Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Jan Kamieth

Scheduling von TDMA Kommunikation in Switch basierten Netzwerken

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jan Kamieth

Scheduling von TDMA Kommunikation in Switch basierten Netzwerken

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Franz Korf
Zweitgutachter: Prof. Dr.-Ing. Andreas Meisel

Abgegeben am 24. Januar 2015

Jan Kamieth

Titel der Masterarbeit
Scheduling von TDMA Kommunikation in Switch basierten Netzwerken
Stichworte
Echtzeit-Ethernet, TTEthernet, Swich basierte Netzwerke, TDMA, Scheduling, Multiprozessor, Problembeschreibung, Optimierung, Latenz, Framework
Kurzzusammenfassung
Die wachsende Anzahl elektronischer Systeme im Automobil und deren vielfältigen Anforderungen an das zugrundeliegende Netzwerk führt zu einem Bedarf an neuer Kommunikationstechnik, um sowohl eine hohe Bandbreite, als auch niedrige Latenzen und einen niedrigen Jitter zu bieten. TT-Ethernet ist ein vielversprechender Kandidat, um dieses Problem zu lösen.
In dieser Arbeit wird eine Lösung für das Scheduling Problem erarbeitet, welches bei dieser Art von Netzwerken auftritt. Dazu wird das Problem in den Bereich des Multiprozessor-Schedulings transformiert, um auf bekannte Lösungsansätze aufbauen zu können. Zusätzlich wird ein Optimierungsziel und eine dazugehörige Bewertungsfunktion definiert, die auf die Eigenschaften und Anforderungen des TT-Ethernet Protokolls zugeschnitten sind. Zur Umsetzung des Schedulings wird ein Framework umgesetzt, welches es erlaubt mehre Lösungsansätze zu implementieren und aus mehreren Ergebnissen das Beste wählen zu können.

Title of the master thesis
Scheduling of TDMA communication in switch based networks
Keywords
Real-time Ethernet, TTEthernet, switch based networks, TDMA, scheduling, multi-processor, problem description, optimization, latency, framework
Abstract
The increasing amount of electronic control units in the automotive domain and the versatile requirements towards the underlying network leads to the demand of a new comunication technology to provide high bandwidth as well as a low latency and jitter. TT-Ethernet is a promising candidate to solve this problem.
This thesis developes a solution for the schedulig problem, which is implied by these kind of networks. To achieve this goal, the problem is transformed into the domain of multiprocessor scheduling to be able to reuse and extend well known scheduling approaches. Furthermore, a optimization goal and the associated evaluation function are developed, to respect the characteristics and requirements of the TT-Ethernter protocol. A framework is implemented to realize the scheduling, which copes multiple approaches to solve the problem. This enables the user to choose the most suitable schedule out of a batch of solutions.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Seit der Erfindung des Automobils schreitet die technologische Entwicklung in einer immer höheren Geschwindigkeit voran. Der aktuelle Höhepunkt in diesem Fortschritt ist der Übergang in die Elektromobilität und der Einsatz komplexer elektronischer Systeme. Moderne Kraftfahrzeuge besitzen bereits über 100 elektronische Module in der Motorsteuerung, in Sicherheitssystemen und in der Komfortelektronik. Diese Zahl ist stark ansteigend und wächst jährlich um einen Faktor von 1,45. Die Größe der verwendeten Softwarekomponenten wächst gleichzeitig um einen Faktor von 4,5 (vgl. Abelein u. a. (2012)). Gleichzeitig handelt es sich bei diesen Komponenten um hoch vernetzte Systeme, die ca. 2500 unterschiedliche Nachrichten austauschen (vgl. Schäuffele und Zurawka (2013)).

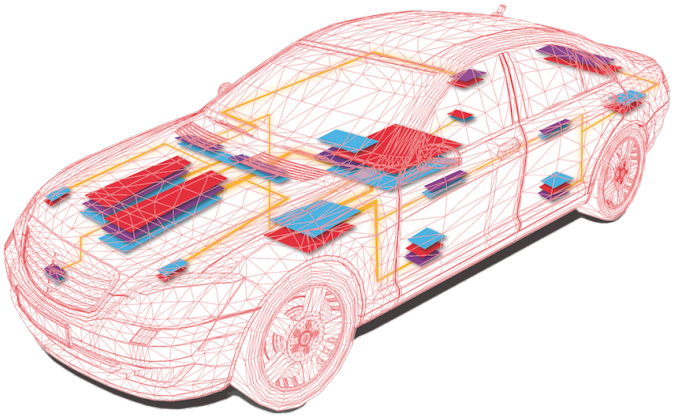

Abbildung 1.1: *Vernetzte Komponenten eines Automobils (Quelle: CoRE-Arbeitsgruppe)*

Einen großen Anteil an dieser Entwicklung haben moderne Fahrerassistenzsysteme, der Einsatz von X-By-Wire Systemen, Informationssysteme und Unterhaltungselektronik. Zwischen den Steuergeräten werden die Daten mittels unterschiedlicher Feldbusse ausgetauscht, die entsprechend den Anforderungen ihrer Anwendungsdomäne entwickelt wurden, wie zum Beispiel Busse für die Motorsteuerung oder Infotainment. Führende Systeme sind das *Controller Area Network* (CAN) (vgl. Robert Bosch GmbH), *Media Oriented Systems Transport* (MOST) (vgl. MOST Cooperation) und *FlexRay* (vgl. FlexRay Consortium (2010)). Für die Kommunikation zwischen den unterschiedlichen Bussystemen werden Gateways eingesetzt, um eine domänenübergreifende Funktionalität zu ermöglichen. Diese heterogenen Strukturen

erhöhen die Komplexität und gestalten die Entwicklung und das Testen dieser Systeme sehr aufwendig.

Zudem verlangen neue Anwendungen, die Kamera oder Laser basierte Scanner einsetzen, immer höhere Bandbreiten zur Übertragung der Daten, die die Kapazitäten der aktuellen Feldbusse überschreiten. In aktuellen Fahrzeugen werden teure Verkabelungen wie dem *Low Voltage Differential Signaling* (LVDS) eingesetzt, um die auftretenden Datenmengen zu bewältigen. Um diese Anforderungen an die Bandbreite auf eine kostengünstigere Weise zu erfüllen und zur gleichen Zeit auch die Komplexität der Vernetzung zu reduzieren, zeigt die Industrie ein steigendes Interesse an einem Einsatz der Ethernet Technologie als fahrzeuginternes Netzwerk (vgl. Bello (2011)). Die aktuelle Forschung beschäftigt sich daher unter anderem mit der Frage, wie die Ethernet Technologie zur Lösung der domänenspezifischen Probleme beitragen kann. Untersucht werden dabei Protokolle wie *Ethernet/IP* (vgl. Kern u. a. (2011)), *Ethernet AVB* (vgl. Queck (2012)) und Time-Triggered Ethernet (vgl. Steinbach u. a. (2011a)).

Time-Triggered Ethernet Protokolle erweisen sich dabei als vielversprechende Kandidaten zur Erfüllung der strikten Anforderungen. Die Vorteile sind die Eigenschaften einer zeitgesteuerten Kommunikation, wie niedrige Latenz, niedrigem Jitter und Determinismus und die erhöhte Bandbreite gegenüber FlexRay und CAN. Die IEEE *Time-Sensitive Networking* (TSN) Arbeitsgruppe hat daher damit begonnen, Time-Triggered Ethernet unter der Bezeichnung IEEE 802.1Qbv (vgl. Institute of Electrical and Electronics Engineers (2013)) als eine Erweiterung des *Audio/Video Bridging* (AVB) Protokolls zu standardisieren.

Eine der größten Herausforderungen beim Einsatz eines Time-Triggered Ethernet Protokolls ist das Erstellen eines gültigen und effizienten Schedules für die Applikationen auf den Steuergeräten und den zeitgesteuerten Nachrichten des Netzwerks. Während sich bei zeitgesteuerten Feldbussen wie FlexRay alle Kommunikationsteilnehmer einen Bus teilen müssen und so nur ein Schedule für die komplette Topologie benötigt wird, basiert Switched Ethernet auf Punkt-zu-Punkt-Verbindungen zwischen den Steuergeräten und dem Switch. In diesem Fall muss für jede Verbindung ein Schedule für die zeitgesteuerten Nachrichten berechnet werden. Hinzu kommt, dass die einzelnen Schedules aufeinander abgestimmt werden müssen, um die spezifischen Echtzeit Ende-zu-Ende Anforderungen aller Anwendungen zu erfüllen.

Eine Eigenschaft der Time-Triggered Ethernet Protokolle ist es, dass durch den Schedule für jede zeitgesteuerte Nachricht ein Übertragungsfenster definiert wird, dass nur von dieser speziellen Nachricht genutzt werden kann. Dies verhindert das Versenden von nichtzeitgesteuerten Verkehr, so dass Übertragungslücken zwischen den zeitgesteuerten Übertragungsfenstern benötigt werden. Um diese Anforderung zu erfüllen, werden Heuristiken und Bewertungsfunktionen benötigt, die es ermöglichen, einen gültigen und optimierten Schedule zur Designzeit des Netzwerks zu erstellen.

In dieser Arbeit wird ein Modell vorgestellt, welches es ermöglicht, dass Scheduling-Problem eines Time-Triggered Ethernet Netzwerks in den Bereich des Echtzeit-Multiprozessor Schedulings abzubilden. Dies ermöglicht es, bestehende Scheduling-Strategien zu nutzen und gegebenenfalls den neuen Anforderungen anzupassen. Zur Unterstützung dieses Verfahrens wird ein Framework entwickelt, dass es erlaubt, mit wenig Aufwand mehrere dieser Heuristiken auf ein Time-Triggered Switched Ethernet Netzwerk anzuwenden und auf diesem Weg mehrere Schedules generiert. Um aus dieser Lösungsmenge das qualitativ hochwertigste Ergebnis zu ermitteln, wird ein Optimierungsziel festgelegt und die dazugehörige Bewertungsfunktion entwickelt. Eine experimentelle Evaluation dieser Methoden wird anhand eines Netzwerks vorgenommen, das die Kommunikationsstruktur eines aktuellen Serienfahrzeugs nachbildet.

Die beiden folgenden Kapitel 2 und 3 befassen sich mit den Grundlagen zum Time-Triggered Ethernet Protokoll und zum generellen Scheduling. In Kapitel 4 wird die Transformation des Netzwerks in die Domäne des Echtzeit-Multiprozessor Schedulings beschrieben und in Kapitel 5 wird darauf aufbauend die Bewertungsfunktion und dafür passende Heuristiken entwickelt. Kapitel 6 enthält eine Evaluation der Ergebnisse und Kapitel 7 eine Beschreibung vom Scheduling-Framework. Das Kapitel 8 schließt diese Arbeit mit einer Zusammenfassung der Ergebnisse und einem Fazit ab.

2 Das Time-Triggered Ethernet Protokoll

Das Time-Triggered Ethernet (TT-Ethernet) Protokoll ist eine Erweiterung der Ethernet Technologie, die in dem IEEE-Standard 802.3 (vgl. Institute of Electrical and Electronics Engineers (2008)) spezifiziert ist. Entwickelt wurde das Protokoll von der Real-Time Systems Group (vgl. Real Time Systems Group (RTS)) der TU Wien. Später übernahm die Firma TTTech Computertechnik AG (vgl. TTTech Computertechnik AG) das Konzept und entwickelte es weiter. Die TT-Ethernet Spezifikation (vgl. Steiner (2008)) wurde von der Society of Automotive Engineers (SAE) in der Arbeitsgruppe AS-2D standardisiert (vgl. SAE - AS-2D Time Triggered Systems and Architecture Committee (2009)).

TT-Ethernet erweitert das Standard-Ethernet um Echtzeit-Fähigkeiten, so dass es möglich ist, normalen Ethernet Verkehr (best-effort) zusammen mit Nachrichten, die harte Echtzeit-Anforderungen erfüllen müssen, im selben Netzwerk zu übertragen. Um die Echtzeit-Fähigkeit zu gewährleisten, wird zusätzlich, zu dem beim Ethernet eingesetzten CSMA/CD-Algorithmus, auf ein TDMA basiertes Zugriffsprotokoll zurück gegriffen. Dieses Protokoll erlaubt es, für einzelne Nachrichten feste zyklische Übertragungsfenster zu konfigurieren, so dass eine gesicherte Übertragung mit niedriger Latenz und niedrigem Jitter garantiert werden kann. Auf technischer Seite bedeutet dies, dass für diese *time-triggered* Nachrichten jedes an der Kommunikation teilnehmende Gerät eine lokale Uhr und einen vorkonfigurierten Schedule besitzt, in dem genau festgelegt ist, wann welche TT-Nachricht übertragen wird. Die lokalen Uhren der Sender und Empfänger werden durch ein zusätzliches fehlergesichertes Synchronisierungsprotokoll aus der TTEthernet-Spezifikation synchronisiert, so dass eine globale Uhrzeit im Netzwerk herrscht.

Für eine ereignisbasierte Kommunikation stehen *rate-constrained* Nachrichten zur Verfügung, die auf dem AFDX Protokoll (vgl. Aeronautical Radio Incorporated (2009a)) basieren. Anstatt eines Schedules wird für RC-Nachrichten eine Menge an Bandbreite konfiguriert, die diese verbrauchen dürfen. Als dritte Option können *best-effort* Nachrichten versendet werden, die den Standard Ethernet Nachrichten entsprechen. Durch die Kompatibilität zum Standard Ethernet ist es möglich, Geräte an das Netzwerk anzuschließen, die nicht zu dem TT-Ethernet Protokoll kompatibel sind.

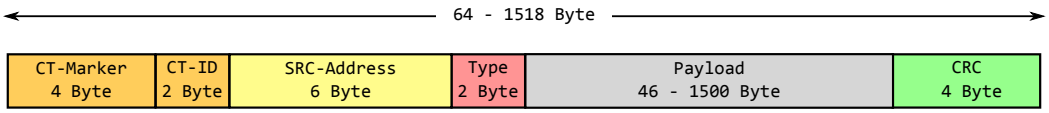

Abbildung 2.1: *Aufbau eines TT-Ethernet Frames*

2.1 Virtuelle Links

Das Versenden von zeitkritischen Nachrichten erfolgt innerhalb des Netzwerks anhand von virtuellen Links. Diese Links ersetzen das ursprüngliche Routing im Ethernet anhand der Ziel-Adresse. Stattdessen besitzt jede TT- und RC-Nachrichten eine *critical-traffic* Adresse, die aus einem CT-Marker und einem CD-Identifier besteht. Der Aufbau eines TT-Ethernet Frames ist in Abbildung 2.1 dargestellt. Der CT-Marker erlaubt es, den Frame durch ein einfaches Matching als zeitkritisch zu erkennen, während der Identifier genutzt wird, um die Nachricht einem virtuellen Link zuzuordnen.

Virtuelle Links stellen eine logische Verbindung zwischen Nachrichten, Sendern und Empfängern her. Zu jedem virtuellen Link gehört eine Nachricht, die anhand der CT-ID erkannt wird, ein Sender und ein oder mehrere Empfänger. Empfängt ein Switch eine zeitkritische Nachricht, erfolgt anhand der CT-ID eine Zuordnung zu dem entsprechenden virtuellen Link, so dass der Switch weiß, an welche Empfänger die Nachricht weitergeleitet werden muss. Das Routing der Nachrichten durch das Netzwerk wird durch die virtuellen Links direkt kontrolliert und mit den Links statisch konfiguriert. In der Abbildung 2.2 ist ein TT-Ethernet Netzwerk zu sehen, das aus einem Switch und vier Steuergeräten besteht. In diesem Netzwerk sind drei virtuelle Links konfiguriert. Über den Link $VL - TT_1$ wird eine TT-Nachricht vom Gerät N_1 an die Geräte N_3 und N_4 gesendet. Anhand des Links $VL - TT_2$ wird eine TT-Nachricht von N_2 nach N_4 übertragen, während der Link $VL - RC_1$ für die Übertragung einer RC-Nachricht von N_2 nach N_3 konfiguriert ist.

2.2 TT-Ethernet Nachrichten-Klassen

Die drei unterschiedlichen Nachrichten-Klassen des Protokolls bilden das Kernstück eines TT-Ethernet Netzwerks. In diesem Abschnitt wird im Detail auf ihre Eigenschaften eingegangen und die benötigte Konfiguration beschrieben.

Time-Triggered Nachrichten Time-Triggered (TT) Nachrichten sind die wichtigsten Nachrichten im System. Jede TT-Nachricht wird periodisch über den zugehörigen virtuellen Link versendet. Die Zeitpunkte der Übertragung und das Routing der Nachricht

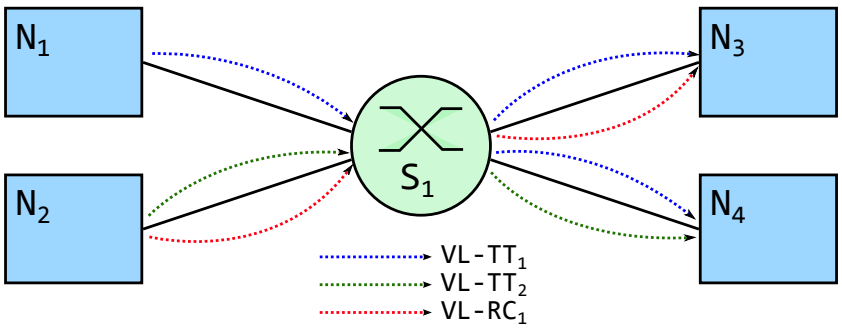

Abbildung 2.2: *TT-Ethernet Netzwerk mit drei virtuellen Links*

sind vollständig statisch konfiguriert. Jedes an der Übertragung beteiligte Gerät der Nachricht besitzt für diese einen festen Schedule, der den Zeitraum für die Sende- und Empfangsfenster definiert. Innerhalb dieser Fenster darf nur diese Nachricht gesendet, bzw. empfangen werden. Versucht eine Applikation auf dem Sender innerhalb dieses Fensters eine andere Nachricht senden, wird diese blockiert. Dieses Verhalten ermöglicht ein deterministisches Übertragen von TT-Nachrichten, so dass diese eine Kommunikation mit harten Echtzeitanforderungen ermöglichen. Das Versenden der Nachrichten erfolgt zyklisch. Jede TT-Nachricht besitzt eine Periode, deren Länge den Abstand zwischen der wiederholten Übertragung der Nachricht bestimmt.

Als Beispiel soll ein 1518 Byte großer Frame als TT-Nachricht über den virtuellen Link $VL - TT_1$ aus dem Beispiel aus Abbildung 2.2 gesendet werden. Das Netzwerk operiert mit 100 Mbit/s, so dass die Übertragung des Frames 121,44 µs dauert. Innerhalb des Switch wird die Nachricht durch das Weiterleiten um 50 µs verzögert. In Tabelle stehen die entsprechenden Sende- und Empfangs-Zeitpunkte für die beteiligten Geräte.

	Empfangsfenster		Sendefenster	
Gerät	Start	Ende	Start	Ende
N_1	-	-	0 µs	130 µs
S_1	0 µs	130 µs	180 µs	310 µs
N_3	310 µs	440 µs	-	-
N_4	310 µs	440 µs	-	-

Tabelle 2.1: *Sende- und Empfangsfenster für den virtuellen Link $VL - TT_1$*

Rate-Constrained Nachrichten Rate-Constrained (RC) Nachrichten sind ebenfalls für die Übertragung von Echtzeitdaten vorgesehen. Im Gegensatz zu TT-Nachrichten werden für RC-Nachrichten aber keine Übertragungsfenster und Perioden definiert, sondern so

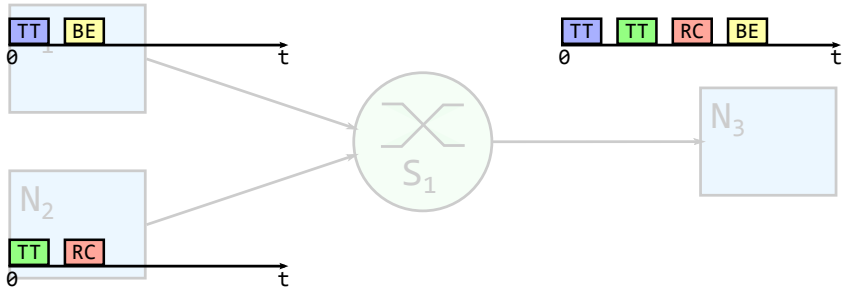

Abbildung 2.3: *Beispiel eines Schedules mit allen Nachrichten-Klassen*

genannte Bandwidth Allocation Gap-Accounts (BAG). Eine BAG definiert ein minimales Intervall für eine RC-Nachricht, das zwischen zwei Nachrichten mit der selben CT-ID eingehalten werden muss. Versucht eine Applikation, die Nachricht öfter zu senden, wird diese vom Switch verworfen. Zusätzlich legt die BAG fest, wie groß die Nachricht maximal sein darf. Da RC-Nachrichten eine niedrigere Priorität haben als TT-Nachrichten, werden diese im Switch verzögert, sollten auf dem Ausgangs-Port TT-Nachrichten zum Versenden vorliegen. Dadurch lässt sich weder die Latenz noch die Höhe des Jitters für diese Nachrichten vorhersagen. Stattdessen wird nur die konfigurierte Bandbreite garantiert, so dass sich diese Nachrichten für eine asynchrone Kommunikation eignen.

Best-Effort Nachrichten Die Übertragung von Best-Effort (BE) Daten entspricht dem Versenden von Nachrichten im Standard-Ethernet. Da BE-Nachrichten die geringste Priorität gegenüber dem TT- und RC-Datenverkehr haben, werden diese Nachrichten nur übertragen, wenn auf dem Kanal keine Echtzeitdaten gesendet werden. Zudem können BE-Nachrichten von jedem Gerät im Netzwerk verworfen werden, sollte durch ihre Präsenz das Einhalten von TT- oder RC-Timings gefährdet sein. Daher kann weder eine bestimmte Latenz noch eine garantierte Auslieferung der Nachrichten sichergestellt werden.

Die Abbildung 2.3 demonstriert ein Netzwerk mit einem Switch und drei Teilnehmern, in dem jeweils zwei TT-, eine RC- und eine BE-Nachricht von den Steuergeräten N_1 und N_2 zum Steuergerät N_3 gesendet werden. In dem Beispiel wird die Priorisierung der unterschiedlichen Nachrichten-Klassen deutlich. Auf dem Link zum Empfänger N_3 verdrängen die TT-Nachrichten die RC-Nachricht. Das BE-Paket kann erst übertragen werden, nachdem die Echtzeit-Nachrichten gesendet wurden.

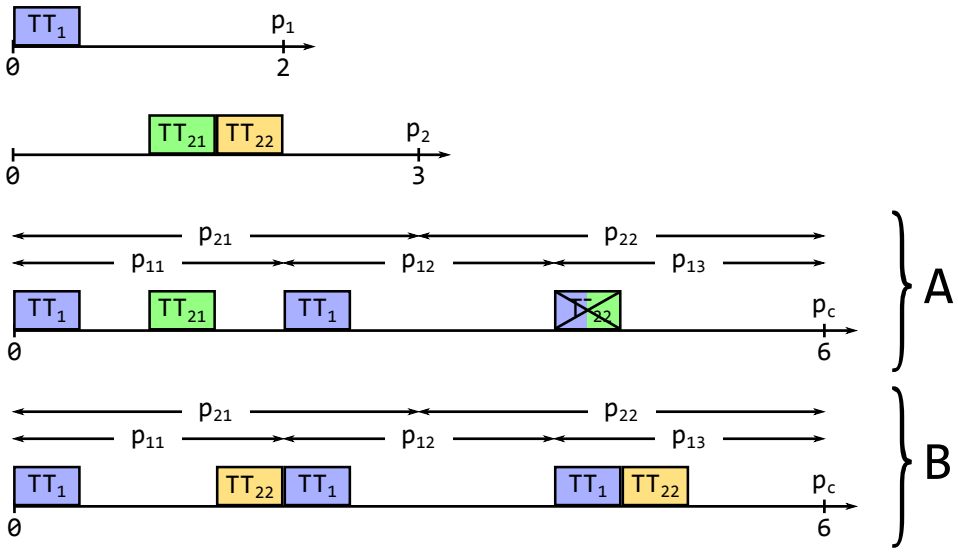

Abbildung 2.4: *Link- und Cluster-Perioden in zwei Versionen*

2.3 Perioden

Eine wichtige Komponente innerhalb des TT-Ethernet Protokolls sind Perioden, auch Zyklen genannt, die für jede einzelne TT-Nachricht konfiguriert werden. Diese Perioden legen fest, in welchem zeitlichen Abstand eine Nachricht wiederholt gesendet wird. Wird der Sendezeitpunkt einer Nachricht festgelegt, kann dieser innerhalb der ersten Instanz der Periode frei gesetzt werden. In den nachfolgenden Instanzen ist es erforderlich, dass der Zeitpunkt der Übertragung innerhalb der Periode der gleiche ist, relativ gesehen zum Beginn der Instanz. Damit ist der zeitliche Abstand zweier aufeinanderfolgender Instanzen einer TT-Nachricht immer derselbe. Neben diesen Perioden der TT-Nachrichten gibt es noch eine logische Cluster-Periode. Dieser Zyklus beschreibt den Zeitraum, in dem jede TT-Nachricht des Netzwerks mindestens einmal gesendet werden muss. Eine weitere Eigenschaft dieser Periode ist, dass ihre Länge das größte gemeinsame Vielfache aller Perioden der TT-Nachrichten sein muss. Das Erstellen der Cluster-Periode ist wichtig, um den Schedule der TT-Sendefenster zu überprüfen. Betrachtet man nur jeweils die ersten Instanzen der Perioden, können Überschneidungen von Übertragungsfenstern bei späteren Instanzen übersehen werden. Nur die Überprüfung der Cluster-Periode garantiert einen gültigen Schedule.

In der Abbildung 2.4 sind die Perioden eines Netzwerks illustriert, in dem zwei virtuelle Links mit TT-Nachrichten konfiguriert sind. Die Perioden der Nachrichten TT_1 und TT_2 haben eine unterschiedliche Länge, so dass die Cluster-Periode p_c durch das größte gemeinsame Vielfache

der beiden Perioden gebildet wird. Die Periode p_1 hat die Länge 2 und p_2 die Länge 3, so dass sich die Länge der Cluster-Periode zu 6 ergibt. In p_2 sind zwei TT-Nachrichten abgebildet, TT_{21} und TT_{22}, die jeweils für unterschiedliche Sendefenster innerhalb der Periode stehen. Die Zeitleisten *A* und *B* zeigen jeweils die Cluster-Periode. In Zeitleiste *A* wurde die Position TT_{21} in der zweiten Periode gewählt. Dies führt aber zu einem Fehler im Schedule, da es zu einer Überschneidung in den Instanzen der Perioden p_{13} und p_{22} kommt. Die Zeitleiste *B* zeigt einen korrekten Schedule, in dem die Position in der zweiten Periode mit TT_{22} weiter hinten gewählt wurde, so dass es zu keiner Überschneidung kommt.

2.4 Synchronisation

Um ein systemweites Scheduling zu ermöglichen, müssen sämtliche Teilnehmer ihre Systemzeit angleichen, damit ein zeitlich korrekter Ablauf der Aufgaben möglich wird. Da nicht gewährleistet werden kann, dass eingebaute Taktgeber immer synchron zueinander laufen, divergiert die Systemzeit nach einer gewissen Zeitspanne. Eine fortlaufende Synchronisation ist damit erforderlich.

TT-Ethernet sieht hierfür eine fehlertolerante Zwei-Wege-Synchronisation vor, die durch eine Master-/Slave Architektur erreicht wird. Die Teilnehmer nehmen dafür unterschiedliche Rollen im Netzwerk an.

Synchronisation-Master (SM) initialisieren die Synchronisation. Dafür versenden sie einen Synchronisations-Frame, einen so genannten Protocol-Control-Frame (PCF). Dieser enthält die aktuelle Systemzeit des Masters.

Compression-Master (CM) empfangen die gesendete Zeit von jedem SM und berechnen anhand dieser eine neue globale Systemzeit, die sie nun wieder an alle Clients senden.

Synchronisation-Client (SC) aktualisieren nach dem Erhalt der neuen globalen Systemzeit vom Compression-Master ihre lokalen Systemzeiten.

Für die Synchronisation senden zunächst alle Synchronisations-Master ihre lokale Systemzeit mittels eines PCF-Frames an den Compression-Master. Dieser berechnet anhand dieser Zeiten die neue globale Systemzeit und sendet diese an die Synchronisations-Clients. Im Netzwerk können dabei mehrere Synchronisations-Master definiert werden, damit auch bei einem Ausfall eines Masters die Synchronisation weiterhin möglich ist.

3 Scheduling Grundlagen

Im Allgemeinen versteht man unter Scheduling die Verwaltung von mehreren Aufgaben, die für ihre Ausführung bestimmte Ressourcen benötigen. Der Scheduler berechnet dafür die zeitliche Ordnung der Aufgabe und versucht dabei, diese Ordnung auf eine definierte Kostenfunktion hin zu optimieren, wie zum Beispiel die bestmögliche Ausnutzung der Ressourcen. Scheduling wird in den unterschiedlichsten Themengebieten benötigt, wie zum Beispiel in der Logistik, der Fertigung und nicht zuletzt auch in der Informatik. Die Methoden zur Darstellung und Lösung von Scheduling Problemen stammen aus der diskreten Mathematik und dem Bereich der kombinatorischen Optimierung.

In der Informatik wird mit dem Scheduling die Zuordnung von Prozessen auf Prozessoren beschrieben. Dabei hängen die Lösungsansätze stark von der vorhandenen Prozessor-Umgebung ab. Unterschieden wird zwischen Umgebungen mit nur einem Prozessor und Umgebungen mit mehreren Prozessoren. Mehrprozessorsysteme unterteilt man weiter in parallele und dedizierte Systeme. In parallelen Systemen kann jeder Prozess auf jedem Prozessor ausgeführt werden, während in dedizierten Systemen jeder Prozess nur auf einem bestimmten Prozessor ausgeführt werden kann.

Das Problem des Schedulings eines TT-Ethernet Systems ist in dem Gebiet der dedizierten Mehrprozessorsysteme angesiedelt und wird in den folgenden Kapiteln formal und im Detail beschrieben. Dafür beginnen wir mit den Grundlagen und benötigten Termini der Scheduling Theorie um darauf aufbauend die eigentliche Problembeschreibung durchführen. Danach wird auf mögliche Metriken eingegangen, die dabei helfen, errechnete Lösungen zu bewerten. Abschließend werden mögliche Lösungsansätze erarbeitet, um das TT-Ethernet Scheduling Problem zu lösen.

3.1 Grundlagen

In diesem Abschnitt werden die nötigen Termini und theoretischen Grundlagen erklärt, die benötigt werden, um das Scheduling-Problem zu beschreiben. Angefangen bei den Begriffen Prozessor und Prozess, die umgangssprachlich in diesem Bereich verwendet werden, muss man diese in der Schedulingtheorie näher definieren, um Verwechslungen zu vermeiden.

Folgend wird nicht mehr der Begriff Prozess benutzt, sondern von einem *Task* gesprochen, wie es in der klassischen Schedulingtheorie üblich ist. Weiterhin beschreibt der Begriff *Prozessor* eine einzelne Recheneinheit, auf der jeweils immer nur ein Task zur selben Zeit ausgeführt werden kann. Eine Menge von Tasks bildet oft eine logische Einheit, die durch Abhängigkeiten zwischen den Tasks gebildet werden. Diese Einheiten werden als *Jobs* bezeichnet.

Weiterhin besitzen Tasks zwei Charakteristika die ausschlaggebend für das Scheduling sind. Zuerst wären die Timing Eigenschaften der Tasks zu nennen. Diese bestehen aus einer Menge von Zeitangaben, die das Verhalten der Tasks detailliert beschreiben. Diese werden im Detail im nächsten Abschnitt definiert. Das zweite Charakteristikum sind die Abhängigkeiten der Tasks untereinander. Diese werden mit Hilfe der Graphentheorie behandelt und im übernächsten Abschnitt näher erläutert.

Ein klassisches Scheduling Problem in einer Mehrprozessor Umgebung besteht aus einer Menge von Jobs $J = \{J_1, J_2, \ldots, J_m\}, m \in \mathbb{N}$, die auf einer Menge von Prozessoren $P = \{P_1, P_2, \ldots, P_n\}, n \in \mathbb{N}$ ausgeführt werden müssen. Jeder Job J_i besteht wiederum aus einer Menge von Tasks mit $T_i = \{T_{i1}, T_{i2}, \ldots, T_{io}\}, o \in \mathbb{N}$. Auf diesen Mengen gilt es zwei Teilprobleme zu lösen: die räumliche Zuordnung der Tasks auf die Prozessoren (*Allocation Problem*), also auf welchem Prozessor ein Task ausgeführt wird und die zeitliche Zuordnung der Tasks auf einem Prozessor (*Scheduling Problem*), also wann und in welcher Reihenfolge die Tasks ausgeführt werden müssen (vgl. Zapata und Alvarez (2005)).

3.1.1 Task Timings

Im Bereich des Schedulings von Real-Time Systemen besitzt jeder Task eine Reihe von Timing Eigenschaften, die das Verhalten und einzuhaltenden Bedingungen der Tasks festlegen. Das wichtigste Timing in einem Real-Time System ist die *Deadline*. Die Deadline legt fest, zu welchem Zeitpunkt ein Task spätestens vollständig ausgeführt sein muss. Eine verpasste Deadline führt zu einem ungültigen Schedule, so dass Deadlines oftmals eine zentrale Rolle in Scheduling-Algorithmen einnehmen. Für jeden Task T_i lassen sich folgende Timings definieren (vgl. Buttazzo (2011)):

Definition 1 (*Release Time*)

Die Release Time r_i definiert den Zeitpunkt, an dem der Task T_i bereit ist, ausgeführt zu werden, dass heißt, alle notwendigen Abhängigkeiten erfüllt sind.

Definition 2 (*Start Time*)

Die Start Time s_i definiert den Zeitpunkt, an dem mit der Ausführung des Tasks T_i begonnen wird.

Definition 3 (*Computation Time*)

Die Computation Time c_i definiert den Zeitraum, den der Task T_i den Prozessor blockiert, bis der Task vollständig ausgeführt wurde.

Definition 4 (*Finishing Time*)

Die Finishing Time f_i definiert den Zeitpunkt, an dem der Task T_i komplett ausgeführt wurde. Er ist gegeben durch:

$$f_i = s_i + c_i \tag{3.1}$$

Definition 5 (*Deadline*)

Die Deadline d_i legt den Zeitpunkt fest, an dem die Ausführung des Tasks T_i spätestens beendet sein muss. Für einen gültigen Schedule muss gelten:

$$f_i \leq d_i \tag{3.2}$$

Definition 6 (*Lateness*)

Die Lateness l_i eines Tasks T_i spiegelt den Zeitraum wieder, der zwischen der Finish Time f_i und der Deadline d_i des Tasks liegt. Sie ist definiert durch:

$$l_i = f_i - d_i \tag{3.3}$$

Der Wert ist negativ, wenn die Finish Time kleiner als die Deadline ist, also wenn der Schedule gültig ist. Dagegen nimmt die Lateness einen positiven Wert an, wenn die Deadline überschritten wurde.

Definition 7 (*Response Time*)

Mit der Response Time R_i bezeichnet man die Zeit, die zwischen der Release Time r_i und der Start Time s_i vergeht. Sie ist definiert durch:

$$R_i = s_i - r_i \tag{3.4}$$

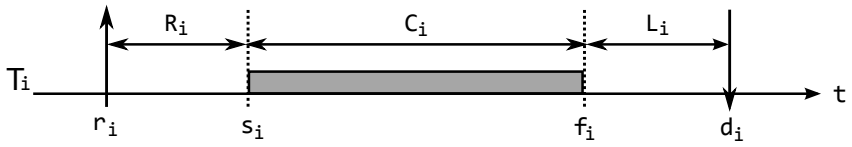

Abbildung 3.1: *Timing Eigenschaften eines Tasks*

Abbildung 3.1 illustriert diese Eigenschaften beispielhaft für einen Task T_i und stellt die einzelnen Timings in einen chronologischen Zusammenhang.

