Route mit der Route einer TT-Nachricht hat, kann parallel mit dieser übertragen werden.

8 Zusammenfassung und Ausblick

Das Ziel dieser Arbeit ist es, ein analytisches Framework zur Migration von FlexRay-Systemmodellen nach TTEthernet-Systemmodellen mit dem Fokus auf dem Kommunikationsmodell zu entwickeln. In diesem Kapitel werden die für seine Realisierung geleisteten Arbeiten zusammengefasst und ein Ausblick auf zukünftige Arbeiten gegeben.

8.1 Zusammenfassung der Arbeit

Durch den kontinuierlichen Anstieg eingesetzter elektrischer und elektronischer Systeme steigen die Anforderungen an Automobil-Kommunikationssysteme immer weiter an. Von zukünftigen Automobil-Kommunikationssystemen werden zuverlässige Datenübertragung, geringere Latenzen und Jitter, Fehlertoleranz und hohe Bandbreite gefordert. Die momentan eingesetzten Techniken sind diesen Anforderungen nicht gewachsen. So werden neue Ansätze benötigt. Es gibt momentan zwei vielversprechende Techniken — FlexRay und TTEthernet. TTEthernet hat Aufgrund dessen, dass es auf Ethernet basiert, weitere Vorteile gegenüber FlexRay. Beispiele hierfür sind: mehr Bandbreite, preiswert, seit über 30 Jahren auf dem Markt, bessere Übertragungstechnik (Full-Duplex, Multicast, etc.). So ist der Wechsel von FlexRay zu TTEthernet lohnenswert. Da FlexRay und TTEthernet auf den gleichen Prinzipien beruhen, kann viel aus vorhandenen FlexRay-Projekten bzw. -Designentscheidungen in neuen TTEthernet-Systemen wiederverwendet werden. Hierfür sind Migrationsstrategien bezüglich des Designs unerlässlich. Diese wurden im Rahmen der Abschlussarbeit entwickelt.

Für die Entwicklung der Migrationsstrategien wurden zunächst die Architektur und ein Konzept in Kapitel 2 vorgestellt. Dazu gehört auch die Eingrenzung der Ziele der Arbeit, welche anhand einer E/E-Architektur festgelegt wurden. In dieser Arbeit sind nur drei der sechs Schichten der E/E-Architektur relevant, nämlich die Anforderungs-, Funktionsmodell- und Topologiemodell-Schicht. Diese sind auch die wesentlichen Schichten für die Erstellung und Validierung eines Kommunikationsmodells. Die Funktionsmodelle und das Topologiemodell wurden zusammen als Systemmodell gekennzeichnet. Das Konzept der Migration ist es, aus einem vorhandenen FlexRay-Systemmodell und FlexRay-Schedule ein TTEthernet-Systemmodell und TTEthernet-Schedule zu generieren, welcher die Systemspezifikation und

die zeitlichen Anforderungen der im System gesendeten Nachrichten einhält. Beim letzteren wurde besonders Acht auf die Deadlines gegeben, da diese für ein Echtzeitsystem wichtig sind. Bei der Entwicklung der Migrationsstrategien wurden sowohl das statische als auch das dynamische Segment von FlexRay in Betracht gezogen.

Um die entworfenen Konzepte der Migration zu realisieren, wurden zunächst grundlegende Arbeiten durchgeführt. Dazu gehört die Untersuchung von FlexRay (Kapitel 3) und TTEthernet (Kapitel 4), die Festlegung eines Modellierungs- und Beschreibungs-Frameworks für verteilte Echtzeitsysteme mit Task-Graphen (Kapitel 5) und die Untersuchung von verwandten und vergleichbaren Arbeiten (Kapitel 6). Die Beherrschung von FlexRay und TTEthernet war für eine erfolgreiche Entwicklung der Migrationsstrategien unabdingbar. Hierfür wurden die FlexRay-Spezifikationen und die zur Standardisierung eingereichte Version der TTEthernet-Spezifikation durchgearbeitet. Bei der Untersuchung von verwandten und vergleichbaren Arbeiten wurden zwei Ansätze zur Migration von Automobil-Kommunikationsprotokollen anhand von zwei Realisierungen beschrieben. Diese Ansätze sind: Migration mit Hilfe eines Gateways und vollständige Migration der Anwendungen. Es wurde im Rahmen dieser Arbeit auf den letzteren Ansatz mit dem Fokus auf das Design zurückgegriffen. Denn der Gateway-Ansatz macht das System teuer, komplexer und verursacht zusätzliche Verzögerungszeiten der Nachrichten. Es wurde zudem bei der Untersuchung von verwandten Arbeiten keine veröffentlichte Arbeit bezüglich der Migration von FlexRay nach TTEthernet gefunden.

Die Entwicklung der Migrationsstrategien fand im Kapitel 7 satt. Hierfür wurden zunächst FlexRay und TTEthernet verglichen. Ziel hierbei war es, die Unterschiede der beiden Protokolle herauszukristallisieren. Die festgestellten wesentlichen Unterschiede sind: unterschiedliche Anzahl und unterschiedlicher Aufbau der Kommunikationszyklen, Länge der time-triggered Kommunikationsslots, verschiedene Ansätze für die event-triggered Kommunikation, maximale und minimale Payload sowie Protokoll-Overhead, Bandbreite und Topologien. Diese Unterschiede haben die Migration stark geprägt. Die Entwicklung der Migrationsstrategien wurde in zwei Teile aufgeteilt — Migration des statischen und des dynamischen Segments.

