Fakultät für Elektrotechnik
und Informationstechnik

Lehrstuhl für
Kommunikations-
technik

technische universität
dortmund

Masterarbeit M03-2018

Optimierte Ressourcenzuteilung für hybride kaskadierte Netze in der Industrieautomation

von
Calvin Timmer

Abgabedatum: 23. Oktober 2018

Inhaltsverzeichnis

1 Einleitung

Die moderne Fertigungslandschaft ist zunehmend geprägt durch die massive Vernetzung der Produktionsmittel. Inzwischen werden Werkzeuge jeder Größenordnung – vom Drehmomentschlüssel bis zum Schwerlastkran – mit diversen Sensoren und Aktoren ausgestattet und in ein Fertigungsnetzwerk integriert. Die treibende Kraft hinter dieser Entwicklung ist der Wunsch nach hoch-automatisierter Fertigung. Indem Software die Entscheidungs- und Produktionsprozesse in einer Fabrik übernimmt, lassen sich enorm komplexe oder variantenreiche Fertigungen kostengünstig und schnell durchführen.

Diese Entwicklung führt dazu, dass leistungsfähigen Kommunikationsnetzen eine immer wichtigere Rolle in der Industrie zukommt. Der Ausfall eines einzelnen Netzelements kann durch den entstehenden Produktionsausfall bereits zu hohen Kosten führen. Gleichzeitig steigt der Installations- und Wartungsaufwand durch die Größe der Netzwerke immer weiter an.

Für kleinere Geräte, insbesondere solche, die noch teilweise von Menschen bedient werden, wie beispielsweise ein vernetzter Drehmomentschlüssel, haben Funksysteme große Vorteile. Sie besitzen im Vergleich zu Kabeln einen geringeren Installationsaufwand, haben weniger Verschleiß und lassen sich auch mobil verwenden. Hoch-automatisierte Anlagen müssen jedoch bis heute auf kabelgebundene Technologien zurückgreifen, da aktuelle Funksysteme die Anforderungen bezüglich Zuverlässigkeit, Latenz und Datenrate nicht erfüllen können.

Die Entwicklung neuer Funkstandards, die diese Anforderungen erfüllen können, stößt deshalb auf großes Interesse. Derzeit findet ein großer Teil dieser Entwicklung unter dem Dach des 5G Mobilfunks statt. Abseits davon beschäftigen sich diverse Forschungsprojekte auch mit anderen Ansätzen. Im Rahmen des ParSec-Projekts [1] wird beispielsweise ein Funksystem auf Basis des bisher noch nicht verbreiteten Parallel Sequence Spread Spectrum (PSSS) Verfahrens entwickelt. Frequenzspreizverfahren wie PSSS sind vielversprechende Kandidaten, da sie eine hohe Störsicherheit gegenüber schmalbandigen Störungen besitzen und mit gut mit anderen Technologien ko-existieren können. Das ist gerade in dicht bebauten Fabriken interessant und umso mehr, wenn ein lizenzfreies Frequenzband genutzt werden soll.

In den seltensten Fällen werden Fabriken von Grund auf neu errichtet. Stattdessen wird neue Technologie oft zur Aufrüstung bestehender Anlagen verwendet. Deshalb sollte bei der Entwicklung eines neuen Kommunikationssystems die Kompatibilität mit bestehenden Systemen

beachtet werden. Im Falle der Nachrüstung von automatisierten Produktionsanlagen bedeutet das, dass eine Mischung der bestehenden Feldbusse oder Industrial-Ethernets (IEs) mit den neuen Funktechnologien entsteht. Diese Teile müssen dabei so auf einander abgestimmt sein, dass das entstehende hybride Netzwerk weiterhin reibungslos die Anforderungen der Anlage erfüllen kann.

In dieser Arbeit wird ein Medium Access Control (MAC) Protokoll für ein ParSec-System entworfen, das speziell auf den Aufbau derartiger hybrider Netzwerke ausgelegt ist. Es soll in der Lage sein eine drahtlose Schnittstelle zwischen mehreren IE-Netzwerken aufzubauen, ohne dass die bestehenden IE-Komponenten dazu aktualisiert werden müssten. Konzeptionell soll es unabhängig vom verwendeten IE nutzbar sein; in dieser Arbeit wird aber speziell Sercos III betrachtet.

Aufbau der Arbeit

In Kapitel 2 wird eine Einführung in die bestehenden Technologien gegeben, die in dieser Arbeit verwendet werden. Dazu gehört eine detailliertere Betrachtung des industriellen Umfelds, eine Einführung in Sercos III und die Vorstellung des PSSS-Verfahrens.

Danach wird in Kapitel 3 das entwickelte MAC-Protokoll beschrieben und erklärt. Speziell das Scheduling der echtzeitkritischen Daten wird ausführlich behandelt. Kapitel 4 beschreibt daraufhin die Implementierung dieses Scheduling-Algorithmus.

Um den entwickelten Ansatz zu evaluieren, werden in Kapitel 5 zwei beispielhafte Szenarien auf der Industrie beschrieben. Anschließend wird die erstellte Implementierung simulativ verifiziert.

Im letzten Kapitel folgt ein abschließendes Fazit und ein Ausblick auf weiterführende Fragestellungen.

2 Grundlagen

In diesem Kapitel werden die Grundlagen der für diese Arbeit relevanten Themenbereiche präsentiert. Zuerst wird eine Begriffserklärung der verschiedenen Echtzeitklassen eingeführt. Dann wird in Abschnitt 2.2 ein Einstieg in die Anforderungen der industriellen Kommunikation gegeben. In Abschnitt 2.3 wird der grundlegende Aufbau von Industrial-Ethernet-Systemen vorgestellt. Abschließend wird in Abschnitt 2.4 die Parallel Sequence Spread Spectrum (PSSS) Funktechnologie präsentiert. Hier wird mitunter auf die Codierung und Decodierung von PSSS-Symbolen, die Möglichkeiten zum Multiplexing unter PSSS, die Bitfehlerverteilung eines PSSS-Systems und die Implementierung von PSSS im ParSec-Projekt [1] eingegangen.

2.1 Echtzeitanforderungen

Es wird zwischen vier Klassen der Echtzeitanforderungen eines Systems unterschieden. Die folgenden typischen Werte für Latenz und Jitter sind aus [2] entnommen.

- Systeme mit *weichen Echtzeitanforderungen* sind darauf angewiesen, dass gesendete Daten den Empfänger nicht viel später als zu einem vorgegebenen Zeitpunkt erreichen. Je größer die Verspätung der Daten, umso schlechter ist die Systemperformance. Typische Anforderungen an die Latenz betragen 10 ms bis 100 ms.
- Bei Systemen mit *harten Echtzeitanforderungen* wird angenommen, dass ein verspätetes Eintreffen eines Datums sofort dazu führt, dass dieses nicht mehr verwertet werden kann. Jede Verspätung entspricht somit einem Paketfehler. Zusätzlich besteht meistens eine extrem strikte Anforderung an die Paketfehlerwahrscheinlichkeit. Die Latenzen müssen sich typischerweise im Bereich von 1 ms bis 10 ms und der Jitter unter 1 ms bewegen.
- *Isochrone Echtzeitanforderungen* fügen den harten Echtzeitanforderungen die Bedingung hinzu, dass die Daten nicht zu früh eintreffen dürfen. Die dazu notwendigen Latenzen bewegen sich üblicherweise unter 1 ms und der Jitter unter 1 µs.

In dieser Arbeit werden vor allem Systeme mit harten Echtzeitanforderungen betrachtet.

2.2 Fabrikautomation

In der Industriekommunikation kann nach [2] grob zwischen drei Anforderungsfeldern unterschieden werden:

CM Beim „Condition Monitoring" (oder zu deutsch: der Zustandsüberwachung) werden Sensordaten gesammelt, die den Status einer Anlage überwachen und zur frühzeitigen Erkennung von drohenden Ausfällen genutzt werden können. Die Netzwerke können sehr viele Sensoren umfassen und über eine große Fläche verteilt sein. Da die Daten nicht direkt zur Steuerung der Anlagen verwendet werden und teilweise sogar noch durch einen menschlichen Bediener interpretiert werden müssen, bestehen keine hohen Anforderungen an die Zuverlässigkeit und Latenz.

PA Die Prozessautomation beschäftigt sich mit der Verarbeitung von Volumengütern, wie beispielsweise von Flüssigkeiten, Gasen oder endlos-Produkten wie Drähten. Die Verarbeitung läuft entsprechend mit hoher Kontinuität ab. Deshalb sind nur langsame Steuereingriffe üblich, allerdings ist zur Qualitätssicherung eine enge Überwachung des Produkts notwendig. Aus diesem Grund fallen sehr große Mengen an Messdaten an, die zuverlässig übertragen werden müssen. Gleichzeitig besitzen die Anlagen häufig sehr große Ausdehnungen, sodass die Sensoren über eine weite Fläche verteilt sind.

FA In der Fabrikautomation geht es um die Probleme der diskreten Fertigung. Zahlreiche Produkteinheiten, zum Beispiel Getränkedosen, müssen zwischen verschiedenen Anlagen transportiert und von jeder Anlage in einem oder mehreren Verarbeitungsschritten manipuliert werden. Viele Arbeitsschritte erfordern dabei die Kooperation unterschiedlicher Aktoren und müssen mit einer hohen Genauigkeit ausgeführt werden. Daher werden diese Systeme oft in einem geschlossenen Regelkreis ausgeführt. Für einen großen Produktdurchsatz sind außerdem hohe Arbeitsgeschwindigkeiten erforderlich. Fehlerhafte Daten können leicht zu defekten Produkten oder einer Beschädigung der Anlage führen. Weil einzelne Anlagenteile separat gesteuert werden können, ist die räumliche Ausdehnung der Netzwerke aber vergleichsweise gering.

Für CM und PA existieren mit ZigBee [3], Bluetooth [4], WSAN [5] und WirelessHART [6] bereits geeignete Funktechnologien, die auch in der Praxis angewendet werden. Im Bereich der FA gibt es hingegen bisher keinen Funkstandard, der alle Anforderungen zufriedenstellend erfüllt. Wie mitunter in [7] und [8] ausführlich behandelt wird, ist eine der wichtigsten Zielsetzungen des kommenden 5G-Standards eine bessere Abdeckung dieses Aufgabenfelds. Dennoch verbleibt ein großer Forschungsbedarf in diesem Feld.