3.1.2 Taskgraphen

In der Schedulingtheorie werden Taskgraphen genutzt, um die Abhängigkeiten der Tasks untereinander abzubilden (Kwok und Ahmad, 1999). In vielen Fällen wird dafür ein gerichteter azyklischer Graph (directed acyclic graph), im folgenden als *DAG* abgekürzt, genutzt. In einem DAG steht jeder Knoten für einen Task und jede Kante für eine Relation zwischen zwei Tasks. Die Abbildung 3.2 demonstriert solch einen Graphen. Definiert wird der DAG durch das Tupel:

$$G = (V, E) \tag{3.5}$$

$V = \{V_1, V_2, \ldots, V_n\}, n \in \mathbb{N}$ ist eine Menge von Knoten und $E = \{E_1, E_2, \ldots, E_m\}, m \in \mathbb{N}$ eine Menge von Kanten mit $E \subseteq V \times V$. Jede Kante wird durch ein Tupel aus zwei Knoten gebildet:

$$E_i = (V_j, V_k); i, j, k \in \mathbb{N} \tag{3.6}$$

Da der Graph gerichtet ist, wird durch dieses Tupel eine Vorgänger und Nachfolger Relation zwischen den Knoten hergestellt. Dies bedeutet, dass $(V_j, V_k) \neq (V_k, V_j)$ ist. V_j ist damit ein Vorgänger von V_j und V_k ein Nachfolger von V_j. Ein Knoten ohne Vorgänger wird *Entry Node* und ein Knoten ohne Nachfolger *Exit Node* genannt. Dabei ist es möglich das ein Knoten mehrere Vorgänger, bzw. Nachfolger hat. Die Vorgänger- und Nachfolgerbeziehungen sind folgendermaßen definiert:

Definition 8 (*Vorgängerbeziehung*)

Für die Vorgänger eines Knotens lassen sich folgende Relationen aufstellen. Die Menge der Vorgänger des Entry Nodes ist die leere Menge:

$$\mathbf{pred}(EntryNode) = \emptyset \tag{3.7}$$

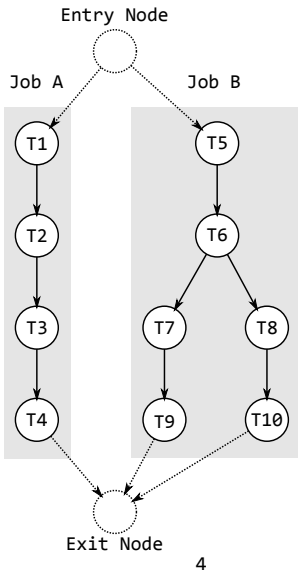

Abbildung 3.2: *Beispiel eines Taskgraphen*

Für die anderen Knoten ergeben sich die Vorgänger durch:

$$\mathbf{pred}(V_i) = \bigcup_{j=1}^{m} \begin{cases} \{V_j\} & \textit{wenn } (V_j, V_i) \in E \\ \emptyset & \textit{sonst} \end{cases} \tag{3.8}$$

Die Nachfolgerrelation lässt sich analog dazu definieren:

Definition 9 (*Nachfolgerbeziehung*)

Für die Nachfolger eines Knotens lassen sich folgende Relationen aufstellen. Die Menge der Nachfolger des Exit Nodes ist die leere Menge:

$$\mathbf{succ}(ExitNode) = \emptyset \tag{3.9}$$

Für die anderen Knoten ergeben sich die Nachfolger durch:

$$\mathbf{succ}(V_i) = \bigcup_{j=1}^{m} \begin{cases} \{V_j\} & \textit{wenn } (V_i, V_j) \in E \\ \emptyset & \textit{sonst} \end{cases} \tag{3.10}$$

Aus diesen Relationen lassen sich auch eine weitere Eigenschaft der Taskgraphen definieren, die Pfade. Ein Pfad ist eine Menge von Knoten, die aus dem Ausgangsknoten und seinen Nachfolgern besteht. Daraus folgt, dass sich für jeden Knoten ein Pfad aus Nachfolgern erstellen lässt. Die Konstruktion eines Pfades folgt gemäß folgenden Definitionen:

Definition 10 (*Pfade im Taskgraphen*)

Der Pfad des Exit Nodes ist der Exit Node selber:

$$\mathbf{path}(ExitNode) = \{V_{Exit}\} \tag{3.11}$$

Die Pfade der anderen Knoten lassen sich nun rekursiv erzeugen:

$$\mathbf{path}(V_i) = \bigcup_{j=1}^{m} \begin{cases} \{V_i\} \cup \mathbf{path}(V_j) & \textit{wenn } (V_i, V_j) \in E \\ \emptyset & \textit{sonst} \end{cases} \tag{3.12}$$

Innerhalb der Knotenmenge eines Pfades liegt wieder eine partielle Ordnung zwischen den Knoten vor. Diese wird wieder durch die Vorgänger- und Nachfolgerbeziehung der Knoten bestimmt, so dass sich die Ordnung der Knoten folgendermaßen notieren lässt:

Definition 11 (*Partielle Ordnung der Knoten innerhalb eines Pfades*)

- *$V_a \rightarrow V_b$ definiert, dass der Knoten V_a ein direkter Vorgänger von V_b ist. Es existiert eine gerichtete Kante von V_a nach V_b: $(V_a, V_b) \in E$.*
- *$V_a \prec V_c$ definiert, dass der Knoten V_a ein Vorgänger von V_c ist. Es existiert ein Pfad von V_a nach V_c:*

$$\mathbf{path}(V_a) = \{V_a, V_b, V_c\} \tag{3.13}$$

$$V_a \rightarrow V_b \rightarrow V_c \tag{3.14}$$

Jedem Pfad innerhalb des Taskgraphen lässt sich eine Länge zuweisen. Dieser Wert bestimmt, wie viel Zeit benötigt wird, um alle Tasks des Pfades auszuführen.

Definition 12 (*Pfadlänge*)

Sei

$$P = \{V_1, \ldots, V_n\}, n \in \mathbb{N}$$

die durch $P = \mathbf{path}(V_1)$ erzeugte Menge der Knoten eines Pfades und

$$C = \{c_1, \ldots, c_n\}$$

die Menge der zu den Knoten gehörenden Computation Times. Dann ergibt sich die Länge eines Pfades durch die Funktion:

$$l(P) = \sum_{i=1}^{n} c_i.$$

3.2 Shop Problem Scheduling

Wie schon zu Beginn dieses Kapitels erwähnt, unterscheidet man beim Scheduling in einer Mehrprozessor-Umgebung zwischen parallelen und dedizierten Systemen. Bei den dedizierten Systemen unterscheidet man weiterhin zwischen drei verschiedenen Arten von *Shop* Problemen, dem *Open Shop*, *Flow Shop* und *Job shop* Scheduling (vgl. Drozdowski (2010)). Gemein haben diese drei Typen, dass sie das Scheduling von Jobs behandeln, deren Taskmenge gleichmächtig zu der Prozessormenge des Problems ist:

Open Shop In dem *Open Shop Scheduling Problem (OSSP)* ist die Reihenfolge, in der die Tasks ausgeführt werden müssen, nicht fest vorgeschrieben. Die einzigen Einschränkungen sind, dass immer nur ein Task pro Job gleichzeitig ausgeführt werden darf und das auf einem Prozessor immer nur ein Task zur gleichen Zeit bearbeitet werden darf (vgl. Blazewicz, J. u. a. (2007)).

Job Shop Das *Job Shop Scheduling Problem (JSSP)* unterscheidet sich von OSSP in der Reihenfolge, in der die Tasks ausgeführt werden müssen. In JSSP müssen die Tasks in einer vordefinierten Reihenfolge ausgeführt werden. Die Abbildung der Tasks auf die Prozessoren kann sich dabei von Job zu Job unterscheiden (vgl. Zalzala und Fleming (1997) und French (1982)).

Flow Shop Das *Flow Shop Scheduling Problem (FSSP)* ist ein Spezialfall des JSSP. Jeder Job besitzt genau einen Task für jeden Prozessor. Die Tasks aller Jobs müssen in der gleichen Reihenfolge auf den Prozessoren ausgeführt werden (vgl. Emmons und Vairaktarakis (2012)).

3.2.1 Disjunktives Graphenmodell für Shop-Probleme

Das disjunktive Graphenmodell ist eine Erweiterung der Taskgraphen und wird gerne genutzt, um das JSSP optisch darzustellen (vgl. Roy und Sussmann (1964), Błażewicz u. a. (1996)

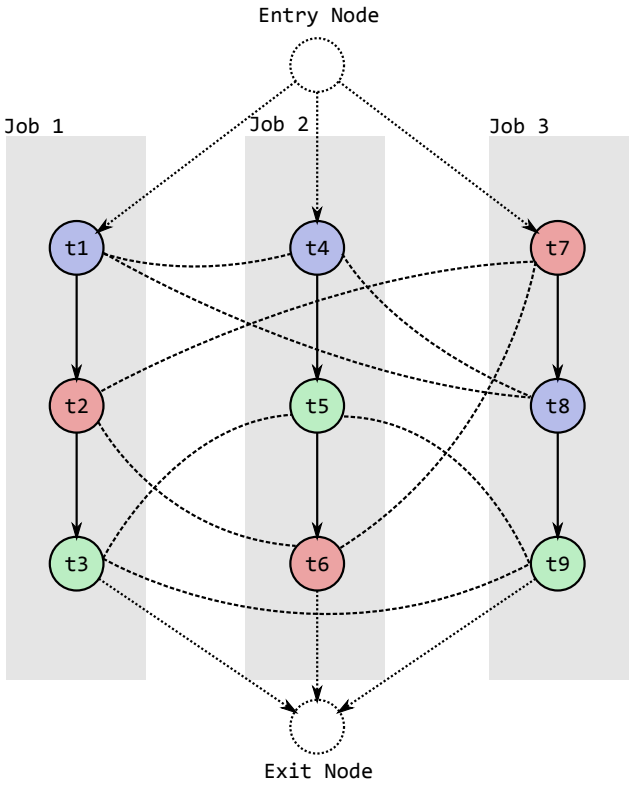

Abbildung 3.3: *Disjunktiver Graph mit 3 Jobs und 3 Prozessoren*

und Mason und Oey (2003)).

Definition 13 (*Disjunktiver Graph*)

Ein disjunktiver Graph $G = (V, E \cup D)$ besteht aus:

- *V ist die Menge von Knoten, die die Tasks repräsentieren. Zusätzlich gibt es einen Entry Node und einen Exit Node, die für den Beginn und das Ende des Schedules stehen.*
- *E ist die Menge von gerichteten (konjunktiven) Kanten, die die Reihenfolge-Relationen zwischen den Tasks repräsentieren.*
- *D ist die Menge von ungerichteten (disjunktiven) Kanten. Jede ungerichtete Kante verbindet zwei Tasks, die auf dem selben Prozessor ausgeführt werden müssen und noch nicht durch eine Kante aus E verbunden sind.*

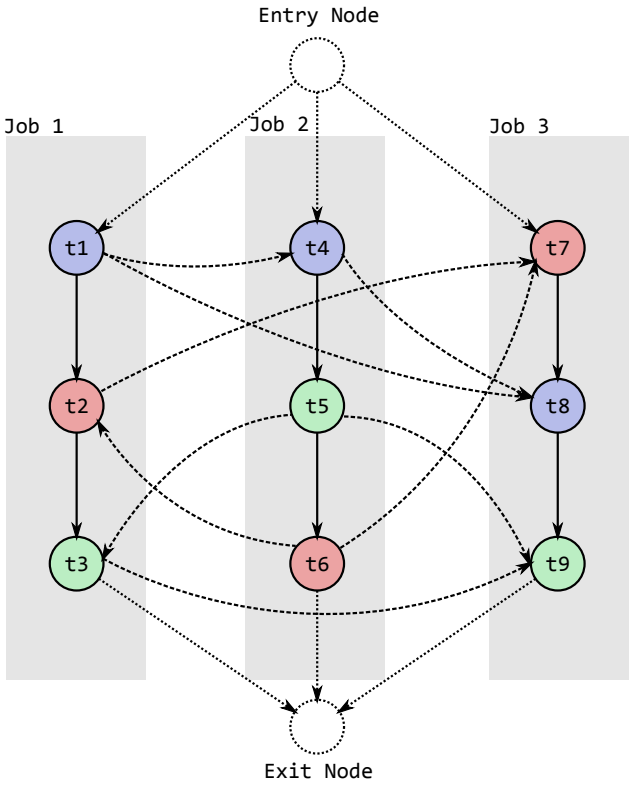

Abbildung 3.4: *Scheduling Lösung in dem disjunktiven Graphenmodell*

Die Abbildung 3.3 demonstriert einen disjunktiven Graphen, der aus drei Jobs besteht. Jeder Job beinhaltet drei Tasks die jeweils auf drei verschiedenen Prozessoren ausgeführt werden müssen. Die konjunktiven Kanten zwischen den Tasks bilden die chronologischen Relationen zwischen den Tasks ab. Für den Job $J_1 = \{t_1, t_2, t_3\}$ gilt damit folgend $t_1 \rightarrow t_2 \rightarrow t_3$. Die disjunktiven Kanten verbinden jeweils die Tasks, die auf dem selben Prozessor ausgeführt werden müssen. Demnach dürfen die Tasks t_1, t_2 und t_3 nicht parallel zur Ausführung gebracht werden.

Um das Shop Scheduling Problem zu lösen, muss für die Tasks, die auf dem selben Prozessor zugeordnet sind, die Reihenfolge der Ausführung bestimmt werden. In dem disjunktiven Graphenmodell bedeutet dies, dass alle ungerichteten Kanten in gerichtete Kanten umgewandelt werden müssen. Dadurch wird für jeden Prozessor eine Sequenz gebildet, in deren Reihenfolge die zugeordneten Tasks Ausgeführt werden müssen. Es gilt zu beachten, dass durch die Umwandlung der ungerichteten zu gerichteten Kanten ein zyklenfreier Graph erzeugt werden muss. Die Zyklenfreiheit ist eine notwendige Bedingung für einen gültigen Schedule.

In der Abbildung 3.4 wird eine mögliche Lösung des Problems aus Bild 3.3 aufgezeigt. Der Graph besteht nur noch aus gerichteten Kanten und ist zyklenfrei. Für die einzelnen Prozessoren ergeben sich damit folgende Sequenzen: $P_1 = (t_1, t_4, t_8)$, $P_2 = (t_6, t_2, t_7)$ und $P_3 = (t_5, t_3, t_9)$.

3.3 Optimierungsziele und Bewertungsfunktionen

Ein weiterer wichtiger Bestandteil des Schedulings sind die Optimierungsziele und Bewertungsfunktionen. Durch das Optimierungsziel wird festgelegt, was mit einem Schedule erreicht werden soll. Dieses Ziel wird zusätzlich zu den Bedingungen definiert, die ein Schedule erfüllen muss, wie zum Beispiel das Einhalten aller Deadlines. Über das Optimierungsziel wird daher kein weiteres Gültigkeitskriterium definiert, sondern ein Merkmal, an dem sich die Qualität eines Schedules messen lässt. Zu jedem Optimierungsziel gehört eine Bewertungsfunktion, im die Qualität eines Schedules numerisch zu erfassen. Die Wahl des Optimierungsziel hängt von der Problemstellung eines Schedules ab und muss unter Betrachtung der Anforderungen gewählt werden. Dabei ist es möglich, zur Beurteilung eines Schedules mehrere Ziele und Bewertungsfunktionen zu wählen, um seinen Schedule auf mehrere Faktoren hin zu optimieren. Im Bereich des Uni- und Mehr-Prozessor Schedulings gibt es zahlreiche Bewertungsfunktionen (vgl. Buttazzo (2011) und Drozdowski (2010)), von denen eine kleine Auswahl aus dem Mehr-Prozessor Scheduling vorgestellt werden soll.

Schedule Länge Beschreibt die Länge eines Schedules, also den Zeitraum, der benötigt wird, um alle Tasks auszuführen. Definiert wird diese Länge durch die späteste *Finishing Time*.

$$F_{max} = max_i(f_i)$$

Das Ziel dieser Optimierung ist es, den F_{max} Wert zu minimieren und die Ausführungszeit des Schedules damit zu minimieren.

Average und Maximal Lateness Die Lateness eines Tasks beschreibt den Zeitraum, um den der Task seine Deadline verpasst hat. Die maximale Lateness betrifft damit den Task, dessen *Finishing Time* am längsten nach seiner *Deadline* liegt.

$$L_{avg} = \frac{\sum_{i=0}^{n} l_i}{n}$$
$$L_{max} = max_i(l_i)$$

Diese Werte können negativ sein, wenn alle Tasks ihre Deadline erfüllen. Der Schedule sollte in die Richtung optimiert werden, so dass die *Lateness* minimiert wird.

Average und Maximal Response Time Die *Response Time* ist der Zeitraum, der zwischen der *Release Time* und seiner *Start Time* liegt. Je größer dieser Wert ist, desto länger muss ein Task auf seine Aktivierung warten.

$$R_{avg} = \frac{\sum_{i=0}^{n} R_i}{n}$$
$$R_{max} = max_i(R_i)$$

Je kleiner diese Werte sind, desto früher werden die Tasks zur Ausführung gebracht, so dass mögliche Ergebnisse früher vorliegen.

3.4 Algorithmen zur Lösung des Scheduling Problems

Echtzeitsysteme basieren darauf, dass alle Ergebnisse der Operationen rechtzeitig zur Verfügung stehen. In der Scheduling-Theorie bedeutet dies nichts anderes, als dass alle Tasks vor dem Erreichen ihrer Deadline ausgeführt und beendet werden müssen. Mathematisch betrachtet, ist das Scheduling von Tasks ein Suchproblem, bei dem mit der Hilfe der Kombinatorik versucht wird, einen gültigen Schedule zu finden. Der Suchraum wird in diesem Fall von einem Baum gebildet, dessen innere Knoten partielle Schedules und dessen Blätter vollständige Schedules darstellen. In den meisten Fällen stellen nicht alle Blätter auch einen gültigen Schedule dar, so dass unter Umständen eine erschöpfende Suche durchgeführt werden muss. Mehrere Untersuchungen haben gezeigt, dass das Finden eines optimalen Schedules schon für einfache Probleme NP-vollständig ist (Hou u. a. (1994), Correa u. a. (1999), Bokhari (1981)). Hinzu kommt, dass es zwar für Systeme mit nur einem Prozessor Verfahren gibt, die einen optimalen Schedule finden (Dertouzos, 1974a), dies aber nicht mehr der Fall ist, wenn es sich um ein Mehrprozessorsystem handelt, in dem die Ausführung der Tasks durch wechselseitigen Ausschluss beschränkt ist (Mok und Dertouzos, 1978).

Aus diesen Gründen nutzt man Heuristiken, um in akzeptabler Zeit eine zulässige Lösung zu finden. Dazu erweitert man das Suchproblem zu einem Optimierungsproblem. Das Optimierungsproblem besteht aus dem Suchraum, einer Bewertungsfunktion und einem Optimierungsziel, welche im Abschnitt 3.3 besprochen wurden. Durch eine Heuristik wird in den meisten Fällen nicht die optimale Lösung gefunden, sondern versucht eine Lösung so nah wie möglich am Optimum zu finden. Zum Einsatz kommen dabei Heuristiken, die aus einem Durchgang bestehen, sowie auch sogenannte Metaheuristiken, die eine Folge von Schritten definieren. Bei der Entwicklung der Heuristiken kommen zahlreiche Techniken zum Einsatz, die von *branch-and-bound*, *integer programming* und *list scheduling*, über *randomization*, bis zu *genetic algorithms*, *simulated annealing* und *tabu search* reichen. Zu diesen Heuristiken gibt es zahlreiche Arbeiten, die die unterschiedlichen Heuristiken vergleichen und bewerten. Für einen Auszug siehe: McCreary u. a. (1994), Zomaya u. a. (1999), Davidović und Crainic (2006), Hwang u. a. (2008) und Jin u. a. (2008).

Algorithm 1 Pseudo Code eines einfachen List Schedulers

```
1: function ListSchedule
2:     for all processing elements pe_i, i = 1,2,...,N_pe do
3:         free_i = 0
4:     end for
5:     schedule P_0 at t = 0
6:     initialize ready process list LsReady with direct successors of P_0
7:     while LsReady is not empty do
8:         p = Head(LsReady)
9:         schedule p at t = free_M(p)
10:        free_M(p) = t + t_p
11:        delete p from LsReady
12:        LsReady = LsReady + processes wich become ready after p
13:    end while
14: end function
```

3.4.1 List Scheduling Heuristiken

Die so genannten List Scheduling Heuristiken werden häufig eingesetzt, um Scheduling Probleme zu lösen, die in einem DAG organisiert sind und dessen Tasks daher asynchrone *Release Times* aufweisen. Diese Heuristiken basieren auf der Idee, die ausführbaren Tasks in einer Liste zu organisieren, die nach den Prioritäten der Tasks sortiert ist. Auf dieser Liste werden nun folgende Schritte wiederholt durchgeführt, bis alle Tasks ausgeführt wurden:

1. Wähle den Task mit der höchsten Priorität aus der Liste.
2. Führe den Task auf dem zugehörigen Prozessor aus.
3. Füge die neu ausführbaren Tasks zur Liste hinzu.
4. Ist die Liste leer, beende den Vorgang. Ansonsten wiederhole ab Schritt 1.

Den passenden Pseudocode zu dieser Methode ist im Block Algorithmus 1 aufgeführt. Der Algorithmus beinhaltet eine Menge von Prozessoren $pe_i, i = 1,2,\ldots,N_{pe}$, die zu Beginn ($t = 0$) der Initialisierung alle im *Idle* Zustand sind ($free_{pe_i} = 0$), der Menge aller Prozesse P und einer Liste $LsReady$, die in der Initialisierung mit allen Tasks gefüllt wird, die zum Systemstart direkt ausführbar sind, also Nachfolger der Quelle des Taskgraphen sind. Ist das System initialisiert, wird das erste Element aus der $LsReady$ Liste entnommen ($p = Head(LsReady)$) und dem entsprechendem Prozessor zugeordnet, welcher über die Funktion $M(p)$ gefunden wird. p wird nun zu dem Zeitpunkt ausgeführt, an dem der Prozessor im *Idle* Zustand ist ($free_{M(p)}$) und die Zeit des Prozessors aktualisiert ($free_{M(p)} = t + t_p$). Der aktuell gewählte Prozess wird nun von der $LsReady$ Liste entfernt. Hinzu kommen

die Prozesse, die nach der Ausführung des Prozesses bereit werden. Dieser Vorgang wird so lange wiederholt, bis die $LsReady$ Liste leer ist, also alle Prozesse einem Prozessor zugeordnet wurden und einen Startzeitpunkt haben.

Eine ausführlichere Beschreibung dieses Verfahrens findet sich in Eles u. a. (2000). Das Festlegen der Prioritäten erfolgt zu Beginn des Schedulings und kann nach unterschiedlichen Kriterien erfolgen. Eine Auswahl und Bewertung von unterschiedlichen List Scheduling Heuristiken ist in Adam u. a. (1974), Liao u. a. (1994) und Liu u. a. (2005) nachzulesen. Im Rahmen dieser Arbeit wird auf die folgenden Heuristiken zurückgegriffen:

First Come First Served

Bei diesem Algorithmus werden alle Tasks in der Reihenfolge ihrer Ankunft ausgeführt. Eine Sortierung nach einem definierten Priorität findet nicht statt (vgl Tanenbaum (2007)). Der Vorteil dieser Heuristik ist, dass sie sich sehr einfach implementieren lässt.

Earliest Deadline First (EDF)

Der Earliest Deadline First Algorithmus wird häufig in Echtzeit Betriebssystemen verwendet und ist optimal für präemptive Uni-Prozessoren (vgl. Dertouzos (1974b)). Doch auch im Bereich des Multi-Prozessor Schedulings wird diese Heuristik eingesetzt (vgl. Thakor und Shah (2011)). EDF sortiert die Prioritätenliste aufsteigend nach Deadline, so dass der Task, der am nächsten an seiner Deadline liegt, als erstes ausgeführt wird.

Longest Job First (LDF) und Shortest Job First (SJF)

Die Longest und Shortest Job First Algorithmen orientieren sich für die Ordnung innerhalb der Prioritätenliste anhand der in Abschnitt 3.1.2 definierten Pfadlänge im Taskgraphen. Der Pfad eines Tasks besteht aus sich selbst und all seinen Nachfolgern. Je nach Heuristik wird jeweils der Task aus der Liste gewählt, der den längsten oder kürzesten Pfad aufweist.

3.4.2 Metaheuristiken

Unter Metaheuristiken versteht man Algorithmen, die eingesetzt werden, um Optimierungsprobleme näherungsweise zu lösen. Metaheuristiken definieren eine Folge von Schritten, die auf unterschiedliche Problemstellungen angewandt werden können. Die einzelnen Schritte bestehen oftmals wieder aus normalen Heuristiken, die speziell für das Problem implementiert werden. Metaheuristiken finden ihre Anwendung bei besonders schweren kombinatorischen

Optimierungsproblemen, für die keine andere Lösung bekannt ist. Der Erfolg dieser Algorithmen hängt stark von der Implementierung der einzelnen Schritte ab, so dass die gefundene Lösung nicht optimal sein muss. Das Grundprinzip vieler Metaheuristiken basiert auf der lokalen Suche:

1. Finde eine Lösung *L*.
2. Erstelle eine *Nachbarschaft* von zu *L* ähnlichen Lösungen.
3. Suche die *Nachbarschaft* nach der besten Lösung ab.

Metaheuristiken werden oft zur Lösung von Scheduling-Problemen eingesetzt. Verwendete Verfahren sind dabei unter anderem die *Tabu Suche* (vgl. Ponsich und Coello Coello (2013) und Vilcot und Billaut (2011)), *Ameisenalgorithmen* (vgl. Ferrandi u. a. (2010) und Ying und Lin (2006)) und *simulierte Abkühlung* (vgl. Wang u. a. (2011) und Zorin und Kostenko (2014)). Da in dieser Arbeit nicht auf Metaheuristiken zurückgegriffen wird, wird auf eine detaillierte Beschreibung dieser Algorithmen verzichtet. Es ist allerdings wichtig, die Existenz dieser Verfahren im Hinterkopf zu behalten, da sie eine Möglichkeit darstellen, den in dieser Arbeit beschriebenen Lösungsweg für komplexere Probleme zu optimieren.

3.5 Verwandte Arbeiten

In diesem Abschnitt soll eine Auswahl von verwandten Arbeiten vorgestellt werden, die sich mit den für diese Arbeit relevanten Themen des Scheduling beschäftigen. Eine gute Basis bietet das Scheduling von periodischen Tasks, welches in der Vergangenheit ausführlich erforscht wurde (vgl. Ramamritham (1995), Hou und Shin (1997) und Peng u. a. (1997)). In diesen Arbeiten wird detailliert auf das Scheduling von periodischen Tasks in verteilten Echtzeit-Systemen mit Kommunikationsaufwand und multiplen Abhängigkeiten eingegangen. Auf der anderen Seite fehlt in diesen Texten der Bezug zu aperiodischen Tasks, welche bei der Betrachtung eines Time-Triggered Ethernet Netzwerks unerlässlich sind. Weiterhin sind die verwendeten Optimierungsziele, die häufig auf klassischen Metriken basieren, nicht übertragbar.

In Hinsicht auf aperiodische Task geht die Arbeit *Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems* (vgl. Fohler (1995)) weiter auf das Problem ein und beschreibt eine Lösung für ein verteiltes Echtzeit-System mit periodischen und aperiodischen Tasks. Es werden *Time Division Multiple Access* Verfahren genutzt, um die verarbeitenden und Kommunizierenden Knoten des Systems zu modellieren, für welches zunächst ein statischer Schedule der periodischen Tasks berechnet wird. Anhand dieses Schedules werden nun die freien Kapazitäten für jeden Knoten berechnet, so dass zur Laufzeit des Systems entschieden werden kann, ob als nächstes ein periodischer oder aperiodischer Task ausgeführt werden soll. Obwohl das beschriebene Problem in vielen

Aspekten Ähnlichkeit aufweist, ist die Lösung nicht auf ein Time-Triggered Ethernet System anwendbar, da TT-Ethernet nur statische Schedules für TT-Nachrichten erlaubt und zur Laufzeit nicht mehr entschieden werden kann, ob zu einem bestimmten Zeitpunkt eine TT- oder RC-Nachricht versendet wird. Um trotzdem eine faire Übertragung der RC-Nachrichten zu ermöglichen, muss bereits zur Designzeit des Systems der Schedule so angelegt werden, dass zu jedem Zeitpunkt genug Kapazitäten für die aperiodischen RC-Nachrichten vorhanden sind.

Bei den Ethernet Technologien finden sich verwandte Arbeiten im Bereich des *AFDX* (vgl. Aeronautical Radio Incorporated (2009b)) und *AVB* (vgl. IEEE AVB Task Group (2014)) Protokolls. Das *AFDX* Protokoll ist TT-Ethernet sehr ähnlich, beschränkt sich jedoch auf ereignisbasierte Nachrichten. So werden unter anderem Ansätze vorgeschlagen, dass auftretende Scheduling Problem mit *Earliest Deadline First* (vgl. Zhang u. a. (2011)) und *Deficit Round Robin* (vgl. Hua und Liu (2011)) Heuristiken zu lösen. Die Arbeit *Modeling of Ethernet AVB Networks for Worst-Case Timing Analysis* (vgl. Diemer u. a. (2012)) erweist sich als interessant, da eine Transformation des Netzwerks in ein Modell vorgeschlagen wird, um das Scheduling von AVB-Systemen analysieren zu können. Ein Ansatz, der in ähnlicher Form in dieser Arbeit für das TT-Ethernet Protokoll durchgeführt wird.

Da das Gebiet des Time-Triggered Ethernet Scheduling noch relativ neu ist, gibt es zu diesem Thema erst wenig Ergebnisse. In der Arbeit *An Evaluation of SMT-based Schedule Synthesis For Time-Triggered Multi-Hop Networks* (vgl. Steiner (2010)) wird eine Lösung auf Basis der *Satisfiability Modulo Theories* (SMT) vorgeschlagen. Es wird das bereits existierende SMT-Werkzeug *Yices* (vgl. Dutertre und De Moura (2006)) genutzt, um einen Schedule für alle TT-Nachrichten zu erzeugen. Es wird gezeigt, dass es mit dieser Methode möglich ist, mehrere Zehntausend TT-Nachrichten zu planen und eine Auslastung von bis zu 90% der Netzwerk-Kapazität erreicht wird. In dieser Betrachtung fehlt jedoch das Einbeziehen der RC- und BE-Nachrichten, welches gerade bei einer hohen Auslastung mit TT-Nachrichten sehr wichtig ist.
Diese Lücke wird versucht in der Arbeit *Synthesis of Communication Schedules for TTEthernet-based Mixed-criticality Systems* (vgl. Tamas-Selicean u. a. (2012)) zu schließen. Anstatt sich nur auf die TT-Nachrichten zu konzentrieren, wird auch bewusst auf die RC-Nachrichten eingegangen, indem anhand des *End-to-end delay* überprüft wird, ob ein gültiger Schedule erzeugt wurde. Um dies sicher zu stellen, wird eine *Tabu Search* Heuristik verwendet, um einen auf dieses Ziel hin optimierten Schedule zu finden. Der verfolgte Ansatz geht damit in eine ähnliche Richtung, wie sie in dieser Arbeit vorgeschlagen wird.

Ein weiterer Aspekt dieser Arbeit ist die Implementierung eines Frameworks, dass es ermöglicht, unterschiedliche Methoden zur Erzeugung eines Schedules zu nutzen. In der Literatur finden sich zu diesem Thema eine Reihe von Ansätzen in unterschiedlichen Scheduling Gebieten. In *SmartGRID: A fully decentralized Grid Scheduling Framework supported by Swarm Intelligence* (vgl. Huang u. a. (2008)) wird ein Framework beschrieben, dass aus mehreren

entkoppelten Schichten besteht, um ein generelles und modular aufgebautes Werkzeug zum Ressourcen Management im Grid-Computing Bereich bereit zu stellen. Es werden Schwarmintelligenz Algorithmen genutzt, um ungenutzte Ressourcen zu entdecken, während ein Plugin System es ermöglicht, diese Ressourcen mit unterschiedlichen Strategien neu zu verteilen.
Auch beim klassischen CPU Scheduling befinden sich Frameworks im Einsatz. So wird in der Arbeit *ExSched: An External CPU Scheduler Framework for Real-Time Systems* (vgl. Asberg u. a. (2012)) eine Software beschrieben, die den Einsatz von unterschiedlichen Scheduling-Strategien für unterschiedliche Betriebssysteme erlaubt. Hierfür ist kein Eingriff in das Betriebssystem erforderlich, sondern der Zugriff erfolgt über ein einheitliches Interface. Dies erlaubt das Implementieren von neuen Algorithmen als externe Erweiterungen. In der Arbeit wurden beispielhaft die Strategien *Fixed-Priority Scheduling* und *Earliest Deadline First* implementiert.
Für das Scheduling von periodischen real-time Tasks empfiehlt sich die Arbeit *Compositional Real-Time Scheduling Framework with Periodic Model* (vgl. Shin und Lee (2008)), in der formale Abstraktionen eingesetzt werden, um zusammengesetzte real-time Scheduling Probleme zu lösen. Das Framework basiert auf einem *periodic interface* und einem *periodic resource model*, um den Zugriff auf einzelne Komponenten zu beschreiben. Schließlich werden diese Probleme mit den Heuristiken *Earliest Deadline First* und *Rate Monotonic Scheduling* gelöst.
All diesen Arbeiten ist gemein, dass Abstraktionen und Schnittstellen genutzt werden, um einen generellen Zugriff zu erlauben und so eine Möglichkeit schaffen, unterschiedliche Lösungsstrategien zu implementieren, bzw. um eine leichte Erweiterbarkeit zu bestehenden Methoden einzuführen. Dies sind Ansätze, die in ähnlicher Form auch in dieser Arbeit umgesetzt werden. Leider verbietet sich der Einsatz der genannten Lösungen, da das TT-Ethernet Protokoll nicht zu den verwendeten Modellen kompatibel ist.

4 Modellierung des Scheduling-Problems

Um das Scheduling Problem der TDMA Kommunikation in einem TT-Ethernet Netzwerk zu lösen, kann das System transformiert werden, um bekannte Methoden zur Problemlösung aus der Schedulingtheorie anwenden zu können. Diese Transformation des Netzwerks zu einem formalen Modell soll in diesem Kapitel erläutert werden.

Die Abschnitte sind dabei wie folgt aufgebaut. Zuerst wird das TT-Ethernet Netzwerk und seine Bestandteile formal definiert. Nach der Definition werden Beispiele vorgestellt, an denen die Transformation und Modellierung zu Demonstrationszwecken durchgeführt wird. Das Beispiel dient damit als Eingangsparameter für die Transformation. Der zweite Abschnitt beschreibt die eigentliche Transformation. Dabei wird getrennt auf die physikalischen und logischen Komponenten des Systems eingegangen. Die beiden folgenden Abschnitte beschäftigt sich mit dem Job- und Task-Modell und den Deadline Anforderungen der Tasks. Danach wird der Bezug des Problems zum Job Shop Scheduling Problem hergestellt und das Problem formal beschrieben. Abschließend wird noch auf die Anforderungen eingegangen, die eine Lösung des Problems erfüllen muss.

4.1 Definition des Netzwerks

Ein TT-Ethernet Netzwerk und seine unterschiedlichen Komponenten sind die Ausgangsbasis für das Scheduling. Dazu gehören nicht nur die physikalischen Komponenten des Netzwerks, die Hardware, sondern auch die Software Bestandteile des Systems. Die Hardware besteht aus den Steuergeräten, Switches und Links, während auf Softwareseite Applikationen und Nachrichten existieren. Die formale Definition für ein Netzwerk sei:

Definition 14 (*TT-Ethernet Netzwerk*)

Ein TT-Ethernet Netzwerk X wird definiert durch das Tupel:

$$X = (N, S, L, A, M)$$

- *Einer Menge von Steuergeräten* $N = \{n_1, \ldots, n_m\}, m \in \mathbb{N}$
- *Einer Menge von Switches* $S = \{s_1, \ldots, s_n\}, n \in \mathbb{N}$

- *Einer Menge von Links* $L = \{l_1, \ldots, l_o\}, o \in \mathbb{N}$
- *Einer Menge von Applikationen* $A = \{a_1, \ldots, a_p\}, p \in \mathbb{N}$
- *Einer Menge von Nachrichten* $M = \{m_1, \ldots, m_q\}, q \in \mathbb{N}$

Die einzelnen Komponenten sind folgendermaßen definiert:

Definition 15 (*Steuergerät*)

Ein Steuergerät n_m dient als Endgerät in einem Netzwerk und ist immer über einen Link mit einem Switch verbunden. Auf einem Steuergerät können beliebig Applikationen ausgeführt werden.

Definition 16 (*Switch*)

Ein Switch s_n ist ein zentrales Element des Netzwerks und verbindet mehrere Steuergeräte untereinander. Es ist möglich, dass ein Switch mit einem anderen Switch verbunden ist.

Definition 17 (*Link*)

Ein Link l_o verbindet ein Steuergerät mit einem Switch oder zwei Switches untereinander. Man muss zwischen einem physikalischen und einem logischen Link unterscheiden. Da die physikalischen Links in einem Vollduplex Modus operieren, kann man jeden physikalischen Link durch zwei logische Links ersetzen. Jeweils einen logischen Link für jede Übertragungsrichtung.