Bei der Migration des statischen Segments wurde zunächst gezeigt, wie ein vorhandenes FlexRay-Systemmodell in ein TTEthernet-Systemmodell transformiert werden kann. Dabei konnte man feststellen, dass die Funktionsmodelle unverändert in TTEthernet übernommen werden können, weil sich die System-Spezifikation und die zeitlichen Anforderungen nicht ändern. Das Topologiemodell konnte jedoch nicht beibehalten werden, da im TTEthernet nur Switch-Topologien zum Einsatz kommen. Es wurden Vorschläge gemacht, wie FlexRay-Topologien in TTEthernet-Topologien abgebildet werden können. Letztendlich ist es möglich, jede FlexRay-Topologie auf eine beliebige TTEthernet-Topologie abzubilden, sofern die Sys-

temspezifikation und die zeitlichen Anforderungen der Nachrichten eingehalten werden. Um das letztere zu überprüfen müsste Aufgrund der Unterschiede zwischen den Kommunikationszyklen und der Topologiemodelle der beiden Protokolle ein neuer Scheduling-Algorithmus für TTEthernet-TT-Nachrichten entwickelt werden. Dieser generiert mit Hilfe der TTEthernet-Funktions- und Topologie-Modelle und des vorgegebenen FlexRay-Schedules einen Schedule für jede TTEthernet-TT-Nachricht. Um diese Schedules zu validieren, wird überprüft, ob diese die ursprünglichen festgelegten Zeitanforderungen (Deadlines) der Nachrichten einhalten. Wenn dies der Fall ist, dann sind die Anwendungen migrierbar ansonsten nicht. Die Strategie für die Migration des statischen Segments wurde anhand von Automobil verteilten Echtzeit-Anwendungen (EPS und TC) validiert. Man konnte dabei feststellen, dass obwohl die zu migrierenden Nachrichten eine sehr kleine Payload (max. $2Byte$) hatten, ihre Deadlines in TTEthernet eingehalten werden konnten.

Für die Migration des dynamischen Segments wurden zwei Strategien entwickelt. Bei der ersten Strategie werden DYN-Nachrichten auf TT-Nachrichten abgebildet und bei der zweiten DYN-Nachrichten auf RC-Nachrichten. Die erste Strategie entspricht der zur Migration des statischen Segments mit dem Unterschied, dass bei der Vergabe der time-triggered Zeitslots mit höher priorisierten Nachrichten angefangen wird und die Releasezeiten der Nachrichten nicht überprüft werden, da diese bei event-triggered Nachrichten nicht bekannt ist. Mit der ersten Strategie kann nicht immer ein Übertragungsslot für jede Nachricht gefunden werden, denn je kleiner die Länge des Minislots und je mehr davon konfiguriert wurden, desto schwieriger wird es, einen TT-Slot für jede DYN-Nachricht zu finden, also alle DYN-Nachrichten auf TT-Nachrichten abzubilden. Außerdem entfallen die Prioritäten der Nachrichten. Die zweite Strategie versucht, anhand eines Algorithmus eine RC-BAG für jede zu migrierende DYN-Nachricht so festzulegen, dass diese die ursprünglichen Eigenschaften der Nachricht (wie die Priorität) einhält. Aufgrund dessen, dass man bei der zweiten Strategie mit nicht deterministischen event-triggered Anwendungen zu tun hat, konnte die Validierung dieser Strategie nur durch den Vergleich der Worst-Case-Response-Times (WCRT) der Nachrichten durchgeführt werden. Eine Anwendung ist mit dieser Strategie nur dann migrierbar, wenn die WCRTs jeder DYN-Nachricht kleiner ist als der entsprechenden RC-Nachricht. Diese Strategie wurde anhand von zwei event-triggered Anwendungen (Driver-Assistant und Diagnose-Tool) validiert. Man konnte feststellen, dass die WCRTs der Nachrichten bei TTEthernet wesentlich kleiner und gebunden sind. Außerdem wurden nur 16 % der Bandbreite in TTEthernet in Anspruch genommen, so dass noch 84 % für zukünftige Anwendungen zur Verfügung standen. Grund für die geringere verwendete Bandbreite ist die Berücksichtigung der parallelen Übertragung von Nachrichten bei der Berechnung der WCRTs.

8.2 Ausblick auf zukünftige Arbeiten

Trotz der sorgfältigen Erarbeitung dieser Arbeit kann sie nicht vollkommen sein. Das Ziel war es, Strategien, mit derer Hilfe ein FlexRay-System zu einem TTEthernet-System migriert werden kann, zu entwickeln. Dabei wurde der Fokus auf eine frühere Phase der Systementwicklung gelegt, das Design des Kommunikationsmodells der verteilten Anwendungen. So ist diese Arbeit ein Proof-of-Concept und stellt außerdem eine solide Basis für weiterführende Arbeiten dar. Es ist insbesondere wichtig, dass das entwickelte Framework für die Migration von FlexRay nach TTEthernet in einer Tool-Chain implementiert wird, welche das auf Task-Graphen basierte Modellierungs-Framework und die Migrationsstrategien von der Anforderung, der Modellierung über die Simulation bis zur Generierung von Codes für die jeweiligen Zielplattforme unterstützt.

Obwohl das hier entwickelte Migrations-Framework gute Ergebnisse, sowohl für das statische als auch für das dynamische Segment, aufgewiesen hat, kann dieses noch verbessert werden. Es wurde z. B. bei der Entwicklung der TTEthernet-Scheduling-Algorithmen von statischen Topologien und Routen ausgegangen. Weiterführende Gedanken wären Algorithmen zu entwickeln, welche für eine gegebene Anzahl an Netzwerk-Knoten die beste Topologie und die besten Routen finden. Dies ist ein iterativer Prozess und ein NP-Hartes (nicht-deterministisch Polynomiale) Problem. Weiterhin wurde die Kommunikation zwischen den event-triggered und time-triggered Anwendungen nicht berücksichtigt. Dies ist zwar nicht üblich, kann jedoch in der Praxis nicht ausgeschlossen werden.