2.2.1 Grundlegender Aufbau der Anlagen und Netzwerke

Eine Anlage besteht aus einer Vielzahl von Sensoren und Aktoren, sowie einer zentralen speicherprogrammierbaren Steuerung (SPS). Die Steuerung besitzt ein Modell der gesamten Anlage und enthält die vollständige Anwendungslogik. Sie empfängt Messwerte von den Sensoren, wertet diese aus und ermittelt damit den aktuellen Ist-Zustand. Durch das Anwendungsprogramm ist der Soll-Zustand des Systems im nächsten Zeitschritt vorgegeben. Anhand des Anlagenmodells bestimmt die Steuerung dann die optimale Steuerfolge für die Aktoren und sendet ihnen diese. Die Aktoren führen die Befehle aus und die Sensoren messen anschließend die aktualisierten Werte. Somit ist ein geschlossener Regelkreis realisiert, der die präzise Steuerung des gesamten Systems erlaubt.

Damit die Steuerung die einzelnen Messwerte zu einem gemeinsamen Ist-Zustand kombinieren kann, müssen die Aufnahmezeitpunkte der Messwerte genau bekannt sein. Im Idealfall wurden sogar alle Messwerte zeitgleich erhoben, da dann keine Interpolation der Werte nötig ist. Gleiches gilt für die Steuerdaten. Damit die optimale Steuerfolge bestimmt werden kann, muss genau definiert sein zu welchen Zeitpunkten die Steuerbefehle umgesetzt werden. Daher besitzen FA-Systeme harte oder isochrone Echtzeitanforderungen.

Aus diesen Rahmenbedingungen ergeben sich die folgende Prinzipien, nach denen Netzwerke für die FA üblicherweise entworfen sind:

- Die logische Netzwerk-Topologie ist eine Sterntopologie mit der SPS als zentralem Element.
- Es werden *„global sampling points"* (GSPs) definiert. Zu diesen Zeitpunkten nehmen alle Sensoren ihre Messwerte und alle Aktoren aktivieren ihre Steuerwerte. Damit das funktioniert müssen die GSPs sehr präzise zwischen allen Teilnehmern synchronisiert sein. Die GSPs treten in einem regelmäßigen und unveränderlichen Zyklus auf.
- Nach einem GSP fragt die SPS die Messdaten der Sensoren ab oder diese senden der SPS ihre Daten unaufgefordert zu. Die SPS berechnet daraus neue Stellwerte und sendet diese noch vor dem nächsten GSP an die Aktoren.
- Der Kanalzugriff erfolgt nach einem deterministischen Schema, damit die hohen Anforderungen definitiv jederzeit erfüllt werden. Bei nichtdeterministische Zugriffsverfahren kann während der Auslegungsphase nicht garantiert werden, dass die Anforderungen unter allen Lastszenarien erfüllt werden, sodass ein Restrisiko beim Betrieb der Anlage bestehen würde.

2.3 Industrial Ethernet

Aufgrund der hohen Anforderungen der Fabrikautomation sind spezialisierte Netzwerktechnologien entstanden, die nur auf diesen Anwendungszweck zugeschnitten sind. In der Vergangenheit wurden dazu eine Vielzahl an Technologien und Protokollen genutzt. Seit Anfang des Jahrhunderts setzen sich aber zunehmend Systeme durch, die auf dem IEEE 802.3 Ethernet-Standard [9] aufbauen. Diese ergänzen den Ethernet-Standard üblicherweise um ein alternatives MAC-Protokoll, das auf die Erfüllung der Echtzeitanforderungen ausgelegt ist. Gleichzeitig bleibt meist ein Teil der ungenutzten Übertragungskapazität für normalen Ethernet-Verkehr erhalten, was die Anbindung von Teilnehmern mit Standard-Ethernet-Technologie erlaubt. Dadurch und durch die breite und kostengünstige Verfügbarkeit kompatibler Ethernet-Hardware, haben diese Industrial Ethernet (IE) Systeme die klassischen Feldbusse heute zu einem Großteil abgelöst.

2.3.1 Sercos III

In dieser Arbeit wird der Sercos III Standard als Grundlage für die Beschreibung einer IE Architektur herangezogen. Die hier verwendete Spezifikation des Kommunikations-Protokolls ist in [10] zu finden.

Der physische Aufbau eines Sercos-Netzwerks kann entweder als Doppelring oder Doppellinie erfolgen, wobei die Hin- und Rückrichtung über verschiedene Adernpaare im gleichen Ethernet-Kabel erfolgt. In der Praxis ist vor allem die Doppel-Ring-Topologie wichtig, da sie beim Ausfall einer Verbindung den automatischen Rückfall auf eine Doppel-Linien-Topologie erlaubt und damit eine höhere Ausfallsicherheit besitzt.

Das Data Link Layer (DLL) Protokoll von Sercos III unterscheidet zwischen zwei Typen von Teilnehmern: dem IE-Master (IE-M) und den IE-Slaves (IE-S). Der IE-M ist der Koordinator des Netzwerks und befindet sich in der SPS. Ein Sercos Netzwerk muss immer genau einen IE-M besitzen. Die IE-S sind die restlichen Teilnehmer des Netzwerks, die sogenannten Feldgeräte. Sie umfassen Sensoren, Aktoren, Antriebe, Ein-/Ausgabe-Geräte und mehr. Ein physisches Feldgerät kann dabei mehrere logische IE-S umfassen, wenn es unterschiedliche Funktionen umsetzt. Im Netzwerk können bis zu 511 IE-S registriert werden. Für viele typische Funktionen existieren standardisierte Geräteprofile, die einen minimalen Funktionsumfang und das zugehörige Interface festlegen. Die Adressierung der einzelne Datenelemente erfolgt über sogenannte *„identification numbers"* (IDNs). Zu jeder IDN existiert im Standard eine Definition des Datenformats und der Bedeutung der entsprechenden Daten. Zusätzlich können Gerätehersteller eigene IDNs definieren, über die Funktionen bereitgestellt werden können, die nicht in den Standardprofilen enthalten sind.

Für den Zugriff auf IDNs – beziehungsweise auf die Daten hinter den IDNs – stehen zwei Wege zu Verfügung. Um zyklisch und mit Echtzeitanforderungen auf ein IDN zuzugreifen, existieren die Connections. Eine Connection wird während der Konfiguration des Netzwerk angelegt und besitzt danach fest zugewiesene Übertragungsressourcen. IE-M und IE-S übertragen in festen Intervallen und ohne Aufforderung oder Signalisierung die Daten des zugehörigen IDNs über eine Connection. Soll dagegen nur sporadisch und ohne Echtzeitanforderung auf ein IDN zugegriffen werden, kann der IE-M eine Anfrage über den Service-Channel (SVC) jedes IE-S senden. Der angesprochene IE-S antwortet dann mit den angefragten Daten ebenfalls über den SVC. Der SVC für jeden IE-S besitzt eine feste Größe von 4 Byte Nutzdaten. Da die IDNs beliebige Größen besitzen können, werden sie gegebenenfalls fragmentiert übertragen. Ein mit TCP vergleichbares Protokoll koordiniert den Datenaustausch. Aufgrund der begrenzten Größe ist der SVC nicht in der Lage Echtzeitanforderungen zu erfüllen. Er wird beispielsweise für die Konfiguration der Feldgeräte genutzt.

IE-Zyklus

Wie schon mehrfach erwähnt wurde, ist der Datenaustausch im Netzwerk stark an einem periodischen IE-Zyklus orientiert. Dieser existiert um die Anforderungen der RT-Daten erfüllen zu können, er wirkt sich dadurch aber automatisch auch auf alle anderen Datentypen aus. Sercos III überträgt neben den Connections auch die SVC-Daten, die Status-Daten der einzelnen IE-S und den Takt zur Synchronisation des GSPs über die gleiche zyklische Datenstruktur. Sercos verwendet bis zu vier Ethernet-Frames als Downlink und vier Ethernet-Frames als Uplink. Die Downlink-Frames werden als Master-Data-Telegramme (MDT) bezeichnet und die Uplink-Frames als Acknowledge-Telegramme (AT). MDTs werden vor jedem GSP vom Master an die Slaves gesandt. Nach jedem GSP sendet der Master leere ATs, die während der Übertragung „on-the-fly” von jedem Slave mit seinen eigenen Daten ergänzt werden. Dazu ist jedem Datum von jedem Slave eine eigene feste Position in den Telegrammen zugewiesen. Wenn die ATs aufgrund der Topologie des Netzwerks zum Master zurückkehren enthalten sie somit die Daten aller Slaves. Der zeitliche Ablauf ist in Abbildung 2.1 dargestellt.

Jedes Telegramm enthält dabei eine Mischung aus allen genannten Datentypen, wobei die interne Struktur teils vorgegeben ist und teils vom IE-M frei gestaltet werden kann. In jedem Fall muss die Struktur aber allen Teilnehmern während der Konfigurationsphase des Netzwerks mitgeteilt werden und darf sich danach nicht mehr ändern. Da jedes Telegramm Daten mit unterschiedlichen Quellen und Zielen bündelt, werden die Telegramme auch Summentelegramme genannt. Im Gegensatz zu eigenen Ethernet-Frames für jedes Datum, wird durch Summentelegramme der Overhead reduziert und somit die Nettodatenrate gesteigert. Der Nachteil ist, dass bei einem einzigen Übertragungsfehler der gesamte Frame verworfen werden muss und

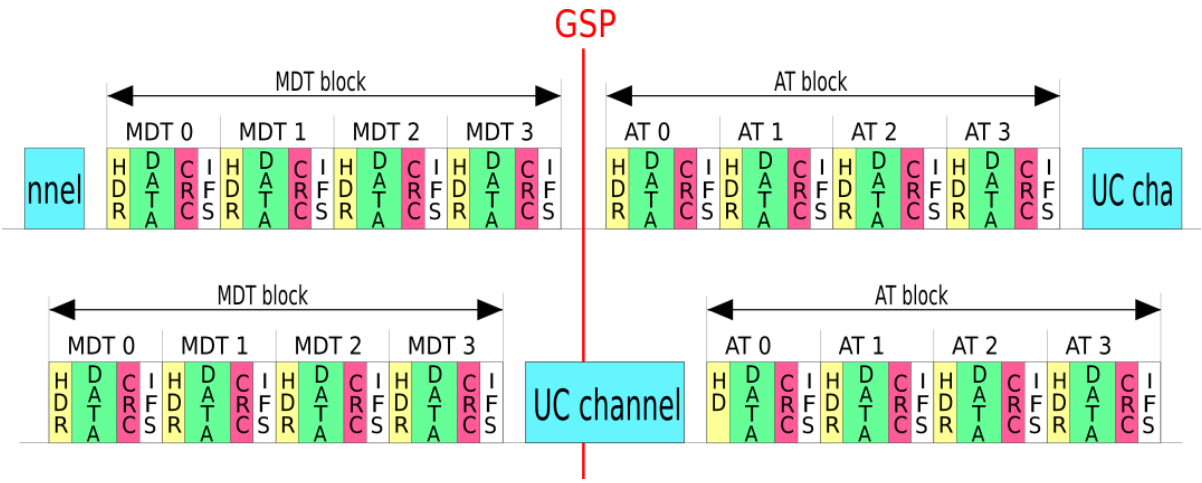


Abbildung 2.1: Die zwei Varianten zum Aufbau eines Sercos-Zyklus. Sie unterscheiden sich nur in der Platzierung des UC-Kanals. Der dargestellte Zyklus wiederholt sich periodisch in beide Richtungen. Die Grafik ist an eine Abbildung aus [10, S. 210] angelehnt.

somit viele Daten auf einmal ausfallen. Weil die Bitfehlerrate eines Ethernet-Netzwerks sehr gering ist, fällt das Problem allerdings nicht ins Gewicht. Sercos verwendet daher auch keinen Fehlerkorrektur-Mechanismus.