Definition 18 (*Applikation*)

Eine Applikation ist ein Programm, das auf einem Steuergerät ausgeführt. Applikationen können als Sender und/oder Empfänger fungieren. Ein Sender erzeugt eine Nachricht während ein Empfänger die Nachricht konsumiert. Es ist möglich, dass eine Applikation mehrere Nachrichten erzeugt und/oder konsumiert. Eine Applikation a_p wird durch folgendes Tupel definiert:

$$a_p = (n, c, SendM, RecvM)$$

mit

- *n ist das Steuergerät, auf dem die Applikation ausgeführt wird, $n \in N$.*
- *c ist die Computation Time der Applikation in Sekunden, $c \in \mathbb{N}$.*
- *$SendM$ ist die Menge der Nachrichten, die die Applikation erzeugt. Die Menge kann leer sein. $SendM \subset M$ oder $SendM = \emptyset$.*

- *$RecvM$ ist die Menge der Nachrichten, die die Applikation konsumiert. Die Menge kann leer sein. $RecvM \subset M$ oder $RecvM = \emptyset$.*

Definition 19 (*Nachricht*)

Nachrichten dienen zum Informationsaustausch zwischen Applikationen. Sie werden zwischen den Steuergeräten über Links und Switches übertragen. Nachrichten sind eng mit den Eigenschaften der virtuellen Links des TT-Netzwerks verknüpft. Eine Nachricht m_q ist folgend definiert:

$$m_q = (size, period, sendA, RescA)$$

mit

- *$size$ definiert die Größe der Nachricht in Byte.*
- *$period$ definiert die Periodenlänge. Eine Nachricht wird alle $period$ Sekunden verschickt.*
- *$sendA$ ist die Applikation, von der diese Nachricht gesendet wird: $sendA \in A$.*
- *$RescA$ ist die Menge der Applikationen, die diese Nachricht empfangen: $ResvA \subset A$.*

4.1.1 Beispielsystem

Das Beispielsystem aus diesem Abschnitt wird durch das gesamte Kapitel hindurch genutzt, um die Transformation und Modellierung zu veranschaulichen und zu überprüfen. In der Abbildung 4.1 ist ein kleines Time-Triggered Ethernet Netzwerk aus zwei Switches zu sehen, an die jeweils zwei Steuergeräte angeschlossen sind. Verbunden werden die Komponenten durch Links, die im Vollduplex Modus operieren. In der Abbildung sind daher für jeden physikalischen Link zusätzlich zwei logische Links dargestellt. Einen Link für jede Übertragungsrichtung.

Auf diesem Netzwerk aufbauend werden drei unterschiedliche Steuerkreise implementiert, aus deren Kombination sich beliebig große und Komplexe Steuerkreise bilden lassen. Sie bilden damit die Basisbausteine für größere Netzwerke. Die Abbildung 4.2 demonstriert einen einfachen Steuerkreis aus einem Sender und einem Empfänger. Als Sender fungiert die Applikation A_1, die auf dem Gerät N_1 ausgeführt wird. Der Empfänger ist die Applikation A_2, die auf dem Gerät N_3 läuft. Übertragen wird die Nachricht M_1, die zunächst von dem Gerät N_1 über den Link L_1 an den Switch S_1, dann über Link L_5 an Switch S_2 und schließlich über den Link L_8 an das Gerät N_3 gesendet wird. Entsprechend sind die beiden anderen

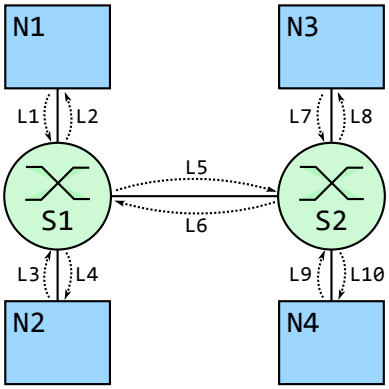

Abbildung 4.1: *Beispielnetzwerk zur Modellierung*

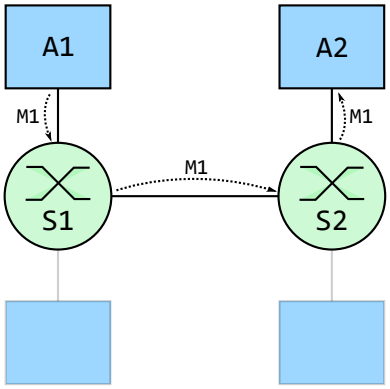

Abbildung 4.2: *Steuerkreis 1*

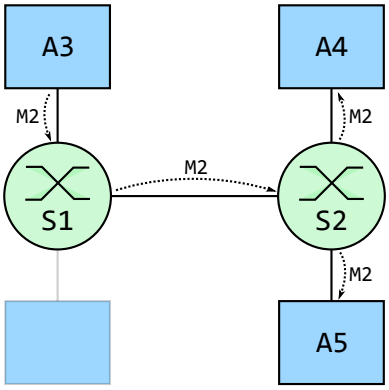

Abbildung 4.3: *Steuerkreis 2*

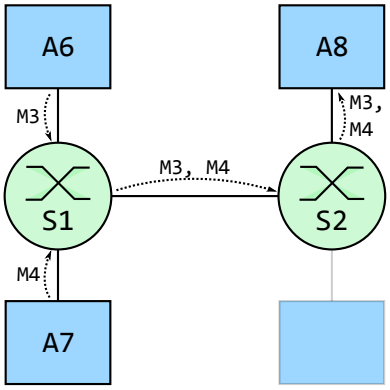

Abbildung 4.4: *Steuerkreis 3*

Steuerkreise aufgebaut. Der zweite Steuerkreis in Bild 4.3 besteht aus einem Sender und zwei Empfängern. Das letzte Bild 4.4 zeigt einen Steuerkreis mit zwei Sendern, die an einen Empfänger senden.

Für die Applikationen und Nachrichten sind folgende Werte festgelegt:

Applikation	n	c	SendM	RecvM
a_1	n_1	100µs	$\{m_1\}$	$\emptyset$
a_2	n_3	200µs	$\emptyset$	$\{m_1\}$
a_3	n_1	300µs	$\{m_2\}$	$\emptyset$
a_4	n_3	300µs	$\emptyset$	$\{m_2\}$
a_5	n_4	200µs	$\emptyset$	$\{m_2\}$
a_6	n_1	400µs	$\{m_3\}$	$\emptyset$
a_7	n_2	100µs	$\{m_4\}$	$\emptyset$
a_8	n_3	100µs	$\emptyset$	$\{m_3, m_4\}$

Nachricht	size	period	sendA	RescA
m_1	200 B	5 ms	a_1	$\{a_2\}$
m_2	1500 B	10 ms	a_3	$\{a_4, a_5\}$
m_3	20 B	4 ms	a_6	$\{a_8\}$
m_4	100 B	100 ms	a_7	$\{a_8\}$

4.2 TT-Ethernet Transformation

Der erste Schritt, um ein Scheduling der TT-Nachrichten in dem Netzwerk durchzuführen, ist eine Transformation des Ausgangssystems. Durch diese Transformation wird jedes Element des Netzwerks auf ein Element der klassischen Schedulingtheorie abgebildet. Diese Abbildung ermöglicht es, Strategien und Lösungen aus dem Bereich des Multiprozessor-Scheduling auf das Netzwerk anzuwenden. Die Abbildung wird definiert durch die Funktion:

$$\phi : X \to Y \tag{4.1}$$

Die Funktion f bildet jedes Element aus dem Definitionsbereich X in den Wertebereich Y ab. X repräsentiert das Netzwerk und wurde in Abschnitt 4.1 definiert. Y steht für die Elemente aus dem Multiprozessor-Scheduling und wird folgendermaßen definiert:

Definition 20 (*Wertebereich der Abbildung*)

Der Wertebereich für die Abbildung eines TT-Ethernet Netzwerks ist gegeben durch das Tupel:

$$Y = (P, T)$$

Mit den Objekten:

- *Einer Menge von Prozessoren* $P = \{p_1, \ldots, p_r\}, r \in \mathbb{N}$
- *Einer Menge von Tasks* $T = \{t_1, \ldots, t_s\}, s \in \mathbb{N}$

Die fünf unterschiedlichen Mengen des Definitionsbereichs werden auf nur noch zwei unterschiedliche Mengen im Wertebereich reduziert. Nach der Abbildung existiert das Modell vorerst nur noch aus Prozessoren und Tasks. Die Abbildungsvorschrift ist so definiert, dass sämtliche Hardware-Komponenten des Netzwerks auf die Prozessoren abgebildet werden und die Software-Elemente als Tasks interpretiert werden.

Jedes Steuergerät und jeder Link und jeder Switch aus unserem Netzwerkmodell bildet einen Prozessor in unserem Schedulingmodell. Es kommen dabei zwei verschiedene Typen von Prozessoren zum Einsatz. Sequenzielle und parallele Prozessoren. Steuergeräte und Links werden auf sequenzielle und Switches auf parallele Prozessoren abgebildet. Auf den Steuergeräten und Links kann immer nur ein Task zur selben Zeit ausgeführt werden, während auf einem Switch mehrere Tasks parallel bearbeitet werden können. Weiterhin ist bei dieser Abbildung zu beachten, dass die logischen Links und nicht die physikalischen Links betrachtet werden. Für jede Übertragungsrichtung eines physikalischen Links wird ein Prozessor benötigt, um dem Umstand gerecht zu werden, dass in beide Richtungen parallel übertragen werden kann.

Die Tasks werden in unserem Schedulingmodell aus den Applikationen und Nachrichten gebildet. Jede Applikation ist genau ein Task. Bei den Nachrichten ist die Abbildung komplexer und hängt von der Topologie des Netzwerks und dem Routing der Nachrichten ab. Diese Punkte werden im Abschnitt 4.5 über das Task Modell näher beleuchtet. Allgemein gilt, dass für jeden Link und für jeden Switch über den die Nachrichten übertragen werden muss, im Schedulingmodell ein Task erzeugt werden muss.

Mit diesen Eigenschaften lässt sich die Abbildung folgendermaßen definieren:

Definition 21 (*Abbildungsfunktion und Vorschrift*)

Durch den Definitions- und Wertebereich ergibt sich die Funktion zu:

$$\phi : (N, S, L, A, M) \rightarrow (P, T)$$

Mit folgender Abbildungsvorschrift:

$$\phi(x) = \begin{cases} y : y \in P & \textit{wenn } x \in N \cup S \cup L \\ z : z \in T & \textit{wenn } x \in A \cup M \end{cases}$$

4.2.1 Transformation des Beispiels

Um diese Abbildung zu veranschaulichen, soll das Verfahren an dem Beispiel aus Abschnitt 4.1.1 durchgeführt werden. Die Details der Abbildung in Bezug auf die Abhängigkeiten der Tasks und das Erstellen von mehreren Tasks für eine Nachricht folgen im Abschnitt 4.5. Für den Definitionsbereich $X = (N, S, L, A, M)$ der Abbildung des Beispiels ergibt sich:

$$
\begin{aligned}
N &= (n_1, n_2, n_3, n_4) \\
S &= (s_1, s_2) \\
L &= (l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8, l_9, l_{10}) \\
A &= (a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8) \\
M &= (m_1, m_2, m_3, m_4)
\end{aligned}
$$

Nach dem Anwenden der Abbildungsvorschrift $\phi : (N, S, L, A, M) \rightarrow (P, T)$ ergibt sich die Menge der Prozessoren P und Tasks T zu:

$$
\begin{aligned}
P &= (n_1, n_2, n_3, n_4, s_1, s_2, l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8, l_9, l_{10}) \\
T &= (a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, m_1, m_2, m_3, m_4)
\end{aligned}
$$

4.3 Prozessoren, Tasks und Computation Time im Modell

Nach der Transformation besteht unser Modell nur noch aus Prozessoren und Tasks. Eines der wichtigsten Eigenschaften der Tasks, die Computation Time, wurde dabei noch nicht betrachtet. Die Computation Time definiert den Zeitraum, wie lange ein Task den Prozessor blockiert. Diese Zeit steht in Abhängigkeit zu dem Prozessor, auf dem Task ausgeführt wird und zu dem Ursprung des Tasks. Dabei treten unterschiedliche Kombinationen aus Prozessoren und Tasks auf, deren Verhalten in Bezug auf die Computation Time in diesem Abschnitt näher erläutert werden soll.

Steuergeräte und Applikationen
Steuergeräte sind in dem Modell homogene Prozessoren ohne weitere Eigenschaften. Das heißt, dass der Prozessor keinen Einfluss auf die Ausführungszeit des Tasks hat und die Applikation den Prozessor exakt die Zeit blockiert, die über die Computation Time der Applikation festgelegt wurde.

Switches und Nachrichten
Im Gegensatz zu den Steuergeräten stellen Switches Prozessoren da, die mehrere Tasks parallel ausführen können. Jede eingehende Nachricht wird als Task betrachtet, der sofort zur Ausführung gebracht wird. Dies entspricht dem Kopieren der Nachricht aus dem Puffer des empfangenden Ports in den Puffer des Ports, über den die Nachricht weitergeschickt wird.

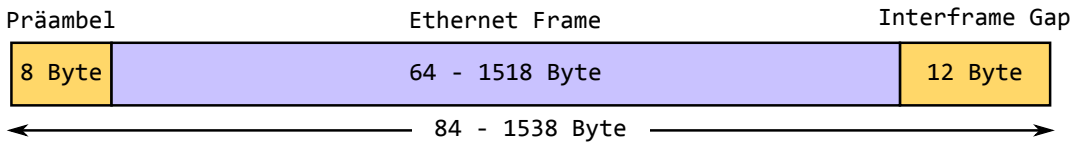

Abbildung 4.5: *Ethernet Paket einer Nachricht*

Die Zeit, die der Kopiervorgang benötigt ist nicht abhängig von der Größe der Nachricht, sondern wird für jeden Switch fest konfiguriert. Ein Min-Size Frame wird genau so lange verzögert, wie ein Max-Size Frame. Es gilt zu beachten, dass die Nachricht auf dem Link des Ausgangs-Ports nicht direkt im Anschluss des Kopierens versendet werden muss, sondern für einen beliebigen Zeitraum gepuffert werden kann.

Links und Nachrichten
Die Computation Time der Tasks, die als Nachrichten auf den Links ausgeführt werden, hängt von der Übertragungsgeschwindigkeit des Netzwerks und der Menge der zu übertragenden Daten ab. In einem TT-Ethernet Netzwerk ist eine Datenrate von 100 Mbit/s oder 1 Gbit/s üblich. Die Datenmenge, das Ethernet Paket, setzt sich zusammen aus dem Ethernet Frame (vgl. 2.2) und den nötigen Protokolldaten des Ethernet Netzwerkprotokolls. Diese bestehen aus der Präambel (8 Byte) und dem Interframe Gap, der aus 96 Bitzeiten (12 Bytes) besteht. Damit müssen neben den Nutzdaten insgesamt 38 Byte an Metadaten übertragen werden. Die Abbildung 4.5 illustriert die komplette Datenmenge, die zur Berechnung der Computation Time des Tasks hinzugezogen wird. Die Computation Time für ein minimales und maximales Paket in einem 100 Mbit/s und in einem 1 Gbit/s Netzwerk ist in der Tabelle 4.1 dargestellt.

Datenrate	84 Byte	1538 Byte
100 Mbit/s	6,72 µs	123,04 µs
1 Gbit/s	672 ns	12,3 µs

Tabelle 4.1: *Computation Time für Ethernet Pakete*

4.3.1 Computation Time im Beispiel

Nachdem die Berechnungsvorschriften der Computation Time der Tasks festgelegt wurde, soll diese nun zur Veranschaulichung am Beispiel durchgeführt werden. Die Computation Time für die Tasks, die ihren Ursprung in den Applikationen haben, kann direkt übernommen werden und findet sich in Tabelle 4.2 wieder.

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
100 µs	200 µs	300 µs	300 µs	200 µs	400 µs	100 µs	100 µs

Tabelle 4.2: *Computation Time der Beispiel-Tasks*

Für die Berechnung der Computation Time c einer Nachricht müssen die Nutzdaten und Metadaten addiert werden und anschließend durch die Übertragungsrate des Netzwerks dividiert werden:

$$c = \frac{\text{Nutzdaten} + \text{Metadaten}}{\text{Datenrate}}$$

Es wird davon ausgegangen, dass das Netzwerk des Beispiels mit einer Rate von 100 Mbit/s überträgt. Die Computation Time der Nachrichten-Tasks ergibt sich dann zu:

m_1	m_2	m_3	m_4
19,04 µs	123,04 µs	6,72 µs	11,04 µs

Tabelle 4.3: *Computation Time der Beispiel-Tasks*

Die Computation Time der Nachrichten-Tasks, die auf einem Switch-Prozessor ausgeführt werden, wird durch die Verzögerung beim Kopiervorgang innerhalb des Switches definiert. Diese wird in unserem Beispiel mit 200 µs definiert.

4.4 Routing der Nachrichten

Bevor das Job und Task Modell im Detail definiert werden kann, muss die Routing-Strategie der Nachrichten im Netzwerk festgelegt werden. Im Falle des TT-Ethernet Protokolls kommt für TT-Nachrichten nur ein statisches Routing in Frage, da bereits zur Designzeit festgelegt werden muss, welchen Weg eine Nachricht nimmt. Die erzeugten Routen stellen die Basis für den Abhängigkeitsgraphen der Tasks, der für das Scheduling benötigt wird.

Im Rahmen dieser Arbeit wird das Routing anhand des kürzesten Wegs gewählt. Zu diesem Zweck wird das Netzwerk als Graph dargestellt und mit Hilfe des Dijkstra-Algorithmus (vgl. Dijkstra (1959)) der kürzeste Weg zwischen den Sendern und Empfängern ermittelt. Steuergeräte und Switches werden in solch einem Graph als Knoten dargestellt, während die Links als gerichtete Kanten repräsentiert werden, so dass alle Prozessoren in dem Graphen abgebildet werden. Neben den Kanten wird deren Länge angegeben, die für alle Kanten den Wert Eins hat. Eine Route beinhaltet alle Kanten und Knoten einer Nachricht, bis auf die Sende- und Empfangsknoten. Dadurch ergeben sich alle Prozessoren in der entsprechenden Reihenfolge, die für die Übertragung der Nachricht benötigt werden. Zu beachten ist, dass eine Nachricht mehrere Empfänger haben kann und daher auch die Route zu mehreren Zielen

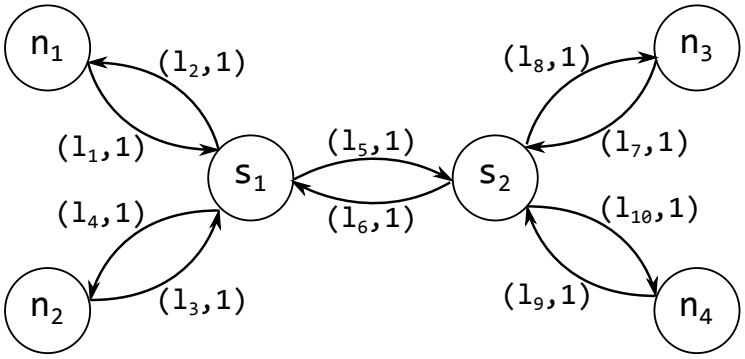

Abbildung 4.6: *Netzwerk Graph mit Kantenlängen*

führen kann. In diesem Fall erzeugt man für jeden Empfänger die Route separat und fügt die Routen hinterher vom Sender beginnend zusammen, so lange die gemeinsamen Teilstrecken identisch sind.

4.4.1 Routing im Beispielnetzwerk

Der Graph für das Beispielnetzwerk ist in der Abbildung 4.6 zu sehen. Wendet man nun den Dijkstra-Algorithmus auf diesen Graphen an, ergeben sich für die Nachrichten aus den Steuerkreisen folgende Routen:

$$
\begin{aligned}
m_1 &= l_1, s_1, l_5, s_2, l_8 \\
m_2 &= l_1, s_1, l_5, s_2, \begin{cases} l_8 \\ l_{10} \end{cases} \\
m_3 &= l_1, s_1, l_5, s_2, l_8 \\
m_4 &= l_3, s_1, l_5, s_2, l_8
\end{aligned}
$$

Die Nachricht m_1 wird demnach über die Prozessoren l_1, s_1, l_5, s_2, l_8 übertragen. Dafür wird der Task, auf den die Nachricht m_1 abgebildet wird, fünf mal angelegt und jeweils nacheinander auf den Prozessoren ausgeführt. Bei der Nachricht m_2 gilt zu beachten, dass mehrere Empfänger vorliegen und sich die Route nach dem zweiten Switch aufspaltet.

4.5 Job und Task Modell

In der Scheduling Theorie mit mehreren Prozessoren lassen sich Tasks häufig logisch zu Jobs gruppieren (Drozdowski, 2010, Seite 56). Ein Job besteht dann aus einer Menge von Tasks, die untereinander zur Relation stehen. Diese Gruppierung lässt sich nach der Transformation auch an unserem Modell durchführen und erleichtert die Anwendung von bereits bestehenden Scheduling Strategien.

Für jeden virtuellen Link einer TT-Nachricht lässt sich ein Job erstellen. Die Basis für diesen Job bilden die Tasks, die sich aus der Transformation dieser Nachricht ergeben. Durch das Routing ergibt sich für jede Nachricht eine Menge von Prozessoren, über die die Nachricht übertragen werden muss. Für jeden Prozessor in dieser Route wird ein separater Task erzeugt, die letztendlich in einem Job gruppiert werden. Zusätzlich zu den Nachrichten-Tasks werden auch die Tasks zu dem Job hinzugefügt, die sich aus der Abbildung des Senders und der Empfänger der Nachricht ergeben. Da eine Applikation mehr als eine Nachricht empfangen kann, ist es möglich, dass ein Empfangs-Task zu mehreren Jobs gehört.

Die Taskmenge eines Jobs ist nicht ungeordnet, sondern bildet eine Reihenfolge ab, in der die Tasks ausgeführt werden müssen. Diese Reihenfolge lässt sich anhand der logischen Abhängigkeiten der Tasks untereinander ableiten und durch die partielle Ordnung

$$(T, \rightarrow)$$

formal beschrieben. Durch diese Ordnung wird eine Vorgänger- und Nachfolger-Relation zwischen den Tasks hergestellt, die die Ausführungsreihenfolge klar definiert. Hat eine Nachricht nur einen Empfänger, ist die Ordnung sogar total, da sie nur aus einer Kette besteht. Hat eine Nachricht mehrere Empfänger, existiert für jeden Empfänger eine Kette innerhalb der Ordnung. Der Sende-Task ist immer die obere Schranke jeder Kette, während die Empfangs-Tasks die unteren Schranken bilden. Zwischen diesen Schranken sind die Nachrichten-Tasks angesiedelt, deren Ordnung sich aus dem Routing ergibt.

4.5.1 Der Taskgraph des Beispiels

In unserem Beispielnetzwerk liegen vier virtuelle Links vor und für jede TT-Nachricht ergibt sich ein Job. Die Taskmenge des Jobs j_1 für die Nachricht m_1 beinhaltet den Sende-Task a_1 und den Empfangs-Task a_2. Die Nachricht passiert in der Übertragung die fünf Prozessoren

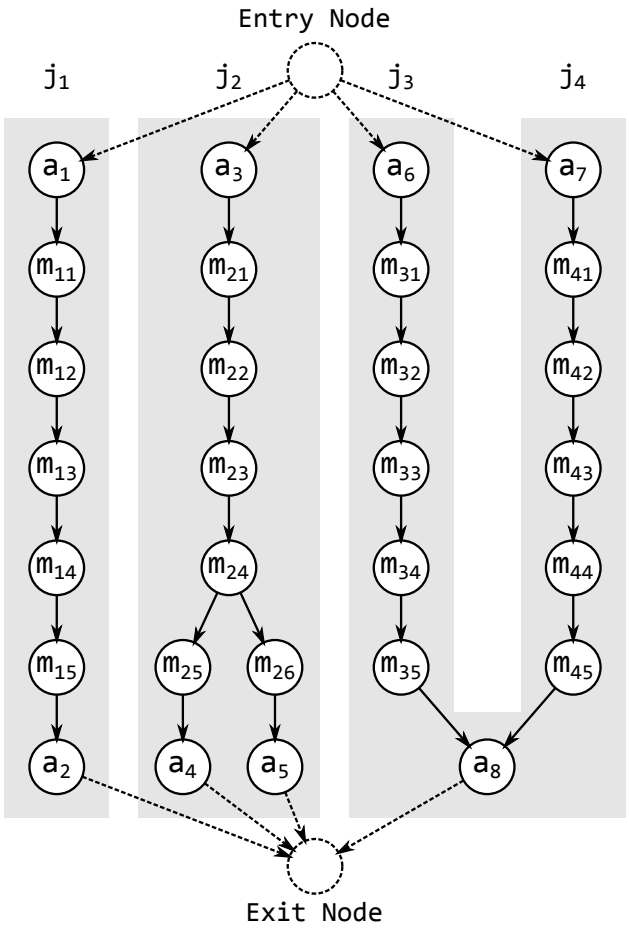

Abbildung 4.7: *Taskgraph des Beispielnetzwerks*

l_1, s_1, l_5, s_2, l_8 und benötigt daher fünf Nachrichten-Tasks $m_{11}, m_{12}, m_{13}, m_{14}, m_{15}$. Analog zu diesem Vorgehen ergeben sich die Jobs des Beispiels damit zu:

$$
\begin{aligned}
j_1 &= \{a_1, m_{11}, m_{12}, m_{13}, m_{14}, m_{15}, a_2\} \\
j_2 &= \{a_3, m_{21}, m_{22}, m_{23}, m_{24}, m_{25}, m_{26}, a_4, a_5\} \\
j_3 &= \{a_6, m_{31}, m_{32}, m_{33}, m_{34}, m_{35}, a_8\} \\
j_4 &= \{a_7, m_{41}, m_{42}, m_{43}, m_{44}, m_{45}, a_8\}
\end{aligned}
$$

Die Ordnung der Taskmenge eines Jobs und damit die Vorgänger- und Nachfolger-Relationen der Tasks lässt sich mittels eines Taskgraphen, wie er in Abschnitt 3.1.2 beschrieben wurde,

visuell darstellen. In Abbildung 4.7 ist der Taskgraph für das Beispielnetzwerk veranschaulicht und die Ketten der Mengen sind leicht ersichtlich. Während die Mengen der Jobs j_1, j_3 und j_4 nur aus einer Kette bestehen, enthält j_2 zwei Ketten, da die Nachricht m_2 zwei Empfänger besitzt. Gleichzeitig ist zu erkennen, dass der Task a_8 zu Job j_3 und j_4 gehört, da die Applikation sowohl die Nachricht $m3$ als auch $m4$ empfängt.

4.6 Perioden und Task-Deadlines

Im Abschnitt 2.3 wurde das periodische Zeitverhalten der TT-Nachrichten erläutert. Diese Perioden machen es nötig, dass die TT-Nachrichten einem Scheduling unterzogen werden müssen. Erst ein gültiger Schedule garantiert, dass alle Nachrichten fristgerecht in ihrer Periode gesendet werden. Die Periodenlänge lässt sich damit direkt auf die Deadline der Tasks übertragen. Diese Übertragung auf die Tasks des dazugehörigen Jobs bedeutet, dass die Deadline auf den letzten Task einer jeden Kette festgelegt wird. Alle Vorgänger dieses Tasks stehen in Abhängigkeit zu dieser Deadline und müssen entsprechend im Schedule beachtet werden. Um dieses Problem zu lösen, werden diese Abhängigkeiten aufgelöst und eine Deadline für alle Vorgänger des Empfangs-Tasks berechnet (Chetto u. a., 1990). Zur Veranschaulichung betrachten wir einen Job mit drei Tasks und folgenden Abhängigkeiten:

$$j = \{t_1, t_2, t_3\}$$
$$t_1 \rightarrow t_2 \rightarrow t_3$$

In diesem vereinfachten Beispiel ist t_1 der Sende-, t_2 der Nachrichts- und t_3 der Empfangs-Task. Zur Definition der Timing Eigenschaften der Tasks siehe 3.1.1. Der Task t_3 bekommt, abgeleitet von der Periode der Nachricht, die Deadline d_3 zugewiesen. Es muss ein Schedule gefunden werden, der diese Deadline erfüllt und dabei die Abhängigkeiten der Tasks beachtet. Task t_1 und t_2 müssen zeitlich so ausgeführt werden, dass t_3 vor seiner Deadline beendet werden kann. Es muss gelten:

$$f_3 \leq d_3$$
$$f_2 \leq d_3 - c_2$$
$$f_1 \leq d_3 - c_2 - c_1$$

Für das Scheduling werden diese zu erfüllenden Timings auf die Deadlines umgerechnet und damit die Abhängigkeiten zwischen den Tasks aufgelöst, indem garantiert wird, dass der Vorgänger eines Tasks beendet ist, bevor er selber ausgeführt wird. Die Berechnung der Deadlines kann auf unterschiedlichen Wegen durchgeführt werden. Folgend wird eine Strategie vorgestellt, die den Tasks eine größtmögliche Freiheit bei der Platzierung im Schedule ermöglicht.

Diese Methode sieht vor, dass die Deadline eines Tasks so angepasst wird, dass sie der spätesten möglichen Release Time ihres Nachfolgers entspricht. Das heißt, die Deadline ist der Zeitpunkt, an dem der Nachfolger spätestens mit seiner Ausführung beginnen muss, um seine eigene Deadline noch einzuhalten. Da ein Task mehrere Nachfolger besitzen kann, muss das Minimum aller berechneten Deadlines gewählt werden. Gegeben sei ein Task t_i und die Menge seiner Nachfolger $S = \mathbf{succ}(t_i)$. Der zugehörige Job habe die Periodenlänge p. Die Deadline des Tasks ergibt sich dann zu:

$$d_i = \begin{cases} min(\{d_j - c_j | t_j \in S\}) : \text{wenn } S \neq \emptyset \\ p : \text{sonst} \end{cases} \tag{4.2}$$

Diese Berechnung wird Schritt für Schritt für alle Tasks eines Jobs durchgeführt, beginnend am Ende einer Kette. Man erreicht durch dieses Verfahren eine größtmögliche Freiheit, da die Deadlines so spät wie möglich terminiert werden. Die Tasks können in ihrer Ausführung maximal verzögert werden.

4.6.1 Deadlines im Beispiel

Wie in den vorangegangen Abschnitten, wenden wir diese Berechnung der Deadlines zunächst auf das Beispiel an, um das Verfahren zu verdeutlichen. Als Ausgangslage nutzen wir den Taskgraphen aus Abschnitt 4.5.1. In diesem sind die einzelnen Jobs und Abhängigkeiten deutlich ersichtlich, so dass wir, beginnend vom *Exit Node*, die Deadlines für jeden einzelnen Task berechnen können. Exemplarisch wollen wir dies am Job j_2 durchführen.

$$\begin{aligned}
d_{a_4} &= 10000\text{µs, da } \mathbf{succ}(a_4) = \emptyset \\
d_{a_5} &= 10000\text{µs, da } \mathbf{succ}(a_5) = \emptyset \\
d_{m_{25}} &= d_{a_4} - c_{a_4} = 9700\text{µs, mit } \mathbf{succ}(m_{25}) = \{a_4\} \\
d_{m_{26}} &= d_{a_5} - c_{a_5} = 9800\text{µs, mit } \mathbf{succ}(m_{26}) = \{a_5\} \\
d_{m_{24}} &= d_{m_{25}} - c_{m_{25}} = 9576,96\text{µs, mit } \mathbf{succ}(m_{24}) = \{m_{25}, m_{26}, a_4, a_5\} \\
d_{m_{23}} &= d_{m_{24}} - c_{m_{24}} = 9376,96\text{µs, mit } \mathbf{succ}(m_{23}) = \{m_{24}, m_{25}, m_{26}, a_4, a_5\} \\
d_{m_{22}} &= d_{m_{23}} - c_{m_{23}} = 9253,92\text{µs, mit } \mathbf{succ}(m_{22}) = \{m_{23}, \ldots, m_{26}, a_4, a_5\} \\
d_{m_{21}} &= d_{m_{22}} - c_{m_{22}} = 9053,92\text{µs, mit } \mathbf{succ}(m_{21}) = \{m_{22}, \ldots, m_{26}, a_4, a_5\} \\
d_{a_3} &= d_{m_{21}} - c_{m_{21}} = 8930,88\text{µs, mit } \mathbf{succ}(a_3) = \{m_{21}, \ldots, m_{26}, a_4, a_5\}
\end{aligned}$$

Die Deadlines der Tasks der anderen Jobs werden analog zu diesem Verfahren berechnet.

4.7 Job Shop und TT-Ethernet

Bevor das Scheduling-Problem formal beschrieben wird, hilft eine Einordnung in die unterschiedlichen Kategorien der Scheduling-Theorie. Nach der Transformation eines TT-Ethernet Netzwerks wird deutlich, dass wir uns im Bereich des Multi-Prozessor Schedulings mit dedizierter Zuordnung der Tasks auf die Prozessoren bewegen. Bei dieser Art des Schedulings sind die Tasks in Jobs organisiert und man unterscheidet zwischen *Open Shop*, *Flow Shop* und *Job Shop* Systemen.

Dieses Prinzip kann ohne Probleme auf das TT-Ethernet Modell übertragen werden, da durch die virtuellen Links des Netzwerks bereits eine Strukturierung der Tasks in Jobs vorgegeben ist. Alle Tasks die zu einem virtuellen Link gehören, also das Senden, Übertragen und Empfangen einer Nachricht übernehmen, bilden einen Job (vgl. Abschnitt 4.5). Die drei *Shop* Systeme unterscheiden sich in der Art und Weise, wie die Jobs und Tasks auf die Prozessoren verteilt werden.

Vergleicht man die Definitionen der *Shop Probleme* aus Abschnitt 3.2, liegt es nahe, das Scheduling eines TT-Ethernet Netzwerks dem *Job Shop Problem* zuzuordnen. Beim *Open Shop* können die Tasks in beliebiger Reihenfolge ausgeführt werden. Dies ist aber bei einem Netzwerk eindeutig nicht der Fall, da der Sende-, Übertragungs- und Empfangsvorgang in einer unveränderbaren Sequenz definiert ist. Beim *Flow Shop* wiederum ist definiert, dass in jedem Job genau ein Task pro Prozessor vorhanden sein muss. Das würde bedeuten, dass jede Nachricht über jeden Link übertragen werden müsste. Es ist leicht zu sehen, dass dies bei unserem System nicht der Fall ist. Das *JSSP* sieht dagegen vor, dass die Zuteilung der Tasks auf die Prozessoren und die Ausführungsreihenfolge vom Problem abhängig ist, also je nach Aufbau des Systems, definiert werden kann. Zur Formulierung und Lösung des Problems ergibt es daher Sinn, sich an dem Bereich der *Job Shop Scheduling Probleme* zu orientieren.

4.8 Formale Problembeschreibung

Das Scheduling-Problem von TDMA Kommunikation in einem Switch basierten Netzwerk lässt sich formal folgendermaßen beschreiben. Es gibt eine Menge von Jobs

$$J = \{j_1, j_2, \ldots, j_n\}, n \in \mathbb{N},$$

eine Menge von Tasks

$$T = \{t_1, t_2, \ldots, t_m\}, m \in \mathbb{N}$$

und eine Menge von Prozessoren

$$P = \{p_1, p_2, \ldots, p_o\} o \in \mathbb{N}.$$

Die Taskmenge T besteht aus n Untermengen, die den entsprechenden Jobs zugeordnet sind. Jeder Job j besteht aus einer Folge von m_j Tasks

$$j_i = (t_{j1}, t_{j2}, \ldots, t_{jm_j}),$$

die in dieser Reihenfolge ausgeführt werden müssen. Die Ausführung eines Tasks $t_i \in T$ darf nicht unterbrochen werden und benötigt eine Ausführungszeit $c_i > 0$. Zusätzlich besitzt jeder Task eine Release Time r_i und eine Deadline d_i, so dass für jeden Task gelten muss

$$s_i \geq r_i \wedge f_i \leq d_i.$$

Die Reihenfolge, in der die Tasks ausgeführt werden müssen, ergibt sich aus dem Taskgraphen $G = (V, E)$ (vgl. Abschnitt 3.1.2). E definiert eine partielle Ordnung innerhalb der Taskmenge. Aus $t_1 \rightarrow t_2$ folgt, dass t_1 vor t_2 ausgeführt werden muss.

Die Funktion

$$\sigma : T \rightarrow P$$

beschreibt die Abbildung der Tasks auf die Prozessoren. Jeder Task t_i muss auf einem Prozessor

$$\sigma(t_i) = p_k$$

ausgeführt werden. Durch diese Relation wird die Taskmenge T in weitere Untermengen

$$\Omega_k = \{t_i \in T \wedge \sigma(t_i) = p_k\}$$

geteilt, die jeweils die Tasks enthalten, die auf dem Prozessor p_k ausgeführt werden müssen. Es wird davon ausgegangen, dass jeder Task eines Jobs auf einem anderen Prozessor ausgeführt werden muss und jeder Prozessor nur maximal einen Task zur selben Zeit ausführen kann. Ein Schedule lässt sich daher auch als ein Tupel

$$\pi = (\pi_1, \ldots, \pi_o)$$

darstellen, wobei

$$\pi_k = (\pi_k(1), \ldots, \pi_k(|\Omega_k|))$$

eine Permutation von Ω_k ist. π ist ein Tupel, dass für jeden Prozessor aus P ein weiteres Tupel π_k enthält, dessen Elemente den Tasks entsprechen, die auf dem Prozessor $k \in P$ ausgeführt werden müssen. Jedes Tupel π_k hat dabei die Länge von $|\Omega_k|$. $\pi_k(i)$ definiert das Element in Ω_k welches an der Position i von π_k ist. Die Abbildung

$$\pi_k(i) = t_i$$

definiert den Task $t_i, t \in T$, der auf dem Prozessor p_k an der Stelle i ausgeführt werden muss. Dadurch sind alle Tasks einem Prozessor mit einer definierten Ausführungsreihenfolge

zugeordnet. Wenn Π_k nun die Menge aller Permutationen von Ω_k darstellt, dann ist $\pi \in \Pi$, mit

$$\Pi = \Pi_1 \times \Pi_2 \times \ldots \times \Pi_o.$$

Π bildet somit die komplette Lösungsmenge ab und π stellt eine spezielle Lösung des Problems dar. Dieser Zusammenhang lässt sich auch durch einen disjunktiven Taskgraphen darstellen und lösen, wobei Π dem Graphen entspricht, der noch ungerichtete Kanten enthält und π dem Graphen, in dem alle Kanten gerichtet sind (vgl. Abschnitt 3.2.1).