Momentan werden, wie bereits angesprochen, mehrere Bussysteme (CAN, LIN, FlexRay, MOST) eingesetzt, um die verschiedenen Anforderungen eines Automobil-Netzwerks zu erfüllen. In dieser Arbeit wurden jedoch nur Migrationsstrategien von FlexRay nach TTEthernet entwickelt. Sinnvoll wäre auch, solche Migrationsstrategien für die anderen Protokolle (LIN, CAN, MOST) zu erarbeiten. Sicherlich wird es nicht möglich sein, ein gesamtes Automobil-Netzwerk auf einem Schlag mit TTEthernet zu ersetzten. TTEthernet kann jedoch im ersten Schritt als Backbone eingesetzt werden und die Netze, die aus den anderen Protokollen bestehen, miteinander verbinden. Im zweiten und zukünftigen Schritte können dann die anderen Protokolle Schritt für Schritt durch TTEthernet ersetzt werden, um ein einheitliches Netzwerk zu bekommen.

Danksagung

Hiermit möchte ich mich bei allen bedanken, die mich bei der Erstellung dieser Arbeit unterstützt haben. Insbesondere gilt mein Dank

Herrn Prof. Dr. Franz Korf für die hervorragende Betreuung,

Herrn Till Steinbach, Herrn Florian Bartols und Herrn Kai Müller für ihre Unterstützung während des gesamten Masterstudiums,

und Herrn Christian Roß für das Korrekturlesen.

Glossar

Backbone	Backbone (deutsch: Basisnetz) bezeichnet den zentralen Kernbereich eines Telekommunikationsnetzes.
Cluster	Ist ein Kommunikationssystem aus mehreren Netzwerk-Knoten, die über mindestens einen Kommunikationskanal direkt miteinander verbunden sind.
Constraint	Constraint (deutsch: Zwangsbedingung) bezeichnet Bedingungen, welche eingehalten werden müssen, damit ein Wert in ein System übernommen wird.
Determinismus	In einem deterministischen System können die Ausgaben mit hoher Sicherheit vorhergesagt werden.
ET/TT-Kommunikation	Bei einer event-triggered (zeitgesteuerten) Kommunikation werden die Nachrichten beim Eintreffen von Ereignissen gesendet, es ist also nicht immer bekannt, wann die Nachrichten gesendet werden. Bei einer time-triggered (zeitgesteuerten) Kommunikation hingegen werden die Nachrichten nach einem festgelegten Zeitplan (Schedule) gesendet.

Fehlertoleranz	Fehlertoleranz ist die Fähigkeit eines Systems, seine Funktionsweise aufrechtzuerhalten, wenn unvorhergesehene Eingaben oder Fehler in der Hard- oder Software auftreten.
Jitter	(deutsch: Schwankung) bezeichnet in der Nachrichtentechnik die Varianz der Übertragungszeit von Nachrichten.
Kommunikationsmodell	Mit Kommunikationsmodell wird das Sender-Empfänger-Modell, welches die Kommunikation als Übertragung einer Nachricht von einem Sender zu einem oder mehreren Empfängern über einen Kommunikationskanal definiert, gemeint.
Latenz	Die Latenz (englisch: latency) ist die Zeit, die eine Nachricht zum Durchqueren eines Netzwerkes vom Sender zum Empfänger benötigt.
Proof-of-Concept	Proof-of-Concept (deutsch: Machbarkeitsnachweis) bezeichnet einen Schritt in der Entwicklung, bei dem die prinzipielle Durchführbarkeit bewiesen wird.
Skalierbarkeit	Die Skalierbarkeit gibt an, wie gut ein Hard- oder Software-System sich wachsenden Anforderungen anpassen kann. Zum Beispiel ist ein skalierbares Netzwerk, ein Netzwerk, welches mit wenigen Teilnehmern starten kann, jedoch auf tausende von Teilnehmern erweitert werden kann.

Toolchain	Als Toolchain (deutsch: Werkzeugkette) wird in der Softwareentwicklung eine systematische Sammlung von Werkzeug-Programmen bezeichnet, welche zur Erzeugung eines Produktes (meist eines anderen Programmes oder eines Systems von Programmen) Verwendung findet. Die Bezeichnung erklärt sich damit, dass die Werkzeug-Programme in der Regel in Form einer Kette nacheinander eingesetzt werden.
Topologie	Mit Topologie wird in Computernetzen die Struktur der Verbindungen mehrerer Teilnehmer untereinander bezeichnet.
WCRT	Die Worst-Case-Response-Time (WCRT) einer Nachricht ist die Zeit, welche vergeht, vom Zeitpunkt, an dem die Nachricht beim Sender zur Übertragung bereitsteht (Release-Zeit), bis diese bei allen Empfängern angekommen ist (Endzeit).