Daten, die nicht von IE-Komponenten selbst stammen, sondern nur durch das Netzwerk getunnelt werden sollen, werden anders behandelt. Sercos garantiert für diese Daten weder eine bestimmte Zuverlässigkeit noch eine spezielle Latenz oder Datenrate. In dieser Arbeit werden sie daher als Best-Effort-Daten (BE-Daten) bezeichnet. Sie werden in normalen Ethernet-Frames übertragen, wenn keine anderen Übertragungen anstehen. Sercos bezeichnet diesen Zeitraum als Unified Communication (UC) Kanal. Der Kanalzugriff erfolgt währenddessen nicht deterministisch durch den IE-M gesteuert, sondern über das normale Ethernet-CSMA/CD-Schema. Der Vorteil dieses Vorgehens besteht darin, dass der nicht-deterministische Kanalzugriff für die BE-Daten besser geeignet ist, da sie nur ein statistisches Auftreten besitzen. Die Platzierung des UC Kanals kann an zwei Stellen im Zyklus erfolgen, wie in Abbildung 2.1 zu sehen ist.

2.4 Parallel Sequence Spread Spectrum

Parallel Sequence Spread Spectrum (PSSS) ist eine Funktechnologie, das auf der Methode der Codespreizung mit quasiorthogonalen Spreizfolgen beruht und Code Division Multiple Access (CDMA) erlaubt. Obwohl die Technik bereits 2004 in [11] beschrieben und in [12] patentiert wurde, sowie 2006 als Teil des IEEE-802.15.4-Standards [13] eingesetzt wurde, wird sie in der Praxis bisher kaum verwendet. Durch die hohe Flexibilität und geringe Latenzzeit ist PSSS jedoch ein vielversprechendes Verfahren für die Echtzeitkommunikation.

Da die Technik sich noch in der aktiven Weiterentwicklung befindet, sind nur wenige Quellen zum PSSS-Verfahren verfügbar. Die folgende Erklärung der Technik folgt [14], außer dort wo andere Quellen angegeben sind.

2.4.1 Grundlagen der PSSS Übertragung

PSSS ist, wie auch das bekanntere Direct Sequence Spread Spectrum (DSSS) Verfahren, ein Codespreizverfahren. Bei DSSS wird jedes zu übertragende Bit mit einer festen Spreizfolge multipliziert, wobei ein Spreizfolge erstmal einer gewöhnlichen Bitfolge entspricht. Das Produkt wird anschießend impulsgeformt, moduliert und übertragen. Ein Sendesymbol entspricht daher einer Spreizfolge und trägt ein Bit Information. Zur Rekonstruktion wird am Empfänger das Signal mit der Spreizfolge korreliert, also punktweise multipliziert und aufsummiert. Daraus ergibt sich das ursprüngliche Nutzbit. Um die Intra-System Kompatibilität sicherzustellen, werden als Spreizfolgen für verschiedene Systeme meist orthogonale Pseudozufallsrauschsequenzen (PN-Sequenzen) verwendet. Da die Kreuzkorrelation der Spreizfolgen beider Systeme somit Null ist, beeinflussen sich die beiden Übertragungen nicht.

Der Ansatz von DSSS wird bei PSSS durch die parallele Nutzung mehrerer Spreizfolgen ergänzt. Anstatt dass mehrere Systeme parallel senden, wird die Orthogonalität der Spreizfolgen dazu genutzt mit einem einzelnen System mehrere Bits zeitgleich zu übertragen. Die PN-Sequenzen, die dabei als Spreizfolgen zum Einsatz kommen, werden über eine Maximum Length Sequence (MLS oder m-Sequenz) erzeugt.

m-Sequenzen

Eine m-Sequenz besitzt die Eigenschaft, dass alle zyklischen Rotationen der Sequenz quasiorthogonal zueinander sind. Daher wird für jede Spreizfolge eine andere zyklische Rotation derselben m-Sequenz genutzt. Beträgt die Länge einer m-Sequenz n Chips, dann lassen sich entsprechend n zyklische Rotationen generieren und damit n Bits parallel übertragen. Mit dem Begriff Chips werden die „Quasi-Bits" einer Spreizfolge bezeichnet. Die Erzeugung von m-Sequenzen erfolgt mithilfe von Linear Feedback Shift Registern (LFSRs). Weil sich dabei die Länge der m-Sequenz über $n = 2^p - 1$ aus der Anzahl p der LFSRs ergibt, sind keine Folgen beliebiger Länge möglich, sondern nur solche die diese Bedingung mit ganzzahligen p erfüllen.

Als Notation für eine m-Sequenz, die aus den Chips m_0 bis m_{n-1} mit $m_i \in \{+1, -1\}$ besteht wird im Folgenden

$$\mathbf{m}_{\to 0} = [m_0 \quad m_1 \quad \cdots \quad m_{n-1}]^\mathsf{T} \tag{2.1}$$

verwendet. Um die zyklische Rotation von $\mathbf{m}_{\to 0}$ um i Stellen auszudrücken wird $\mathbf{m}_{\to i}$ verwendet. Die erste zyklische Rotation würde dementsprechend

$$\mathbf{m}_{\to 1} = [m_{n-1} \quad m_1 \quad \cdots \quad m_{n-2}]^{\mathsf{T}} \tag{2.2}$$

sein. Werden alle Verschiebungen einer m-Sequenz hintereinander gesetzt, entsteht die Matrix

$$\mathbf{E} = [\mathbf{m}_{\to 0} \quad \mathbf{m}_{\to 1} \quad \cdots \quad \mathbf{m}_{\to n-1}] = \begin{bmatrix} m_0 & m_{n-1} & \cdots & m_1 \\ m_1 & m_0 & \cdots & m_2 \\ \vdots & \vdots & \ddots & \vdots \\ m_{n-1} & m_{n-2} & \cdots & m_0 \end{bmatrix}. \tag{2.3}$$

Wie bereits erwähnt sind m-Sequenzen „quasiorthogonal”. Der Begriff wird verwendet, da keine echte Orthogonalität besteht, aber die Korrelation so gering ist, dass m-Sequenzen sich in der Praxis wie orthogonale Sequenzen verwenden lassen. Die Kreuzkorrelation zweier unterschiedlicher Rotationen der gleichen m-Sequenz ergibt dagegen $-\frac{1}{n}$ anstatt Null. Die angegebenen Werte gelten genau dann, wenn die Amplitude der m-Sequenz auf $\pm\sqrt{n}$ gesetzt wird. Mit dem Korrelationsoperator $\otimes$ ist die Quasiorthogonalität formal als

$$\mathbf{m}_{\to i} \otimes \mathbf{m}_{\to j} = \sum_{k=0}^{n-1} m_{(k-i) \bmod n} \cdot m_{(k-j) \bmod n} = \begin{cases} +1 & \text{für } i = j \\ -\frac{1}{n} & \text{sonst} \end{cases} \tag{2.4}$$

definiert.

PSSS-Codierung

Zu übertragen sind n bipolare Datensymbole d_i, die analog zur m-Sequenz, in einem Vektor

$$\mathbf{d} = [d_0 \quad d_1 \quad \cdots \quad d_{n-1}]^{\mathsf{T}} \tag{2.5}$$

zusammengefasst werden.

Die Erzeugung eines PSSS-Symbols wird jedes Nutzbits d_i mit einer anderen Spreizfolge $\mathbf{m}_{\to i}$ gespreizt, also multipliziert. Die resultierenden Sequenzen $d_i \cdot \mathbf{m}_{\to i}$ werden anschließend über alle i aufsummiert, sodass sich eine einzige n-wertige Sequenz, das sogenannte PSSS-Symbol, ergibt, das anschließend übertragen wird. Das Vorgehen ist in Abbildung 2.2 skizzenhaft dargestellt.

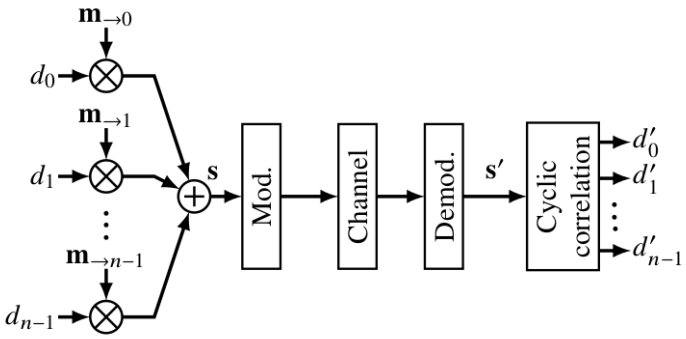


Abbildung 2.2: Das Schema der PSSS-Codierung. Die Abbildung ist leicht modifiziert aus [14] übernommen.

Der ganze Vorgang kann effizient über eine Matrixmultiplikation der Encodermatrix **E** mit dem Datenvektor **d** berechnet werden. Das Ergebnis

$$\mathbf{s} = \mathbf{E} \cdot \mathbf{d}. \tag{2.6}$$

ist der PSSS-Symbolvektor **s**.

PSSS-Decodierung ohne Kanaleinfluss

Zur Verdeutlichung des Prinzips der Decodierung wird zuerst der Kanaleinfluss außer Acht gelassen. Somit entstehen keine Verzerrungen oder Übertragungsfehler beim Empfang und der Detektion von **s**. Bei der Decodierung wird die Orthogonalität der rotierten m-Sequenzen ausgenutzt. Der Symbolvektor **s** wird wieder mit den jeweiligen m-Sequenzen korreliert. Dazu wird nicht die Encodermatrix **E**, sondern die Decodermatrix

$$\mathbf{D} = \mathbf{E}^{\mathrm{T}} \tag{2.7}$$

verwendet. Das Ergebnis der Zwischenrechnung

$$\mathbf{c} = \mathbf{D} \cdot \mathbf{s} = [c_0 \quad c_1 \quad \cdots \quad c_{n-1}]^{\mathrm{T}} \tag{2.8}$$

entspricht noch nicht dem ursprünglichen Ausgangsvektor d, da die m-Sequenz nicht orthogonal sondern nur quasiorthogonal sind. Die unterschiedlichen Spreizfolgen haben daher einen störenden Einfluss aufeinander. Wegen des fehlenden Kanaleinflusses lassen sich Ursprungs-

daten fehlerfrei über eine Schwellwertentscheidung

$$d_i = \begin{cases} +1 & \text{für } c_i \geq 0 \\ -1 & \text{für } c_i < 0 \end{cases} \tag{2.9}$$

aus **c** zurückgewinnen.