Eine Lösung für das Problem muss weiterhin die folgenden Bedingungen erfüllen:

Definition 22 (*Schedule*)

Ein Schedule ist eine Funktion $S : T \to \mathbb{N}$, die für jeden Task t_i einen Startpunkt s_{t_i} definiert. Ein Schedule ist gültig, wenn gilt:

$$\begin{aligned}
\forall t \in T &: s_t \geq 0 \\
\forall t \in T &: s_t + c_t \leq d_t \\
\forall t_1, t_2 \in T, t_1 \to t_2 &: s_{t_1} + c_{t_1} \leq s_{t_2} \\
\forall t_1, t_2 \in T, t_1 \neq t_2, \pi(t_1) = \pi(t_2) &: s_{t_1} + c_{t_1} \leq s_{t_2} \vee s_{t_2} + c_{t_2} \leq s_{t_1}
\end{aligned}$$

4.9 Eine Lösung des Beispiels

Zum Abschluss dieses Kapitels soll eine Lösung, also ein gültiger Schedule für das Beispielnetzwerk, mit den erarbeiteten formalen Mitteln aufgeschrieben werden und die Definitionen zur Gültigkeit eines Schedules an dem Beispiel angewendet werden. Der Schedule wurde mit der *First Come First Served* Heuristik bestimmt, so dass sich die Ausführungsreihenfolge der Tasks auf den Prozessoren folgendermaßen ergibt:

$$\begin{aligned}
&\pi_{n_1} = (a_3, a_6, a_1), && \pi_{n_2} = (a_7), \\
&\pi_{n_3} = (a_4, a_2, a_8), && \pi_{n_4} = (a_5), \\
&\pi_{l_1} = (m_2, m_3, m_1), && \pi_{l_3} = (m_4), \\
&\pi_{l_5} = (m_4, m_2, m_3, m_1), && \pi_{l_8} = (m_4, m_2, m_3, m_1), \\
&\pi_{l_{10}} = (m_2)
\end{aligned}$$

Anschaulicher werden diese Sequenzen, wenn sie mit der Hilfe eines disjunktiven Taskgraphen (vgl. Abschnitt 3.2.1) dargestellt werden, welcher für dieses Beispiel in Abbildung 4.8 wiedergegeben wird. Alle Tasks wurden so eingefärbt, dass dadurch die Zugehörigkeit zu

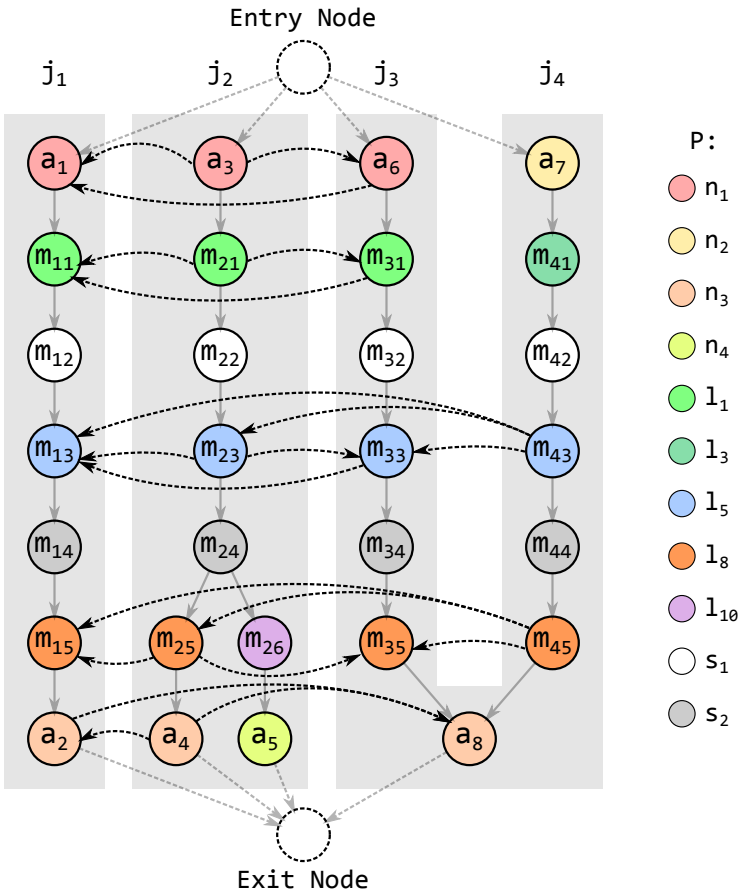

Abbildung 4.8: *Disjunktiver Taskgraph des Beispielnetzwerks*

einem Prozessor hergestellt wird. Alle Tasks einer Farbe werden auf dem selben Prozessor ausgeführt. Die Tasks a_7, m_{41}, m_{26} und a_5 sind jeweils die einzigen Tasks, die auf dem entsprechenden Prozessor ausgeführt werden, so dass keine Reihenfolge in der Ausführung durch den disjunktiven Graphen hergestellt werden muss. Die Tasks m_{i2} und m_{j4}, welche in weiß und grau dargestellt werden, stehen für die Tasks, die auf den Switches ausgeführt werden. Da ein Switch die Tasks parallel verarbeitet, wird hier keine Reihenfolge benötigt, so dass in diesen Fällen keine Einbindung in den disjunktiven Graphen erfolgt.

In der Tabelle 4.4 sind alle Timing-Daten zu dem Beispiel in Nanosekunden angegeben. Für jeden Job ist dabei nur jeweils die erste Instanz der Tasks aufgeführt, da eine komplette Auflistung zu umfangreich wäre. Für die späteren Instanzen muss die jeweilige Periodenlänge der

Start Time (ns)	Finish Time (ns)	Deadline (ns)	Prozessor	Task
0	100000	3666880	n_2	a_7
0	300000	9130880	n_1	a_3
100000	111040	3677920	l_3	m_{41}
211040	222080	3788960	l_5	m_{43}
300000	700000	3679840	n_1	a_6
300000	423040	9253920	l_1	m_{21}
322080	333120	3900000	l_8	m_{45}
523040	646080	9476960	l_5	m_{23}
700000	800000	4542880	n_1	a_1
700000	706720	3686560	l_1	m_{31}
746080	869120	9800000	l_8	m_{25}
746080	869120	9700000	l_{10}	m_{26}
800000	819040	4561920	l_1	m_{11}
806720	813440	3793280	l_5	m_{33}
869120	1069120	10000000	n_4	a_5
869120	1169120	10000000	n_3	a_4
913440	920160	3900000	l_8	m_{35}
919040	938080	4680960	l_5	m_{13}
1038080	1057120	4800000	l_8	m_{15}
1169120	1369120	5000000	n_3	a_2
1369120	1469120	4000000	n_3	a_8

Tabelle 4.4: *Scheduling Tabelle des Beispielnetzwerks*

Jobs hinzuaddiert werden. Anhand dieser Timing-Daten und der in Definition 22 aufgestellten Regeln lässt sich die Gültigkeit des Schedules ermitteln. Für die erste Bedingung

$$\forall t \in T : s_t \geq 0$$

ist schnell ersichtlich, dass diese erfüllt ist. Die *Start Times* aller Tasks sind größer oder gleich 0. Auch die Gültigkeit der zweiten Bedingung

$$\forall t \in T : s_t + c_t \leq d_t$$

lässt sich ohne Probleme überprüfen, da alle *Finish Times* kleiner als die zugehörige *Deadline* sind. Die Bedingung

$$\forall t_1, t_2 \in T, t_1 \rightarrow t_2 : s_{t_1} + c_{t_1} \leq s_{t_2}$$

besagt, dass alle *Finish Times* der Vorgänger eines Tasks kleiner oder gleich der *Start Time* des Nachfolgers sein müssen. Überprüft man dies anhand der Nachricht m_4, sieht man, dass

die *Finish Time* von m_{41} kleiner ist als die *Start Time* von m_{43} und so weiter. Die letzte Bedingung

$$\forall t_1, t_2 \in T, t_1 \neq t_2, \pi(t_1) = \pi(t_2) : s_{t_1} + c_{t_1} \leq s_{t_2} \vee s_{t_2} + c_{t_2} \leq s_{t_1}$$

ist erfüllt, wenn auf keinem Prozessor zu keiner Zeit zwei Tasks parallel ausgeführt werden. Am Beispiel des Prozessors $n1$ sieht man, dass es zu keinen Überschneidungen bei den *Start Times* und *Finish Times* der Tasks gibt. Die Tasks a_3, a_6 und a_1 werden aufeinander folgend ausgeführt und nicht parallel.

5 Konzeption der Scheduling Heuristik

Nachdem in Kapitel 4 erläutert wurde, wie ein TT-Ethernet Netzwerk in den Raum der Mehrprozessor Scheduling-Probleme transportiert wird und das Problem abschließend formal definiert wurde, soll in diesem Kapitel diskutiert werden, auf welchem Weg für dieses Problem ein gültiger und sinnvoller Schedule gefunden werden kann. Zur Entwicklung einer passenden Heuristik muss zunächst das Optimierungsziel ausgearbeitet und darauf aufbauend die Bewertungsfunktion festgelegt werden. Mit diesen beiden Punkten können nun Algorithmen entwickelt werden, die das Problem zielgerichtet lösen. Die dazu erforderlichen Schritte sollen zum Abschluss dieses Kapitels aufgezeigt werden.

5.1 Optimierungsziel des Schedules

Für das Ausarbeiten der Anforderungen an den Schedule muss man zunächst betrachten, wie die Kommunikation innerhalb des Netzwerks abläuft und was dieses in unserem Modell für die Tasks bedeutet. Die Timing Eigenschaften der drei Traffic Klassen, die in Abschnitt 2.2 erläutert wurden, spielen dabei eine ausschlaggebende Rolle.

5.1.1 Optimierung des TT-Traffic

Die TT-Nachrichten sind die wichtigste Nachrichtenklasse, da sie dafür genutzt werden, Steuerkreise zu implementieren, die oftmals sicherheitskritische Funktionen erfüllen. Zu diesem Zweck werden diese Nachrichten periodisch versendet. Auf das Scheduling bezogen, bedeutet dies, dass wir einen Task t haben, der mit der Periode p ausgeführt wird. Da der Task jeweils innerhalb seiner zugewiesenen Periode ausgeführt werden muss, ist die Deadline d des Tasks mit der Periodenlänge gleichzusetzen (vgl. Abschnitt 4.6).

Die Abbildung 5.1 zeigt einen Ausschnitt eines Schedules mit zwei TT-Nachrichten über drei Perioden hinweg. An diesem Diagramm lässt sich gut erkennen, welche Punkte in Hinsicht auf das Scheduling wichtig sind. Zunächst wird deutlich, dass es keinen Sinn ergibt, den Schedule auf seine Länge zu optimieren, da das periodische Ausführen der Tasks keine zeitlichen Grenzen hat. Weiterhin kann man urteilen, dass die Position des Tasks innerhalb seiner Periode keine Rolle spielt, so lange die Deadline des Tasks nicht durch die Positionierung

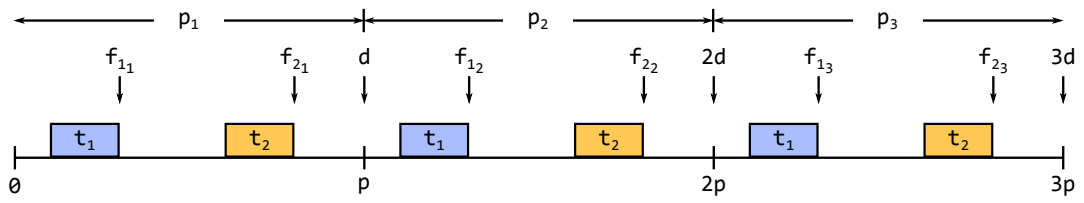

Abbildung 5.1: *Beispiel eines Schedules mit zwei TT-Nachrichten*

verletzt wird. Dies wird gestützt durch die Tatsache, dass der Zeitpunkt der Übertragung einer Nachricht innerhalb der Perioden immer derselbe sein muss (vgl. Abschnitt 2.3). Die Positionen der Instanzen eines Tasks sind, relativ gesehen zum Beginn der Periode, immer die gleichen. Der Abstand zwischen zwei Instanzen eines Tasks ist unveränderlich und es gilt:

$$f_{1_{i+1}} - f_{1_i} = f_{1_{i+2}} - f_{1_{i+1}}$$

Die Positionierung des Tasks hat zwar einen Einfluss auf seine Lateness, so ist zum Beispiel die Lateness von Task t_1 geringer als die von Task t_2, weil

$$f_{1_1} - d < f_{2_1} - d$$

ist, aber die Lateness sagt in diesem Fall nichts über die Qualität des Schedules aus, da nur ausschlaggebend ist, dass der Task seine Deadline nicht verletzt. Der Schedule kann daher nicht durch die Positionierung der TT-Nachrichten optimiert werden. Die Verwendung traditioneller Metriken, wie sie in Drozdowski (2010) und Buttazzo (2011) beschrieben werden, kommt daher nicht in Frage.

5.1.2 Optimierung des RC-Traffic

Der RC-Traffic verlangt eine getrennte Betrachtung, da dieser nicht periodisch abläuft, sondern Event-basiert operiert. Dies bedeutet, dass sich diese Art der Nachrichten nicht statisch planen lassen, sondern dynamisch zur Laufzeit berechnet werden müssen. Bei der statischen Berechnung des Schedules für die TT-Nachrichten können aber die Anforderungen der RC-Nachrichten berücksichtigt werden. Anhand der Abbildung 5.2, in der zwei unterschiedliche Schedules *A* und *B* illustriert werden, soll dies näher erläutert werden

Die Schedules zeigen drei TT-Nachrichten TT_1, TT_2 und TT_3, die mit der Periode p versendet werden und zwei RC-Nachrichten RC_4 und RC_5, die zufällig in dieses Übertragungsfenster gefallen sind. Der Zeitpunkt ihres Auftretens im System ist entsprechend mit den *Release Times* r_4 und r_5 markiert. Der Unterschied zwischen den beiden Schedules besteht darin, dass die Tasks für die TT-Nachrichten in *Schedule A* direkt aufeinander folgend geplant sind und

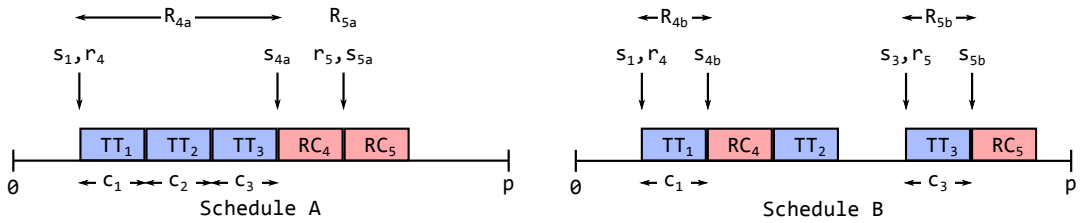

Abbildung 5.2: *Zwei Schedules mit TT- und RC-Nachrichten*

in *Schedule B* Lücken zwischen den TT-Tasks existieren. Diese Lücken haben einen direkten Einfluss auf die *Response Times* der RC-Tasks. Die *Release Time* r_4 des Tasks RC_4 fällt zusammen mit der *Start Time* s_1 des ersten TT-Tasks. Durch die Verkettung der TT-Tasks kann der RC-Task nun erst zum Zeitpunkt s_4 gestartet werden. Dies führt zu einer *Response Time*

$$R_{4a} = s_4 - r_{4a} = c_1 + c_2 + c_3$$

für den Task RC_4. Betrachtet man nun die *Response Time* für den gleichen Task im *Schedule B*, fällt auf, dass diese geringer ausfällt mit

$$R_{4b} = s_4 - r_{4b} = c_1,$$

da zwischen den TT-Tasks genügend Raum ist, den Task RC_4 auszuführen. Bei dem Task RC_5 ergibt sich ein anderes Bild, da sich hier die *Response Time* verschlechtert. Beträgt sie in *Schedule A* noch

$$R_{5a} = s_{5a} - r_5 = 0,$$

vergrößert sie sich in *Schedule B* auf

$$R_{5b} = s_{5b} - r_5 = c_3.$$

Durch diese Betrachtung wird deutlich, dass der Schedule der TT-Nachrichten Einfluss auf die *Response Time* der RC-Nachrichten nimmt. Dies kann für einzelne Nachrichten in positiver wie auch in negativer Weise erfolgen. Da die *Release Times* der RC-Nachrichten nicht vorhergesagt werden können und rein zufällig auftreten, ist es nicht gültig zu sagen, dass die *Response Time* für jede RC-Nachricht verbessert wird. Stattdessen wird versucht durch die Lücken, die durchschnittliche Wartezeit zu minimieren und die Worst Case Szenarien zu entschärfen. Verdeutlichen lässt sich dies an der Abbildung 5.3, in der drei TT-Nachrichten mit der gleichen Periode p gesendet werden. In diesem idealisierten Schedule hat jede Nachricht die Länge 1. In beiden Schedules blockiert der TT-Traffic 50% der Übertragungszeit, der jedoch unterschiedlich verteilt ist. Dies führt dazu, dass die durchschnittliche *Response Time* und der Worst Case Fall eventueller RC-Nachrichten in *Schedule D* geringer ist.

Abbildung 5.3: *Zwei Schedules mit unterschiedlich verteilten TT-Nachrichten*

Nachfolgend wird ein Tupel (x, y) dazu genutzt, den Zeitraum zwischen den Punkten x und y innerhalb des Schedules zu bezeichnen. Um den Umstand nachzuweisen, dass die Lücken einen positiven Einfluss auf die durchschnittliche *Response Time* hat, wird davon ausgegangen, dass RC-Nachrichten zu jedem beliebigen Zeitpunkt eintreffen können und über die ganze Periode p gleich verteilt sind. Vernachlässigt werden die RC-Nachrichten, die exakt zu Beginn eines Zeitfensters auftreten, da sie die Betrachtung verkomplizieren und statistisch gesehen, zu vernachlässigen sind. Es ist weiterhin wichtig, nicht nur eine Periodenlänge zu betrachten, sondern auch die darauf folgende. So wird eine RC-Nachricht, die im Zeitraum $(5, p)$ in *Schedule C* eintrifft, durch die Nachricht TT_1 im darauf folgenden Zyklus blockiert.

Das der Worst Case in *Schedule C* schlechter ist als in *D*, ist leicht zu erkennen. Für *C* tritt der Fall ein, wenn eine RC-Nachricht am Anfang des Fensters $(5, p)$ eintrifft, da sie durch die Nachricht TT_1 im nächsten Zyklus blockiert wird. Die Worst Case *Response Time* beträgt in diesem Fall 4 Zeiteinheiten, also den Raum $(5, 9)$. Für den *Schedule D* beträgt die längste Wartezeit dagegen nur 2 Zeitfenster. Diese tritt auf, wenn eine RC-Nachricht zu Beginn eines freien Fensters eintrifft, zum Beispiel $(1, 2)$. In diesem Fall wird die Nachricht durch den Raum $(1, 3)$ verzögert.

Die Berechnung der durchschnittlichen *Response Time* der Schedules erfolgt durch die Betrachtung der blockierten und freien Räume. Für *Schedule C* ist der Raum $(5, 9)$ periodenübergreifend blockierend. Fällt in diese Zeit eine RC-Nachricht wird sie durchschnittlich um $\frac{9-5}{2} = 2$ Zeiteinheiten verzögert. Diese Verzögerung findet demnach in 4 von 6 Zeiteinheiten statt. Der Raum $(3, 5)$ ist dagegen frei und es findet keine Verzögerung statt. Für 2 Zeiteinheiten ist die Verzögerung 0. Die durchschnittliche Verzögerung für die komplette Periode ergibt sich damit zu:

$$\frac{(2 * 4) + (0 * 2)}{6} = \frac{4}{3}$$

Der Wert für *Schedule D* berechnet sich ähnlich, nur das hier theoretisch kein freier Raum vorliegt, da wir den exakten Beginn eines Zeitfensters nicht berücksichtigen. Stattdessen liegen zwei blockierende Räume vor. Der erste ist $(0, 1)$, in dem RC-Nachrichten durchschnittlich um $\frac{1-0}{2} = \frac{1}{2}$ Einheiten blockiert werden. Der Zweite ist $(1, 3)$. Hier findet eine Verzögerung von $\frac{3-1}{2} = 1$ Einheit statt. Beide Fälle treten jeweils drei mal innerhalb einer

Periode auf. Damit ergibt sich die durchschnittliche Verzögerung für die komplette Periode in *Schedule D* zu:

$$\frac{(0,5 * 3) + (1 * 3)}{6} = \frac{3}{4}$$

Damit wäre nachgewiesen, dass sowohl die Worst Case, wie auch die durchschnittliche Verzögerung im zweiten Schedule geringer ist. Obwohl das Beispiel stark konstruiert und idealisiert ist, lässt sich schlussfolgern, dass ein Schedule durch entsprechende Lücken im TT-Schedule nicht verschlechtert wird, so lange die Lücken groß genug sind. Eine Lücke sollte mindestens so groß sein, dass alle auftretenden RC-Frames in dem Zeitfenster übertragen werden können. Da für RC-Nachrichten auch eine quantitative Aussage über die Frames getroffen werden kann, sollte dies beim Festlegen der Lücken berücksichtigt werden.

5.1.3 Optimierung des BE-Traffic

Für BE-Nachrichten gelten die gleichen Aussagen, die zum RC-Traffic getroffen wurden. Auch die BE-Nachrichten treten im Netzwerk spontan auf und können daher nicht vorher statisch im Schedule berücksichtigt werden. Lücken im TT-Traffic sorgen dafür, dass auch die *Response Times* der BE-Nachrichten minimal bleiben. Jedoch fällt es in diesem Fall schwer, Aussagen über die Größe der Lücken zu treffen, da die Menge des Traffics und die Größe der Frames unbekannt ist. Diese Art der Nachrichten wird daher nur untergeordnet berücksichtigt werden.

5.1.4 Zusammenfassung

In den vorangegangenen Abschnitten wurde ausführlich darauf eingegangen, welche Traffic Klassen in welcher Form Einfluss auf die Qualität des Schedules haben. Für die Wahl des Optimierungsziels ist es wichtig, nicht nur den Traffic zu beachten, der statisch geplant wird, sondern den gesamten Datenverkehr in die Betrachtung mit einzubeziehen. Die Minimierung der *Response Times* der RC- und BE- Nachrichten, also das schnellstmögliche Weiterleiten dieses Traffics, bietet sich als Optimierungsziel an. Um dieses zu erreichen muss der TT-Traffic so geplant werden, dass über dessen gesamten Zyklus möglichst große und gleichmäßig verteilte Lücken im Schedule vorhanden sind. Als Beispiel können die Schedules aus Abbildung 5.3 heran gezogen werden, in der *Schedule D* den optimierten Schedule gegenüber *Schedule C* darstellt. Die Wahl der Größe der Lücken zwischen den TT-Nachrichten sollte sich dabei an der Größe der RC-Nachrichten orientieren, da diese zum Zeitpunkt der Erstellung des Schedules bekannt ist.

5.2 Vorbemerkungen

In den folgenden Abschnitten werden mathematische Berechnungsvorschriften entwickelt, die es ermöglichen sollen, die Qualität eines Schedules objektiv zu bewerten. Um die Komplexität bei der dieser Betrachtung zu reduzieren, wird das Modell in bestimmten Punkten eingeschränkt. Die Aussagen in den folgenden Abschnitten beziehen sich hauptsächlich auf das Verhältnis von Lücken im TT-Schedule in Bezug auf die *Response Time* von RC-Nachrichten. Die getroffenen Aussagen behalten dabei nur so lange Gültigkeit, wie kein Stauverhalten beim Versenden der RC-Nachrichten auftritt.

Dies bedeutet, dass bei diesem Modell immer nur eine RC-Nachricht zur selben Zeit durch eine TT-Nachricht blockiert wird und sich nicht mehrere RC-Nachrichten durch einen blockierenden Sendevorgang aufgestaut werden. Scheint dies zunächst eine kritische Einschränkung zu sein, die eine allgemeine Gültigkeit des Modells verhindert, so relativiert sich dieses Ausmaß durch eine zusätzliche Betrachtung des TT-Traffics und dem dazugehörigen Füllgrad des Schedules. In modernen Fahrzeugsystemen beträgt die Menge der TT-Nachrichten nur um die 5% des Signalaufkommens, bei dem die TT-Nachrichten ausschließlich die Größe eines minimalen Ethernet-Frames (64 Byte) erreichen (vgl. Kamieth u. a. (2014)).

Der Zeitraum, in dem RC- oder BE-Nachrichten blockiert werden, ist daher relativ gering, wenn man ihn mit der Zeit vergleicht, die im Cluster-Zyklus verfügbar ist. Sind nun ausreichend Große Lücken zwischen den TT-Nachrichten, so dass immer nur ein Frame von 64 Byte die Übertragung blockiert, verringert sich das Risiko einer Staubildung. Dies ermöglicht es uns, die genannten Einschränkungen vorzunehmen und bietet eine weitere Motivation, in den folgenden Abschnitten passende Bewertungsfunktionen und Heuristiken in Bezug auf Lücken im Schedule zu entwickeln.

5.3 Die Bewertungsfunktion der Heuristik

Nachdem im vorherigen Abschnitt das Optimierungsziel definiert wurde, muss als zweiter Schritt eine entsprechende Bewertungsfunktion entwickelt werden, um einen Schedule in Hinsicht auf das Optimierungsziel zu beurteilen. Diese Funktion bildet einen Schedule auf eine Kennzahl ab, die es erlaubt, eine Menge von Schedules untereinander zu vergleichen. Für diese Kennzahl gibt es mehrere Ansatzpunkte, die unter anderem auf den Lücken zwischen den TT-Nachrichten basieren. Als Metriken bieten sich die Größe der Lücken an und die Verteilung und Varianz der TT-Nachrichten innerhalb der Periode und die durchschnittliche *Response Time* für RC-Nachrichten. Folgend finden sich detaillierte Erklärungen für diese Metriken. Abschließend erfolgt eine Beurteilung auf Verwendbarkeit von einzelnen Metriken sowie von möglichen Kombinationen. Zunächst sollen die Lücken formal definiert werden.

Definition 23 (*Lücken im Schedule*)

Zur Definition der Lücken innerhalb eines Schedules muss die Cluster-Periode betrachtet werden. Nur in dieser Aufstellung sind alle TT-Nachrichten im Schedule eingetragen. Eine Einzelbetrachtung der jeweiligen Perioden der virtuellen Links ist nicht ausreichend. Eine Lücke besteht immer zwischen zwei einzelnen TT-Nachrichten, also zwischen der Finish Time der ersten und der Start Time der zweiten Nachricht. Um den Beginn und das Ende einer Periode zu berücksichtigen, wird eine Lücke zwischen der letzten Nachricht einer Cluster-Periode und der ersten Nachricht der darauf folgenden Instanz dieser Periode gemessen. Sei

$$T = \{t_1, \ldots, t_n\}, n \in \mathbb{N}$$

die Menge der Tasks der TT-Nachrichten und s_i und f_i die jeweiligen Start und Finish Times. Die Menge ist vollständig geordnet anhand ihrer Start Times s_i. Eine Lücke ist ein Tupel (x, y), welches den Anfangs- und Endpunkt der Lücke innerhalb der Cluster-Periode definiert. Die Menge der Lücken L innerhalb des Schedules einer Cluster-Periode ergibt sich durch

$$L = \bigcup_{i=2}^{n} (f_{i-1}, s_i) \cup (f_n, s_1 + p),$$

wobei p die Länger der Cluster-Periode ist.

5.3.1 Bewertung der Lückengrößen

Da die Grundlage für die Bewertung des Schedules auf den Lücken zwischen den TT-Nachrichten basiert, ist es ein naheliegender Schritt, die Größe dieser Lücken zu betrachten. In einem ersten Schritt gilt es dabei, die einzelnen Lücken im Schedule auf ihre Gültigkeit zu überprüfen. Da eine Lücke nur dann für eine Übertragung nutzbar ist, wenn sie groß genug ist, eine komplette Nachricht zu übertragen, können alle Lücken, die kleiner als die maximale RC-Nachricht sind, ignoriert werden. Die Beurteilung der Größe der noch nutzbaren Lücken lässt sich nach drei unterschiedlichen Kriterien durchführen, nach der minimalen, maximalen und durchschnittlichen Lückengröße.

Definition 24 (*Lückengröße*)

Sei

$$L = \{l_i, \ldots, l_m\}, m \in \mathbb{N}$$

die Menge aller Lücken eines Schedules und ein Tupel $l_i = (x_i, y_i)$ beschreibt jeweils die Start- und Endpunkte einer Lücke. Die Größe einer Lücke wird beschrieben durch die Funktion $g : L \to \mathbb{R}$ mit

$$g(l_i) = y_i - x_i.$$

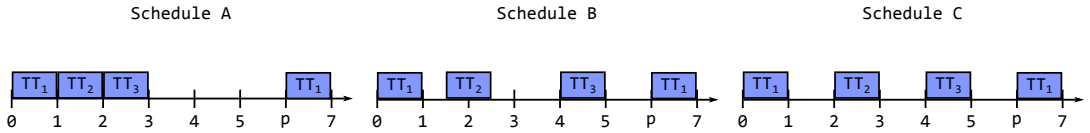

Abbildung 5.4: *Drei Schedules mit unterschiedlichen Lücken*

Definition 25 (*Gültige Lücke*)

Sei L die Menge aller Lücken eines Schedules und g die Funktion zur Berechnung der Größe der Lücke. Weiterhin sei c_{rc} die maximale Computation Time aller RC-Nachrichten. Damit ergibt sich die Menge der gültigen Lücken zu

$$\Lambda = \bigcup_{i=1}^{m} \begin{cases} \{l_i\} & \text{wenn } g(l_i) \geq c_{rc} \\ \emptyset & \text{sonst} \end{cases}$$

Definition 26 (*Minimale, maximale, gesamte und durchschnittliche Lückengröße*)

Sei $\Lambda = \{\lambda_1, \ldots, \lambda_o\}, o \in \mathbb{N}$ die Menge aller gültigen Lücken. Dann ergibt sich die minimale und maximale Größe einer nutzbaren Lücke eines Schedules durch

$$g_{min} = min(\{g(\lambda_i) \mid \lambda_i \in \Lambda\})$$
$$g_{max} = max(\{g(\lambda_i) \mid \lambda_i \in \Lambda\}).$$

Die insgesamt nutzbare Zeitraum aller Lücken ist

$$g_{sum} = \sum_{i=1}^{o} g(\lambda_i),$$

so dass sich die durchschnittliche Lückengröße berechnet durch

$$g_{avg} = \frac{g_{sum}}{o}.$$

Um eine Übersicht über die Aussagekraft dieser Werte zu bekommen, können die drei Schedules aus Abbildung 5.4 analysiert werden. Die Schedules enthalten jeweils drei TT-Nachrichten mit der Länge 1, die zyklisch mit einer Periode der Länge 6 gesendet werden. Die auftretenden RC-Nachrichten haben ebenfalls die Länge 1. Überprüft man nun die einzelnen Lücken auf ihre Gültigkeit, fällt einem auf, dass in *Schedule B* die Lücke zwischen den Nachrichten

	Anzahl	g_{sum}	g_{min}	g_{max}	g_{avg}
Schedule A	1	3	3	3	3
Schedule B	2	2,5	1	1,5	1,25
Schedule C	3	3	1	1	1

Tabelle 5.1: *Anzahl und Größe der Lücken aus Abbildung 5.4*

TT_1 und TT_2 nur eine Größe von 0,5 aufweist und damit zu klein ist, so dass diese bei der Berechnung der Werte nicht mit einbezogen werden darf. In der folgenden Tabelle 5.1 finden sich die Ergebnisse für alle drei Schedules. Die zweite Spalte beinhaltet die Anzahl der gültigen Lücken, die dritte die Gesamtgröße der Lücken und die folgenden Spalten die minimale, maximale und durchschnittliche Lückengröße.

Eine Bewertung der Schedules anhand dieser Zahlen erweist sich als schwierig. Die Gesamtgröße der Lücken ist in allen Schedules fast gleich und wäre ein hoher Wert in einer der drei Kategorien g_{min}, g_{max} oder g_{avg} erstrebenswert, wäre *Schedule A* am hochwertigsten. Wäre dagegen eine möglichst hohe Anzahl an nutzbaren Lücken ein Qualitätsmerkmal, müsste *Schedule C* gewählt werden. In Abschnitt 5.1.2 wurde für den Schedule *Schedule C* eine bessere durchschnittliche *Response Time* für RC-Nachrichten berechnet als für *Schedule A*, so dass eine hohe Anzahl an Lücken wichtiger zu sein scheint, als möglichst große Lücken.

Problematisch an der bisherigen Betrachtung ist die Beschränkung auf kleine Schedules mit einem hohen Anteil an TT-Nachrichten. In der Realität ist dieser Anteil aber geringer, so dass die Aussagekraft der Lückengrößen geringer wird. Dies kann anhand der beiden Schedules deutlich gemacht werden, die in Abbildung 5.5 zu sehen sind, in denen der Anteil der TT-Nachrichten nur noch ein Viertel beträgt. Die Schedules haben jeweils die gleiche Anzahl an Lücken zu bieten, nur das diese unterschiedlich verteilt sind. In *Schedule D* konzentrieren sich die Nachrichten auf die erste Hälfte der Periode, während sie in *Schedule E* gleichmäßig verteilt sind. Da alle Lücken nutzbar sind, hat dies zur Folge, dass die durchschnittliche Lückengröße bei beiden Schedules gleich ist. Daraus kann man schließen, dass auch die durchschnittliche *Response Time* der RC-Nachrichten in beiden Schedules identisch wäre, wenn alle RC-Nachrichten gleich verteilt sind. Betrachtet man die beiden Hälften der Periode aus *Schedule D* getrennt, erhält man eine durchschnittliche *Response Time* von $\frac{3}{4}$ für die erste Hälfte und für die zweite einen Wert von 0, so dass sich für die gesamte Periode eine durchschnittliche *Response Time* von $\frac{9}{22}$ ergibt. Für *Schedule E* beträgt die durchschnittliche *Response Time* ebenfalls $\frac{9}{22}$ (vgl. Abschnitt 5.3.3). Da diese aber über die gesamte Periode gleich ist, unabhängig von der *Release Time* der RC-Nachrichten, ist der Schedule fairer als *Schedule D*. Da in beiden Schedules die Summe und die durchschnittliche Größe der Lücken identisch ist, scheint es vorteilhaft für einen Schedule zu sein, wenn die minimal und maximal Werte nahe an dem Durchschnittswert liegen, da aus diesem Zusammenhang auf eine ausgewogene Verteilung der Lücken geschlossen werden kann.

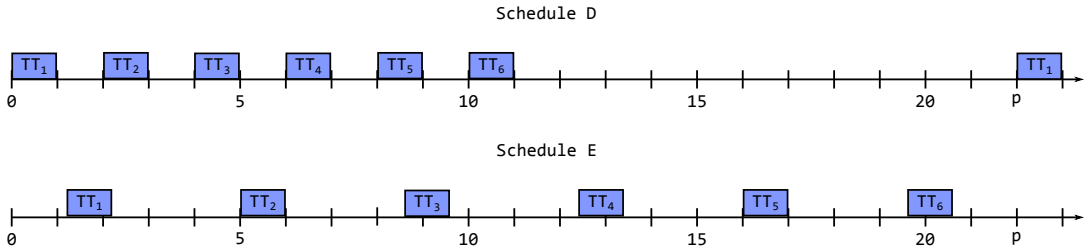

Abbildung 5.5: *Schedules mit weniger TT-Anteil und unterschiedlichen Lücken*

	Anzahl	g_{sum}	g_{min}	g_{max}	g_{avg}
Schedule D	6	16	1	11	2,6
Schedule E	6	16	2,6	2,6	2,6

Tabelle 5.2: *Anzahl und Größe der Lücken aus Abbildung 5.5*

5.3.2 Bewertung der Nachrichten-Verteilung und Varianz

Neben der Betrachtung der eigentlichen Lückengrößen ist es auch möglich, sich auf die Positionierung der TT-Nachrichten zu konzentrieren, aus der letztendlich die Lücken entstehen. Mit einer möglichst gleichmäßigen Verteilung und einer hohen Varianz der *Start Times* der TT-Nachrichten ergibt sich automatisch auch eine gleichmäßige Verteilung der Lücken. In diesem Abschnitt werden Berechnungsvorschriften vorgestellt, die es ermöglichen, Verteilung und Varianz in eine Kennzahl zu kombinieren und so für eine Bewertung des Schedules heran gezogen werden kann. Dazu wird zunächst die Verteilung, dann die Varianz und letztendlich die Kombination dieser beiden Werte berechnet.