Akronyme

ACC	Adaptive Cruise Control
AFDX	Avionics Full DupleX Switched Ethernet
ASR	Antriebsschlupfregelung
BAG	Bandwidth Allocation Gap
BE	Best-Effort
CAN	Controller Area Network
CC	Communication Cycle
CRC	Cyclic Redundancy Check
CT	Critical Traffic
CT-ID	Critical Traffic ID
CTDMA	Coordinated Time Division Multiple Access
DRT	Data Ready Time
DTDMA	Dynamic Time Division Multiple Access
DYN	Dynamisch
E/E	Elektrische/Elektronische
ECU	Electronic Control Unit
EPS	Electric Power Steering
ES	End System
ESP	Elektronisches Stabilitätsprogramm
ET	Event-Triggered

IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
LIN	Local Interconnect Network
MAC	Media Access Control
MOST	Media Oriented Systems Transport
NIT	Network Idle Time
NMV	Network Management Vector
RC	Rate-Constraint
SAE	Society of Automotive Engineers
ST	Statisch
SW	Switch
SWin	Symbol Window
TC	Traction Control
TDMA	Time Division Multiple Access
TT	Time-Triggered
TTCAN	Time-Triggered Controller Area Network
TTE	Time-Triggered Ethernet
TTEthernet	Time-Triggered Ethernet
UDP	User Datagram Protocol
VDM	Vehicle Dynamics Management
WCRT	Worst Case Response Time

List of Symbols

B	Bus
$CCRep(m)$	Anzahl der Kommunikationszyklen, nach denen die Übertragung der Nachricht m wiederholt wird
$CtId$	Critical-Traffic-ID einer Nachricht
DG	Abhängigkeits-Graph (engl. dependency graph)
D_{max}	maximale Datenmenge, die mit einer Nachricht übertragen werden kann
G	Graph
N_{CC}	Anzahl der Kommunikationszyklen
N_{MSlot}	Anzahl der Minislots eines dynamischen Segments
N_{STSlot}	Anzahl der statischen Slots eines statischen Segments
$N_{rep}(m)$	Anzahl der wiederholten Übertragung der Nachricht m innerhalb eines Kommunikationszyklus
O_p	Protokoll-Overhead
R	Route
TG	Task-Graph
TPG	Topologie eines Systems Γ
T_{BAG}	minimaler Zeit-Abstand, mit dem eine Nachricht gesendet werden soll
T_{CC}	Dauer eines Kommunikationszyklus
T_{DYNS}	Dauer eines dynamischen Segments
T_{MSlot}	Dauer eines Minislots
T_{NIT}	Dauer der Network Idle Time
T_{STSlot}	Dauer eines statischen Slots
T_{STS}	Dauer eines statischen Segments
T_{SWin}	Dauer eines Symbol-Windows

Γ	verteiltes System
$\Re_{DYN}(m)$	Worst-Case-Response-Time der dynamischen Nachricht m
$\Re_{RC}(m)$	Worst-Case-Response-Time der rate-constraint Nachricht m
$\mathbf{B}$	endliche Menge von Bussen
$\mathbf{E}$	endliche Menge von Kanten (engl. edges)
$\mathbf{L}$	Menge der physikalischen Kommunikations-Links
$\mathbf{M}$	endliche Menge der Nachrichten einer Anwendung $\mathcal{A}$
$\mathbf{P}$	endliche Menge von Prozessoren
$\mathbf{S}$	endliche Menge von Switches
$\mathbf{TG}$	Menge der Task-Graphen eines Systems Γ
$\mathbf{T}$	endliche Menge von Tasks einer Anwendung $\mathcal{A}$
$\mathbf{V}$	endliche Menge von Knoten (engl. vertices)
$\mathbf{conf}(m)$	Menge der Nachrichten, deren Routen sich mit der von m überschneiden
$\mathbf{pred}(v)$	Menge aller Vorgänger des Knoten v
$\mathbf{sink}(G)$	Menge der Senke-Knoten des Graphen G
$\mathbf{source}(G)$	Menge der Quell-Knoten des Graphen G
$\mathbf{succ}(v)$	Menge aller Nachfolger des Knoten v
$\mathcal{A}$	Anwendung
$\mathcal{ML}$	Nachrichten-Liste
$\mathcal{S}$	Schedule
$\mathcal{S}_{DYNS}$	Schedule eines dynamischen Segments
$\mathcal{S}_{DYN}(m)$	Schedule der dynamischen Nachricht m
$\mathcal{S}_{FR}$	Schedule eines FlexRay-Systems
$\mathcal{S}_{RC}(m)$	Schedule der rate-constraint Nachricht m
$\mathcal{S}_{STS}$	Schedule eines statischen Segments
$\mathcal{S}_{ST}(m)$	Schedule der statischen Nachricht m
$\mathcal{S}_{TTE}$	Schedule eines TTEthernet-Systems
$\mathcal{S}_{TT}(m)$	Schedule der TTEthernet time-triggered Nachricht m
$\mathcal{S}_{\mathbf{DYN}}$	Menge der Schedules aller dynamischen Nachrichten

$\mathcal{S}_{\mathbf{RC}}$	Menge der Schedules aller rate-constraint Nachrichten
$\mathcal{S}_{\mathbf{ST}}$	Menge der Schedules aller statischen Nachrichten
$\mathcal{S}_{\mathbf{TT}}$	Menge der Schedules aller TTEthernet time-triggered Nachrichten
τ	Task
$\zeta_B(m_i)$	maximale Übertragungszeit der Nachricht m_i vom Sender-Task $s_m(m_i)$ zu allen Empfänger-Tasks $r_m(m_i)$ bei einem Bus-System
$\zeta_{SW}(m_i)$	maximale Übertragungszeit der Nachricht m_i vom Sender-Task $s_m(m_i)$ zu allen Empfänger-Tasks $r_m(m_i)$ bei einem Switch-System
$b(l_i)$	relative Datenrate des Links l_i
$bCC(m)$	erster Kommunikationszyklus, in dem die Nachricht m gesendet wird
$bSlot(m)$	erster Slot innerhalb eines statischen Segments, in dem die Nachricht m gesendet wird
$c(m_i)$	Kommunikations-Kosten der Nachricht m_i
$e_m(m_i)$	Menge der Kanten, über die die Nachricht m_i gesendet wird
$e_{[u,v]}$	Kante zwischen Knoten u und Knoten v
l	physikalischer Link
$ltMSlot(m)$	letzter Minislot, in dem die Übertragung der dynamischen Nachricht m noch starten kann
m	Nachricht
$mSlot(m)$	Nummer des Minislots der Nachricht m
p	Prozessor
$proc(\tau)$	Prozessor, auf dem der Task τ ausgeführt wird
$r_m(m_i)$	Menge der Tasks, an die die Nachricht m_i gesendet wird
s	Switch
$s_m(m)$	Task, welcher die Nachricht m sendet
$slotRep(m)$	Anzahl der Slots, nachdem die Übertragung der Nachricht m innerhalb eines Kommunikationszyklus wiederholt wird