PSSS-Decodierung mit Kanaleinfluss

In der Realität wird das PSSS-Symbol *s* durch den Kanaleinfluss zu

$$\mathbf{s}' = \mathbf{s} * \mathbf{h} + \mathbf{n}(t) \tag{2.10}$$

verzerrt, wobei **h** die Kanalimpulsantwort im betrachteten Zeitabschnitt und $\mathbf{n}(t)$ ein additives Rauschen beschreibt. Zur Vereinfachung wird das Rauschen im Folgenden vernachlässigt. In der Kanalimpulsantwort sind alle Einflüsse durch das Ausbreitungsmedium, die Umwelt und die Funkkomponenten zusammengefasst.

Um die Decodierung nach dem obigen Schema durchführen zu können, muss $\mathbf{s}'$ vom Kanaleinfluss bereinigt werden. Der Prozess bei dem Kanal- und Funkbeeinträchtigungen kompensiert werden, nennt sich Kanalentfaltung. Dazu wird eine Schätzung der Kanalimpulsantwort benötigt, um die inverse Kanalimpulsantwort $\mathbf{h}_{ZF}$ zu berechnen. Sie ergibt sich durch invertieren der Fourier-Transformierten **H** des Kanals:

$$\begin{aligned} \mathbf{h} \quad &\circ\!\!-\!\!\bullet \quad \mathbf{H} \\ \mathbf{H}^{-1} \quad &\bullet\!\!-\!\!\circ \quad \mathbf{h}_{ZF} \quad . \end{aligned} \tag{2.11}$$

Anstatt zuerst $\mathbf{s}'$ zu **s** zu entfalten und dann nach obigem Schema vorzugehen, kann auch die Decodermatrix entsprechend modifiziert werden. Da dieser Schritt nur einmalig nach der Kanalschätzung durchgeführt werden muss, kann dadurch die Verarbeitungslatenz der Daten beim Empfang reduziert werden. Um das modifizierte $\mathbf{D}'$ zu erhalten, werden die einzelnen Spreizfolgen mit $\mathbf{h}_{ZF}$ gefaltet:

$$\mathbf{m}'_{\to 0} = \mathbf{h}_{ZF} * \mathbf{m}_{\to 0} \tag{2.12}$$

und anschließend – analog zu Gleichung 2.7 – zu $\mathbf{D}'$ zusammengesetzt.

Zur Decodierung wird dann analog wie vorher vorgegangen. Zuerst wird $\mathbf{c}'$ mittels

$$\mathbf{c}' = \mathbf{D}' \cdot \mathbf{s}' \tag{2.13}$$

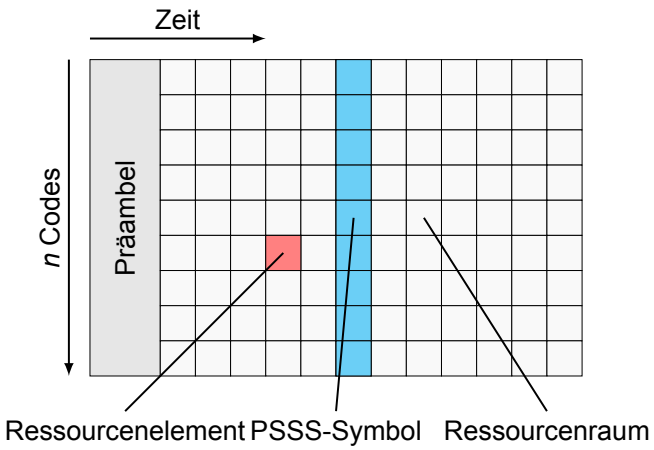


Abbildung 2.3: Der Aufbau eines PSSS Frames.

berechnet und anschließend per Schwellwertentscheidung $\mathbf{d}'$ gebildet.

Da durch unvollständige Kompensation des Kanaleinflusses und Rauschen das Signal verfälscht wird, treten bei der Schwellwertentscheidung zwangsläufig Fehler auf. Da durch die Quasiorthogonalität der Spreizfolgen aber bereits ohne Störungen Werte nahe der Entscheidungsschwelle angenommen werden, ist die Fehlerwahrscheinlichkeit mit diesem simplen Decoder recht hoch. Auf ein erweitertes Decoderschema, das die wechselseitige Störung der quasiorthogonalen Spreizfolgen zu kompensieren versucht, wird in Abschnitt 2.4.5 eingegangen.

2.4.2 Aufteilung in PSSS-Frames

Die Übertragung von Nutzdaten kann nicht ununterbrochen stattfinden. In regelmäßigen Abständen muss die Übertragung durch sogenannte Präambeln unterbrochen werden. Diese sind sowohl für die Synchronisation als auch zur Kanalabschätzung wichtig. Der Abstand der Präambel wird so gewählt, dass der Kanal $\mathbf{h}$ zwischen zwei Präambeln annähernd konstant ist. Er ergibt sich somit aus der Kanalkohärenzzeit. Weil während der Präambeln keine Nutzdaten übertragen werden können, wird der Abstand aber möglichst groß und somit nur knapp kleiner als die minimale erwartete Kanalkohärenzzeit gewählt.

Da eine Präambel jeweils die Grundlage für alle Übertragungen bis zur nächsten Präambel darstellt, bilden diese eine logische Einheit. Die Vereinigung der Präambel mit den folgenden PSSS-Symbolen wird PSSS-Frame genannt. Nach einem PSSS-Frame und vor der nächsten Präambel kann ein Inter-Frame-Space (IFS), also eine kurze Sendepause, eingefügt werden. Dieser kann genutzt werden um die Frequenz, in der Frames gesendet werden, zu reduzieren; er ist aber optional. Daher gibt es für die Zuordnung von Daten zu PSSS-Frames zwei

verschiedene Prinzipien beziehungsweise Deutungen. Wird jeder Frame als eine abgeschlossene Einheit betrachtet, dann spricht man von einem Frame-basierten Schema. Werden dagegen alle PSSS-Symbole als ein kontinuierlicher Datenstrom betrachtet, der gelegentlich von Präambeln unterbrochen wird, dann wird das als Stream-basiertes Schema bezeichnet. Ein Frame-basiertes Schema wird häufig mit IFS kombiniert um die Framerate mit anderen Systemen abzugleichen. Vorteil eines Stream-basierten Schemas ohne IFS ist hingegen, dass eine maximale Datenrate erreicht wird. Außerdem ermöglicht die Loslösung vom Denken in separaten Frames eine höhere Flexibilität beim Scheduling der Daten. Genauere Informationen zu den beiden Schemata sind in [15] zu finden. In dieser Arbeit wird das Stream-basierte Schema genutzt um die höhere Flexibilität und Datenrate ausnutzen zu können.

Die Inhalte der Frames, die zur Übertragung von Daten genutzt werden können, werden als Ressourcen bezeichnet. Die kleinste informations-tragende Einheit stellt dabei das Ressourcenelement (RE) dar, das genau ein Bit fassen kann. Die Gesamtheit aller Ressourcenelemente wird Ressourcenraum genannt. Weil der MAC-Layer die Kontrolle über alle Ressourcen im Ressourcenraum hat, werden diese teilweise auch MAC-Ressourcen genannt. In Abbildung 2.3 werden diese Begriffe veranschaulicht.

2.4.3 Multiplexing

Bisher wurde nur die Kommunikation zwischen einem einzelnen Sender und Empfänger betrachtet. Der Anwendungsfall in der Industriekommunikation sieht allerdings mehrere Teilnehmer im selben Kanal vor. Daher muss ein Multiple-Access-Verfahren eingeführt werden. PSSS erlaubt den Einsatz von Time Division Multiple Access (TDMA), Frequency Division Multiple Access (FDMA) und Code Division Multiple Access (CDMA). Insbesondere in [15], aber auch in [16] und [17] werden die Mehrzugriffsverfahren tiefgehender besprochen.

Die am einfachsten umzusetzende Topologie ist die Sterntopologie. Diese korrespondiert außerdem gut mit dem vorgesehenen Anwendungsfall, da in der Industriekommunikation eine logische Sternstruktur vorzufinden ist. Die zentrale Funkkomponente wird nachfolgend als PSSS-Master und die anderen Teilnehmer werden als PSSS-Slaves bezeichnet. Zum Duplexing wird Frequency Division Duplex (FDD) gewählt. Dadurch können der Downlink (DL) und Uplink (UL) vollkommen unabhängig voneinander verwendet werden und pro Richtung steht die volle Datenrate zur Verfügung, die mit einem PSSS-System erreicht werden kann.

Der DL unterscheidet sich in technischer Sicht nicht von der bereits beschriebenen Punkt-zu-Punkt Kommunikation, da es nur einen Sender gibt und es für diesen irrelevant ist, ob seine Übertragung von einem oder mehreren Teilnehmern empfangen wird. Auf logischer Ebene, im Data Link Layer (DLL), wird sowohl Zeit- als auch Codemultiplex verwendet. Das ist aber kein

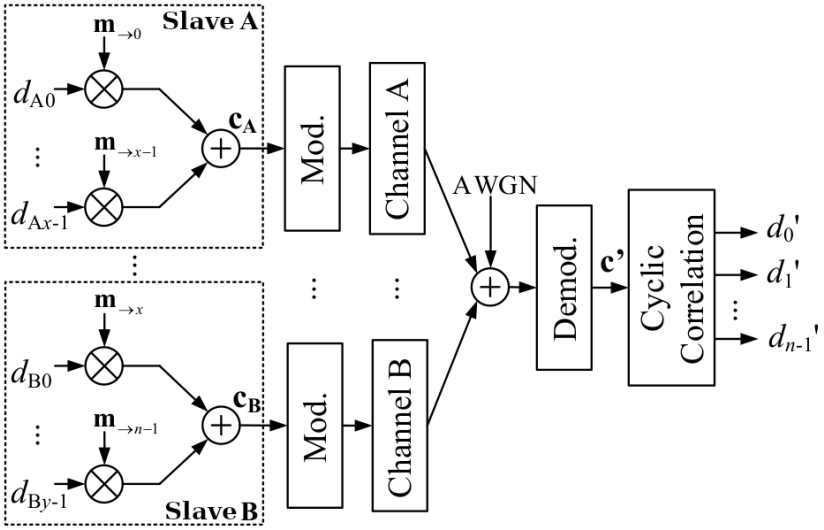
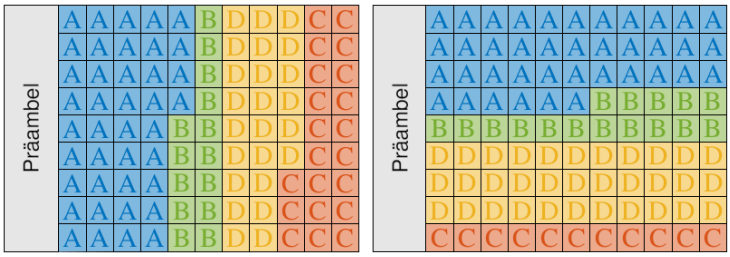


Abbildung 2.4: Die Überlagerung von zwei PSSS-Signalen am Empfänger. Die Abbildung ist leicht modifiziert aus [18] übernommen.