Berechnung der Verteilung der TT-Nachrichten

Für einen optimalen Schedule ist es erstrebenswert, die TT-Nachrichten möglichst gleichmäßig über die Cluster-Periode zu verteilen. Um diesen Umstand in eine Kennzahl zu fassen, werden mehrere Schritte vorgenommen:

1. Der Mittelpunkt der Cluster-Periode wird berechnet und dient für die folgenden Abstandsberechnungen als Nullpunkt. Werte links von diesem Punkt haben einen negativen Wert, während die Punkte auf der rechten Seite positiv sind.
2. Der Abstand einer TT-Nachricht zum Nullpunkt wird ermittelt. Dazu wird zunächst der Mittelpunkt der TT-Nachricht berechnet und der Nullwert von diesem Wert subtrahiert.

3. Der Abstand wird normalisiert, indem er durch halbe Länge der Periode dividiert wird.
4. Alle Werte werden aufsummiert und der Durchschnittswert gebildet.
5. Der Betrag dieses Wertes ist die Kennzahl für die Verteilung.

Definition 27 (*Verteilung der TT-Nachrichten*)

Sei

$$T = \{t_1, \ldots, t_n\}, n \in \mathbb{N}$$

die Menge der TT-Nachrichten, s_i und f_i die jeweiligen Start Times und Finish Times, p die Länge der Cluster-Periode und

$$a = \frac{p}{2}$$

der Mittelpunkt dieser Periode und damit der logische Nullpunkt für die Abstandsberechnung. Der normalisierte Abstand einer TT-Nachricht wird durch die Funktion $x : t \rightarrow [-1, 1]$ bestimmt, mit

$$x(t_i) = \frac{s_i + \frac{(f_i - s_i)}{2} - a}{a}.$$

Damit ergibt sich die Verteilung aller Nachrichten durch die Funktion $y : T \rightarrow [0, 1]$, mit

$$y(T) = |\frac{\sum_{i=1}^{n} x(t_i)}{n}|.$$

Das Ergebnis für den Verteilungswert der Nachrichten liegt in dem abgeschlossenen Intervall $[0, 1]$. Je näher der Wert an der 0 ist, desto gleichmäßiger sind die TT-Nachrichten über die Cluster-Periode verteilt. Ist der Wert dagegen sehr nahe an der 1, liegen die Nachrichten hauptsächlich in der vorderen oder hinteren Hälfte der Periode. Für einen Schedule ist daher ein möglichst niedriger Wert anzustreben.

Um dieses Verfahren zu verdeutlichen, soll beispielhaft die Verteilung der beiden Schedules aus Abbildung 5.5 berechnet werden. Die Länge der Cluster-Periode ist $p = 22$ und die Nachrichten der Schedules *D* und *E* mit ihren *Start Times* und *Finish Times* (s_i, f_i) sind:

$$T_D = \{(0,1), (2,3), (4,5), (6,7), (8,9), (10,11)\}$$
$$T_E = \{(1.2, 2.2), (5,6), (8.6, 9.6), (12.4, 13.4), (16,17), (19.6, 20.6)\}$$

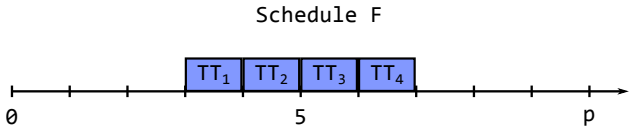

Abbildung 5.6: *Schedule mit gleichmäßiger zentrierter Verteilung*

Die Verteilungswerte ergeben sich damit zu:

$$y(T_D) = |\frac{-\frac{10,5}{11} - \frac{8,5}{11} - \frac{6,5}{11} - \frac{4,5}{11} - \frac{2,5}{11} - \frac{0,5}{11}}{6}|$$
$$= 0,5$$
$$y(T_E) = |\frac{-\frac{9,3}{11} - \frac{5,5}{11} - \frac{1,9}{11} + \frac{1,9}{11} + \frac{5,5}{11} - \frac{10,1}{11}}{6}|$$
$$= 0,13$$

Mit diesen Ergebnissen wird deutlich, dass *Schedule E* eine deutlich bessere Verteilung der TT-Nachrichten aufweist als *Schedule D*, was auch mit bloßem Auge leicht zu erkennen ist und mit den Ergebnissen aus Abschnitt 5.3.1 korreliert. Das jedoch ein perfektes Ergebnis in der Verteilung für einen guten Schedule nicht ausreichend ist, wird in Abbildung 5.6 deutlich. Obwohl sich für den *Schedule F* ein perfekter Verteilungswert von

$$y(T_F) = |\frac{-\frac{1,5}{5} - \frac{0,5}{5} + \frac{0,5}{5} + \frac{1,5}{5}}{4}|$$
$$= 0$$

errechnet, ist leicht zu erkennen, dass die Verteilung nicht optimal ist. Für eine korrekte Beurteilung muss auch die Varianz der Nachrichten über die Cluster-Periode beachtet werden.

Berechnung der Varianz der TT-Nachrichten

In den beiden vorhergehenden Abschnitten wurde deutlich gemacht, dass nicht nur die Verteilung der TT-Nachrichten, sondern auch deren Streuung für einen ausgewogenen Schedule wichtig ist, so dass sich die nutzbaren Lücken gleichmäßig über die gesamte Cluster-Periode verteilen. Für eine optimale Varianz haben die Lücken zwischen den TT-Nachrichten möglichst die gleiche maximale Größe. Um diese Varianz in einer Kennzahl auszudrücken, werden folgende Schritte durchgeführt:

1. Berechnung des möglichen Raums für Lücken.

2. Die Anzahl der möglichen Lücken ermitteln und die optimale durchschnittliche Größe der Lücken berechnen.
3. Die Differenz in der Größe der aktuellen Lücken zum optimalen Durchschnitt feststellen und aufsummieren.
4. Die Summe bildet die Kennzahl für die Varianz.

Definition 28 (*Varianz der TT-Nachrichten*)

Sei

$$T = \{t_1, \ldots, t_n\}, n \in \mathbb{N}$$

die Menge der TT-Nachrichten eines Schedules S, c_i die jeweilige Computation Time der Tasks und p die Länge der Cluster-Periode. Der maximal mögliche nutzbare Raum für Lücken L_{sum} errechnet sich dann durch:

$$L_{sum} = p - \sum_{i=1}^{n} c_i.$$

Die maximale Anzahl an Lücken innerhalb des Schedules ergibt sich aus der Anzahl der TT-Nachrichten n, so dass sich die optimale durchschnittliche Größe L_{avg} der Lücken berechnet durch:

$$L_{avg} = \frac{L_{sum}}{n}.$$

Weiterhin sei

$$\Lambda = \{\lambda_1, \ldots, \lambda_m\}, m \in \mathbb{N}$$

die aus T ermittelte Menge der nutzbaren Lücken und g die Funktion zur Berechnung der Lückengröße. Die Varianz der Nachrichten wird bestimmt durch die Funktion $z : \Lambda \to \mathbb{R}$, mit

$$z(\Lambda) = \sum_{i=1}^{m} |L_{avg} - g(\lambda_i)|$$

Die Varianz in der Verteilung der TT-Nachrichten wird durch diesen Algorithmus indirekt über die Größe der einzelnen Lücken bestimmt, da diese in Wechselwirkung miteinander stehen. Je geringer der Unterschied der aktuellen Lückengrößen zu der optimalen durchschnittlichen Größe ist, also je kleiner das Ergebnis z ist, desto besser ist der Schedule in Hinsicht auf das Optimierungsziel. Zur Verdeutlichung dieses Verfahrens wird die Varianz erneut an den beiden Schedules aus Abbildung 5.5 bestimmt.

Die Schedules *D* und *E* beinhalten jeweils die Taskmengen T_D und T_E, die aus $n = 6$ Tasks mit einer *Computation Time* von $c_i = 1$ bestehen. Die Länge der Cluster-Periode ist $p = 22$. Der maximal nutzbare Raum für Lücken errechnet sich somit zu

$$L_{sum} = 22 - \sum_{i=1}^{6} 1 = 16$$

und die daraus folgende optimale durchschnittliche Größe für die Lücken ist

$$L_{avg} = \frac{16}{6} = \frac{8}{3} \approx 2,6.$$

Die Mengen Menge der nutzbaren Lücken in den Schedules ergibt sich aus T_D und T_E per Definition zu:

$$\Lambda_D = \{(1,2),(3,4),(5,6),(7,8),(9,10),(11,22)\}$$
$$\Lambda_E = \{(2.2,5),(6,8.6),(9.6,12.4),(13.4,16),(17,19.6),(20.6,23.2)\}$$

Anhand dieser Mengen ergibt sich nun durch die Funktion z die Varianz folgend:

$$\begin{aligned} z(\Lambda_D) &= |2,6-1| + |2,6-1| + |2,6-1| + |2,6-1| + |2,6-1| + |2,6-11| \\ &= 16,4 \\ z(\Lambda_E) &= |2,8-2,6| + |2,6-2,6| + |2,8-2,6| + |2,6-2,6| + |2,6-2,6| + |2,6-2,6| \\ &= 0,4 \end{aligned}$$

Das Ergebnis fällt aus wie erwartet und belegt für den *Schedule E* eine bessere Varianz der TT-Nachrichten als in *Schedule D*. Mit einem Wert von $0,4$ ist die Streuung in *Schedule E* sehr nah am Optimum, welches nur aufgrund von gerundeten *Start Times* im Schedule nicht erreicht wird.

5.3.3 Berechnung der durchschnittlichen RC-Response Time

Die vierte Möglichkeit zur Bewertung des Schedules ist die Berechnung der durchschnittlichen *Response Time* der RC-Nachrichten, wie sie zum Beispiel in Abschnitt 5.1.2 durchgeführt wurde. Das Ergebnis definiert den Zeitraum, den eine RC-Nachricht durchschnittlich nach ihrer *Release Times* warten muss, bevor sie versendet wird. Diese Angabe bezieht sich immer auf die Größte im System auftretende RC-Nachricht und behält Gültigkeit unter der Annahme, dass die *Release Times* der RC-Nachrichten gleichmäßig über die gesamte Cluster-Periode verteilt sind. Eine weitere Bedingung ist, dass die RC-Nachrichten seriell auftreten, das heißt, dass keine RC-Nachricht durch eine andere blockiert wird. Sollten mehrere RC-Nachrichten parallel auftreten, verlängert sich die *Response Time* der anderen RC-Nachrichten entsprechend. In diesem Abschnitt wird nun die Berechnung dieser Zeit, wie sie bereits in Abschnitt 5.1.2 und 5.3.1 vorgenommen wurde, formal beschrieben.

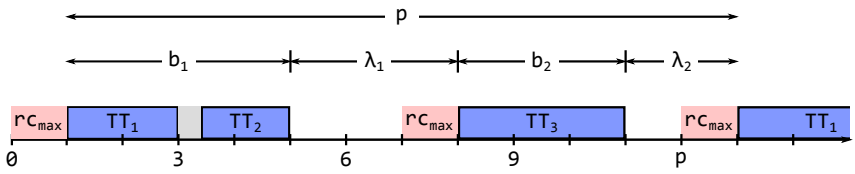

Abbildung 5.7: *Schedule mit blockierten und nutzbaren Räumen*

Als Basis zur Berechnung der durchschnittlichen *Response Time* der RC-Nachrichten dient die Menge Λ der nutzbaren Lücken, da sich aus dieser die blockierten Räume im Schedule ableiten lassen. Würde man direkt von den TT-Nachrichten als blockierte Stellen ausgehen, könnten Lücken übersehen werden, die zu klein für eine RC-Nachricht sind und dadurch auch als blockierend gelten. In Abbildung 5.7 werden diese Räume exemplarisch in einem Schedule aufgezeigt. Betrachtet wird jeweils immer ein Zeitraum, der die Länge p der Cluster-Periode hat und mit der *Start Time* der ersten TT-Nachricht beginnt und bis zur *Start Time* der gleichen Nachricht in der nächsten Instanz der Periode reicht. Die größte RC-Nachricht hat eine Länge von rc_{max}, so dass λ_1 und λ_2 die beiden nutzbaren Lücken darstellen. b_1 und b_2 repräsentieren entsprechend die beiden blockierten Räume. Der freie Raum zwischen den TT-Nachrichten TT_1 und TT_2 zählt dabei mit zu b_1, da er kleiner als rc_{max} ist und daher nicht für eine Übertragung genutzt werden kann. Die rot eingefärbten Zeitfenster haben die Länge rc_{max} und liegen jeweils vor einem blockierten Raum, da eine RC-Nachricht nicht mehr direkt übertragen werden kann, wenn ihre *Start Time* in diesen Zeitraum fällt. Der Sendevorgang wird in diesem Fall bis zum Ende des blockierten Raums verzögert. Dies muss bei der Berechnung der durchschnittlichen Verzögerung beachtet werden, die anhand der folgenden Schritte durchgeführt wird:

1. Aus den nutzbaren Lücken werden die blockierten Stellen errechnet.
2. Für jede blockierte Stelle wird die durchschnittliche Verzögerung berechnet.
3. Die einzelnen Zeiten werden summiert und der Durchschnitt über die gesamte Cluster-Periode gebildet.

Definition 29 (*Durchschnittliche RC-Response Time eines Schedules*)

Sei

$$\Lambda = \{\lambda_1, \ldots, \lambda_n\}, n \in \mathbb{N}$$

die Menge der nutzbaren Lücken, die aufsteigend nach Startposition der Lücken geordnet ist, ein Tupel $\lambda_i = (x_i, y_i)$ ein Element dieser Menge und p die Länge der Cluster-Periode. Die Menge B der blockierten Zeiträume ergibt sich dann durch:

$$B = \bigcup_{i=2}^{n} (x_{i-1}, y_i) \cup (y_n - p, x_1).$$

Sei rc_{max} die Länge der größten RC-Nachricht. Dann errechnet sich die Verzögerung für eine blockierte Stelle $b_j = (x_j, y_j)$ durch die Funktion $v : b \to \mathbb{R}$, mit

$$v(b_j) = \frac{(rc_{max} + y_j - x_j)^2}{2}.$$

Die durchschnittliche Response Time einer RC-Nachricht über die gesamte Cluster-Periode berechnet sich dann durch die Funktion $w : B \to \mathbb{R}$, mit

$$w(B) = \frac{\sum_{j=1}^{|B|} v(b_j)}{p}.$$

Für den Fall, dass $\Lambda = \emptyset$ ist, ist $B = \{(0, p)\}$ und das Ergebnis ist $w(B) = \infty$.

Für den Schedule aus Abbildung 5.7 mit der Cluster-Periode $p = 12$ ergeben sich die folgenden Mengen:

$$\Lambda = \{(5,8), (11,13)\},$$
$$B = \{(1,5), (8,11)\}.$$

Die durchschnittliche *Response Time* für die maximalen RC-Nachricht $rc_{max} = 1$ berechnet sich dann durch:

$$w(B) = \frac{\frac{(1+5-1)^2}{2} + \frac{(1+11-8)^2}{2}}{12}$$
$$= \frac{41}{24} \approx 1,7$$

5.3.4 Zusammenfassung und Fazit

In diesem Abschnitt wurden vier unterschiedliche Methoden vorgestellt, einen Schedule zu bewerten: Lückengrößen, Verteilung und Varianz der TT-Nachrichten und die durchschnittliche *Response Time* für RC-Nachrichten. Da das Optimierungsziel für den Schedule ist, die

Response Time der RC-Nachrichten zu minimieren, liegt es zunächst nahe, diese Berechnung zur Bewertung eines Schedules zu nutzen. Problematisch ist dabei die Tatsache, dass das Ergebnis nur von der insgesamt nutzbaren Lückengröße g_{sum} abhängt und die Verteilung der TT-Nachrichten keinen Einfluss ausübt, so lange zwischen den einzelnen TT-Nachrichten genug Platz für die maximale RC-Nachricht ist. Die beiden Schedules aus Abbildung 5.5 liefern das gleiche Ergebnisse für die *Response Time*, obwohl die Verteilung in *Schedule E* deutlich besser und fairer ist.

Der Weg über die Lückengröße liefert die Werte $g_{sum}, g_{min}, g_{max}, g_{avg}$, über die sich eine gesicherte Aussage über die Qualität des Schedules treffen lässt. Es gilt den Schedule in die Richtung zu optimieren, dass der Wert g_{sum} maximiert wird und gleichzeitig die Werte g_{min} und g_{max} dem von g_{avg} anzunähern. Im besten Fall haben diese drei Werte die gleiche Größe, so dass sich eine ausgewogene Verteilung der TT-Nachrichten und Lücken ergibt. Beispielhaft ist dies wieder an der Abbildung 5.5 und in den Ergebnissen in Tabelle 5.2 zu sehen. In *Schedule E* sind alle Lücken nutzbar, so dass g_{sum} maximal ist. Die anderen drei Werte sind identisch, so dass die Verteilung und Varianz der TT-Nachrichten optimal ist. Der Nachteil an der Bewertung anhand der Lückengröße ist die Tatsache, dass insgesamt vier Werte betrachtet werden müssen und keine eindeutige Kennzahl vorliegt.

Diese Kennzahl liefert die Berechnung der Verteilung und der Varianz der TT-Nachrichten, welche auf den Daten zu den Lückengrößen aufbaut und so eine Art Erweiterung darstellt. Diese Berechnungen liefern jeweils einen einzelnen Wert als Ergebnis. In Abschnitt 5.3.2 wurde jedoch gezeigt, dass ein gutes Ergebnis bei der Berechnung der Verteilung eines Schedules nicht zwingend auf einen guten Schedule hinweist, wie es am *Schedule F* in Abbildung 5.6 zu sehen ist. Es fehlt ein Bezug zu den Lücken im Schedule. Die Varianz dagegen korreliert direkt mit den Lückengrößen aus Abschnitt 5.3.1 und ermöglicht daher eine fundierte Bewertung des Schedules in Hinsicht auf die Verteilung der Lücken. Die Bewertungsfunktion ist damit folgend definiert:

Definition 30 (*Bewertungsfunktion für einen TT-Ethernet Schedule*)

Sei

$$T = \{t_1, \ldots, t_n\}, n \in \mathbb{N}$$

die Menge der TT-Nachrichten,

$$\Lambda = \{\lambda_1, \ldots, \lambda_m\}, m \in \mathbb{N}$$

die Menge der gültigen Lücken und p die Länger der Cluster-Periode. Dann ergibt sich die Qualität eines Schedules S durch die Funktion $z : S \to \mathbb{R}$, mit

$$z(\Lambda) = \sum_{i=1}^{m} |L_{avg} - g(\lambda_i)|$$

Dabei gilt: ist ein Schedule A besser als Schedule B, folgt daraus, dass $z(S_A) < z(S_B)$ ist.

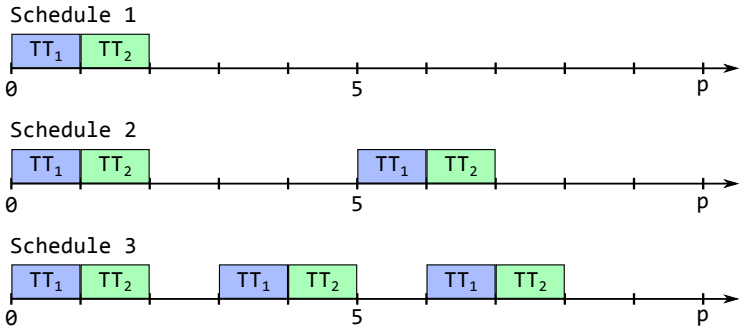

Abbildung 5.8: *Drei Schedules mit steigender Periodizität*

5.4 Varianz der Nachrichten bei steigender Periodizität

In den vorherigen Abschnitten wurde ein Aspekt des TT-Ethernet Protokolls vernachlässigt, um die Komplexität der Beispiele gering zu halten. Nämlich der Umstand, dass die TT-Nachrichten periodisch versendet werden können. In den bisherigen Beispielen wurde immer davon ausgegangen, dass die Perioden der einzelnen Nachrichten mit der Cluster-Periode übereinstimmen. Dies ist insofern kein Problem, da der Wahrheitsgehalt der getroffenen Aussagen durch steigende Periodizität nicht betroffen ist. Bei der Berechnung der Varianz der Nachrichten spielt es keine Rolle, ob eine Nachricht zum ersten Mal auftritt oder eine neue Instanz einer bereits versendeten Nachricht ist.

Es gilt jedoch zu beachten, dass Schedules, deren Nachrichten eine Periodizität aufweisen, bei der Berechnung der Varianz ein besseres Ergebnis erzielen. Dieses Verhalten kann anhand der Abbildung 5.8 näher erläutert werden, in der drei Schedules dargestellt sind, in denen zwei Nachrichten TT_1 und TT_2 zum Versenden geplant sind. In *Schedule 1* haben die beiden Nachrichten jeweils nur eine Periodizität von 1, in *Schedule 2* eine Periodizität von 2 und in *Schedule 3* schließlich eine von 3, so dass TT_1 und TT_2 in der Cluster-Periode p jeweils drei mal versendet werden. In der Tabelle 5.3 stehen die Ergebnisse aus der Berechnung der Varianz der drei Schedules und es ist deutlich zu sehen, dass der Abstand zum Optimum 0 mit steigender Periodizität sinkt, auch wenn noch keine künstlichen Lücken in den Schedule eingefügt wurden. Schedules mit hoher Periodizität bieten daher weniger Raum für Optimierungen, benötigen diese aber auch in gleichem Maße weniger.

	Schedule 1	Schedule 2	Schedule 3
Varianz	0,8	0,6	0,4

Tabelle 5.3: *Bewertung der Schedules aus Abbildung 5.8*

5.5 Einfluss der Einschränkungen auf die Bewertungsfunktion

In Abschnitt 5.2 wurde das Modell, auf dem die Bewertungsfunktion entwickelt wurde, in Hinsicht auf die *Arrival Times* der RC-Nachrichten eingeschränkt. Es bleibt daher zu klären, auf welche Art und Weise dies Einfluss auf die entwickelten Bewertungsfunktionen hat. Eingeschränkt wurde das Modell, um Aussagen über die *Response Times* der RC-Nachrichten zu treffen, die nur eine Gültigkeit haben, so lange sich keine RC-Nachrichten aufstauen. Es ist damit anzunehmen, dass die Bewertungsfunktion aus Abschnitt 5.3.3, die direkt die *Response Time* der RC-Nachrichten berechnet, durch diese Einschränkungen keine gesicherten Ergebnisse liefert und sich in der Praxis anders verhält. Bei den anderen Metriken, die sich nur auf die Größe und Position der Lücken beziehen, ist dies jedoch nicht der Fall. So wird auch bei der Berechnung der Varianz kein Bezug zu dem Übertragungsverhalten der ereignisbasierten Nachrichten gezogen. Die Aussagen basieren damit alleine auf der Annahme, dass entsprechende Lücken zu einem besseren Schedule führen. Dies behält Gültigkeit, so lange die RC-Nachrichten gleichmäßig über die Cluster-Periode verteilt sind. Obwohl dies im voraus nicht bestimmbar ist, kann man davon ausgehen, dass die stochastisch auftretenden Nachrichten über einen längeren Zeitraum ein gleichmäßig verteiltes Bild erzeugen.

5.6 Berechnung des optimalen Schedules

Ein gutes Werkzeug zur Einordnung der Qualität eines Schedules ist die Berechnung des optimalen Schedules, um eine Referenz zu haben, an der sich die erzeugten Schedules vergleichen lassen. Wendet man die Bewertungsfunktion auf einen optimalen Schedule S mit den nutzbaren Lücken Λ an, so ergibt sich die Varianz immer zu:

$$z(\Lambda) = 0,$$

da die TT-Nachrichten perfekt über die Cluster-Periode verteilt sind und die Lücken zwischen den Nachrichten alle die gleiche maximale Größe besitzen. In der Realität wird sich solch ein Schedule in den meisten Fällen nicht realisieren lassen, da bei der Platzierung der Nachrichten auf deren Deadlines Rücksicht genommen werden muss. Die Lücken werden daher selten alle auf die maximale Größe festlegbar sein, so dass sich für die Varianz ein Wert von

$$z(\Lambda) > 0$$

ergeben wird. Die Berechnung des optimalen Schedules liefert daher nicht nur einen Hinweis, wie weit der aktuelle Schedule vom Optimum entfernt ist, sondern liefert auch gleichzeitig die Größe für die optimale Lücke. Dieses Ergebnis kann als Richtwert für die beim Scheduling eingesetzten Lücken genutzt werden. So kann es sich beim iterativen Scheduling als sinnvoll erweisen, die Lückengröße nicht beginnend von 0 zu steigern, sondern zu versuchen, von der

optimalen Größe ausgehend, die Lücken in jedem Durchgang zu verkleinern, bis ein gültiger Schedule gefunden wurde. Für einen Schedule S berechnet sich die optimale Lückengröße durch:

Definition 31 (*Berechnung der optimalen Lückengröße*)

Sei

$$T = \{t_1, \ldots, t_n\}, n \in \mathbb{N}$$

die Menge der TT-Nachrichten des Schedules S, c_i die Computation Time eines Tasks und p die Länge der Cluster-Periode. Die Anzahl a der möglichen Lücken ergibt sich aus der Kardinalität der Taskmenge

$$a = |T|$$

und der vorhandene Platz b im Schedule zu

$$b = p - \sum_{i=1}^{n} c_i.$$

Die maximal mögliche Größe ϵ für die Lücken berechnet sich nun durch

$$\epsilon = \frac{b}{a}.$$

5.7 Mögliche Nachteile der Lücken

Künstlich erzeugte Lücken im Schedule wurden in diesem Kapitel als Möglichkeit etabliert, die Qualität eines zu erhöhen. Bevor dieses Prinzip genutzt wird, um passende Heuristiken zu entwickeln, muss noch ein Blick auf mögliche Nachteile geworfen werden, die durch dieses Verfahren entstehen können. Ein wichtiger Punkt ist, dass durch Lücken Bandbreite verschwendet werden kann. Dieses Problem tritt auf, wenn die Größe der Lücken dazu führt, dass RC- oder BE-Nachrichten einer bestimmten Größe nicht mehr vor der nächsten TT-Nachricht übertragen werden können. Anhand der Abbildung 5.9 soll dieser Zusammenhang näher erläutert werden.

In der Abbildung 5.9 sind zwei Schedules zu sehen, in denen jeweils drei TT- und RC-Nachrichten übertragen werden, die zyklisch innerhalb der Periode p übertragen werden. Die TT-Nachrichten besitzen alle die Deadline d und die RC-Nachrichten haben unterschiedliche Prioritäten, wobei RC_1 die höchste und RC_3 die niedrigste Priorität hat. RC_1 und RC_2 haben die Länge 1, während RC_3 die Länge $1{,}5$ hat. Die beiden Schedules werden nun mit unterschiedlichen Strategien geplant. *Schedule 1* wird ohne und *Schedule 2* mit

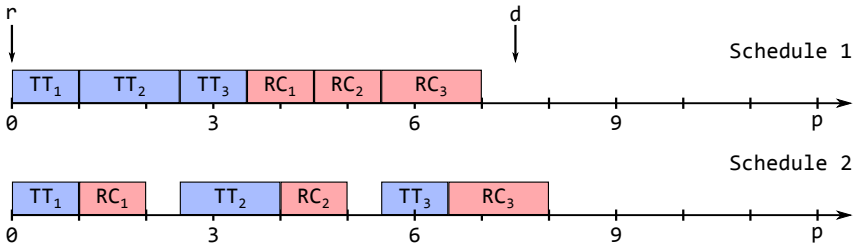

Abbildung 5.9: *Verschwendung von Bandbreite beim Einfügen von Lücken*

Lücken berechnet. Da die Größe der RC-Nachrichten im Vorfeld bekannt ist, orientieren sich die geplanten Lücken an der maximalen RC-Nachricht RC_3. Nehmen wir für dieses Beispiel nun an, dass die RC-Nachrichten alle zufällig die *Release Time r* besitzen, also direkt zu Beginn der Periode bereit zum Versenden sind, ergibt sich das dargestellte Bild. In *Schedule 1* werden zunächst die TT- und darauf folgend die RC-Nachrichten in Reihenfolge ihrer Prioritäten übertragen. Im Gegensatz dazu ist in *Schedule 2* genügend Zeit zwischen den TT-Nachrichten, um jeweils eine RC-Nachricht zu übertragen. In diesem Fall ist nun zu erkennen, dass die Bandbreite nicht optimal genutzt wird, da die Lücken nicht groß genug sind, um RC_1 und RC_2 aufzunehmen. Es werden jeweils $0{,}5$ Zeiteinheiten an Übertragungszeit verschwendet und gegen verbesserte *Response Times* für RC_1 und RC_2 eingetauscht.

Dieses Beispiel ist natürlich stark konstruiert und spiegelt nicht den generellen Fall wieder. Ob dieser Fall eintritt hängt stark von den tatsächlichen *Release Times* der RC-Nachrichten ab. Zudem darf der BC-Traffic nicht vergessen werden, der auch zufällig so auftreten kann, dass er die restlichen Lücken auffüllt. Mit diesem speziellen Beispiel soll nur darauf hingewiesen werden, dass durch geplante Lücken auch Nachteile entstehen können. Bei einer hohen Auslastung der Bandbreite könnte es in so einem Fall dazu kommen, dass nicht alle Nachrichten fristgerecht versendet werden können.

5.8 Scheduling von virtuellen Links

In der bisherigen Diskussion wurden immer nur einzelne Schedules betrachtet, obwohl es sich bei einem TT-Ethernet Netzwerk um ein Mehrprozessorsystem handelt, so dass im Endeffekt eine Menge von Schedules erzeugt werden. Daher ist es sinnvoll, die Schedules auch prozessorübergreifend zu untersuchen, insbesondere die Prozessoren, die innerhalb eines virtuellen Links gruppiert sind. Betrachtet man dazu ein einfaches Beispiel, in dem eine RC-Nachricht über einen virtuellen Link übertragen werden muss, der aus zwei Prozessoren besteht: *Link A* und *Link B*. Zusätzlich werden auf der gleichen Übertragungsstrecke noch drei

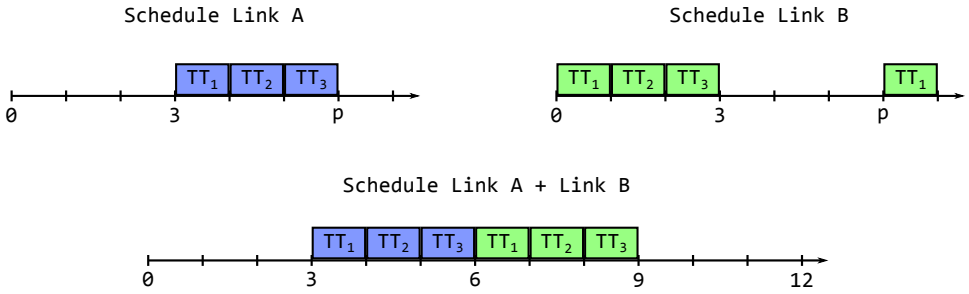

Abbildung 5.10: *Kombinierter Schedule eines virtuellen Links*

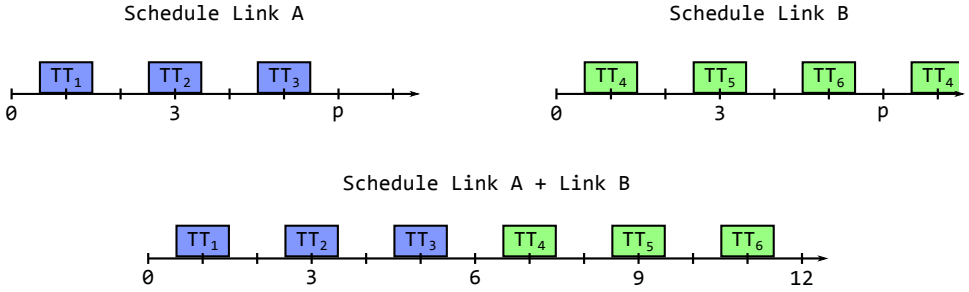

Abbildung 5.11: *Optimierter Schedule eines virtuellen Links*

TT-Nachrichten mit der Periode $p = 6$ gesendet. Im schlechtesten Fall sehen die einzelnen Schedules für die beiden Links aus wie in Abbildung 5.10 dargestellt.

In dieser Konfiguration ist die hintere Hälfte in *Schedule Link A* und die vordere Hälfte in *Schedule Link B* durch TT-Nachrichten blockiert, so dass sich übergreifend ein langes Zeitfenster ergibt, dass für andere Übertragungen nicht genutzt werden kann. Dies führt zu großen Latenzen beim Versenden von RC- und BE-Nachrichten, sollten sie in diesen Zeitraum fallen. Die Verteilung der TT-Nachrichten über den Zeitraum der Cluster-Periode sollte daher nicht nur innerhalb des einzelnen Schedules ausgewogen sein, sondern auch im Bezug auf folgende Links im Netzwerk.

Durch die Ausrichtung der Bewertungsfunktion auf eine zentrierte Verteilung der TT-Nachrichten innerhalb des Schedules, läuft die Optimierung genau auf diese Konstellation hinaus. In der Abbildung 5.11 sind die Schedules in idealer Form mit der höchsten möglichen Bewertung zu sehen. Auch wenn diese Idealfälle in der Realität selten produzierbar sind, zeigen sie doch, in welche Richtung sich die Schedules entwickeln werden und das durch die Optimierungen auch das Problem von lange blockierten Zeitfenstern prozessorübergrei-

fend gelöst wird. Große Latenzen bei RC- und BE-Nachrichten können daher durch diesen Problemfall nicht auftreten.

5.9 Entwicklung der Scheduling-Heuristik

Der nächste Schritt nach dem Festlegen des Optimierungsziels und der Definition der dazugehörigen Bewertungsfunktion, folgt nun die Entwicklung einer entsprechenden Heuristik, um einen Schedule nach diesen Richtlinien zu erzeugen. Zu diesem Zweck werden in diesem Abschnitt mehrere Lösungsansätze diskutiert, die unterschiedliche Strategien verfolgen. Ein wesentlicher Punkt beim Scheduling ist die Auswahl des nächsten Tasks, der auf einem freien Prozessor ausgeführt werden soll. Auf diesen Aspekt wird in diesem Abschnitt nicht im Detail eingegangen, da die eingesetzten Heuristiken bereits ausgiebig in der vorhandenen Literatur erläutert wurden. Eine grundlegende Beschreibung der Heuristiken und deren Funktionsweise wurde in Abschnitt 3.4 vorgenommen. Stattdessen konzentriert sich dieser Abschnitt darauf, eine Reihe von Verfahren zu entwickeln, um die in den vorherigen Abschnitten erwähnten Lücken zu erzeugen.

5.9.1 Verfahren A: Dummy-Tasks

Der erste Ansatz basiert auf der Idee, die Lücken im Schedule der TT-Nachrichten durch sogenannte Dummy-Tasks zu erzeugen. Ein Dummy-Task wird jeweils immer nach einer TT-Nachricht eingeplant, um einen fest definierten Zeitraum zwischen zwei TT-Nachrichten zu blockieren und somit für RC- und BE-Nachrichten im laufenden Betrieb des Systems freizuhalten. Die Größe des zu blockierenden Zeitraums wird durch die *Computation Time* des Tasks konfiguriert. Um möglichst große Lücken zwischen den TT-Nachrichten zu erreichen, erfolgt dieses Verfahren iterativ, so dass die *Computation Time* der Dummy-Tasks schrittweise erhöht werden kann.

Sei c_d die *Computation Time* der Dummy-Tasks, c_{rc} die Größe der maximalen RC-Nachricht und X die gewählte Scheduling Heuristik. Zur Erstellung des Schedules sind für jeden Prozessor die folgenden Schritte auszuführen:

1. Die Größe der Dummy-Tasks wird auf $c_d = 0$ festgelegt.
2. Sortiere die Liste der ausführbaren Tasks nach den Vorgaben der Heuristik X.
3. Wähle den ersten Task aus der Liste und führe ihn aus.
4. Prüfe, ob der Schedule gültig ist. Sollte dies nicht der Fall sein, kann die Heuristik mit Schritt 9 beendet werden.

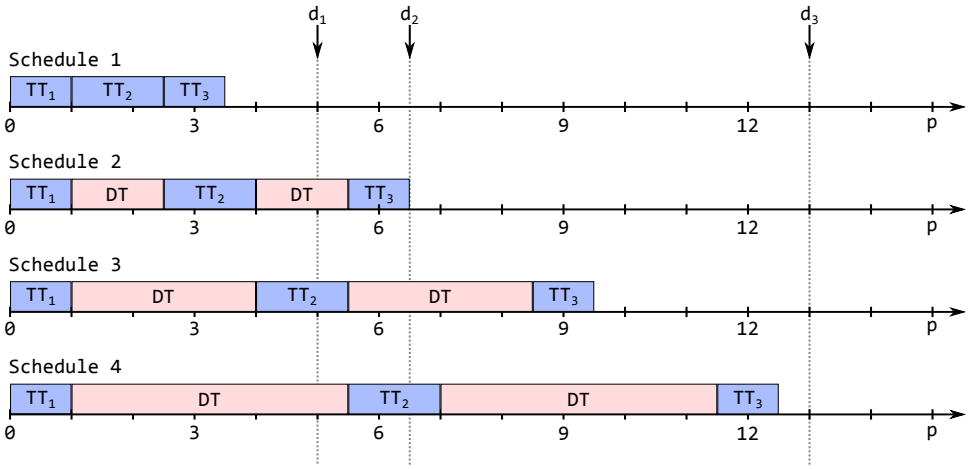

Abbildung 5.12: *Schedule eines Prozessors mit steigenden Dummy Task Größen*

	Schedule 1	Schedule 2	Schedule 3
Qualität	1,02	0,62	0,18

Tabelle 5.4: *Bewertung der Schedules aus Abbildung 5.12*

5. Führe einen Dummy-Task aus.
6. Füge eventuell neu ausführbare Tasks zur Liste hinzu.
7. Wiederhole diesen Vorgang ab Schritt 2, bis alle Tasks ausgeführt wurden.
8. Setze das System wieder auf den Anfangszustand und vergrößere die Dummy-Tasks um $c_d = c_d + c_{rc}$ und erzeuge einen weiteren Schedule ab Schritt 2.
9. Wähle aus der Menge der erzeugten und gültigen Schedules den Besten aus.