$t_W(\tau_i)$	Berechnungskosten des Tasks τ_i
$t_b(l_i)$	Bitzeit des physikalischen Links l_i
$t_{DS}(s_i)$	Zeit, welche der Switch s_i für die Bearbeitung einer Nachricht vom Ein- zum Ausgangsport benötigt
$t_{d\tau}(\tau_i)$	Zeitpunkt, an dem die Ausführung des Tasks τ_i abgeschlossen sein muss
$t_{dg}(TG_i)$	Zeitpunkt, an dem die Ausführung des Task-Graphen TG_i abgeschlossen sein muss
$t_{dm}(m_i)$	Zeitpunkt, an dem die Übertragung der Nachricht m_i abgeschlossen sein muss
$t_{dr}(\tau_i)$	frühester Zeitpunkt, an dem der Task τ_i seine Ausführung starten kann
$t_{f\tau}(\tau_i)$	Zeitpunkt, an dem die Ausführung des Tasks τ_i endet
$t_{fm}(m_i)$	Zeitpunkt, an dem die Übertragung der Nachricht m_i abgeschlossen wird
$t_{pm}(m)$	Zeit, nach der das Senden der Nachricht m innerhalb eines Kommunikationszyklus wiederholt wird
$t_{r\tau}(\tau_i)$	Zeitpunkt, an dem der Task τ_i zur Ausführung bereit ist
$t_{rm}(m_i)$	Zeitpunkt, an dem die Nachricht m_i zur Übertragung bereitsteht
$t_{s\tau}(\tau_i)$	Zeitpunkt, an dem die Ausführung des Tasks τ_i startet
$t_{sg}(TG_i)$	Zeitpunkt, an dem die Ausführung des Task-Graphen TG_i startet
$t_{sm}(m_i)$	Zeitpunkt, an dem die Übertragung der Nachricht m_i gestartet wird

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[Aeronautical Radio Incorporated 2009] Aeronautical Radio Incorporated: Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network / ARINC. 2009 (ARINC Report 664P7-1). – Standard

[Albert u. a. 2003] Albert, A. ; Strasser, R. ; Trächtler, A.: Migration from CAN to TTCAN for a Distributed Control System. In: *9th international CAN Conference, iCC 2003, Munich*, 2003, S. 9–15

[Armengaud u. a. 2009] Armengaud, E. ; Tengg, A. ; Driussi, M. ; Karner, M. ; Steger, C. ; Weiss, R.: Automotive software architecture: Migration challenges from an event-triggered to a time-triggered communication scheme. In: *Intelligent solutions in Embedded Systems, 2009 Seventh Workshop on*, june 2009, S. 95 –103

[AUDI AG, TTTech Automotive GmbH 2009] AUDI AG, TTTech Automotive GmbH: *FlexRay-Based Optimization of Data Communication: Audi A8 - The Sportiest Sedan in the Luxury Class*. Dezember 2009. – URL `http://www.tttech-automotive.com/fileadmin/content/pdf/Audi-Casestudy/TTTech-Audi-Casestudy-A8.pdf`. – Zugriffsdatum: 2011-01-18

[Bartols 2010] Bartols, Florian: *Leistungsmessung von Time-Triggered Ethernet Komponenten unter harten Echtzeitbedingungen mithilfe modifizierter Linux-Treiber*. Hamburg, HAW Hamburg, Bachelorthesis, Juli 2010. – Bachelorthesis

[Bauer u. a. 2009] Bauer, H. ; Scharbarg, J.-L. ; Fraboul, C.: Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network. In: *Emerging Technologies Factory Automation, 2009. ETFA 2009. IEEE Conference on*, sept. 2009, S. 1 –8. – ISSN 1946-0759

[Bauer u. a. 2010] Bauer, H. ; Scharbarg, J.-L. ; Fraboul, C.: Improving the Worst-Case Delay Analysis of an AFDX Network Using an Optimized Trajectory Approach. In: *Industrial Informatics, IEEE Transactions on* 6 (2010), nov., Nr. 4, S. 521 –533. – ISSN 1551-3203

[Bauer u. a. 2011] Bauer, Henri ; Scharbarg, Jean-Luc ; Fraboul, Christian: Applying Trajectory approach with static priority queuing for improving the use of available AFDX resources. In: *Real-Time Systems* (2011), S. 1–33. – URL http://dx.doi.org/10.1007/s11241-011-9142-9. – 10.1007/s11241-011-9142-9. – ISSN 0922-6443

[BMW Deutschland] BMW Deutschland: *Home*. – URL http://www.bmw.de/de/de/index.html. – Zugriffsdatum: 2012-03

[Bosch 2011] Bosch: *Bosch Kraftfahrzeugtechnik: Vehicle Dynamics Management* . 2011. – URL http://www.bosch-kraftfahrzeugtechnik.de/de/de/driving_safety/driving_safety_systems_for_passenger_cars_1/vehicles_dynamics_management/vehicles_dynamics_management_1.html. – Zugriffsdatum: 2012-01-20