Mehrfachzugriff, da der Sender nicht mit anderen Sender um den Kanal konkurrieren muss. Es ist anzumerken, dass durch unterschiedliche Ausbreitungspfade zu den verschiedenen Empfängern die Empfangsqualität und der Empfangszeitpunkt zwischen den Empfängern abweichen.

Beim UL ist die technische Umsetzung aufwendiger, da alle Sender koordiniert auf den Kanal zugreifen müssen. TDMA ist bei ausreichender Synchronizität der Sender leicht umzusetzen, indem alle Sender während der ihnen nicht zugewiesenen PSSS-Symbole inaktiv bleiben. Da die Synchronisation auf den Präambeln beruht, muss kein durchgängiger Träger vorhanden sein. PSSS ermöglicht darüber hinaus den Einsatz von CDMA. Damit die Überlagerung der Signale mehrerer Sender beim Empfänger ein konsistentes PSSS-Symbol erzeugt, muss die Leistung aller Signale am Empfänger identisch sein. Da das nur an einem Punkt der Fall sein kann, ist die Verwendung von CDMA auf eine Sterntopologie beschränkt. Abbildung 2.4 zeigt das Schema einer UL-Übertragung. In der Abbildung sind zwei Slaves – Node A und Node B – dargestellt. Node A hat die Codes von $\mathbf{m}_{\rightarrow 0}$ bis $\mathbf{m}_{\rightarrow x-1}$ zugewiesen, um die Daten $\mathbf{d_A} = (d_{A_0}, \cdots, d_{A_{x-1}})$ zu senden. Node B hat entsprechend die anderen Sequenzen zugewiesen. Jeder Node sendet seine Daten analog zum bisher bekannten Vorgehen, codiert aber nur mit dem entsprechenden Teil der Encodermatrix E. Am Empfänger überlagern sich die Signale, was einer Addition der Teilsignale gleichkommt. Der Empfänger erhält somit einen Symbolvektor s', den er wie gehabt decodieren kann. Tiefer gehende Informationen sind in [17] und [18] zu finden.

Die Zuweisung der Codes zu den Sendern erfolgt nicht statisch, sondern kann sich in jedem PSSS-Symbol ändern. Sie wird durch den MAC-Layer geregelt. Die Zuweisung erfolgt in der



(a) Sequentielle Anordnung **(b)** Parallele Anordnung

Abbildung 2.5: Mögliche Anordnungen der Ressourcenblöcke. Die Abbildung ist leicht modifiziert aus [15] übernommen.

Form von Ressourcen Blöcken (RBs), die jeweils eine zusammengehörige Gruppe Ressourcenelemente darstellen. Sie können in verschiedenen Konfigurationen angeordnet werden, wie in Abbildung 2.5 beispielhaft dargestellt ist.

2.4.4 Transmit Power Control und Timing Advance

Ein Problem beim TDMA und CDMA im UL ist, dass die Signale synchron und mit der gleichen Leistung beim Master ankommen müssen. Unter realen Bedingungen sind die Slaves jedoch über eine breite Umgebung verteilt. Dadurch weisen die Verbindungen sehr unterschiedliche Ausbreitungspfade und somit sehr verschiedene Eigenschaften auf. Würden alle Slaves mit derselben Leistung senden, kämen die Signal daher mit unterschiedlicher Signalstärke beim Master an. Zusätzlich würde das Signal der näher gelegenen Slaves früher eintreffen als das Signal der weiter entfernten Slaves. Durch mobile Funkkomponenten und eine veränderliche Umgebung sind diese Eigenschaften zusätzlich zeitvariant.

Um diese Unterschiede zu kompensieren, müssen zwei Techniken angewandt werden. Die erste, die Transmit Power Control (TPC), ist eine aktive Regelung der Sendeleistung aller Slaves, sodass die jeweilige Empfangsleistung beim Master für alle Slaves identisch ist. Dazu muss der Master alle ankommenden Signale messen und den Slaves mitteilen, in welchem Maß sie ihre Sendeleistung anpassen müssen. Um einen möglichst guten Stör-Rausch-Abstand zu erhalten, muss die Sendeleistung dabei außerdem so geregelt werden, dass der Slave mit den schlechtesten Kanaleigenschaften mit seiner maximalen Sendeleistung sendet. Alle anderen Slaves müssen entsprechend herunter geregelt werden. Die maximale Sendeleistung ergibt sich dabei durch gesetzliche Vorgaben für das jeweilige Frequenzband. Um außerdem den zeitlichen Versatz zu kompensieren muss als zweite Technik das Timing Advance (TA) eingesetzt werden. TA ist in vielen Punkten vergleichbar mit TPC, stellt im Unterschied dazu aber eine Regelung des Sendezeitpunkts jedes Slaves dar. Die Slaves bekommen vom

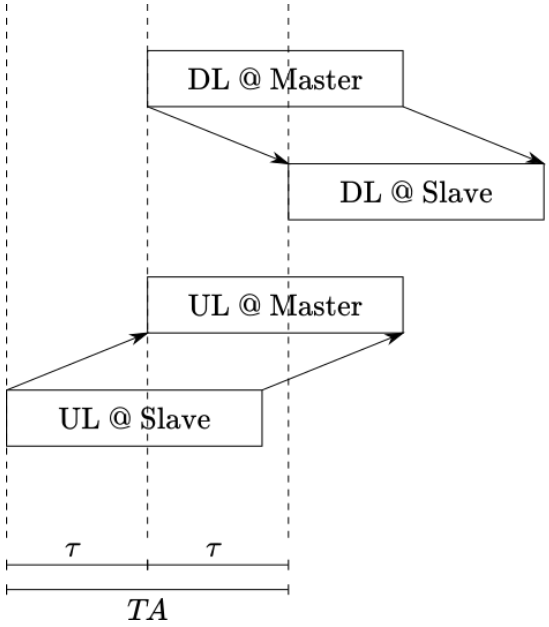


Abbildung 2.6: Veranschaulichung des TA-Mechanismus. Das dargestellte UL-Symbol soll am Master keinen Versatz zum DL-Symbol aufweisen. Dazu muss das UL-Symbol um $TA = 2\tau$ früher gesendet werden als das DL-Symbol empfangen wird, wobei τ die Laufzeit über den Ausbreitungspfad bezeichnet.

Master abhängig von der Länge ihres Signalpfads einen Offset vorgegeben, um den sie ihren Sendezeitpunkt anpassen sollen. Der Begriff „Sendezeitpunkt" bezieht sich hier auf den genauen Zeitpunkt, zu dem die Übertragung eines PSSS-Symbols begonnen wird. Der Offset wird relativ zum Empfangszeitpunkt eines PSSS-Symbols im Downlink angegeben. Das ist in Abbildung 2.6 dargestellt.

In Abschnitt 3.2.3 wird erarbeitet welche Präzision die TPC/TA-Daten besitzen müssen um eine ausreichende Signalanpassung herzustellen. Daraus wird außerdem abgeleitet welches Datenvolumen durch die TPC/TA-Daten erzeugt wird.

2.4.5 Bitfehlerwahrscheinlichkeit

Die Bitfehlerrate vom Spread Spectrum Verfahren ohne ein CDMA-Schema entspricht nach [17] der Bitfehlerrate eines BPSK-Systems

$$BER_{BPSK} = \frac{1}{2}\,\mathrm{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right) \tag{2.14}$$

mit der Bitenergie E_b und der Rauschleistungsdichte N_0 in einem angenommenen AWGN-Kanal. Da die Codes des CDMA-Schemas bei PSSS allerdings nicht vollkommen orthogonal

sind, beeinflussen sich die verschiedenen Signale gegenseitig. In [17] wurde daher das Bitfehlerverhalten eines PSSS-Systems näher untersucht. Es zeigt sich, dass sich durch das Einbringen eines zusätzlichen Korrekturterms, der den wechselseitigen Einfluss der Signale modelliert, die Bitfehlerrate des PSSS-Systems aus der des BPSK-Systems abgeleitet werden kann. Zur tatsächlichen Berechnung der Bitfehlerrate muss dabei zusätzlich eine Annahme über die stochastische Verteilung der Bitwerte im Nutzsignal getroffen werden. In [18] wird allerdings der Einsatz eine Iterativen-Symbol-Decoders vorgeschlagen, der in der Lage ist mit geringem Aufwand die Beeinflussung der Signale größtenteils zu kompensieren und so die Bitfehlerrate zu verbessern. Eine simulative Auswertung dieses Ansatzes durch die Autoren von [18] hat gezeigt, dass die Bitfehlerrate unter diesem Decoder tatsächlich ausreichend exakt mit der Bitfehlerrate des BPSK-Systems modelliert werden kann.

Voraussetzung dafür ist allerdings bei einem Multiple-Access Szenario, dass alle Signale den Empfänger ausreichend synchron erreichen, damit das PSSS-Symbol nicht verfälscht wird. In [17] wird auch die Auswirkung verschiedener Stärken der Desynchronisation untersucht. Bei einem zeitlichen Versatz von weniger als $0.1 \cdot T_{chip}$ kann demnach noch davon ausgegangen werden, dass die Decodierung kaum beeinflusst wird.

Schließlich wurde in [14] die Bitfehlerverteilung bei der Übertragung mit dem Prototypen eines PSSS-Systems experimentell analysiert. Die Ergebnisse legen nahe, dass die Verteilung der Bitfehler über alle Ressourcenelemente eines PSSS-Frames einer Gleichverteilung entspricht, wenn die Framelänge passend zur Kanalkohärenzzeit gewählt wird. Das Fehlen von Bitfehlerclustern ist eine willkommene Eigenschaft für jene Fehlerkorrekturverfahren, die bei Burstfehlern versagen. Ein Scrambler-Schritt, wie er in vielen Kommunikationskanälen eingesetzt wird, die Burst-artige Fehlerverteilungen zeigen, ist damit vermutlich bei PSSS-System nicht nötig.