Die Abbildung 5.12 demonstriert die Resultate dieses Verfahrens, angewendet auf einen Prozessor, auf dem die drei Tasks TT_1, TT_2 und TT_3 ausgeführt werden müssen. Die Deadlines der Tasks sind jeweils mit d_1, d_2 und d_3 gekennzeichnet. Die Ausführungsreihenfolge wurde anhand der *Earliest Deadline First* Heuristik festgelegt. Insgesamt wurden vier Iterationen durchgeführt und die Größe der Dummy-Tasks (DT) jeweils um die Größe der maximalen RC-Nachricht erhöht. Nach dem vierten Durchgang wird das Verfahren abgebrochen, da *Schedule 4* durch das Verletzen der Deadline d_2 ungültig ist. Aus den drei gültigen Schedules wird nun der Beste ausgewählt. Die Tabelle 5.4 zeigt die Ergebnisse für die Schedules die anhand der Bewertungsfunktion aus Definition 30 berechnet wurden. Demnach ist *Schedule 3* der qualitativ hochwertigste.

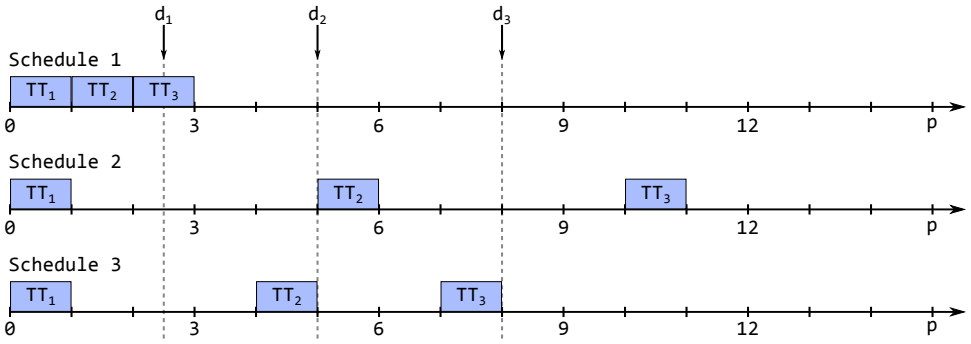

Abbildung 5.13: *Stufenweise Erstellung eines Schedules*

5.9.2 Verfahren B: Erzeugen des Schedules in Stufen

Der zweite Ansatz erzeugt einen Schedule, indem es ein Verfahren mit mehreren Stufen befolgt, welches bereits in ähnlicher Form für das Planen von periodischen Tasks eingesetzt wurde (vgl. Fohler (1995)). In der ersten Stufe, der Scheduling-Stufe, wird mit der Hilfe einer einfachen Heuristik ein Schedule ohne künstliche Lücken berechnet, so dass für jeden Prozessor die Ausführungsreihenfolge der Tasks festgelegt wird und die Abhängigkeiten aufgelöst werden. Auf dieser Basis werden in der Optimierungsstufe die einzelnen Tasks im Schedule zeitlich nach hinten verschoben, um möglichst optimale Lücken zwischen den Tasks zu erzeugen. Für jeden Prozessor werden die folgenden Schritte durchgeführt:

1. Berechnung der *Start Time* s_i aller Tasks durch eine Scheduling Heuristik.
2. Bestimmen der optimalen Lückengröße zwischen den Tasks.
3. Zeitliches Verschieben der Tasks zwischen der *Start Time* s_i und der *Deadline* d_i, um die Abstände zwischen den Tasks an die optimale Lückengröße anzupassen.

Anhand der Abbildung 5.13 kann dieses Verfahren am Beispiel eines einfachen Schedules mit drei Tasks näher erläutert werden. *Schedule 1* spiegelt den Zustand des Schedules wieder, nachdem in der ersten Stufe die Ausführungsreihenfolge der Tasks festgelegt wurde. Für den *Schedule 2* wird zunächst nach Definition 31 die optimale Lückengröße berechnet, mit:

$$a = 3,$$

$$b = 15 - \sum_{i=1}^{3} 1 = 12,$$

$$\epsilon = \frac{12}{3} = 4,$$

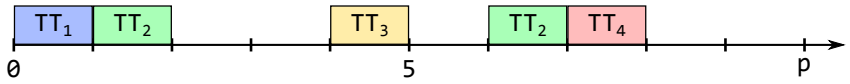

Abbildung 5.14: *Schedule mit vier Tasks*

so dass die Tasks so verschoben werden, dass sie jeweils 4 Zeiteinheiten Abstand zwischen sich haben und dadurch den optimale Schedule präsentieren. Dies führt jedoch zu einer Verletzung der Deadline bei den Tasks TT_2 und TT_3, so dass die *Start Times* dieser Tasks angepasst werden müssen und so weit zurück nach vorne verschoben werden, bis die Deadline nicht mehr verletzt wird. Der daraus folgende finale *Schedule 3* demonstriert nach Definition 30 den bestmöglichen gültigen Schedule für das Problem.

	Schedule 1	Schedule 2	Schedule 3
Varianz	1,06	0	0,4

Tabelle 5.5: *Bewertung der Schedules aus Abbildung 5.13*

Während dieses Vorgehen für einfache Probleme eine schnelle Lösung und Optimierung bietet, steigt die Komplexität bei höheren Periodizitäten der Nachrichten. Dies liegt begründet in der Tatsache, dass beim Verschieben einer periodischen Nachricht alle Instanzen dieser Nachricht verschoben werden müssen und somit für alle Instanzen überprüft werden muss, dass die neuen *Start* und *Finish Times* sich mit keiner anderen Nachricht im Schedule überschneiden. Dieses Suchproblem verhält sich ähnlich zum ursprünglichen Scheduling-Problem, so dass weitere Heuristiken gebraucht werden, um neue gültige Lösungen zu finden. Dies Unterscheidet den Einsatz dieses mehrstufigen Verfahrens vom Einsatz bei den bisherigen Problemen. Beispielhaft kann dies anhand der Abbildung 5.14 demonstriert werden, die einen Schedule mit drei Tasks zeigt, von denen TT_2 periodisch ausgeführt wird. Für einen ersten Optimierungsversuch bietet es sich an, die erste Instanz von TT_2 nach $(2,3)$ zu verschieben. Dies hat jedoch zur Folge, dass auch die zweite Instanz nach $(7,8)$ verschoben werden muss und in Kollision mit dem Task TT_4 tritt. Die Suche nach einer gültigen Lösung in der Optimierungsstufe kann daher bei steigenden Periodizitäten sehr viel Zeit in Anspruch nehmen.

5.9.3 Verfahren C: Gruppieren von Tasks

Das Gruppieren von Tasks ist eher eine Erweiterung zu den bestehenden Verfahren, als ein alleinstehender Ansatz. Beim Einsatz dieser Technik wird zunächst versucht, mehrere Tasks zu gruppieren, bevor Lücken im Schedule erzeugt werden. Dies bedeutet, dass nicht mehr zwischen allen Tasks eine Lücke eingefügt wird, sondern nur noch zwischen Gruppen von

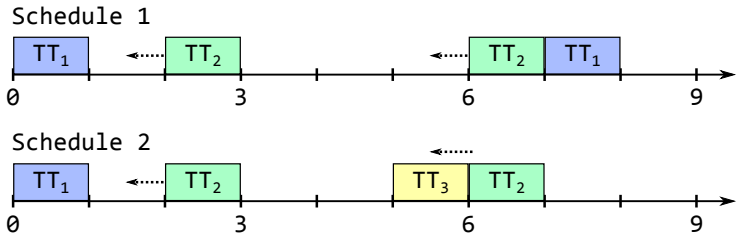

Abbildung 5.15: *Zwei Schedules mit zwei und drei Tasks*

Tasks. Bei der folgenden Berechnung der Varianz werden die gruppierten Tasks als Einheit betrachtet, damit das Ergebnis nicht verfälscht wird. Dies kann besonders ein Vorteil sein, wenn viele kleine Tasks im Schedule sind, wie dies bei einem TT-Ethernet System der Fall sein kann, da der TT-Traffic oft nur einen geringen Anteil am Gesamtaufkommen hat und zusätzlich die heutigen Systeme nur aus Minimum-Ethernet-Frames bestehen (vgl. Abschnitt 6.1.1). Den in Abschnitt 5.7 erwähnten Nachteil der Bandbreiten Verschwendung kann mit diesem Verfahren entgegen getreten werden.

Ähnlich wie in dem vorherigen Verfahren, erschwert die Periodizität der Tasks das Gruppieren derselben. Möchte man eine bestimmte Menge von zeitlich nahe angeordneten Tasks näher zusammen legen, so dass keine Lücken zwischen ihnen mehr bestehen, müssen auch alle anderen Instanzen dieser Tasks in gleichem Maße verschoben werden. Anhand der Abbildung 5.15 lassen sich zwei Probleme aufzeigen, die beim Gruppieren von Tasks auftreten können. *Schedule 1* zeigt einen Ausschnitt einer Cluster-Periode in der zwei periodische Tasks TT_1 und TT_2 geplant sind. In diesem Schedule könnten für eine Optimierung die ersten beiden Instanzen der Tasks gruppiert werden und der Task TT_2 von $(2,3)$ nach $(1,2)$ verschoben werden. Dies führt jedoch dazu, dass auch die zweite Instanz dieses Tasks von $(6,7)$ nach $(5,6)$ wandert und nun eine Lücke zwischen den beiden nachfolgenden Instanzen auftritt. Anstatt eine ungewollte Lücke zu schließen, wurde sie nur verschoben. Das zweite Problem hängt mit der Gültigkeit des Schedules zusammen. Wie am *Schedule 2* zu sehen ist, würde in diesem Fall das Verschieben von Task TT_2 auf die Position $(1,2)$, bzw. $(5,6)$ dazu führen, dass der Schedule ungültig wird, da in diesem Schedule ein Task TT_3 zu dieser Zeit geplant ist.

Äquivalent zum Mehrstufigen Scheduling-Verfahren steigt die Schwierigkeit beim Gruppieren mit der Periodizität der Tasks. Gleichzeitig bleibt die Frage ungeklärt, ob wirklich eine sinnvolle Optimierung des Schedules durch Gruppieren erzielt werden kann. Da es nicht möglich ist, gezielt einzelne Tasks zu verschieben, sondern gezwungen ist, eine Menge von Task-Instanzen in ihren Ausführungszeiten zu ändern. Dies führt erneut zu einem Problem aus der Kombinatorik, für das eigene Heuristiken zur Lösung benötigt wird.

5.9.4 Zusammenfassung

In diesem Abschnitt wurden Ansätze vorgestellt, mit deren Hilfe das Scheduling eines TT-Ethernet Systems in Hinsicht auf die in Abschnitt 5.3 entwickelte Bewertungsfunktion gelöst werden kann. Das *Verfahren A* erweist sich in der Umsetzung einfach und ermöglicht es, in einem iterativen Prozess nach einer optimalen Lösung zu suchen. Die Komplexität des Problems hat dabei nur einen geringen Einfluss auf den eigentlichen Optimierungsprozess. *Verfahren B* verfolgt einen anderen Ansatz und bietet für kleine Probleme einen schnellen Weg, der gegebenenfalls ohne große Suche zu einer maximal optimierten Lösung führen kann. Auf der anderen Seite skaliert der Algorithmus schlecht für Probleme mit Tasks die eine hohe Periodizität aufweisen, so dass die benötigten Ressourcen für das Finden einer optimierten Lösung schnell steigen. Ähnlich verhält es sich mit der Ergänzung aus *Verfahren C*, dessen Verhalten beim Finden einer Lösung mit dem *Verfahren B* zu vergleichen ist.

Für dem weiteren Verlauf und der Implementierung des Schedulers wird aus den genannten Gründen das *Verfahren A* gewählt, um eine robuste und ressourceneffiziente Umsetzung zu gewährleisten.

6 Evaluation der Scheduling-Strategie

In diesem Kapitel soll überprüft werden, ob das in Kapitel 4 aufgestellte Modell und die daran entwickelten Metriken und Scheduling-Strategien aus Kapitel 5 den Anforderungen genügen und zu Ergebnissen führen, die den Einsatz der TT-Ethernet Technologie in der Automobilindustrie ermöglichen. Zu diesem Zweck wird ein Kommunikationsnetz als Beispiel genutzt, wie es einem modernen Serienfahrzeugen eingesetzt wird. Aus Gründen des Datenschutzes muss leider auf eine detaillierte Beschreibung des originalen Netze verzichtet werden, so dass nur das modellierte TT-Ethernet Netz näher erläutert wird.

6.1 Variante A des Systems

Bei dem untersuchten Netz handelt es sich um ein modernes Kommunikationssystem eines Automobils, dessen Aufbau auf die wichtigsten Knoten und Nachrichten reduziert wurde. Weiterhin wurde die Ursprüngliche Topologie des Netzwerks an die Anforderungen von TT-Ethernet angepasst und mittels domänenbasierter Partitionierung (vgl. Steinbach u. a. (2012)) in eine Stern-Topologie überführt.

6.1.1 Beschreibung des Netzwerks

Das resultierende TT-Ethernet Netzwerk ist in Abbildung 6.1 dargestellt und besteht aus 4 Switches, 15 Knoten und 18 Links. Auf den Knoten werden 148 Applikationen ausgeführt, zwischen denen 41 Time-Triggered Nachrichten ausgetauscht werden, so dass 41 produzierende und 107 verbrauchende Applikationen im Netzwerk vorliegen. Für die Applikationen auf den Knoten liegen leider keine Daten vor, so dass deren Laufzeiten im Bereich von 10 µs bis 100 µs festgelegt werden. Das Netzwerk besteht somit aus folgenden Mengen:

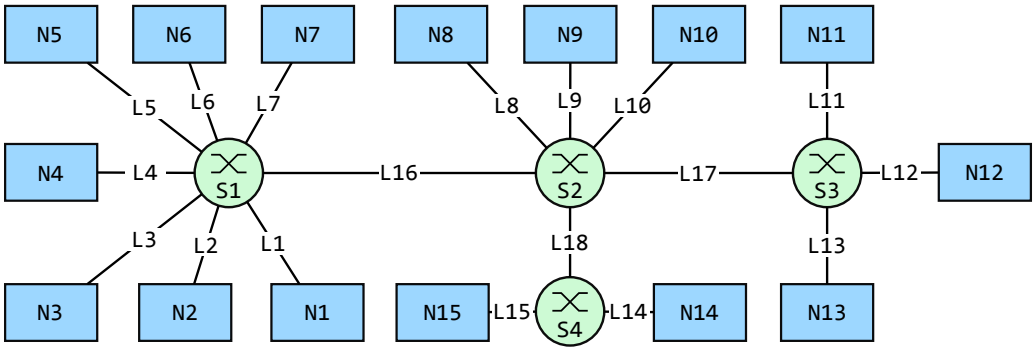

Abbildung 6.1: *Struktur des Beispielnetzwerks A*

$$N = \{n_1, \ldots, n_{15}\}$$
$$S = \{s_1, \ldots, s_4\}$$
$$L = \{l_1, \ldots, l_{17}\}$$
$$A = \{a_1, \ldots, a_{148}\}$$
$$M = \{m_1, \ldots, m_{41}\}$$

Die Größe der ursprünglichen Nachrichten beträgt zwischen 3 Byte und 42 Byte, so dass alle Nachrichten aufgrund der Mindestgröße eines TT-Ethernet Pakets auf 84 Byte anwachsen (vgl. Abschnitt 4.3). Diese zeitkritischen Nachrichten entsprechen ungefähr 5% des gesamten Kommunikationsaufkommens. Zu den 41 TT-Nachrichten gehört jeweils ein virtueller Link, über die das Routing innerhalb des Netzwerks erfolgt. Der Aufbau der virtuellen Links reicht von einfachen Ketten, in denen es nur einen Sender und einen Empfänger gibt, bis zu komplexen Baum-Strukturen, die aus 1 Sender und 8 Empfängern bestehen. Die Abbildung 6.2 demonstriert einen virtuellen Link im Netzwerk mit einem Sender und 8 Empfängern, so dass ein Gefühl für die Übertragungsstruktur entsteht. Eine Übersicht, wie viele Nachrichten jeder Knoten sendet und empfängt, bietet die Tabelle 6.1. In der dargestellten Matrix besteht die erste Zeile aus der Anzahl der Nachrichten, die jeder Knoten produziert und in der zweiten Zeile die Anzahl der verbrauchten Nachrichten.

Alle TT-Nachrichten werden zyklisch übertragen und sind einer von 8 Perioden zugeordnet. Die Spanne der Periodenlängen reicht von 1000 ms bis zu 5 ms, so dass die Cluster-Periode 1000 ms lang ist. In diesem Zeitraum werden alle Nachrichten mindestens einmal übertragen. Nachrichten mit einer Periode von 5 ms werden demnach 200 mal in der Cluster-Periode

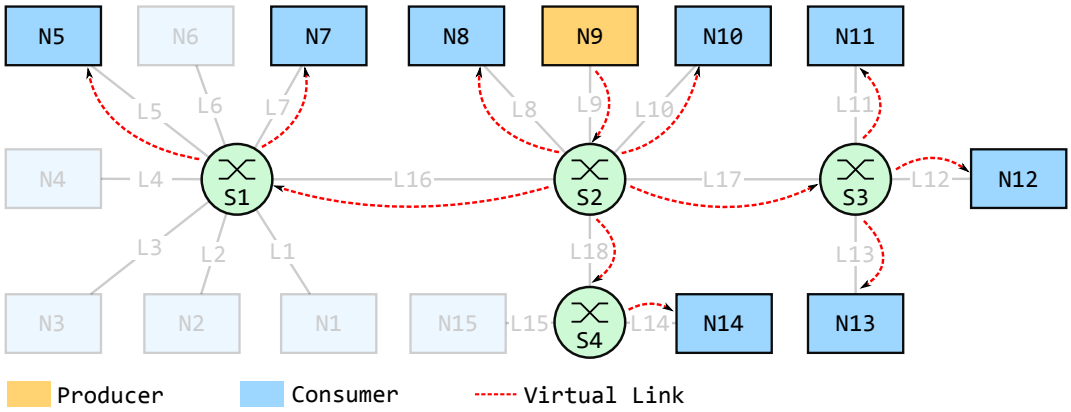

Abbildung 6.2: *Pfadverlauf eines virtuellen Links im Netzwerk A*

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}	n_{11}	n_{12}	n_{13}	n_{14}	n_{15}
P	0	0	0	0	0	0	4	3	1	0	19	4	0	4	6
C	0	0	0	0	8	0	16	14	15	11	5	4	1	14	19

Tabelle 6.1: *Produzenten (P) und Verbraucher (C) des Netzwerks*

Übertragen. Die Informationen zu jeder Periode finden sich in Tabelle 6.2, in der jeweils die Länge der Periode, die Anzahl der zugeordneten Nachrichten und die Anzahl der vorgenommenen Übertragungen innerhalb der Cluster-Periode stehen. Aus diesen Zahlen ergibt sich, dass insgesamt 2303 Übertragungen vorgenommen werden müssen. Da alle Nachrichten nur die Minimum Größe von 64 Byte haben, ist die anfallende Datenmenge gering, so dass die verbrauchte Bandbreite in einem 100 Mbit/s Netzwerk sehr niedrig ist. Auf dem Link l_{18}, über den die meisten Nachrichten gesendet werden, wird nur 1,1% der verfügbaren Bandbreite benötigt, um die TT-Nachrichten zu übertragen.

Periode	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
Länge	1000 ms	500 ms	200 ms	100 ms	50 ms	20 ms	10 ms	5 ms
Anzahl Nachrichten	1	6	2	6	1	12	10	3
Anzahl Übertragungen	1	12	10	60	20	600	1000	600

Tabelle 6.2: *Verteilung der Periodenlängen*

6.1.2 Transformation ins Scheduling Modell

Dieses Netzwerk wird nun, wie in Kapitel 4 beschrieben, in die Domäne des Multiprozessor Schedulings überführt. Gemäß der Definition 21 ergeben sich nach der Transformation folgende Mengen:

$$P = \{p_1, \ldots, p_{55}\}$$
$$T = \{t_1, \ldots, t_{379}\}$$

Das Problem besteht damit aus 55 Prozessoren, auf denen 379 Tasks ausgeführt werden müssen. Die *Computation Time* c_{t_M} der Tasks, die aus den Nachrichten transformiert wurden, berechnet sich für die 84 Byte Pakete zu:

$$c_{t_M} = \frac{84Byte}{100Mbit/s} = 6{,}72\mu s$$

Die *Computation Times* c_{t_A} der Applikationen können ohne Änderungen übernommen werden und auch die Deadlines der Tasks werden direkt von der Länge ihrer zugeordneten Periode übernommen (vgl. Abschnitt 4.6). Der Taskgraph des Scheduling-Problems setzt sich aus den 41 Jobs zusammen, die sich aus den virtuellen Links ergeben (vgl. Abschnitt 4.5). Auf eine grafische Darstellung des Taskgraphen wird verzichtet, da dieser mehrere Seiten beanspruchen würde.

6.1.3 Scheduling ohne Lücken

Auf diesem Modell werden zunächst Schedules anhand der List-Scheduling Algorithmen aus Abschnitt 3.4 erstellt. Auf das Einfügen von künstlich erzeugten Lücken wird dabei zunächst verzichtet, um einen Überblick über die Ausgangslage zu erhalten. Als zusätzliches wichtiges Merkmal, neben der entwickelten Bewertungsfunktion, wird die *Maximal Lateness* der Tasks hinzugezogen. Anhand dieses Wertes lässt sich abschätzen, wie viel Zeitraum noch zur Verfügung steht, um einzelne Tasks zwecks Erzeugung von Lücken zu verzögern. In der Tabelle 6.3 finden sich die Ergebnisse für die *Maximal Lateness* und die Varianz der Tasks für die Heuristiken *First Come First Served (FCFS)*, *Earliest Deadline First (EDF)*, *Shortest Job First (SJF)* und *Longest Job First (LJF)*.

	FCFS	EDF	SJF	LJF
Maximal Lateness (µs)	-3989	-4360	-3953	-3600
Varianz der Tasks	0,472	0,484	0,462	0,483

Tabelle 6.3: *Ergebnisse der Scheduling-Heuristiken ohne künstlich erzeugte Lücken*

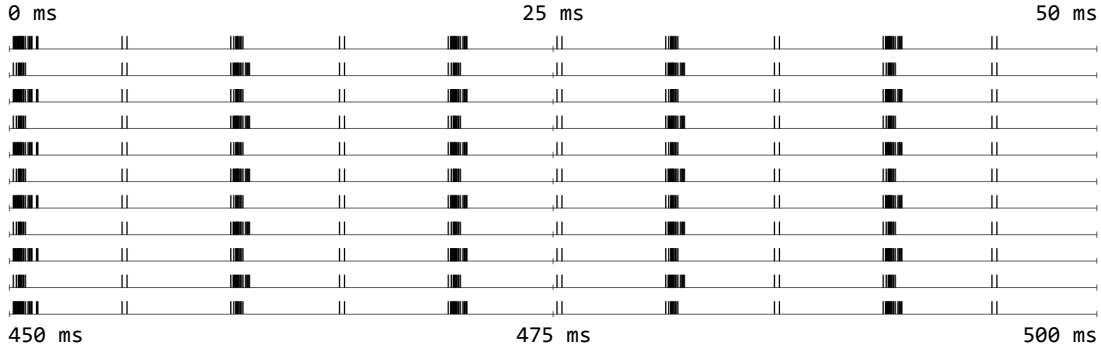

Abbildung 6.3: *Schedule einer halben Periode mit EDF und ohne Lücken*

Die *Maximal Lateness* liegt bei durchschnittlich $-4ms$. Dies ist, in Relation gesehen zu der *Computation Time* der Tasks von $6{,}72\mu s$, relativ viel, besonders in Hinsicht auf die Tatsache, dass die kleinste Periode eine Länge von $5ms$ hat. Es liegt damit ausreichend Zeitraum zur Verfügung, um Dummy-Tasks in den Schedule einzufügen. *Earliest Deadline First* erzeugt das beste Ergebnis mit einer *Lateness* von $-4{,}36ms$, während *Longest Job First* mit $-3{,}6ms$ das schlechteste Ergebnis liefert. Dies ist nicht weiter überraschend, da EDF alle Tasks mit einer kurzen Periodenlänge zuerst ausführt und diese daher nicht unnötig verzögert werden. Die Zeit beim LJF hängt dagegen von der Konfiguration des Systems ab. Es ist genauso möglich, dass SJF schlechtere Ergebnisse erzielt, je nachdem ob den länger oder kürzer laufenden Jobs eine kurze Periodenlänge zugeordnet wird.
Das Ergebnis bei der Varianz korreliert im Falle vom *Earliest Deadline First* Algorithmus mit der *Maximal Lateness*. EDF führt zu besseren Zeiten bei der *Lateness* und damit im gleichen Zug auch dazu, dass die Tasks in ihrer Ausführung alle relativ weit vorne im Schedule angesiedelt sind. Ein Umstand, der sich in der schlechteren Varianz der Tasks über die gesamte Cluster-Periode widerspiegelt. Bei den drei anderen Algorithmen hängt das Ergebnis erneut von der Systemkonfiguration ab, da diese Heuristiken keinen Bezug zur Periodenlänge der Jobs haben. Auf diese Relation wird noch im folgenden Abschnitt 6.1.4 im Detail eingegangen.

In der Abbildung 6.3 sind die ersten $500ms$ des Schedules für den Link l_{18} in der Übertragungsrichtung vom Switch s_2 zum Switch s_4 abgebildet. In diese Richtung hat dieser Link eine Auslastung von $1{,}09\%$ und überträgt damit die meisten Nachrichten im Netzwerk. Der Schedule wurde mit der Heuristik *Earliest Deadline First* erzeugt und spiegelt in dieser Darstellung das periodische Kommunikationsverhalten des Systems wieder. Es ist zu erkennen, wie jeweils ab den Zeitpunkten $0ms$, $10ms$, $20ms$ und den darauf folgenden Vielfachen besonders viele Tasks ausgeführt werden. Dies lässt sich anhand der Tabelle 6.2 überprüfen, in der gerade diesen Perioden besonders viele Tasks zugeordnet sind. Jeder Strich auf der Zeitleiste steht für einen geplanten Task, wobei die Breite des Strichs nicht maßstabsgetreu

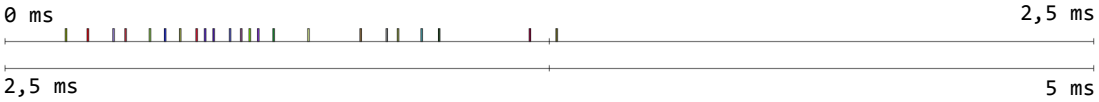

Abbildung 6.4: *Schedule der ersten 5 ms mit EDF und ohne Lücken*

ist und in dieser Darstellung eine höhere Auslastung vortäuscht. Die Abbildung 6.4 zeigt eine Detailansicht der ersten $5ms$ dieses Schedules, aus der ersichtlich wird, dass die erste Instanz aller Tasks bereits innerhalb der ersten $1,3ms$ ausgeführt werden. Es besteht zudem genügend zeitlicher Raum zur Verfügung, um die einzelnen Tasks in ihrer Ausführung zu verschieben. Weiterhin lässt sich anhand dieses Ausschnitts erahnen, dass die Erzeugung eines gültigen Schedules für dieses System keine Schwierigkeiten bedeutet und auch ein Vielfaches an Tasks möglich wäre. Nachdem nun ein Basis-Schedule im Detail vorgestellt wurde, wird im nächsten Abschnitt versucht, anhand der vier bekannten Heuristiken und mit Hilfe von künstlich erzeugten Lücken, einen optimierten Schedule zu finden.

6.1.4 Iteratives Scheduling mit Dummy-Tasks

In diesem Schritt wird erneut das Netzwerk genutzt, um mehrere Schedules mit den vier Heuristiken zu erzeugen. Zusätzlich zu den realen Tasks werden die in Abschnitt 5.9.1 beschriebenen Dummy-Tasks in den Schedule eingefügt. Eingefügt werden diese Tasks nur bei den Prozessoren, die die Nachrichtenübertragung der Links simulieren. Dummy-Tasks auf den Knoten würde einen zu starken Einfluss auf die *Arrival Times* der Nachrichten-Tasks haben. Dies würde dazu führen, dass alle Nachrichten-Tasks im Schedule zeitlich weiter nach hinten geschoben werden und somit die Freiheit beim Verschieben der Nachrichten-Tasks einschränkt wird. Das Interesse bezieht sich in diesem Fall aber hauptsächlich auf der Übertragungsverhalten des TT-Ethernet Netzwerks, so dass die Nachrichten-Tasks im Vordergrund stehen.

Das Ziel ist es, die Heuristiken hinsichtlich der *Maximal Lateness* und der entwickelten Bewertungsfunktion zu beurteilen. Zu diesem Zweck wird für jede Heuristik die Größe der Dummy-Tasks schrittweise erhöht, bis kein gültiger Schedule mehr gefunden werden kann. Sämtliche Beispiele werden erneut anhand des Links l_18 getätigt. In der Tabelle 6.4 stehen einige ausgewählte Werte aus den erzielten Ergebnissen. Zu finden sind die Zahlen für die Lückengrößen eines einzelnen maximal großem Ethernet Pakets mit dem dazugehörigen Interpacket Gap (1538 Byte), der maximal erreichten Lückengröße und die Lückengröße, bei dem das beste Ergebnis hinsichtlich der Varianz erreicht wurde. Für Lücken mit der Größe von 1538 Byte ähneln die Ergebnisse den Werten aus dem vorangegangenem Abschnitt 6.1.3. Die Heuristiken *First Come First Served* und *Shortest Job First* erzielen bei der Varianz das beste Ergebnis mit einem Wert von jeweils 0,394. *Earliest Deadline First* schneidet in

dieser Disziplin erneut am schlechtesten ab, deutet aber mit der besten *Maximal Lateness* auf das meiste Potential hin. *FCFS*, *SJF* und *LJF* erreichen bei jeweils einer Lückengröße von 1510 Byte, 4120 Byte und 3110 Byte ihre Grenzen und erzielen an dieser gleichzeitig ihren besten Wert bei der Varianz. Die bei diesen Lücken auftretende *Maximal Lateness* von nur noch $-6\mu s$ und $-7\mu s$ liefert einen Kontroll-Wert, an dem sich ablesen lässt, dass der Schedule bei noch größeren Lücken ungültig werden würde. *EDF* ermöglicht wesentlich größere Lücken von bis zu 6580 Byte, während der beste Wert bei der Varianz bei 6540 Byte erreicht wird. Das Ergebnis von 0,258 spiegelt gleichzeitig das beste mit diesen Heuristiken erreichbare Ergebnis wieder und legt erneut nahe, dass *Earliest Deadline First* hinsichtlich der Bewertungsfunktion das beste Ergebnis liefert.

	Lückengröße	Maximal Lateness	Varianz
First Come First Served	1538	-2369	0,394
	3510	-6	0,297
Shortest Job First	1538	-2686	0,394
	4120	-7	0,291
Longest Job First	1538	-2145	0.415
	3110	-7	0,345
Earliest Deadline First	1538	-4189	0,433
	6540	-1912	0,258
	6580	-49	0,261

Tabelle 6.4: *Ergebnisse der Heuristiken mit unterschiedlichen Lückengrößen*

Weitere Details liefern die beiden Graphen aus Abbildung 6.5, die zusätzlich zu dem Resultat aus Tabelle 6.4 Werte beinhalten, die mit Schritten von 500 Byte bei den Lücken erzielt wurden. Die Graphen für die beiden Algorithmen *FCFS* und *SJF* zeigen in beiden Systemen ein nahezu lineares Verhalten und demonstrieren, wie die *Maximal Lateness* mit steigender Lückengröße abnimmt, bzw. die Varianz zunimmt. Interessanter erweisen sich die Graphen von *LJF* und *EDF* in Bezug auf die *Maximal Lateness*. In beiden Fällen fällt die Kurve beim Anwachsen der Lückengröße von 0 Byte auf 500 Byte. Erklären lässt sich dies anhand der Laufzeiten der Tasks, die die produzierenden Applikationen repräsentieren. Durch die Lücke werden auch länger laufende Applikationen zu Ende ausgeführt, bevor die nächste Nachricht zum Versenden ausgewählt wird. Die Heuristiken haben damit die Möglichkeit aus einer größeren Menge von Tasks auszuwählen, was gerade im Falle von *EDF* oftmals zu einer besseren *Maximal Lateness* führen kann.
Abweichend erscheinen auch die beiden Sprünge in der Kurve von *EDF* bei den Lücken von 5500 Byte und 6580 Byte. Zieht man die Kurve aus dem Graphen der Varianz hinzu, lässt sich anhand dieser Sprünge eine volle Auslastung des Prozessors erkennen. Während die Varianz nahezu linear weiter steigt, macht die *Maximal Lateness* einen großen Sprung. Mit Auslastung ist in diesem Fall gemeint, dass der Schedule bereits so weit mit normalen und

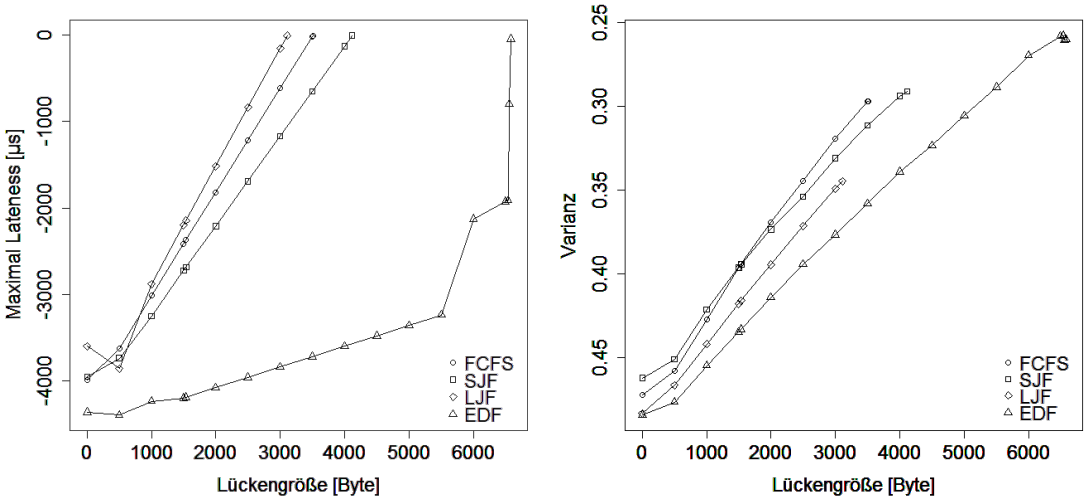

Abbildung 6.5: *Maximale Latenz und Varianz der Heuristiken bei steigenden Lückengrößen*

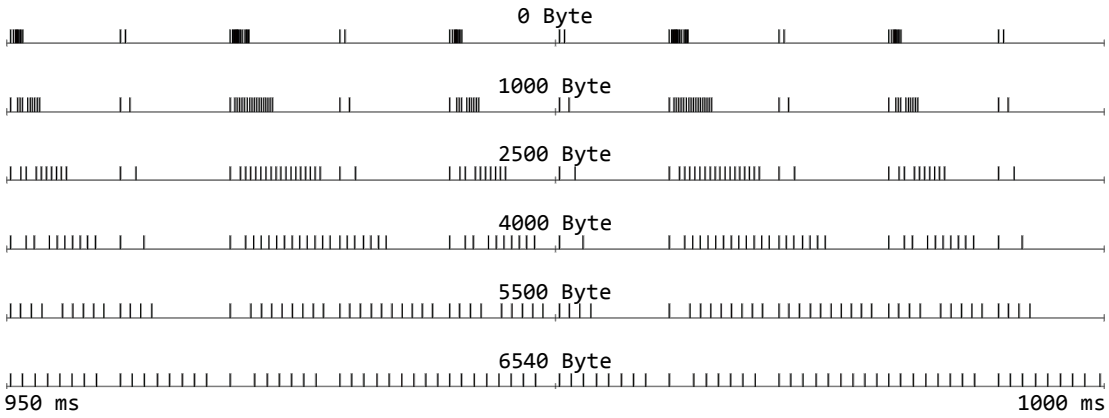

Abbildung 6.6: *Schedule-Ausschnitte (EDF) mit unterschiedlichen Lückengrößen*

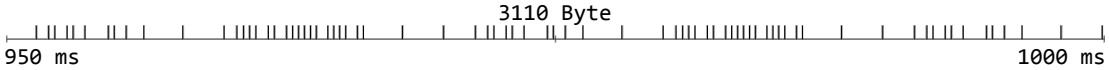

Abbildung 6.7: *Schedule-Ausschnitt (LJF) mit maximaler Lückengrößen*

Dummy-Tasks gefüllt ist, dass neue Tasks zeitlich sehr weit nach hinten geschoben werden müssen, um eine Verbesserung in der Varianz zu erreichen. Die letzten beiden Sprünge nach 6500 Byte deuten darauf hin, dass der Prozessor zu fast 100% mit Tasks ausgelastet ist, da zum gleichen Zeitpunkt ein Maximum bei der Varianz erreicht ist und nach einem Wert von 0,258 wieder zu fallen beginnt.