[Boyer und Fraboul 2008] Boyer, M. ; Fraboul, C.: Tightening end to end delay upper bound for AFDX network calculus with rate latency FIFO servers using network calculus. In: *Factory Communication Systems, 2008. WFCS 2008. IEEE International Workshop on*, may 2008, S. 11 –20

[Buttazzo 2005] Buttazzo, Giorgio: *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications Second Edition* . 2. New York : Springer, 2005. – ISBN 0-387-23137-4

[Cormen u. a. 2003] Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L. ; Stein, Clifford: *Introduction to Algorithms, Third Edition.* 3. London, England : McGraw-Hill Book Company, 2003. – ISBN 978-0-262-03384-8

[Elektrobit] Elektrobit: *EB tresos Designer*. Elektrobit. – URL http://www.eb-tresos-blog.com/solutions/tresos/eb-tresos-designer/. – Zugriffsdatum: 2011-02

[FlexRay Consortium a] FlexRay Consortium: *FlexRay*. – URL http://flexray.com/. – Zugriffsdatum: 2011-02-07

[FlexRay Consortium b] FlexRay Consortium: *FlexRay*. – URL http://flexray.com/. – Zugriffsdatum: 2011-02-07

[FlexRay Consortium 2005a] FlexRay Consortium: FlexRay Communications System Electrical Physical Layer Specification / FlexRay Consortium. Stuttgart, Dezember 2005 (2.1 Revision A). – Specification

[FlexRay Consortium 2005b] FlexRay Consortium: FlexRay Communications System Protocol Specification / FlexRay Consortium. Stuttgart, Dezember 2005 (2.1 Revision A). – Specification

[FlexRay Consortium 2005c] FlexRay Consortium: FlexRay Requirements Specification / FlexRay Consortium. Stuttgart, Dezember 2005 (2.1). – Specification

[Ghosal u. a. 2010] Ghosal, A. ; Zeng, Haibo ; Di Natale, M. ; Ben-Haim, Y.: Computing robustness of FlexRay schedules to uncertainties in design parameters. In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, march 2010, S. 550 –555. – ISSN 1530-1591

[Greifeneder und Frey 2007] Greifeneder, Jürgen ; Frey, Georg: Analyse netzbasierter Automatisierungssysteme. In: *VDI-Berichte* 1980 (2007), S. 23–33

[Hu u. a. 2011] Hu, Ning ; Lv, Tangqi ; Huang, Ning: Applying Trajectory approach for computing worst-case end-to-end delays on an AFDX network. In: *Procedia Engineering* 15 (2011), Nr. 0, S. 2555 – 2560. – URL `http://www.sciencedirect.com/science/article/pii/S1877705811019813`. – <ce:title>CEIS 2011</ce:title>. – ISSN 1877-7058

[Jang u. a. 2011] Jang, K. ; Park, I. ; Han, J. ; Lee, K. ; Sunwoo, M.: Design framework for FlexRay network parameter optimization. In: *International Journal of Automotive Technology* 12 (2011), S. 589–597. – URL `http://dx.doi.org/10.1007/s12239-011-0069-x`. – 10.1007/s12239-011-0069-x. – ISSN 1229-9138

[Jonsson und Shin 1997] Jonsson, J. ; Shin, K.G.: Deadline assignment in distributed hard real-time systems with relaxed locality constraints. In: *Distributed Computing Systems, 1997., Proceedings of the 17th International Conference on*, may 1997, S. 432 –440

[Kern u. a. 2011] Kern, A. ; Streichert, T. ; Teich, J.: An automated data structure migration concept x2014; From CAN to Ethernet/IP in automotive embedded systems (CANoverIP). In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, march 2011, S. 1 –6. – ISSN 1530-1591

[Kim u. a. 2008] Kim, Seung-Han ; Seo, Suk-Hyun ; Kim, Jin-Ho ; Moon, Tae-Moon ; Son, Chang-Wan ; Hwang, Sung-Ho ; Jeon, Jae W.: A gateway system for an automotive system: LIN, CAN, and FlexRay. In: *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, july 2008, S. 967 –972. – ISSN 1935-4576

[Kopetz 2008] Kopetz, Hermann: The Rationale for Time-Triggered Ethernet. In: *Real-Time Systems Symposium, 2008*, Dezember 2008, S. 3–11. – ISSN 1052-8725

[Kopetz u. a. 2005] Kopetz, Hermann ; Ademaj, Astrit ; Grillinger, Petr ; Steinhammer, Klaus: The time-triggered Ethernet (TTE) design. In: *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005.*, Mai 2005, S. 22–33

[LIN-Administration] LIN-Administration: *Local Interconnect Network*. – URL `http://www.lin-subbus.org/`. – Zugriffsdatum: 2011-01-06

[Lukasiewycz u. a. 2011] Lukasiewycz, Martin ; Glaß, Michael ; Jürgen, Teich ; Paul, Milbredt: FlexRay Schedule Optimization of the Static Segment. In: *CODES+ISSS09* (2011), October

[Ma 2008] Ma, Xiaojun: *Schedulability-Driven Reliability Optimization of FlexRay Communications*. Denmark, Informatics and Mathematical Modelling, Technical University of Denmark, Masterthesis, 2008

[Millinger 2011] Millinger, Dietmar: *FlexRay Technology: Media Access*. 2011. – URL `http://www.millinger-consulting.com/dmtc/index.php?option=com_content&view=article&id=11&Itemid=11`. – Zugriffsdatum: 2011-11-20

[MOST Cooperation] MOST Cooperation: *Media Oriented Systems Transport*. – URL `http://www.mostcooperation.com/`. – Zugriffsdatum: 2011-01-06