2.4.6 ParSec

Im Forschungsprojekt ParSec [1] wird ein Funksystem entwickelt, das auf dem PSSS-Verfahren basiert. Es ist eine Kooperation verschiedener Institute und Unternehmen und wird vom Bundesministerium für Bildung und Forschung gefördert. ParSec soll die Vorteile von PSSS – die geringe Latenz, hohe Flexibilität und hohe Zuverlässigkeit – für die Industriekommunikation nutzbar machen. Es ist als Anwendung im unlizensierten 5 GHz Band, zwischen 5,725 GHz und 5,875 GHz geplant. Für die Prototypen wurde bisher eine Chipdauer $T_{chip} = 50\,\text{ns}$ und ein Raised-Root-Cosine-Filter mit einem Roll-Off-Faktor von $r = 1$ verwendet. Alle Angaben entstammen den Veröffentlichungen [16] und [14].

Im Gegensatz zur PSSS-Anwendung im IEEE 802.15.4 Standard, werden bei ParSec alle n verfügbaren Codes zur Übertragung genutzt. Die zusätzliche Datenrate ist für die Nutzung im Industrieumfeld relevant. Gleichzeitig verschwindet dadurch allerdings der Spreizgewinn, der sonst bei Spread Spectrum Verfahren den Stör-Rausch-Abstand (SNR) verbessert. Derzeit ist der Einsatz von m-Sequenzen der Länge $n = 255$ geplant. Zur Reduktion von Inter-Symbol-Interferenzen (ISI) wird jedes Symbol zusätzlich durch einen zyklischen Präfix (CP) von 30 Chips ergänzt. Damit ergibt sich mit der obigen Chiprate eine Symboldauer von $T_{sym} = 14{,}25\,\mu s$.

Die Topologie und das Multiplexing werden wie in Abschnitt 2.4.3 beschrieben umgesetzt. Für das Duplexing wird neben FDD aber auch der Einsatz von CDD in Betracht gezogen.

3 Konzept

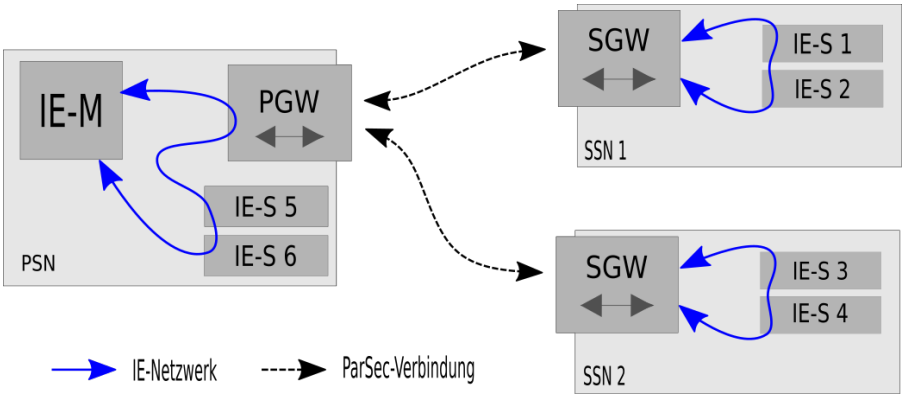
Wie bereits im Einleitungskapitel beschrieben wurde, soll in dieser Arbeit ein Protokoll für ein PSSS-basiertes, hybrides Netzwerk entwickelt werden, das die transparente, drahtlose Erweiterung eines Industrial Ethernets ermöglicht. Der Fokus der Arbeit liegt dabei insbesondere auf dem Entwurf eines anwendungsgerechten Ressourcenzuweisungs-Algorithmus auf MAC-Ebene.

In diesem Kapitel wird der dazu entwickelte Ansatz vorgestellt. Als Erstes wird die Topologie des hybriden Netzwerks beschreiben. Dann wird in Abschnitt 3.2 eine Anforderungsbeschreibung für die verschiedenen zu übertragenden Datentypen erstellt. In Abschnitt 3.3 wird der grundlegende Ablauf der Kommunikation im hybriden Netzwerks beschrieben. Die Algorithmen zum Scheduling der Echtzeitdaten werden schließlich in Abschnitt 3.4 vorgestellt.

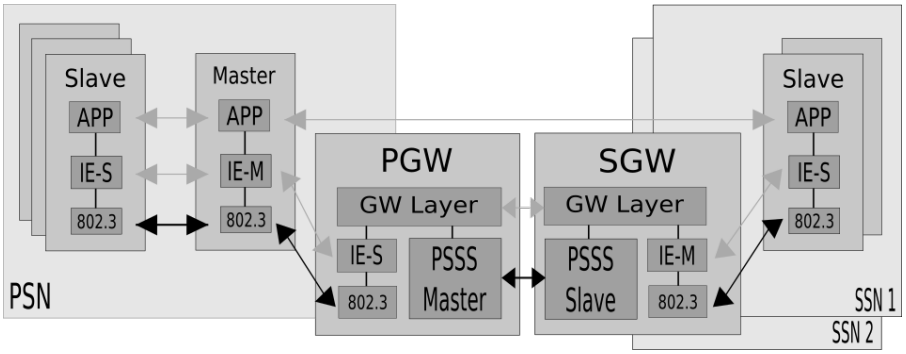
3.1 Topologie

Bei der Erweiterung eines IE durch Funktechnologie stellt vor allem die deterministische Echtzeitanforderung der RT-Daten eine große Hürde dar. In [19] haben die Autoren untersucht welcher Systemaufbau dafür am besten geeignet ist. In diesem Abschnitt wird ihr Ergebnis zusammenfassend vorgestellt. Die Benennung der einzelnen Komponenten weicht von der Quelle ab um hier ein einheitliches Namensschema beizubehalten.

Der Aufbau sieht vor, mehrere, räumlich getrennte IEs über ein PSSS-basiertes Backbone miteinander zu koppeln. Der Verbund der IEs mit dem Backbone wird im Folgenden hybrides Netzwerk (HN) genannt. Auf dem HN wird ein einzelnes logisches IE aufgebaut, das im Folgenden als fusioniertes Netzwerk (FN) bezeichnet wird. Die einzelnen IEs sind die Subnetzwerke (SNs). Davon enthält eines den globalen Industrial Ethernet Master (IE-M) des gesamten FNs. Es wird deshalb als primären Subnetzwerk (PSN) bezeichnet. Die restlichen SNs werden sekundäre Subnetzwerke (SSNs) genannt. Eine schematische Darstellung des gesamten Aufbaus ist in Abbildung 3.1a zu finden.



(a) Topologie des hybriden Netzwerks



(b) Detailansicht der Komponenten

Abbildung 3.1: Der schematische Grundaufbau des hybriden Netzwerks. Die Abbildungen sind an [19] angelehnt.

Um die Kopplung der SNs zu erreichen, wird in jedem davon ein Gateway platziert, das einen Proxy für den Zugang zum restlichen HNs bereitstellt. Damit keine Anpassungen der IE-Komponenten nötig sind, wird der Proxy vollkommen transparent realisiert. Das heißt, dass er selbst nicht als IE-Komponente in Erscheinung tritt, sondern nur die Industrial Ethernet Slaves (IE-S) aus den anderen SNs emuliert. Die Proxys in den SSNs fungieren zusätzlich jeweils als lokaler IE-M. Sie betreiben den IE ihres jeweiligen SSNs so, dass er mit den Vorgaben des globalen IE-M kompatibel ist, aber nicht zwangsläufig identisch. Wie genau dies umgesetzt wird, wird in der Quelle nicht weiter behandelt, sondern in Abschnitt 3.3 selbst erarbeitet.

Das primäre Gateway (PGW) unterscheidet sich derweil nicht nur hinsichtlich des Verhalten des Proxys von den sekundären Gateways (SGWs). Es übernimmt auch im Backbone die Sonderrolle des PSSS-Masters (PS-M). Dies ergibt sich daraus, da das PGW mit allen SGWs in Kontakt stehen muss, während jedes SGW nur mit dem PGW in Kontakt steht. Die SGWs bilden somit die PSSS-Slaves (PS-S). Eine schematische Darstellung der Gateways ist in Abbildung 3.1b zu finden.

3.2 Anforderungsbeschreibung

Dieser Abschnitt soll eine Anforderungsbeschreibung liefern, an der sich der Systementwurf orientiert. Dazu werden die Anforderungen und Einschränkungen diskutiert, die sich aus dem Systemaufbau ergeben und einige Vereinfachungen eingeführt, die in dieser Arbeit angewendet werden.

3.2.1 Betriebsbereich

Das hier entwickelte Konzept spezialisiert sich auf den laufenden Betrieb des hybriden Netzwerks. Dementsprechend wird angenommen, dass alle Parameter der Übertragung bereits im Voraus ausgehandelt wurden und allen Teilnehmern bekannt sind. Weiterhin werden keine Änderungen an der Topologie des Netzwerks unterstützt. Dazu gehören sowohl Änderungen am PSN oder SSN, wie beispielsweise Hot-Plugging oder Veränderungen der Producer-Zyklen, als auch Änderungen an der Topologie des Backbones.

Die Beschränkung auf den laufenden Betrieb erscheint gerechtfertigt, da die Setup-Phasen abweichende Anforderungen aufweist. Damit sind eigene Protokolle für diese Aufgaben sinnvoll. Nicht zuletzt ist die Anmeldephase in PSSS-Netzwerken außerdem ein noch kaum untersuchtes Feld, das den Rahmen dieser Arbeit übersteigen würde.

Die Fixierung aller Netzwerk-Parameter geschieht vor dem Hintergrund, dass die Analyse und Bewertung des Einflusses solcher Parameter sehr komplex ist. Da eine umfassende Betrachtung des Systems damit nicht mehr im Rahmen dieser Arbeit geleistet werden könnte, konzentriert sich der Entwurf auf statisch parametrierte Netzwerke.

3.2.2 Synchronisation

Zur Synchronisation der verschiedenen IEs ist es notwendig, dass das Funksystem eine präzise Zeitbasis bereitstellt. Die Referenzzeit wird dabei durch das Zeitsignal des primären IEs definiert und muss vom Funksystem propagiert werden. Damit die sekundären IEs als synchronisiert angesehen werden können, muss die Frequenz und Phasenlage der IE-Zyklen auf den Zyklus im primären IE abgestimmt werden und darf danach nicht mehr signifikant driften.

Der niederfrequente Frequenzdrift zwischen den Uhren muss dazu bis in alle Ewigkeit vollständig kompensiert werden. Er würde ansonsten zu einer Verschiebung der Zyklen in den verschiedenen IEs führen und somit eine Desynchronisation und ein Systemversagen herbeiführen.