Zu erkennen ist dies auch in Abbildung 6.6, in der sechs Ausschnitte aus unterschiedlichen Schedules zu sehen sind. Die Zeitleisten der vorangegangenen Schedule-Abschnitte bilden ein nahezu identisches Bild und wurden der Übersicht halber weg gelassen. Die Ausschnitte zeigen jeweils die letzten $50ms$ des entsprechenden Schedules, die aufsteigend von 0 Byte bis zu 6540 Byte Lückengröße mit der Heuristik *Earliest Deadline First* erzeugt wurden. Anhand dieser sechs ausgewählten Schritte lässt sich ablesen, wie die Varianz der Tasks mit steigender Lückengröße zunimmt und schließlich sein Maximum erreicht. Bei 6540 Byte liegt der letzte Task kurz vor der $1000ms$ Cluster-Perioden Grenze und verhindert somit das Einfügen von größeren Lücken. Gleichzeitig ist an diesem letzten Ausschnitt zu sehen, dass zwischen fast allen Tasks exakt ein Dummy-Task mit einer Größe von 6540 Byte liegt. Man kann daher sagen, dass der Prozessor fast zu 100% ausgelastet ist. Als Gegenbeispiel lässt sich der Ausschnitt aus Abbildung 6.7 herbei ziehen, in dem ebenfalls die letzten $50ms$ eines Schedules zu sehen sind, der mit *Longest Job First* und der in diesem Fall maximal möglichen Lückengröße von 3110 Byte erzeugt wurde. Im Gegensatz zu dem *EDF* Schedule sind hier auch wesentlich größere Lücken zwischen den Tasks zu erkennen, so dass nicht davon auszugehen ist, dass der Prozessor komplett mit Tasks und Dummy-Tasks ausgelastet ist.

6.1.5 Simulation der Ergebnisse

Nachdem die theoretischen Ergebnisse in den vorherigen Abschnitten ausgiebig diskutiert wurden, sollen die erzeugten Schedules simuliert werden, um realitätsnahe Daten zu erhalten. Die Konzentration liegt dabei auf den Ergebnissen, die mit der *Earliest Deadline First* Heuristik erzeugt wurden, da diese die besten Ergebnisse produziert hat. Die Simulation basiert auf dem OMNeT++ Framework (vgl. OMNeT (2015)), welches um TT-Ethernet Funktionalität erweitert wurde (vgl. Steinbach u. a. (2011b)). Bei der Betrachtung der Simulationsergebnisse wird der Schwerpunkt auf die Analyse der *Response Time*, bzw. der Latenz der Rate-Constrained Nachrichten gelegt, da diese als Optimierungsziel für den Scheduler festgelegt wurde (vgl. Abschnitt 5.1.2). Zu diesem Zweck wählen wir einen virtuellen

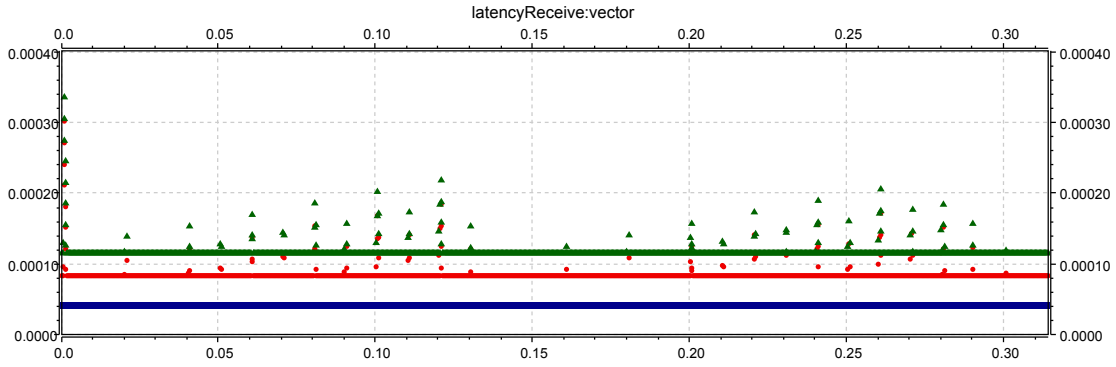

Abbildung 6.8: *Latenzen der RC-Nachricht ohne Scheduling Lücken*

Link, über den eine RC-Nachricht von N_{10} nach N_7 übertragen wird. Die Nachricht hat eine Größe von 416 Byte und wird periodisch alle $125\mu s$ gesendet. In den folgenden Abbildungen wird die Latenz der einzelnen Nachrichten für jeden Empfänger anhand graphischer Symbole dargestellt. Der erste Empfänger ist der Switch S_2 und die Latenz der Nachricht wird für dieses Gerät durch ein blaues Viereck symbolisiert. Es folgt der Switch S_1, an dem die Latenz der Nachricht durch einen roten Punkt widergespiegelt wird. Der letzte Empfänger der Nachricht ist N_7, an dem die Latenz der RC-Nachricht durch ein grünes Dreieck abgebildet wird. Alle Zeitangaben an den Achsen sind in Sekunden angegeben.

Als Ausgangslage für die Analyse nehmen wir den Schedule, der ohne künstlich erzeugte Lücken erstellt wurde. In der Abbildung 6.8 sind die entsprechenden Latenzen der RC-Nachricht für die ersten 0,3 Sekunden des Schedules zu sehen. Am Switch S_2 kommt die Nachricht immer nach ungefähr $41,3\mu s$ an. Da auf dem Link von N_{10} nach S_2 keine TT-Nachrichten versendet werden, wird die RC-Nachricht zu keinem Zeitpunkt blockiert und kann immer sofort übertragen werden, was zu einer konstanten Latenz führt. Die durchschnittliche Latenz am Switch S_1 beträgt $83,3\mu s$ und am abschließenden Empfänger N_7 ergibt sich die Latenz zu $116,6\mu s$. Auf diesem Übertragungsweg treten für einzelne Instanzen der RC-Nachricht auch höhere Latenzen auf, da sie sich den Kanal mit TT-Nachrichten teilen müssen und verzögert werden. Die maximale Latenz für die RC-Nachricht vom Sender N_{10} zum Sender N_7 beläuft sich für die meiste Zeit der Übertragung in diesem Fall auf bis zu $210\mu s$, was fast einer Verdoppelung des Durchschnitts entspricht. Eine Abweichung aus diesem Muster fällt einem bei der Betrachtung der ersten drei Millisekunden der Simulation auf, welche in der Abbildung 6.9 dargestellt sind. Man sieht, dass die RC-Nachricht am Switch S_1 beim fünften Sendevorgang deutlich verzögert wird und sich diese Verzögerung auch erst bei den folgenden Übertragungen langsam abbaut. Am Empfänger ergibt sich eine maximale Latenz von $365\mu s$, die durch den Umstand zu erklären ist, dass in diesem Schedule die TT-Nachrichten alle am Anfang ohne Lücken versendet werden. Aus der Tabelle 6.1 können wir entnehmen, dass das Gerät N_7 insgesamt 16 TT-Nachrichten empfängt, die alle den Switch

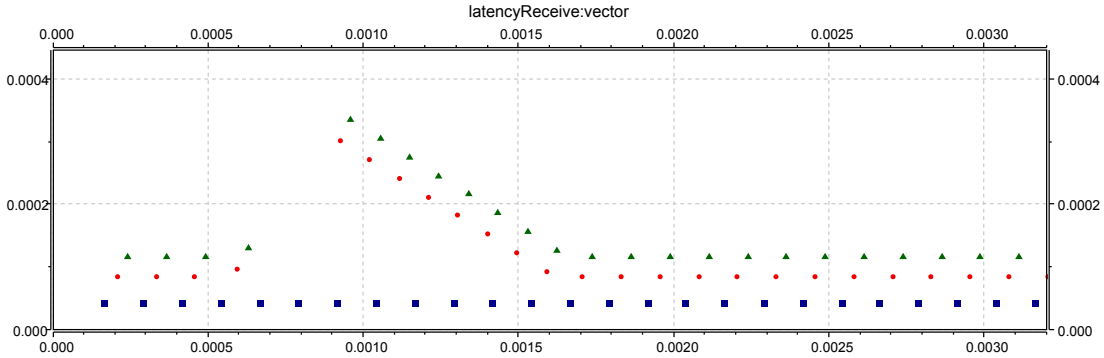

Abbildung 6.9: *Latenzen zu Beginn der Übertragung ohne Scheduling Lücken*

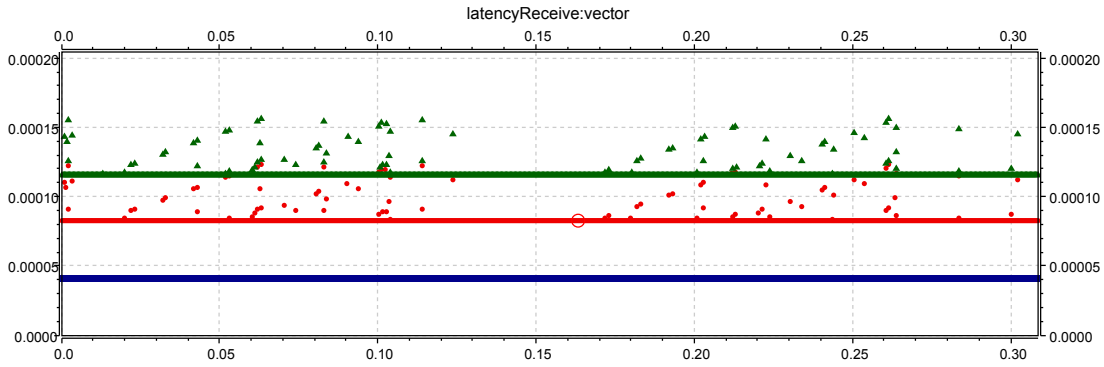

Abbildung 6.10: *Latenzen der RC-Nachricht mit Scheduling Lücken*

S_2 passieren müssen, da an S_1 kein anderes Gerät außer N_7 angeschlossen ist, welches selber TT-Nachrichten versendet. Diese TT-Nachrichten blockieren nun in dem Zeitraum zwischen $500\mu s$ und $900\mu s$ die RC-Nachricht, was letztendlich zu der hohen Latenz führt.

Zum Vergleich werden die Ergebnisse betrachtet, die sich durch den Schedule ergeben, der mit der maximalen Lückengröße von 6540 Byte generiert wurde. Die Abbildung 6.10 zeigt wieder die ersten 0,3 Sekunden aus diesem Schedule. Auffallend ist zunächst, dass sich an den durchschnittlichen Latenzen der RC-Nachricht an den unterschiedlichen Empfängern nichts geändert hat, bzw. die Veränderung verschwindend gering ist. Dies hängt damit zusammen, dass die RC-Nachricht alle $125\mu s$ gesendet wird, während die Perioden der TT-Nachrichten teilweise im hohen Millisekunden Bereich liegen (vgl. Tabelle 6.2). Dadurch kommt es bei einer großen Menge von RC-Nachrichten zu keinen Verzögerungen, so dass die durchschnittliche Latenz gering bleibt. Es ist jedoch auch ersichtlich das im Worst Case Fall, die maximale Latenz geringer ausfällt als bei dem Schedule ohne Lücken. Ohne Lücken kam es immer wieder zu Spitzen in der Latenz von bis zu $210\mu s$ und zum Beginn der Übertragung sogar von

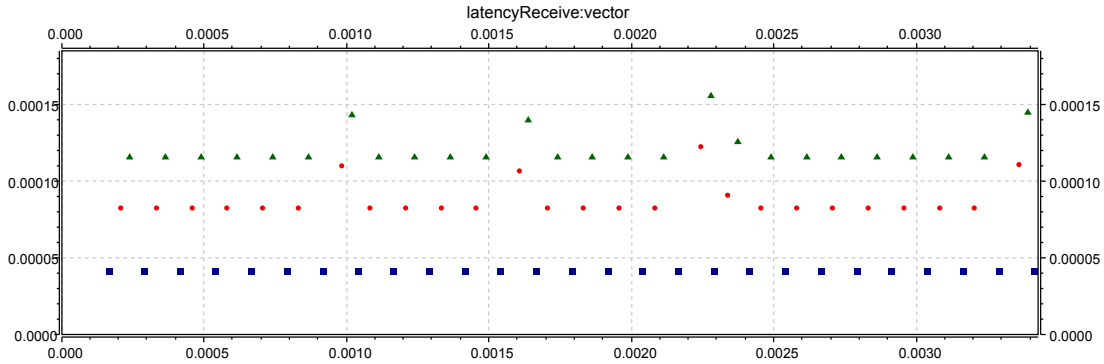

Abbildung 6.11: *Latenzen zu Beginn der Übertragung mit Scheduling Lücken*

bis zu $365\mu s$. Mit Lücken verringert sich die maximale Latenz auf $155\mu s$. Eine Reduktion von über 25 Prozent und nur noch eine Abweichung von $39\mu s$ zum Durchschnittswert. Dieser Unterschied wird auch bei der Betrachtung der ersten drei Millisekunden des Schedules deutlich. Kam es in diesem Zeitraum beim Schedule ohne Lücken noch zu der höchsten Latenz von $365\mu s$ und zu einer Verzögerung bei 7 RC-Nachrichten, ergibt sich beim Schedule mit Lücken ein anderes Bild. Wie in der Abbildung 6.11 zu erkennen ist, treten in diesem Zeitraum nun fast keine Verzögerungen auf. Durch die Lücken im Schedule der TT-Nachrichten ist kein länger zusammenhängender Zeitraum mehr blockiert, so dass nur noch vereinzelte RC-Nachrichten eine Latenz von bis zu $155\mu s$ am Empfänger N_7 aufweisen. Eine deutliche Verbesserung gegenüber dem Schedule ohne Lücken.

6.2 Variante B des Systems

Da im Originalzustand des Systems, wie es in Variante A beschrieben wurde, zu wenig TT-Nachrichten auf dem betrachteten Übertragungskanal gab, wurde das Netzwerk leicht modifiziert und weitere Sendevorgänge hinzugefügt. So kommen auf dem Link zwischen N_{10} und N_7 insgesamt 10 weitere TT-Nachrichten mit einer Größe von 100 Byte hinzu, die jeweils mit einer Periode von $500ms$ direkt hintereinander gesendet werden. Dies führt auf dem Link alle $500ms$ zu einem längeren Zeitraum, der für RC-Nachrichten blockiert ist, somit Stress bei der Übertragung erzeugt und zu hohen Latenzen führt.

Die Abbildung 6.12 zeigt die Latenzen der RC-Nachrichten aus den ersten 1,2 Sekunden des Schedules. Die durchschnittliche Latenz der RC-Nachrichten bleibt nahezu unverändert, aber es ist deutlich zu sehen, wie zu den Zeitpunkten 0,5 Sekunden und 1,0 Sekunden, an denen die zusätzlichen TT-Nachrichten versendet werden, Spitzen bei der Latenz auftreten

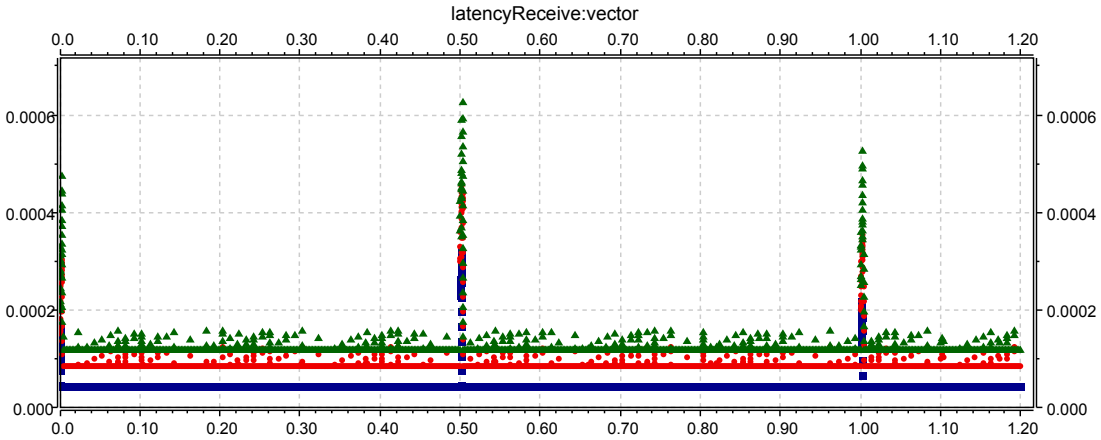

Abbildung 6.12: *Latenzen bei vielen TT-Nachrichten Aufkommen ohne Lücken*

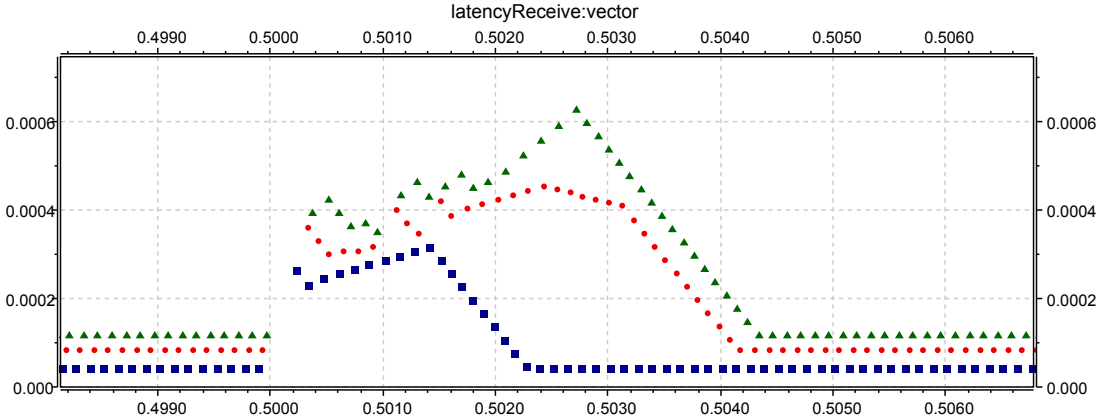

Abbildung 6.13: *Detailansicht der Stresssituation ohne Scheduling Lücken*

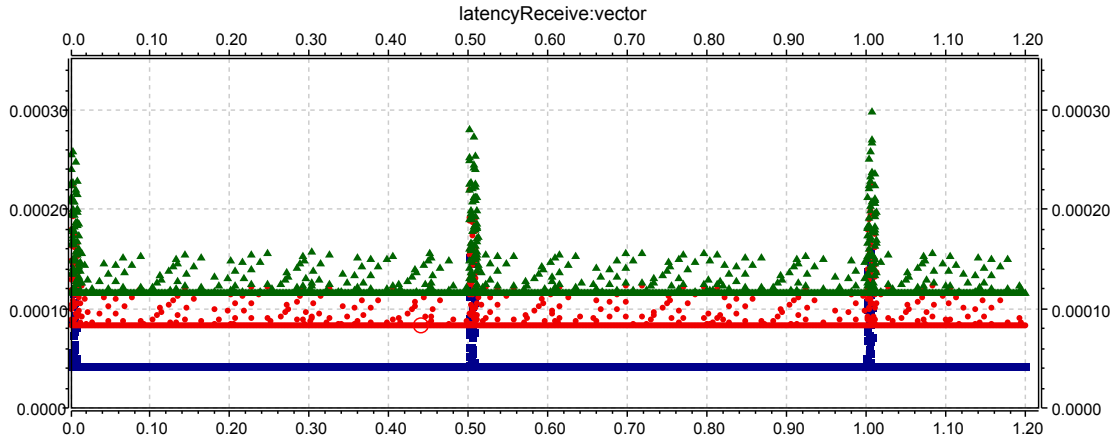

Abbildung 6.14: *Latenzen bei vielen TT-Nachrichten Aufkommen mit Lücken*

und Werte von über $600\mu s$ erreichen. Eine deutliche Steigerung zu den Worst Case Werten aus der Simulation von Variante A. In der Abbildung 6.13 ist eine Detailansicht dieser Stresssituation zu sehen. Diese beginnt genau nach 0,5 Sekunden und dauert insgesamt 4 Millisekunden. Vor und nach diesem Zeitraum ist keine erhöhte Latenz bei den RC-Nachrichten zu erkennen. Innerhalb dieser 4 Millisekunden sieht man, wie die RC-Nachrichten zunächst blockiert und immer weiter verzögert werden. Erst nach 2,9 Millisekunden ist die maximale Latenz am Empfänger N_7 erreicht und verringert sich danach wieder schrittweise bis auf den Durchschnittswert.

Fügt man beim Scheduling dieses Systems Lücken von 6540 Byte zwischen den TT-Nachrichten ein, ergibt sich ein ähnliches Bild, dass jedoch in den Worst Case Fällen wesentlich bessere maximale Latenzen aufweist. Wie in der Abbildung 6.14 zu sehen ist, verringert sich die maximale Latenz um 50 Prozent in den Stresssituationen auf $300\mu s$. Im Gegensatz zu dem Ausschnitt aus Abbildung 6.12 sieht man, dass sich die Spitze in den Latenzen über einen längeren Zeitraum hinzieht. Dies wird besonders deutlich, wenn man die Detailansicht zu diesem Zeitfenster in Abbildung 6.15 betrachtet. Statt über einen Zeitraum von 4 Millisekunden, treten die zusätzlichen TT-Nachrichten durch die Lücken nun insgesamt über einen Zeitraum von 11,5 Millisekunden auf. Dies führt dazu, dass bei den RC-Nachrichten über einen längeren Zeitraum zusätzliche Verzögerungen auftreten, diese aber nicht so hoch ausfallen. Ein unterschied zu dem Schedule ohne Lücken besteht auch in dem Umstand, dass die RC-Nachrichten nicht über den kompletten Zeitraum konstant verzögert werden, bzw. sich die Verzögerung nicht immer weiter steigert, sondern das es auch kurze Zeitfenster gibt, in denen die RC-Nachrichten sofort versendet werden, bevor die nächste TT-Nachricht wieder zu einem blockierendem Zustand führt.

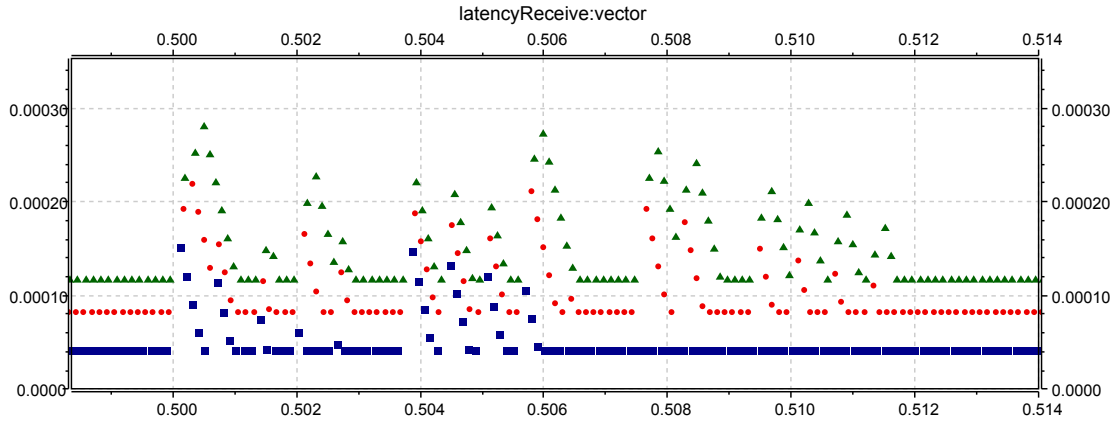

Abbildung 6.15: *Detailansicht der Stresssituation mit Scheduling Lücken*

6.3 Zusammenfassung

In diesem Kapitel wurde versucht die in den vorherigen Kapiteln aufgestellten theoretischen Behauptungen zu bestätigen. Dazu gehört auf der einen Seite die Überprüfung, ob sich das aufgestellte Modell zum Scheduling eignet und auf der anderen Seite, ob die entwickelte Metrik, die auf dem Einfügen von Lücken in den TT-Schedule basiert, zu einem verbesserten Schedule, in Hinsicht auf die *Response Time* der RC-Nachrichten, führt. Diese beiden Punkte konnten jeweils bestätigt werden. Durchgeführt wurde die Evaluation anhand eines Beispiels, dass auf einem Netzwerk eines modernen Serienfahrzeugs basiert. Beim Erzeugen des Schedules ohne Lücken stellte sich heraus, dass alle vier verwendeten List Scheduling Heuristiken dafür geeignet sind, einen gültigen Schedule zu generieren. Die erzielten Ergebnisse bei der *Maximal Lateness* und der *Varianz* unterschied sich zwischen den verschiedenen Algorithmen nur gering. Dies liegt unter anderem an dem Umstand, dass die Menge der zeitgesteuerten Nachrichten in den aktuellen Kommunikationsnetzen der Fahrzeuge relativ gering ist in Relation zu der verfügbaren Bandbreite, die TT-Ethernet Systeme bieten. In dem Fall, dass nur 1 Prozent der Bandbreite von TT-Nachrichten benötigt wird und die Periodenlängen im Millisekundenbereich liegen, stellt es kein Problem dar, die Anforderungen an einen gültigen Schedule zu erfüllen. Damit erübrigt sich aktuell auch der Einsatz von komplexeren Scheduling Algorithmen. Metaheuristiken versprechen bei diesen Problemen keine deutlich besseren Ergebnisse.

Im zweiten Schritt der Evaluation wurde untersucht, wie sich die Strategie der Lücken im Schedule der TT-Nachrichten anwenden lässt. Dafür wurde versucht mit jeder der vier Heuristiken möglichst große Lücken zwischen den einzelnen TT-Nachrichten einzufügen. Es stellte sich heraus, dass in dieser Disziplin der *Earliest Deadline First* Algorithmus die besten Ergebnisse produziert und ungefähr doppelt so große Lücken ermöglicht als die anderen Heuristiken.

Die Metrik zur Varianz der Nachrichten korrelierte dabei mit der Größe der Lücken und erwies sich so als nutzbares Werkzeug zur Bewertung eines Schedules. Anhand einer graphischen Darstellung der Schedules in der Form einer Zeitleiste konnte zudem visuell gezeigt werden, wie die Nachrichten mit zunehmender Lückengröße besser über die gesamte Cluster-Periode verteilt werden, was dazu führt, dass die für RC- und BE-Nachrichten blockierten Zeitfenster weniger konzentriert auftreten.

Im letzten Schritt wurde eine Software zur Simulation von TT-Ethernet Systemen genutzt, um die Aussagen zum Einfluss der Lücken auf die *Response Time* und die Latenz der RC-Nachrichten zu überprüfen. Dabei stellte sich heraus, dass die durchschnittliche Latenz der RC-Nachrichten nur marginal davon zu beeinflussen war. Dies hängt damit zusammen, dass über die Links wesentlich mehr RC- als TT-Nachrichten versendet werden und damit ein Großteil der RC-Nachrichten ohne Verzögerung übertragen werden kann. Jedoch zeigte sich im Bereich des Worst Case, dass die maximale Latenz durch die bessere Verteilung der TT-Nachrichten in bestimmten Fällen um bis zu 50 Prozent reduziert werden konnte. Dies stellt eine nicht unwesentliche Verbesserung im Bereich der Übertragung der RC-Nachrichten dar und bestätigen, dass die gewählte Scheduling Strategie und Metrik zu einer sinnvollen Verbesserung des Netzwerks führt.

7 Das Scheduling Framework

In diesem Kapitel sollen die Grundzüge des Scheduling Framework beschrieben werden, dass als Softwarekomponente implementiert wurde, um die Verfahren praktisch umzusetzen, die in den vorherigen Kapiteln theoretisch erarbeitet wurden. Die Software setzt sich aus zwei Hauptkomponenten zusammen, dem Back- und Frontend, die wiederum aus mehreren Modulen bestehen.

7.1 Backend

Das Backend ist eine in Java implementierte Softwarekomponente, deren Aufbau in der Abbildung 7.1 auf der nächsten Seite als UML Komponentendiagramm dargestellt wird. Das Backend bietet zwei Schnittstellen an, um Daten ein- oder auszulesen. Die erste Schnittstelle ist in *JSON* (vgl. Crockford (2006)) definiert und ermöglicht den Datenaustausch mit dem Frontend, während die zweite mittels *FIBEX* die Zusammenarbeit mit fremden Werkzeugen erlaubt. Zusätzlich lässt sich die Komponente als Bibliothek exportieren und so direkt in andere Programme integrieren. Die wichtigsten Teile des Backend sind die Data, Routing, Algorithm und Scheduling Pakete.

7.1.1 Data-Paket

In dem Data-Paket werden alle erforderlichen Daten in Klassen modelliert und für andere Pakete zur Verfügung gestellt. Die Modellierung der Daten orientiert sich dabei an den Komponenten des TT-Ethernet Protokolls, so dass für alle Elemente eine entsprechende Klasse vorliegt. Dies ermöglicht es auf einfache Art und Weise auf die Daten zuzugreifen und bildet somit eine gute Basis für weiterführende Pakete. Zusätzlich ist in diesem Paket eine Import und Export Funktion integriert, die eine Serialisierung der Daten ins JSON und FIBEX Format ermöglicht.

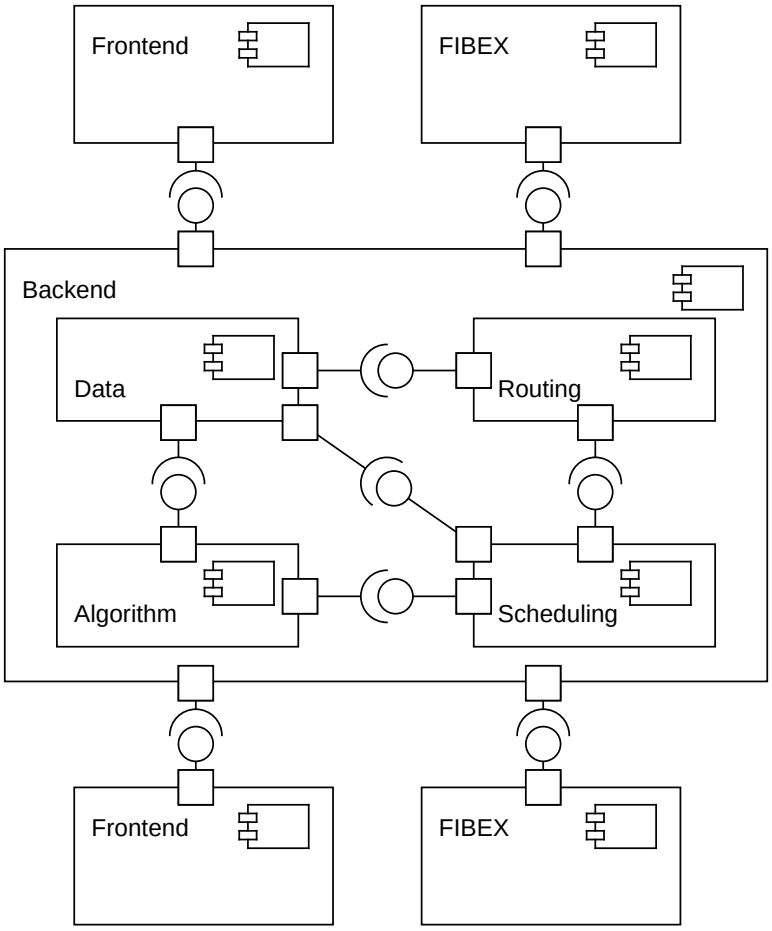

Abbildung 7.1: *UML Komponentendiagramm vom Scheduler Backend*

7.1.2 Routing-Paket

Das Routing-Paket nutzt das Daten-Paket, um auf den modellierten Daten das Routing durchzuführen. Dazu wird ein Graph aufgebaut, dessen Knoten aus den Steuergeräten und Switches besteht, während die Links als Kanten mit der Länge 1 interpretiert werden. Dadurch wird es möglich, Graphen-basierte Routing-Algorithmen zu nutzen, um die Sender- und Empfänger-Informationen, die in den virtuellen Links des TT-Ethernet Protokolls gespeichert sind, auf physikalische Übertragungswege zu übertragen. In der aktuellen Version des Frameworks wird der Dijkstra-Algorithmus genutzt, um den kürzesten Weg durch das Netzwerk zu finden. Es ist aber möglich, auf der Basis des Datenmodells weitere Routing-Verfahren zu implementieren.

7.1.3 Algorithm-Paket

Dieses Paket beinhaltet die Implementierung der Scheduling-Heuristiken. Dazu gehört auch die Möglichkeit, jederzeit neue Heuristiken zu den bestehenden hinzufügen zu können. Zu diesem Zweck müssen alle Algorithmen das folgende Interface implementieren, um mit dem Scheduling-Paket kompatibel zu sein.

Listing 7.1: *Java Quellcode für das Algorithm Interface*

```
public interface AlgorithmInterface {
    public Processable findNextItem(ProcessableList items);
}
```

Dieses Interface ermöglicht es, aus einer Liste von Tasks denjenigen auszuwählen, der als nächstes auf einem Prozessor ausgeführt werden soll. Derzeit wurde dieses Interface für die Heuristiken *First Come First Served*, *Shortest Job First*, *Longest Job First* und *Earliest Deadline First* implementiert.

7.1.4 Scheduling-Paket

Das Scheduling-Paket ist die Komponente in der eigentliche Arbeit verrichtet wird. In diesem Paket wird auch die in Kapitel 4 beschriebene Transformation des Netzwerks in das Scheduling-Modell vorgenommen. Um diese Abbildung umzusetzen, werden, wie in Abbildung 7.2 dargestellt, zwei Klassen *Processor* und *Processable* implementiert, die von den jeweiligen Klassen aus dem Data-Paket erweitert werden. Auf diese Weise werden alle Elemente in ausführende und auszuführende Gruppen unterteilt und erhalten so die notwendigen Eigenschaften für das Scheduling. *Processables* haben einen Status, der signalisiert, ob ein Task blockiert ist, bereit zur Ausführung ist oder bereits ausgeführt wurde. In den Listen

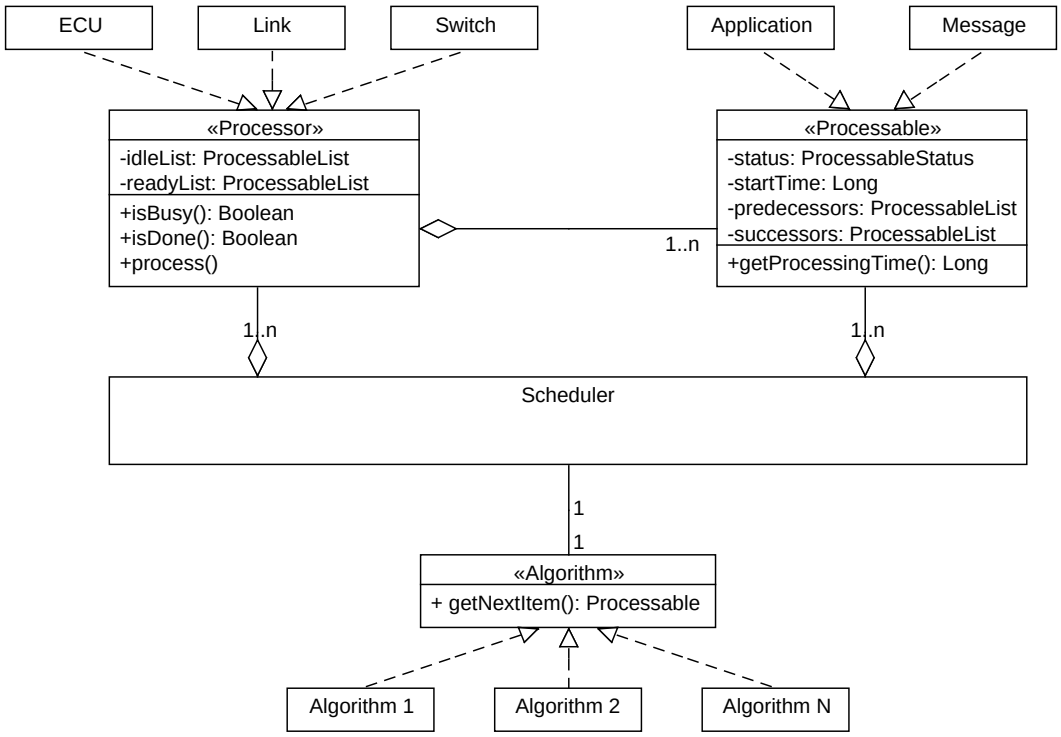

Abbildung 7.2: *Ausschnitt des UML Klassendiagramm des Scheduling Pakets*

predecessors und *successors* wird der Taskgraph (vgl. Abschnitt 3.1.2) abgebildet, so dass jeder Task Zugriff auf seine Vorgänger und Nachfolger hat. Auf diesem Weg können auch alle Nachfolger eines Tasks nach dessen Ausführung benachrichtigt werden, so dass sie vom blockierten in den wartenden Zustand wechseln können. Beim *Processor* sind die beiden wichtigsten Elemente die *idleList* und *readyList*. In diesen Listen werden für jeden Prozessor die Tasks hinterlegt, die noch durch nicht ausgeführte Vorgänger blockiert sind, bzw. Tasks die auf diesem Prozessor bereit sind, ausgeführt zu werden. Die in Abschnitt 3.4.1 beschriebenen List-Scheduling Heuristiken greifen unter anderem auf diese Listen zurück. Mit *isBusy* und *isDone* kann abgefragt werden, ob der Prozessor zum aktuellen Zeitpunkt einen Task ausführt, bzw. ob bereits alle Tasks ausgeführt wurden. So lange dies nicht der Fall ist, wird die Methode *process* wiederholt aufgerufen, bis alle Tasks ausgeführt wurden. Innerhalb von *process* wird dabei auf das Algorithm-Paket zugegriffen und durch den gewählten Algorithmus der nächste auszuführende Task aus der *readList* ausgewählt.