[Nagarajan u. a. 2004] Nagarajan, Kandasamy ; John, Hayes ; Brian, Murray: Dependable Communication Synthesis for Distributed Embedded Systems. In: Palade, Vasile (Hrsg.) ; Howlett, Robert (Hrsg.) ; Jain, Lakhmi (Hrsg.): *Knowledge-Based Intelligent Information and Engineering Systems* Bd. 2788, Springer Berlin / Heidelberg, 2004, S. 275–288. – ISBN 978-3-540-40803-1

[Natale und Stankovic 1994] Natale, M. ; Stankovic, J.A.: Dynamic end-to-end guarantees in distributed real time systems. In: *Real-Time Systems Symposium, 1994., Proceedings.*, dec 1994, S. 216 –227

[Navet u. a. 2005] Navet, N. ; Song, Y. ; Simonot-Lion, F. ; Wilwert, C.: Trends in Automotive Communication Systems. In: *Proceedings of the IEEE* 93 (2005), june, Nr. 6, S. 1204 –1223. – ISSN 0018-9219

[Navet 2008] Navet, Nicolas: *Automating the Configuration of the FlexRay Communication Cycle*. 2008. – URL `http://www.loria.fr/~nnavet/publi/FlexRay_ProductDays_RTaW.pdf`. – Zugriffsdatum: 22-09-2011

[Navet und Simonot-Lion 2009] Navet, Nicolas ; Simonot-Lion, Françoise: *Automotive Embedded Systems Handbook*. New York : CRC Press, 2009. – ISBN 978-0-8493-8026-6

[Petru u. a. 1998] Petru, Eles ; Krzysztof, Kuchcinski ; Zebo, Peng ; Alexa, Doboli ; Pop, Paul: Scheduling of Conditional Process Graphs for the Synthesis of Embedded Systems. In: *Proceedings of the conference on Design, automation and test in Europe, 98*. Washington, DC, USA : IEEE Computer Society, 1998 (DATE '98), S. 132–139. – URL `http://dl.acm.org/citation.cfm?id=368058.368119`. – ISBN 0-8186-8359-7

[Pop 2003] Pop, Paul: *Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems*. Linköping, Linköping University, Dissertation, 2003. – Dissertation

[Pop u. a. 2007a] Pop, T. ; Pop, P. ; Eles, P. ; Peng, Z.: Bus Access Optimisation for FlexRay-based Distributed Embedded Systems. In: *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, april 2007, S. 1 –6

[Pop u. a. 2007b] Pop, Traian ; Pop, Paul ; Eles, Petru ; Peng, Zebo ; Andrei, Alexandru: Timing analysis of the FlexRay communication protocol. In: *Real-Time Systems* 39 (2007), Nr. 1-3, S. 205–235. – ISSN 0922-6443

[Rausch 2008] Rausch, Mathias: *FlexRay: Grundlagen, Funktionsweise, Anwendung*. München : Carl Hanser Verlag, 2008. – ISBN 978-3-446-41249-1

[Real Time Systems Group (RTS)] Real Time Systems Group (RTS): *TTEthernet*. – URL `http://ti.tuwien.ac.at/rts`. – Zugriffsdatum: 2010-12-10

[Reif 2011] Reif, Konrad: *Bosch Autoelektrik und Autoelektronik: Bordnetze, Sensoren und elektronische Systeme*. 6. Wiesbaden : Vieweg und Teubner, 2011. – ISBN 978-3-8348-1274-2

[Richard u. a. 2008] Richard, Murphy ; Walsh, Frank ; Brendan, Jackman: *Migration Framework from CAN to FlexRay*. 2008. – URL `http://repository.wit.ie/1046/1/CAN_to_FlexRay_Migration_Framework_-_AAE_Show_2008.pdf`

[Robert Bosch GmbH] Robert Bosch GmbH: *Controller Area Network*. – URL `http://www.semiconductors.bosch.de/`. – Zugriffsdatum: 2011-02-03

[Rui u. a. 2010] Rui, Zhao ; Gui-he, Qin ; Jia-qiao, Liu: Gateway system for CAN and FlexRay in automotive ECU networks. In: *Information Networking and Automation (ICINA), 2010 International Conference on* Bd. 2, oct. 2010, S. V2–49 –V2–53

[SAE - AS-2D Time Triggered Systems and Architecture Committee 2009] SAE - AS-2D Time Triggered Systems and Architecture Committee: *Time-Triggered Ethernet (AS 6802).* 2009. – URL `http://www.sae.org`. – Zugriffsdatum: 2010-12-11

[Scharbarg u. a. 2009] Scharbarg, J.-L. ; Ridouard, F. ; Fraboul, C.: A Probabilistic Analysis of End-To-End Delays on an AFDX Avionic Network. In: *Industrial Informatics, IEEE Transactions on* (2009), feb., S. 38 –49. – ISSN 1551-3203

[Schmidt und Schmidt 2009] Schmidt, E.G. ; Schmidt, K.: Message Scheduling for the FlexRay Protocol: The Dynamic Segment. In: *Vehicular Technology, IEEE Transactions on* 58 (2009), jun, Nr. 5, S. 2160 –2169. – ISSN 0018-9545

[Schmidt 2011] Schmidt, Karsten: *Architekturen in der automobilen Softwareentwicklung.* Audi Electronics Ventures GmBH. 2011. – URL `http://www.architecture-day.de/wp-content/uploads/2011/06/Architecture.Day_Schmidt.Karsten.pdf`. – Zugriffsdatum: 2011-11-30

[Schmidt u. a. 2010] Schmidt, Klaus ; Güran, Schmidt E. ; Demirci, Ali ; Yuruklu, Emrah ; Karakaya, Utku: An Experimental Study of the FlexRay Dynamic Segment. In: *Advances in Automotive Control, 6th IFAC Symposium Advances in Automotive Control (2010)* (2010). – ISSN 1551-3203