Der hochfrequente Frequenzjitter innerhalb einiger weniger Zyklen muss hingegen lediglich auf einen vorgegebenen Wert beschränkt werden. Jeder IE-Standard definiert hierbei eine maximale Abweichung der Zyklusdauer vom Sollwert. Im Fall von Sercos III sind dies 100 ppm, also beispielsweise bei einer Zyklusdauer von 1 ms unter 100 ns Abweichung pro Zyklus und bei der maximalen Zyklusdauer von 65 ms unter 6,5 µs Abweichung pro Zyklus. Theoretisch erwähnt der Sercos III Standard zudem eine vergrößerte erlaubte Abweichung von 2000 ppm bei kaskadierten Netzen. In dieser Arbeit wird jedoch die präzisere Angabe aus dem normalen Betrieb als Zielsetzung verwendet. Neben der Vorgabe des Standards muss zudem eine weitere Beschränkung beachtet werden. Die Abweichung zwischen den Zyklen der beteiligten IEs muss gering genug sein, dass die im voraus festgelegte Ressourcenzuteilung auf MAC-Ebene jederzeit eine fristgerechte Übertragung garantiert. Es liegt dazu gegebenenfalls auch in der Verantwortung der MAC-Ebene ausreichende Pufferzeiten für diese Schwankungen einzuplanen.

Die Bestimmung der absoluten Phasenlage ist wichtig, damit Zeitangaben aus den verschiedenen IEs auf eine gemeinsame absolute Referenzzeit umgerechnet werden können. Da der Frequenzdrift kompensiert wird, ändert sich die Phasenlage nur im Rahmen des Frequenzjitters. Da dieser sich im selben Rahmen wie bei einem normalen IE bewegen soll, führt er gegenüber dem IE zu keiner zusätzliche Zeitabweichung. Somit genügt es die absolute Phasenlage in der Setup-Phase zu bestimmen und die oben genannten Anforderungen bezüglich Frequenzdrift- und jitter zu erfüllen.

3.2.3 Übertragung verschiedener Datentypen

Das Backbone muss unterschiedliche Datentypen mit verschiedenen Anforderungen transportieren. Einige davon sind IE-Daten, die weitergeleitet werden müssen, während andere zur Aufrechterhaltung des Backbones selbst notwendig sind. Die folgenden Abschnitte gehen jeweils im Detail auf die einzelnen Datentypen ein.

RT-Daten

Der erste Datentyp besteht aus den echtzeit-kritischen Daten der IE-Connections. Da eine IE-Connection immer genau einen Producer und einen oder mehrere Consumer besitzt, haben auch die RT-Daten immer genau ein Quell-SN und ein oder mehrere Ziel-SN. Als Regelfall wird hier die IE-M-zu-IE-S Kommunikation angenommen, aber die wechselseitige Kommunikation zwischen IE-S in einer beliebigen Auswahl von SNs soll unterstützt werden. Falls Connections existieren, die keinen Consumer außerhalb ihres Quell-SSNs besitzen, müssen deren Daten

trotzdem ins PSN übertragen werden, da der IE-M diese Connections auch dann verwaltet, wenn er selbst kein Consumer ist. Ihr Fehlen würde er somit als Netzwerkfehler auffassen.

Die Steuerdaten des IE-M müssen so übertragen werden, dass die IE-S sie zum GSP innerhalb desselben Zyklus anwenden können. Umgekehrt müssen die Messdaten der IE-S am Ende desselben Zyklus den IE-M erreichen, um für die Berechnung der folgenden Steuerdaten bereitzustehen. Für Connections, die zwischen IE-S in verschiedenen SSNs bestehen, kann diese Anforderung im Allgemeinen aber nicht erfüllt werden, ohne dass nicht-abwärtskompatible Änderungen an den IEs und IE-Ss eingeführt würden. Das ergibt sich daraus, dass ein UL-Telegramm, in dem eine solche Connection transportiert wird, im konsumierenden SSNs strikt nach dem entsprechenden UL-Telegramm im produzierenden SSN gesendet werden müsste. Das ist aber bereits unmöglich, falls eine wechselseitige Übertragung stattfinden soll. Auch ansonsten ist die dazu bei kurzen Zykluszeiten notwendige Latenz schwer zu erreichen, da die Daten über das PGW umgeleitet werden müssen. Stattdessen wird die Anforderung so aufgeweicht, dass Consumer, die sich in einem SSNs befinden, das nicht auch den Producer enthält, die Connection-Daten erst im folgenden Zyklus empfangen müssen.

Für die RT-Daten gilt zudem eine sehr hohe Anforderung bezüglich ihrer Fehlerrate. In [2] wird eine Methode beschrieben um die maximale PER einer Übertragungsstrecke abzuschätzen, wenn gegebene Zuverlässigkeitsanforderungen erfüllt werden müssen. Eine Beispielrechnung in dem Paper zeigt, dass eine maximale PER von 10^{-8} eine realistische Anforderung in der FA ist.

Daher ist die Unterstützung eines FEC- oder ARQ-Schemas für die RT-Daten eine notwendig Voraussetzung. Aufgrund der Latenzanforderungen ist ein ARQ-Schema jedoch nicht umsetzbar. Daher verbleibt nur die Möglichkeit ein FEC-Schema zu nutzen. Entsteht bei der Übertragung dennoch eine nicht zu korrigierende Menge an Fehlern, soll dies zumindest zuverlässig erkannt werden. Bei einem unentdeckten Fehler würde das empfangende Gateway die Daten ansonsten in die IE-Telegramme einfügen und – da es annimmt sie seinen korrekt – einen dazu passenden CRC am Ende der Telegramme generieren. Fehlerschutzmechanismen der anderen IE-Teilnehmer könnten dann nicht mehr greifen, da sie einen gültigen CRC feststellen würden. Bei einem erkannten Fehler könnte das Gateway hingegen den CRC der Telegramme ungültig machen und so die anderen IE-Teilnehmern über den Fehler informieren. Diese Anforderung wird über die maximale Unerkannte-Paketfehler-Rate (U-PER) ausdrückt. In der Literatur waren keine Angaben zu finden, welche U-PER ein FA-System erfüllen muss.

In [2] wird zusätzlich erwähnt, dass einige Systeme – darunter Sercos III – standardmäßig den Verlust von Paketen tolerieren, wenn im vorherigen und nachfolgenden Zyklus keine Fehler aufgetreten sind. Die fehlerhaften Daten werden durch Interpolation rekonstruiert. Bei der Berechnung der PER kann dies entsprechend berücksichtigt werden.

SVC-Daten

Der zweite prozess-relevante Datentyp besteht aus den Service-Daten. Sie werden zwar ebenfalls über die DL- und UL-Telegramme der IEs übertragen, haben jedoch keine Echtzeit-Anforderungen und andere Charakteristiken als die RT-Daten. Obwohl in den IEs genügend Kapazität reserviert wird, um zeitgleich den Service-Channel aller IE-S zu bedienen, wird hier als Regelfall angenommen, dass nur eine geringe Anzahl an Service-Channels zeitgleich verwendet wird. Indem die Daten von den Gateways gepuffert werden, lassen sich zusätzlich Schwankungen im Datenaufkommen und der verfügbaren Übertragungskapazität glätten. Es wird daher vereinfachend davon ausgegangen, dass die Übertragung einer geringen Anzahl an SVC-Daten pro Zyklus stets ausreichend ist. Sollte das mittlere Datenaufkommen über der erwarteten Kapazität liegen, dann können nicht alle SVC-Daten mit endlicher Latenz übertragen werden, und das System darf und soll einen Systemausfall melden, sodass durch den Betreiber entsprechende Änderungen vorgenommen werden können. Solange die Kapazität ausreichend ist, soll die Übertragung der SVC-Daten die gleichen Zuverlässigkeitsanforderungen erfüllen wie bei den RT-Daten. Die Latenz soll währenddessen so gering wie möglich ausfallen.

BE-Daten

Der dritte und letzte Nutzdatentyp besteht aus den Best-Effort-Daten (BE-Daten). Dies sind alle Daten ohne Echtzeit- und Zuverlässigkeitsgarantien. Da sie zudem die geringste Priorität aller Datentypen besitzen, werden sie nur in ansonsten ungenutzten Kapazitäten des Backbones übertragen. Ihre Kennzahlen, wie die durchschnittliche Datenrate, Fehlerrate, Latenz und Queue-Länge können aber zur Bewertung der Performance des Netzwerks herangezogen werden. Eine optimal Umsetzung des HNs lässt sich daher darüber definieren, dass alle Anforderungen der RT- und SVC-Daten erfüllt und die Kennzahlen der BE-Daten optimiert werden.

TPC/TA-Daten

Die Übertragung der TPC/TA-Parameter vom PS-M an die PS-S findet über den PSSS-Kanal selbst statt und muss aufgrund des zeitvarianten Kanal in jedem PSSS-Frame wiederholt werden. Weil der Übertragungsweg digital ist, müssen die Werte der Parameter diskretisiert werden. Wegen der häufigen Übertagung dieser Daten ist es wiederum wünschenswert, dass sie

ein möglichst geringes Datenvolumen aufweisen, da ansonsten die Netto-Datenrate des Kanals stark belastet wird. Im Folgenden wird betrachtet, wie die Diskretisierung der Parameter dazu gestaltet werden muss.

Es wird angenommen, dass der Pfadverlust der ausschlaggebende Faktor für die Dämpfung der Empfangsleistung ist. Damit lassen sich sowohl die TPC- als auch die TA-Parameter direkt auf die Länge der Ausbreitungspfade zurückführen. Zu ermitteln ist jeweils die maximale Spannweite des Wertebereichs der Parameter, $TP_{max} - TP_{min}$ und $TA_{max} - TA_{min}$, sowie ihre minimale Auflösung ΔTP_{min} und ΔTA_{min}. Die optimale Diskretisierung besitzt genau so viele Diskretisierungsstufen, dass der resultierende Wertebereich mit der entsprechenden Auflösung darstellt werden kann. Die Länge des längsten und des kürzesten vorkommenden Ausbreitungspfads werden nachfolgend jeweils mit d_{min} beziehungsweise d_{max} bezeichnet.

Wird die Empfangsleistung über den längsten und kürzesten Pfad gleichgesetzt ergibt sich

$$TP_{min} - \eta \cdot 10\log_{10}(d_{min}) - 20\log_{10}(f) + K = TP_{max} - \eta \cdot 10\log_{10}(d_{max}) - 20\log_{10}(f) + K$$

mit der Sendeleistung TP_{min} beziehungsweise TP_{max} des entsprechenden PS-S in dB, der Sendefrequenz f und einer Konstante K. Umgestellt folgt aus dieser Gleichung

$$\begin{aligned} TP_{max} - TP_{min} &= \eta \cdot 10\log_{10}(d_{max}) - \eta 10\log_{10}(d_{min}) \\ &= \eta \cdot 10\log_{10}\left(\frac{d_{max}}{d_{min}}\right) \quad . \end{aligned} \tag{3.1}$$

Damit ist die Spannweite des Wertebereichs des TPC-Parameters bekannt. Die minimale Auflösung ΔTP_{min} kann sich dagegen an der möglichen Auflösung des eingesetzten Transceivers orientieren. Die Sendeleistung eines handelsüblichen AD9361-Transceiver von Analog Devices kann laut Datenblatt [20] beispielsweise in Schritten von

$$\Delta TP = 0{,}25\,\text{dB} \tag{3.2}$$

eingestellt werden.