Der eigentliche Scheduling-Prozess läuft, wie in Abschnitt 5.9.1 beschrieben, iterativ ab und findet innerhalb der Klasse *Scheduler* statt. Die Abbildung 7.3 illustriert den Ablauf innerhalb

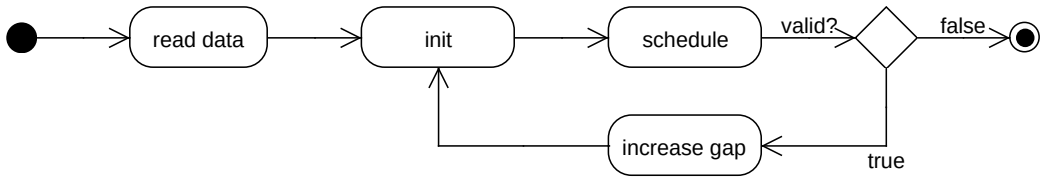

Abbildung 7.3: *UML Aktivitätsdiagramm des Scheduling Pakets*

der Klasse. Zunächst werden alle Daten über das Data-Paket eingelesen und entsprechend der Transformation modelliert. Darauf folgt eine Initialisierung aller für das Scheduling relevanten Daten. Dazu gehört das Festlegen des Routings, die Berechnung aller Deadlines nach Abschnitt 4.6, die Initialisierung der *idleList* aller Prozessoren, sowie die Konfiguration der Scheduling-Heuristik und der Lückengröße. Auf dieser Datenbasis wird nun das Scheduling ausgeführt. Wird ein gültiger Schedule gefunden, wird die Lückengröße erhöht und ein neuer Schedule berechnet. Dies wird so lange wiederholt, bis kein gültiger Schedule mehr gefunden werden kann.

7.2 Frontend

Zum Modellieren von TT-Ethernet Systemen und zur Auswertung der Scheduling Ergebnisse wurde eine grafische Benutzeroberfläche (GUI) entwickelt, um einen leichten Zugriff auf die Daten zu ermöglichen. Die Oberfläche wurde mit Web-Technologien umgesetzt, um eine plattformübergreifende Lösung anzubieten, die keine Installation benötigt, sondern direkt als Webanwendung nutzbar ist. Zur Bearbeitung der Netzwerk-Topologie und des Taskgraphen stehen graphische Komponenten zur Verfügung, die eine Modellierung nach dem *WYSIWYG* („What You See Is What You Get“) Prinzip erlaubt, während der Zugriff auf die Spezifikationen der einzelnen Elemente tabellarisch erfolgt. Zur Auswertung der Ergebnisse steht eine Vielzahl von Werkzeugen zur Verfügung. Dazu gehört eine Übersicht und Gegenüberstellung aller berechneten Schedules eines Systems, die graphische Darstellung aller virtuellen Links und der dazugehörigen Struktur im Netzwerk, eine detaillierte Scheduling-Tabelle mit einer Auflistung aller relevanten Timings wie *Start Time*, *Finish Time* und *Deadline*, eine Übersicht über die Auslastung der Prozessoren, eine Auflistung aller relevanten Metriken und ein ausführliches Protokoll vom Scheduling-Ablauf aus dem Backend. Besonders der Blick auf die Auslastung der Prozessoren und der detaillierte Blick auf den Schedule eines einzelnen Prozessoren ermöglicht die optische Überprüfung des Einflusses, den Lücken auf den Sche-

dule haben. So wurden die graphischen Darstellungen (vgl. Abbildung 6.3) eines Schedules aus dem Kapitel 6 anhand dieser GUI Komponente erstellt.

7.3 Integration durch FIBEX

Um die Integration des Schedules mit externen Werkzeugen zu ermöglichen, wurde ein Datenaustausch mittels des FIBEX Formats implementiert. FIBEX steht für *Field Bus Exchange Format* (vgl. Association for Standardisation of Automation and Measuring Systems (2013)) und ist ein XML-basierter Standard zur Beschreibung von Steuergeräte-Netzwerken. Entwickelt wurde es von der *Association for Standardisation of Automation and Measuring Systems (ASAM e. V.)* und wird von der Automobilindustrie hauptsächlich dazu verwendet, um FlexRay-Netzwerke zu beschreiben. Im Rahmen der CoRE-Arbeitsgruppe (vgl. CoRE-Arbeitsgruppe) wurde dieser Standard erweitert (vgl. Bartols (2012)) um auch TT-Ethernet Netzwerke in diesem Format beschreiben zu können. Für das Scheduling wird die physikalische Beschreibung des Netzwerkes, die logischen Verknüpfungen zwischen Sendern und Empfängern, sowie die Größe und periodischen Eigenschaften der Nachrichten benötigt. Beispielhaft ist im Listing 7.2 die Definition eines virtuellen Links im FIBEX Format enthalten. In diesem ist die Sender und Empfänger Relation für die TT-Nachricht mit der Id *105* beschrieben, die mit der Periode *VL-PERIOD3* von der ECU *ECU3-CL* zu den Geräten *ECU1-MA* und *ECU5-CL* gesendet wird.

Listing 7.2: *Virtual Link in Fibex*

```
1    <ttethernet:VIRTUAL_LINK xsi:type="ttethernet:VIRTUAL-LINK-TT-TYPE" ID="
         VL105">
2        <ho:SHORT_NAME>TTVirtualLink3</ho:SHORT_NAME>
3        <ttethernet:VLINK_SENDER>
4            <fx:ECU_REF ID_REF="ECU3-CL"/>
5        </ttethernet:VLINK_SENDER>
6        <ttethernet:VLINK_RECEIVER>
7            <fx:ECU_REF ID_REF="ECU1-MA"/>
8            <fx:ECU_REF ID_REF="ECU5-CL"/>
9        </ttethernet:VLINK_RECEIVER>
10       <ttethernet:VL_IDENTIFIER>105</ttethernet:VL_IDENTIFIER>
11       <ttethernet:VIRTUAL_LINK_PERIOD_REF ID_REF="VL-PERIOD3"/>
12   </ttethernet:VIRTUAL_LINK>
```

Diese Informationen werden im Backend aufbereitet und durch das Scheduling um Port Timings ergänzt. Diese Timings werden für die logischen Ports an den ECUs und Switches definiert und setzen sich aus der *Start Time* und *Finish Time* eines Tasks zusammen und definieren somit die Sende- und Empfangsfenster, in dem eine TT-Nachricht gesendet, bzw. empfangen werden darf. Das Listing 7.3 demonstriert einen Port an der ECU *ECU5* für den virtuellen Link *VL105* aus dem vorherigen Listing. In dem Port ist das Empfangsfenster für den physikalischen Link von Switch *SW1* nach *ECU5* definiert.

Listing 7.3: *Port Timings in Fibex*

```
<ttethernet:OUTPUT-PORT ID="OP-SW1-ECU5-VL105" xsi:type="ttethernet:ECU-
    PORT-TYPE">
    <fx:FRAME-TRIGGERING-REF ID-REF="FT-VL105"/>
    <ttethernet:TIMINGS>
        <ttethernet:ABSOLUTELY-SCHEDULED-TIMING>
            <ttethernet:WINDOW-START>2100000</ttethernet:WINDOW-START>
            <ttethernet:WINDOW-END>2300000</ttethernet:WINDOW-END>
        </ttethernet:ABSOLUTELY-SCHEDULED-TIMING>
    </ttethernet:TIMINGS>
</ttethernet:OUTPUT-PORT>
```

8 Zusammenfassung, Fazit und Ausblick

In diesem Kapitel findet diese Arbeit zum Scheduling von TDMA Kommunikation in Switch basierten Netzwerken ihren Abschluss. Dazu werden die Aussagen der einzelnen Kapitel zusammengefasst, ein Fazit gezogen und ein Ausblick auf zukünftige Arbeiten gegeben.

8.1 Zusammenfassung der Arbeit

Bei der Entwicklung moderne Automobile wird zunehmend auf den Einsatz komplexer elektronischer Systeme gesetzt. Ausgelöst wird dieser Wandel durch fortgeschrittene Fahrerassistenzsysteme, X-By-Wire Systemen und einem Ausbau der Komfort- und Informationssysteme, die alle unterschiedliche Anforderungen an das zugrunde liegende Netzwerk stellen. Sicherheitskritische Anwendungen benötigen eine sehr niedrige Latenz mit einem niedrigen Jitter, während sensorbasierte Systeme eine hohe Bandbreite für die Daten von Kameras und Laserscannern voraussetzen. Gleichzeitig wird versucht, die Menge an unterschiedlichen Bussystemen und damit auch die Komplexität der Vernetzung zu reduzieren. Ein vielversprechender Kandidat zur Erfüllung dieser Anforderungen ist das Time-Triggered Ethernet Protokoll, welches Lösungen für zeitkritische als auch bandbreitenintensive Anwendungen bietet. Eine große Herausforderung beim Einsatz dieser Technologie stellt das Aufstellen eines passenden Schedules dar, der nicht nur alle Deadlines der zeitgesteuerten Nachrichten erfüllt, sondern auch versucht die Latenz der anderen Nachrichtenklassen zu berücksichtigen. In dieser Arbeit wurde zu diesem Problem eine mögliche Lösung vorgestellt.

Im zweiten Kapitel wird das nötige Hintergrundwissen im Bereich des TT-Ethernet Protokolls vermittelt. Dazu gehören insbesondere die Spezifikationen zu den unterschiedlichen Nachrichtenklassen, die Eigenschaften der virtuellen Links und Informationen zu dem periodischen Sendeverhalten der TT-Nachrichten. Zum Scheduling wird im dritten Kapitel die erforderliche Basis geschaffen. Es werden allgemeine Begriffe erklärt und die Besonderheiten beim zeitlichen Planen von Tasks erläutert. Dazu gehören Werkzeuge zum Darstellen des Problems, wie formale Notationen und graphische Mittel wie der Taskgraph. Anschließend wurde die Thematik von Optimierungszielen und Bewertungsfunktionen näher beleuchtet und welche Rolle diese beim Scheduling spielen. Darauf aufbauend wurden übliche Lösungsansätze präsentiert, die eingesetzt werden können, um diese Ziele zu erfüllen. Abschließend

wurden im dritten Kapitel verwandte Arbeiten vorgestellt, die sich mit einer ähnlichen Thematik beschäftigen wie diese Arbeit, um einen kleinen Überblick über ähnliche Probleme und Lösungen zu bekommen.

Das vierte Kapitel behandelt die Transformation eines TT-Ethernet Netzwerks in die Domäne des Multiprozessor Schedulings. Die einzelnen Komponenten des Netzwerks werden beschrieben und über eine formal beschriebene Abbildung auf Elemente der Scheduling Theorie übertragen. Dieses Modell wird im Detail beschrieben und es wird insbesondere auf das Task- und Job-Modell eingegangen und welche Rolle die Perioden der TT-Nachrichten in Hinsicht auf die Task Deadlines spielen. Anschließend wurde eine allgemeine formale Beschreibung des Scheduling-Problems durchgeführt und die Bedingungen definiert, die für einen gültigen Schedule erfüllt werden müssen. Parallel dazu wurde jeder durchgeführte Schritt an einem Beispiel veranschaulicht.

Aufbauend auf diesem Modell werden im fünften Kapitel mögliche Optimierungsziele für den Scheduler untersucht. Im Detail wird überprüft, wie auf die unterschiedlichen Nachrichtenklassen Einfluss genommen werden kann, um den Schedule zu verbessern. Folgend wurde die Optimierung der RC-Nachrichten durch Lücken im TT-Schedule als Ziel festgelegt und unterschiedliche Bewertungsfunktionen definiert, um dieses Ziel messbar zu machen. Aus diesen Funktionen wurde eine Vorschrift zur Berechnung der Varianz der TT-Nachrichten aufgestellt, mit der sich die Qualität eines Schedules in eine einzelne Zahl fassen lässt. Anschließend wurden zu diesem Optimierungsziel und der dazugehörigen Bewertungsfunktion mögliche Nachteile untersucht und im Detail erläutert, welchen Einfluss zum Beispiel die Eigenschaften der Perioden und virtuellen Links auf das erstellte Konzept haben. Am Ende des Kapitels wurden mehrere Verfahren vorgestellt, die die bekannten List Scheduling Heuristiken erweitern, um die für die Optimierung nötigen Lücken in den Schedule einzufügen. Die Entscheidung fiel für das Verfahren, dass nach jeder TT-Nachricht die Übertragung für einen bestimmten Zeitraum blockiert, um die Lücke zu generieren. Die Größe der Lücke wird dabei schrittweise erhöht, bis kein gültiger Schedule mehr gefunden werden kann. Dies erwies sich als einfachster und stabilster Weg zur Lösung dieses Problems.

Im sechsten Kapitel wurde anhand eines realistischen Beispiels die Evaluation der Scheduling Strategie durchgeführt. Dies wurde in drei Schritten durchgeführt. Zuerst wurden mit den vier List Scheduling Heuristiken Schedules erzeugt und das Resultat hinsichtlich der maximalen Lateness und der Varianz der TT-Nachrichten analysiert. In diesem Schritt fiel der Unterschied zwischen den Heuristiken relativ gering aus. Im zweiten Schritt wurden Lücken eingefügt und diese schrittweise vergrößert. Der *Earliest Deadline First* Algorithmus erwies sich am effektivsten, so dass nahezu doppelt so Große Lücken generiert werden konnten, als bei den anderen Heuristiken. Die Bewertungsfunktion hat dieses Resultat in Hinsicht auf die Varianz bestätigt. Im letzten Schritt wurden die Ergebnisse simuliert um aussagen zu können, ob die Lücken eine Wirkung auf die Latenz der RC-Nachrichten haben und somit ein Weg sind, das Optimierungsziel zu erfüllen. Dies konnte bestätigt werden, auch wenn

keine merkbare Verbesserung bei der durchschnittlichen Latenz erzielt werden konnte, da die Menge an TT-Nachrichten in Relation zu den RC-Nachrichten zu gering war. Die maximale Latenz konnte jedoch um bis zu 50 Prozent reduziert werden.

Das folgende Kapitel enthielt eine einführende Beschreibung des implementierten Scheduling Frameworks und seiner unterschiedlichen Komponenten. Mit diesem Framework ist es möglich, mehrere Scheduling Strategien zu implementieren, um auf diesem Weg mehrere Lösungen zu produzieren, aus denen schließlich die Beste ausgewählt werden kann. Zusätzlich wurde das FIBEX Format vorgestellt, welches es ermöglicht, dass Scheduling Framework in andere Werkzeuge zu integrieren.

8.2 Fazit zum Scheduling von TT-Ethernet Netzwerken

Das Scheduling ist ein wichtiger und komplexer Aspekt im Bereich der TT-Ethernet Netzwerke und erhält durch den steigenden Einsatz elektronischer Systeme im Automobilbereich eine stetig wachsende Relevanz.

In dieser Arbeit wurde ein Modell vorgestellt, welches dieses Problem in den Bereich des Multiprozessor Schedulings transferiert und damit den Einsatz von bekannten und ausgiebig erprobten Scheduling Strategien ermöglicht. Zusätzlich wurde ein sinnvolles Optimierungsziel und die dazugehörige Bewertungsfunktion definiert, die eine begründete Aussage über die Qualität eines TT-Ethernet Schedules erlaubt. Weiterhin wurden bestehende List Scheduling Heuristiken erweitert, um beim Erzeugen eines Schedules dieses Optimierungsziel zu erfüllen. Bei der Evaluation wurde bestätigt, dass sowohl das Optimierungsziel, als auch der gewählte Lösungsansatz in aktuellen Netzwerken zu verbesserten Lösungen führt.

Im relativ jungen Feld des Schedulings von Time-Triggered Ethernet Netzwerken leistet diese Arbeit daher einen sinnvollen Beitrag, der es ermöglicht aktuelle Probleme zu lösen und zu optimieren.

8.3 Ausblick auf zukünftige Arbeiten

Da Time-Triggered Ethernet Netzwerke im Automobilbereich derzeit noch im Forschungsstadium sind, kann diese Arbeit auf keinen Fall als vollständig abschließend in Hinsicht auf dieses Thema gelten. Es wurde eine gute Basis geschaffen, über die aber nur schwer eine Aussage zu treffen ist, wie lange sie ausreichend brauchbare Ergebnisse liefert. Insbesondere bei der Evaluation wurde deutlich, wie gering der Anteil an zeitgesteuerten Nachrichten momentan ist und dass dieser Anteil keine Probleme für das Scheduling darstellt. Es ist jedoch fest davon auszugehen, dass dieser Anteil in den nächsten Jahren stark wachsen wird und die in dieser

Arbeit vorgestellten Lösungsansätze unbrauchbar werden oder zumindest nur noch unbefriedigende Ergebnisse liefern. Potenzial zur Verbesserung bieten zum Beispiel die eingesetzten Heuristiken. In diesem Bereich besteht die Wahrscheinlichkeit, dass komplexere Metaheuristiken dazu geeignet sind, bessere Ergebnisse zu liefern. Insbesondere, wenn das Aufkommen der TT-Nachrichten steigt. Zusätzlich kann und sollte weitere Arbeit in die Optimierung der Schedules investiert werden. So ist nicht nur eine verbesserte Bewertungsfunktion möglich, sondern auch eine Optimierung auf mehrere Ziele.

Literaturverzeichnis

[Abelein u. a. 2012] Abelein, U. ; Lochner, H. ; Hahn, D. ; Straube, S.: Complexity, quality and robustness - the challenges of tomorrow's automotive electronics. In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, 2012, S. 870–871

[Adam u. a. 1974] Adam, Thomas L. ; Chandy, K. M. ; Dickson, J. R.: A Comparison of List Schedules for Parallel Processing Systems. In: *Commun. ACM* 17 (1974), Dezember, Nr. 12, S. 685–690. – URL http://doi.acm.org/10.1145/361604.361619. – Zugriffsdatum: 2014-08-24. – ISSN 0001-0782

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

[Aeronautical Radio Incorporated 2009b] Aeronautical Radio Incorporated: *Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network*. 2009

[Asberg u. a. 2012] Asberg, M. ; Nolte, T. ; Kato, S. ; Rajkumar, R.: ExSched: An External CPU Scheduler Framework for Real-Time Systems. In: *2012 IEEE 18th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, August 2012, S. 240–249

[Association for Standardisation of Automation and Measuring Systems 2013] Association for Standardisation of Automation and Measuring Systems: Data Model for ECU Network Systems (Field Bus Data Exchange Format) / ASAM e.V. Mai 2013 (4.1.0). – Specification

[Błażewicz u. a. 1996] Błażewicz, Jacek ; Domschke, Wolfgang ; Pesch, Erwin: The job shop scheduling problem: Conventional and new solution techniques. In: *European Journal of Operational Research* 93 (1996), August, Nr. 1, S. 1–33. – URL http://www.sciencedirect.com/science/article/pii/0377221795003622. – Zugriffsdatum: 2014-05-12. – ISSN 0377-2217

[Bartols 2012] Bartols, Florian: *Cluster Simulation of Real-time Ethernet based Electronic Control Units in Context of Automotive Applications*. Februar 2012. – Bericht

[Bello 2011] Bello, Lucia L.: The Case for Ethernet in Automotive Communications. In: *SIGBED Rev.* 8 (2011), Dezember, Nr. 4, S. 7–15. – URL http://doi.acm.org/10.1145/2095256.2095257. – Zugriffsdatum: 2014-03-07. – ISSN 1551-3688

[Blazewicz, J. u. a. 2007] Blazewicz, J. ; Ecker, K.H. ; Pesch, E. ; Schmidt, G. ; Weglarz, J.: Open Shop Scheduling. In: *Handbook on Scheduling*. Springer Berlin Heidelberg, Januar 2007 (International Handbook on Information Systems), S. 321–343. – URL http://link.springer.com/chapter/10.1007/978-3-540-32220-7_9. – Zugriffsdatum: 2014-05-15. – ISBN 978-3-540-28046-0, 978-3-540-32220-7

[Bokhari 1981] Bokhari, S.H.: On the Mapping Problem. In: *IEEE Transactions on Computers* C-30 (1981), März, Nr. 3, S. 207–214. – ISSN 0018-9340

[Buttazzo 2011] Buttazzo, Giorgio C.: *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, 2011. – ISBN 1461406765

[Chetto u. a. 1990] Chetto, H. ; Silly, M. ; Bouchentouf, T.: Dynamic scheduling of real-time tasks under precedence constraints. In: *Real-Time Systems* 2 (1990), September, Nr. 3, S. 181–194. – URL http://link.springer.com/article/10.1007/BF00365326. – Zugriffsdatum: 2014-07-01. – ISSN 0922-6443, 1573-1383

[CoRE-Arbeitsgruppe] CoRE-Arbeitsgruppe: *Communication over Real-time Ethernet*. – URL http://core.informatik.haw-hamburg.de. – Zugriffsdatum: 2014-03-24

[Correa u. a. 1999] Correa, R.C. ; Ferreira, A ; Rebreyend, P.: Scheduling multiprocessor tasks with genetic algorithms. In: *IEEE Transactions on Parallel and Distributed Systems* 10 (1999), August, Nr. 8, S. 825–837. – ISSN 1045-9219

[Crockford 2006] Crockford, Douglas: JSON: The fat-free alternative to XML. In: *Proc. of XML* Bd. 2006, 2006

[Davidović und Crainic 2006] Davidović, Tatjana ; Crainic, Teodor G.: Benchmark-problem instances for static scheduling of task graphs with communication delays on homogeneous multiprocessor systems. In: *Computers & Operations Research* 33 (2006), August, Nr. 8, S. 2155–2177. – URL http://www.sciencedirect.com/science/article/pii/S0305054805000055. – Zugriffsdatum: 2014-07-29. – ISSN 0305-0548

[Dertouzos 1974a] Dertouzos, Michael L.: Control Robotics: The Procedural Control of Physical Processes. In: *IFIP Congress'74*, 1974, S. 807–813

[Dertouzos 1974b] Dertouzos, Michael L.: Control Robotics: The Procedural Control of Physical Processes. In: *IFIP Congress*, URL http://dblp.uni-trier.de/db/conf/ifip/ifip74.html#Dertouzos74, 1974, S. 807–813

[Diemer u. a. 2012] Diemer, Jonas ; Rox, Jonas ; Ernst, Rolf: Modeling of Ethernet AVB networks for worst-case timing analysis. In: *MATHMOD, Austria* (2012). – URL `http://seth.asc.tuwien.ac.at/proc12/full_paper/Contribution374.pdf`. – Zugriffsdatum: 2014-03-05

[Dijkstra 1959] Dijkstra, E. W.: A note on two problems in connexion with graphs. In: *Numerische Mathematik* 1 (1959), Dezember, Nr. 1, S. 269–271. – URL `http://link.springer.com/article/10.1007/BF01386390`. – Zugriffsdatum: 2014-03-11. – ISSN 0029-599X, 0945-3245

[Drozdowski 2010] Drozdowski, Maciej: *Scheduling for parallel processing*. Springer, 2010

[Dutertre und De Moura 2006] Dutertre, Bruno ; De Moura, Leonardo: The yices smt solver. In: *Tool paper at http://yices. csl. sri. com/tool-paper. pdf* 2 (2006), S. 2

[Eles u. a. 2000] Eles, P. ; Doboli, A. ; Pop, P. ; Peng, Zebo: Scheduling with bus access optimization for distributed embedded systems. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 8 (2000), Nr. 5, S. 472–491. – ISSN 1063-8210

[Emmons und Vairaktarakis 2012] Emmons, Hamilton ; Vairaktarakis, George: *Flow Shop Scheduling: Theoretical Results, Algorithms, and Applications*. Springer, September 2012. – ISBN 9781461451525

[Ferrandi u. a. 2010] Ferrandi, F. ; Lanzi, P.-L. ; Pilato, C. ; Sciuto, D. ; Tumeo, A.: Ant Colony Heuristic for Mapping and Scheduling Tasks and Communications on Heterogeneous Embedded Systems. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29 (2010), Nr. 6, S. 911–924. – ISSN 0278-0070

[FlexRay Consortium 2010] FlexRay Consortium: Protocol Specification / FlexRay Consortium. Stuttgart, Oktober 2010 (3.0.1). – Specification

[Fohler 1995] Fohler, G.: Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. In: *, 16th IEEE Real-Time Systems Symposium, 1995. Proceedings*, Dezember 1995, S. 152–161

[French 1982] French, Simon: *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Ellis Horwood, 1982. – ISBN 9780853122999

[Hou und Shin 1997] Hou, Chao-Ju ; Shin, K.G.: Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. In: *IEEE Transactions on Computers* 46 (1997), Dezember, Nr. 12, S. 1338–1356. – ISSN 0018-9340

[Hou u. a. 1994] Hou, E. S H. ; Ansari, N. ; Ren, Hong: A genetic algorithm for multiprocessor scheduling. In: *IEEE Transactions on Parallel and Distributed Systems* 5 (1994), Februar, Nr. 2, S. 113–120. – ISSN 1045-9219

[Hua und Liu 2011] Hua, Yu ; Liu, Xue: Scheduling design and analysis for end-to-end heterogeneous flows in an avionics network. In: *2011 Proceedings IEEE INFOCOM*, April 2011, S. 2417–2425

[Huang u. a. 2008] Huang, Ye ; Brocco, A. ; Kuonen, P. ; Courant, M. ; Hirsbrunner, B.: SmartGRID: A Fully Decentralized Grid Scheduling Framework Supported by Swarm Intelligence. In: *Seventh International Conference on Grid and Cooperative Computing, 2008. GCC '08*, Oktober 2008, S. 160–168

[Hwang u. a. 2008] Hwang, Reakook ; Gen, Mitsuo ; Katayama, Hiroshi: A comparison of multiprocessor task scheduling algorithms with communication costs. In: *Computers & Operations Research* 35 (2008), März, Nr. 3, S. 976–993. – URL http://www.sciencedirect.com/science/article/pii/S0305054806001432. – Zugriffsdatum: 2013-05-06. – ISSN 0305-0548

[IEEE AVB Task Group 2014] IEEE AVB Task Group: *IEEE 802.1 AV Bridging Task Group*. 2014. – URL http://www.ieee802.org/1/pages/avbridges.html. – Zugriffsdatum: 2014-12-31

[Institute of Electrical and Electronics Engineers 2008] Institute of Electrical and Electronics Engineers: IEEE 802.3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications / IEEE. September 2008 (IEEE 802.3-2008). – Standard. – ISBN 973-07381-5796-2

[Institute of Electrical and Electronics Engineers 2013] Institute of Electrical and Electronics Engineers: 802.1Qbv - Bridges and Bridged Networks - Amendment: Enhancements for Scheduled Traffic / IEEE. Dezember 2013 (P802.1Qbv/D1.0). – Draft Standard

[Jin u. a. 2008] Jin, Shiyuan ; Schiavone, Guy ; Turgut, Damla: A performance study of multiprocessor task scheduling algorithms. In: *The Journal of Supercomputing* 43 (2008), Januar, Nr. 1, S. 77–97. – URL http://link.springer.com/10.1007/s11227-007-0139-z. – Zugriffsdatum: 2014-07-27. – ISSN 0920-8542, 1573-0484

[Kamieth u. a. 2014] Kamieth, Jan ; Steinbach, Till ; Korf, Franz ; Schmidt, Thomas C.: Design of TDMA-based In-Car Networks: Applying Multiprocessor Scheduling Strategies on Time-triggered Switched Ethernet Communication. In: *19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2014)*. Piscataway, New Jersey : IEEE Press, 2014

[Kern u. a. 2011] Kern, A. ; Zhang, Hongyan ; Streichert, T. ; Teich, J.: Testing switched Ethernet networks in automotive embedded systems. In: *2011 6th IEEE International Symposium on Industrial Embedded Systems (SIES)*, Juni 2011, S. 150–155

[Kwok und Ahmad 1999] KWOK, Yu-Kwong ; AHMAD, Ishfaq: Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. In: *ACM Comput. Surv.* 31 (1999), Dezember, Nr. 4, S. 406–471. – URL http://doi.acm.org/10.1145/344588.344618. – Zugriffsdatum: 2014-03-10. – ISSN 0360-0300

[Liao u. a. 1994] LIAO, G. ; ALTMAN, E.R. ; AGARWAL, V.K. ; GAO, G.R.: A comparative study of multiprocessor list scheduling heuristics. In: *Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences, 1994* Bd. 1, Januar 1994, S. 68–77

[Liu u. a. 2005] LIU, G. Q. ; POH, K. L. ; XIE, M.: Iterative list scheduling for heterogeneous computing. In: *Journal of Parallel and Distributed Computing* 65 (2005), Mai, Nr. 5, S. 654–665. – URL http://www.sciencedirect.com/science/article/pii/S0743731505000043. – Zugriffsdatum: 2014-08-24. – ISSN 0743-7315

[Mason und Oey 2003] MASON, Scott J. ; OEY, Kasin: Scheduling complex job shops using disjunctive graphs: A cycle elimination procedure. In: *International Journal of Production Research* 41 (2003), Nr. 5, S. 981–994. – URL http://dx.doi.org/10.1080/00207540210163009. – Zugriffsdatum: 2014-05-15. – ISSN 0020-7543

[McCreary u. a. 1994] MCCREARY, C.L. ; KHAN, A. A. ; THOMPSON, J. J. ; MCARDLE, M. E.: A comparison of heuristics for scheduling DAGs on multiprocessors. In: *Parallel Processing Symposium, 1994. Proceedings., Eighth International*, April 1994, S. 446–451

[Mok und Dertouzos 1978] MOK, Aloysius Ka-Lau ; DERTOUZOS, Michael L.: *Multiprocessor scheduling in a hard real-time environment*. Domain Specific Systems Group [Massachusetts Institute of Technology], 1978

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

[OMNeT 2015] OMNET: *OMNeT++ Network Simulation Framework*. 2015. – URL http://www.omnetpp.org/. – Zugriffsdatum: 2015-01-16

[Peng u. a. 1997] PENG, D.-T. ; SHIN, K.G. ; ABDELZAHER, T.F.: Assignment and scheduling communicating periodic tasks in distributed real-time systems. In: *IEEE Transactions on Software Engineering* 23 (1997), Dezember, Nr. 12, S. 745–758. – ISSN 0098-5589

[Ponsich und Coello Coello 2013] PONSICH, Antonin ; COELLO COELLO, Carlos A.: A hybrid Differential Evolution—Tabu Search algorithm for the solution of Job-Shop Scheduling Problems. In: *Applied Soft Computing* 13 (2013), Januar, Nr. 1, S. 462–474. – URL http://www.sciencedirect.com/science/article/pii/S1568494612004188. – Zugriffsdatum: 2015-01-01. – ISSN 1568-4946

[Queck 2012] QUECK, R.: Analysis of Ethernet AVB for automotive networks using Network Calculus. In: *2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, Juli 2012, S. 61–67

[Ramamritham 1995] RAMAMRITHAM, K.: Allocation and scheduling of precedence-related periodic tasks. In: *IEEE Transactions on Parallel and Distributed Systems* 6 (1995), April, Nr. 4, S. 412–420. – ISSN 1045-9219

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

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

[Roy und Sussmann 1964] ROY, B ; SUSSMANN, B: Les problemes d'ordonnancement avec contraintes disjonctives. In: *Note DS* (1964), Nr. 9

[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

[Schäuffele und Zurawka 2013] SCHÄUFFELE, Jörg ; ZURAWKA, Thomas: *Automotive Software Engineering*. Wiesbaden : Vieweg und Teubner, 2013. – ISBN 978-3-8348-2469-1

[Shin und Lee 2008] SHIN, Insik ; LEE, Insup: Compositional Real-time Scheduling Framework with Periodic Model. In: *ACM Trans. Embed. Comput. Syst.* 7 (2008), Mai, Nr. 3, S. 30:1–30:39. – URL `http://doi.acm.org/10.1145/1347375.1347383`. – Zugriffsdatum: 2015-01-05. – ISSN 1539-9087

[Steinbach u. a. 2011a] STEINBACH, T. ; KORF, F. ; SCHMIDT, T.C.: Real-time Ethernet for automotive applications: A solution for future in-car networks. In: *2011 IEEE International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, September 2011, S. 216–220

[Steinbach u. a. 2011b] 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*. New York : ACM-DL, März 2011, S. 375–382. – ISBN 978-1-936968-00-8

[Steinbach u. a. 2012] STEINBACH, Till ; LIM, Hyung-Taek ; KORF, Franz ; SCHMIDT, Thomas C. ; HERRSCHER, Daniel ; WOLISZ, Adam: Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802). In: *2012 IEEE Vehicular Technology Conference (VTC Fall)*. Piscataway, New Jersey : IEEE Press, September 2012. – ISSN 1090-3038

[Steiner 2010] Steiner, W.: An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks. In: *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, November 2010, S. 375–384

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

[Tamas-Selicean u. a. 2012] Tamas-Selicean, Domitian ; Pop, Paul ; Steiner, Wilfried: Synthesis of Communication Schedules for TTEthernet-based Mixed-criticality Systems. In: *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. New York, NY, USA : ACM, 2012 (CODES+ISSS '12), S. 473–482. – URL http://doi.acm.org/10.1145/2380445.2380518. – Zugriffsdatum: 2014-04-07. – ISBN 978-1-4503-1426-8

[Tanenbaum 2007] Tanenbaum, Andrew S.: *Modern Operating Systems*. 3rd. Upper Saddle River, NJ, USA : Prentice Hall Press, 2007. – ISBN 9780136006633

[Thakor und Shah 2011] Thakor, D. ; Shah, A.: D_EDF: An efficient scheduling algorithm for real-time multiprocessor system. In: *2011 World Congress on Information and Communication Technologies (WICT)*, 2011, S. 1044–1049

[TTTech Computertechnik AG]

[Vilcot und Billaut 2011] Vilcot, Geoffrey ; Billaut, Jean-Charles: A tabu search algorithm for solving a multicriteria flexible job shop scheduling problem. In: *International Journal of Production Research* 49 (2011), Dezember, Nr. 23, S. 6963–6980. – URL http://dx.doi.org/10.1080/00207543.2010.526016. – Zugriffsdatum: 2015-01-01. – ISSN 0020-7543

[Wang u. a. 2011] Wang, Hui-Mei ; Chou, Fuh-Der ; Wu, Ful-Chiang: A simulated annealing for hybrid flow shop scheduling with multiprocessor tasks to minimize makespan. In: *The International Journal of Advanced Manufacturing Technology* 53 (2011), März, Nr. 5-8, S. 761–776. – URL http://link.springer.com/article/10.1007/s00170-010-2868-z. – Zugriffsdatum: 2015-01-01. – ISSN 0268-3768, 1433-3015

[Ying und Lin 2006] Ying, Kuo-Ching ; Lin, Shih-Wei: Multiprocessor task scheduling in multistage hybrid flow-shops: an ant colony system approach. In: *International Journal of Production Research* 44 (2006), August, Nr. 16, S. 3161–3177. – URL http://dx.doi.org/10.1080/00207540500536939. – Zugriffsdatum: 2015-01-01. – ISSN 0020-7543

[Zalzala und Fleming 1997] Zalzala, Ali M. S. ; Fleming, Peter J.: *Genetic Algorithms in Engineering Systems*. IET, Januar 1997. – ISBN 9780852969021

[Zapata und Alvarez 2005] Zapata, Omar U. P. ; Alvarez, Pedro M.: Edf and rm multiprocessor scheduling algorithms: Survey and performance evaluation. In: *Seccion de Computacion Av. IPN* 2508 (2005). – URL `http://delta.cs.cinvestav.mx/~pmalvarez/multitechreport.pdf`. – Zugriffsdatum: 2014-01-24

[Zhang u. a. 2011] Zhang, Jiandong ; Qiao, Shasha ; Li, Dujuan ; Shi, Guoqing: Modeling and simulation of EDF scheduling algorithm on AFDX switch. In: *2011 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, September 2011, S. 1–4

[Zomaya u. a. 1999] Zomaya, AY. ; Ward, C. ; Macey, B.: Genetic scheduling for parallel processor systems: comparative studies and performance issues. In: *IEEE Transactions on Parallel and Distributed Systems* 10 (1999), August, Nr. 8, S. 795–812. – ISSN 1045-9219

[Zorin und Kostenko 2014] Zorin, D. A. ; Kostenko, V. A.: Simulated annealing algorithm in problems of multiprocessor scheduling. In: *Automation and Remote Control* 75 (2014), Oktober, Nr. 10, S. 1790–1801. – URL `http://link.springer.com/article/10.1134/S0005117914100063`. – Zugriffsdatum: 2015-01-01. – ISSN 0005-1179, 1608-3032

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 24. Januar 2015 Jan Kamieth