[Schwager 2010] Schwager, Jürgen: *Informationsportal für Echtzeit-Ethernet in der Industrieautomation.* Hochschule Reutlingen. März 2010. – URL `http://www.pdv.reutlingen-university.de/rte/`

[Seo u. a. 2006] Seo, Suk-Hyun ; Lee, Sang won ; Hwang, Sung-Ho ; Jeon, Jae W.: Development of Network Gateway Between CAN and FlexRay Protocols For ECU Embedded Systems. In: *SICE-ICASE, 2006. International Joint Conference*, oct. 2006, S. 2256 –2261

[Sinnen 2007] Sinnen, Oliver: *Task Scheduling for Parallel Systems.* 1. New Jersey : John Wiley and Sons, Mai 2007. – ISBN 978-0471735762

[Steinbach 2011] Steinbach, Till: *Echtzeit-Ethernet für Anwendungen im Automobil: Metriken und deren simulationsbasierte Evaluierung am Beispiel von TTEthernet.* Hamburg, Hochschule für Angewandte Wissenschaften Hamburg, Masterthesis, Februar 2011

[Steinbach u. a. 2011] Steinbach, Till ; Dieumo Kenfack, Hermand ; Korf, Franz ; Schmidt, Thomas C.: An Extension of the OMNeT++ INET Framework for Simulating Real-time

Ethernet with High Accuracy. In: *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, 2011. – to appear

[Steinbach u. a. 2010] Steinbach, Till ; Korf, Franz ; Schmidt, Thomas C.: Comparing Time-Triggered Ethernet with FlexRay: An Evaluation of Competing Approaches to Real-time for In-Vehicle Networks. In: *8th IEEE Intern. Workshop on Factory Communication Systems*. Piscataway, New Jersey : IEEE Press, Mai 2010, S. 199–202

[Steiner u. a. 2009] Steiner, W. ; Bauer, G. ; Hall, B. ; Paulitsch, M. ; Varadarajan, S.: TTEthernet Dataflow Concept. In: *8th IEEE International Symposium on Network Computing and Applications, 2009. NCA 2009.*, Juli 2009, S. 319–322

[Steiner 2008] Steiner, Wilfried: *TTEthernet Specification*. TTTech Computertechnik AG. November 2008. – URL `http://www.tttech.com`

[Steiner 2011] Steiner, Wilfried: Synthesis of Static Communication Schedules for Mixed-Criticality Systems. In: *14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW) 2011*, März 2011, S. 11–18

[Traub 2010] Traub, Matthias: *Durchgängige Timing-Bewertung von Vernetzungsarchitekturen und Gateway-Systemen im Kraftfahrzeug*. Karlsruher, Karlsruher Institut für Technologie (KIT), Dissertation, 2010. – Dissertation

[TTTech Computertechnik AG] TTTech Computertechnik AG: *Home*. – URL `http://www.tttech.com`. – Zugriffsdatum: 2011-01-17

[TTTech Computertechnik AG 2008] TTTech Computertechnik AG: *TTEthernet Application Programming Interface*. TTTech Computertechnik AG. Dezember 2008. – URL `http://www.tttech.com`

[TU-Dresden und Lehrstuhl-Fahrzeugmechatronik 2011] TU-Dresden ; Lehrstuhl-Fahrzeugmechatronik: *Fahrzeug-E/E-Architektur*. 2011. – URL `http://tu-dresden.de/die_tu_dresden/fakultaeten/vkw/iad/professuren/fm/forschung/schwpkt/architektur`. – Zugriffsdatum: 22-09-2011

[Vector Informatik] Vector Informatik: *FlexRay Network Designer* . Vector Informatik. – URL `http://www.vector.com/vi_networkdesigner_flexray_de.html`. – Zugriffsdatum: 2011-02

[Voss 2010] Voss, Sebastian: *Integrated Task and Message Scheduling in Time-Triggered Aeronautic Systems.* München, Fakultät Wirtschaftswissenschaften der Universität Duisburg-Essen, Dissertation, 2010. – Dissertation

[Wikipedia a] Wikipedia: *Begriffe: Sender-Empfänger-Modell, Fehlertoleranz, Toolchain.* Wikipedia. – URL `http://de.wikipedia.org/wiki/Wikipedia:Hauptseite`. – Zugriffsdatum: 2012-03

[Wikipedia b] Wikipedia: *terms: scalable, deterministic system.* Wikipedia. – URL `http://en.wikipedia.org/wiki/Main_Page`. – Zugriffsdatum: 2012-03

[Zeng und Natale 2009] Zeng, Haibo ; Natale, Di: *Overview of FlexRay scheduling issues.* 2009. – URL `http://retis.sssup.it/sites/retis.sssup.it/files/lesson10-introduction_to_FlexRay.pdf`. – Zugriffsdatum: 2011-11-30

[Zeng u. a. 2009] Zeng, Haibo ; Zhengzheng, Wei ; Di Natale, M. ; Ghosal, A. ; Giusto, P. ; Sangiovanni-Vincentelli, A.: Scheduling the FlexRay bus using optimization techniques. In: *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, july 2009, S. 874 –877. – ISSN 0738-100X

[Zeng und Song 2009] Zeng, Xingxing ; Song, Dong: The Research on End-to-End Delay Calculation Method for Real-Time Network AFDX. In: *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on*, dec. 2009, S. 1 –4

[Zhao 2011] Zhao, Aijie: *Reliable In-Vehicle FlexRay Network Scheduler Design.* Netherlands, TU Delft, Faculty of Electrical Engineering, Mathematics and Computer Science, Masterthesis, 2011

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

Hamburg, 12. April 2012 Hermand Dieumo Kenfack