Wie sich aus Abbildung 2.6 erkennen lässt, entspricht der TA-Parameter dem Doppelten der Signallaufzeit τ im entsprechen Ausbreitungspfad. Diese berechnet sich wiederum über $\tau = \frac{d}{c_0}$ mit der Vakuumlichtgeschwindigkeit c_0. Also folgt für den Wertebereich des TA-Parameters

$$TA_{max} - TA_{min} = 2\frac{d_{max}}{c_0} - 2\frac{d_{min}}{c_0} \quad . \tag{3.3}$$

Laut [18] sollte die Abweichung der Empfangszeiten verschiedener Signale nicht mehr als

$0.1T_{chip}$ betragen. Damit ist auch die notwendige Auflösung

$$\Delta TA = \frac{T_{chip}}{10} \tag{3.4}$$

gegeben.

Für die Anzahl der Bits des TPC- und TA-Parameters n_{TPC} und n_{TA} gilt damit schließlich

$$n_{TPC} = \left\lceil \log_2 \left(\frac{TP_{max} - TP_{min}}{\Delta TP} \right) \right\rceil = \left\lceil \log_2 \left(\eta \cdot 40 \log_{10} \left(\frac{d_{max}}{d_{min}} \right) \right) \right\rceil \tag{3.5}$$

$$n_{TA} = \left\lceil \log_2 \left(\frac{TA_{max} - TA_{min}}{\Delta TA} \right) \right\rceil = \left\lceil \log_2 \left(20 \frac{d_{max} - d_{min}}{T_{chip} \cdot c_0} \right) \right\rceil \quad . \tag{3.6}$$

Weil jeder PS-S eigene TPC/TA-Parameter übermittelt bekommen muss, beschreiben die obigen Angaben das Datenaufkommen pro PS-S.

Da die TPC/TA-Messungen während der Uplink-Präambeln stattfinden, ist eine Kopplung der Übertragung mit dem dem Uplink-Zyklus sinnvoll. Da Uplink und Downlink aber nicht miteinander gekoppelt sind, folgt, dass die TPC/TA-Daten zu beliebigen Zeitpunkten im Downlink-Zyklus vorkommen können. Da sie, wie die Präambeln, zur Aufrechterhaltung der Verbindung unerlässlich sind, werden sie mit der höchsten Priorität übertragen und unterbrechen die Übertragung jedes anderen Datentyps.

Auch die TPC/TA-Daten sollten durch ein angemessenes FEC-Verfahren geschützt werden um Übertragungsfehler zu reduzieren. Tritt ein nicht zu korrigierender Fehler auf, muss davon ausgegangen werden, dass der gesamte betroffene Kanal für die Dauer des folgenden Frames unbrauchbar ist. Gleiches gilt für die Daten in den Präambeln.

MAC-Daten

Auf der MAC-Ebene entstehen einige weitere Steuerdaten, die beispielsweise für Statusmitteilungen, die Korrektur der Zeitbasis, Zustandsänderungen und dergleichen notwendig sind. Ihre Anforderungen hängen von der Gestaltung des MACs selbst ab und können daher nicht im Voraus analysiert werden.

3.3 Kommunikationsablauf

Nachdem die Topologie des hybriden Netzwerks und die Anforderungen der verschiedenen Datentypen bekannt sind, wird nun der optimale Datenfluss innerhalb des Netzwerks betrach-

tet. Außerdem wird die Aufteilung des PSSS-Ressourcenraums definiert.

Wie bereits in Abschnitt 3.2.3 erwähnt, haben die TPC/TA-Daten die höchste Priorität aller Datentypen. Sie werden daher immer sofort übertragen, sobald sie anfallen.

3.3.1 Übertragung der RT-Daten

Die Übertragung der RT-Daten ist aufgrund der Echtzeit-Anforderungen stark eingeschränkt. In diesem Abschnitt wird untersucht, welcher Kommunikationsablauf eingesetzt werden muss um die Anforderungen erfüllen zu können. Dabei wird zuerst nur der Fall von Connections zwischen IE-M und IE-S betrachtet und später der Umgang mit Connections zwischen verschiedenen IE-S ergänzt.

Im Folgenden wird der Kommunikationsablauf erst einmal nur grob umrissen, um die Rahmenbedingungen für eine präzisere Betrachtung einzuführen. In Abschnitt 3.4 wird dann der genaue Scheduling-Algorithmus für die RT-Daten beschrieben.

Die Arbeitsweise eines IE wurde bereits in Abschnitt 2.3 erklärt; die für die RT-Daten relevanten Information sind aber im Folgenden nochmals kurz zusammengefasst:

- Es gibt einen festen Kommunikations-Zyklus der Dauer T_{cyc}.
- Ein Zyklus beginnt mit einer RT-Datenübertragung vom IE-M zu den IE-S.
- Darauf folgt der GSP, zu dem alle IE-S zeitgleich die vom IE-M übermittelten Steuerdaten anwenden und eigene Messwerte erheben.
- Eine weitere RT-Datenübertragung propagiert die Messwerte der IE-S an alle anderen IE-S und insbesondere den IE-M.
- Abschließend bestimmt der IE-M anhand der Messwerte neue Steuerdaten für den nächsten Zyklus.
- Eine Connection kann mehrere Empfänger besitzen und sowohl von IE-M als auch IE-S ausgehen sowie auch an eine beliebige Mischung von IE-M und IE-S gereichtet sein.

Bereits in [21] wurde ein Kanalzugriffsschema für echtzeitkritische, hybride Netzwerke unter vergleichbaren Voraussetzungen untersucht. Das Ergebnis entspricht in weiten Teilen dem hier entworfenen Schema.

Connections zwischen IE-M und IE-S

Damit die RT-Daten zwischen IE-M und IE-S innerhalb eines Zyklus im hybriden Netzwerk übertragen werden können, obwohl sie in zwei physischen IEs und dem Backbone transportiert werden, ist nachfolgender Kommunikationsablauf notwendig.

1. Der IE-M sendet die DL-Daten im PSN.
2. Das PGW empfängt die DL-Daten und überprüft ihre CRC-Prüfsumme. Anschließend filtert es die Daten, die nicht zu den SGWs weitergeleitet werden müssen, aus. Datum *i* steht daraufhin ab der Releasezeit $T_{DL,rel,i}$ zur Verfügung.
3. Dann ordnet das PGW die gefilterten Daten neu an und bündelt sie zu Blöcken, die es mit einem FEC-Schutz versieht.
4. Schließlich versendet das PGW die Blöcke über den Backbone-Downlink.
5. Alle SGWs empfangen die Blöcke, wenden den FEC an und filtern die Daten aus, die nicht an ihr eigenes SSN gerichtet sind.
6. Jedes SGW verteilt die verbliebenen Daten zu den dazu vorgesehenen Sendezeitpunkten in seinem SSN. Datum *i* muss dafür spätestens zur Deadline $T_{DL,due,i}$ zur Verfügung stehen. Wenn ein Datum an mehrere SSNs gerichtet ist, gilt die früheste Deadline als $T_{DL,due,i}$.
7. Alle IE-S wenden zum GSP die soeben empfangenen Steuerdaten an und erheben neue Messdaten.
8. Im Anschluss initiiert jedes SGW die Übertragung der UL-Daten.
9. Nachdem die UL-Daten das SGW erreicht haben und anhand ihres CRC validiert wurden, sind sie zu den Zeitpunkten $T_{UL,rel,i}$ verfügbar.
10. Die SGWs ordnen die UL-Daten neu an und bündeln sie zu Blöcken, die sie per FEC schützen. Die Daten werden nicht gefiltert, da wie in Abschnitt 3.2 erläutert, alle Connections im PSN vorhanden sein sollen.
11. Dann übertragen die SGWs die Blöcke im Backbone-Uplink zum PGW.
12. Dieses empfängt die Daten und wendet den FEC an.
13. Das PGW leitet die UL-Daten auf Anforderung des IE-M im PSN weiter. Dafür müssen sie spätestens zu den Zeitpunkten $T_{UL,due,i}$ bereitstehen.

14. Zuletzt erreichen die UL-Daten den IE-M, der die Messdaten auswertet und neue Steuerdaten für den nächsten Zyklus erzeugt.

Da die Datenrate des PSSS-PHYs gering im Vergleich zur Datenrate des IE ist, sollte eine möglichst große Zeitspanne zur Verfügung stehen, in der Daten über das Backbone übertragen werden können. Dazu müssen die Werte der Releasezeiten $T_{DL,rel,i}$ und $T_{UL,rel,i}$ minimiert werden und die Werte der Deadlines $T_{DL,due,i}$ sowie $T_{UL,due,i}$ maximiert werden. Daraus ergibt sich, dass die DL-Daten im PSN möglich zeitnah nach ihrer Berechnung durch den IE-M und in den SSNs möglichst kurz vor dem GSP übertragen werden sollten. Für die UL-Daten gilt Entsprechendes.

Connections zwischen verschiedenen IE-S

Da Connections zwischen verschiedenen IE-S als UL-Daten behandelt werden, werden diese bisher nur im SN ihres jeweiligen Producers und im PSN verteilt. Die restlichen SSNs werden noch nicht mit diesen Informationen versorgt. Da für die dadurch betroffenen Connections, wie in Abschnitt 3.2 besprochen, aufgeweichte Anforderungen gelten sollen, müssen diese Daten erst im folgen Zyklus die Ziel-SSNs erreichen. Daher kann der Kommunikationsablauf wie folgt ergänzt werden:

1. Das PGW empfängt die betreffenden Daten zusammen mit den anderen UL-Daten von den SGWs. Das geschieht ohnehin, da ausnahmslos alle Connections im PSN vorhanden sein sollen. Der Zeitpunkt $T_{S2S,rel,i}$, zu dem das i-te Datum am PGW zur Verfügung steht, hängt vom Scheduling der Daten im Backbone-Uplink ab.
2. Das PGW ordnet die Daten nun wieder neu und bündelt sie zu FEC-geschützten Blöcken.
3. Es sendet die betreffenden Daten anschließend über den Backbone-Downlink an alle SGWs.
4. Diese empfangen die Blöcke, wenden den FEC an und filtern die Daten aus, die nicht an sie gerichtet sind.
5. Wenn die SGWs nach dem folgenden GSP die Übertragung nächsten UL-Daten initiieren, übermitteln sie sofort die betreffenden Daten mit, sodass sie zu allen IE-S propagiert werden. Das i-te Datum muss dazu spätestens bis zum Zeitpunkt $T_{S2S,due,i}$ verfügbar sein, der sich nach dem GSP aber vor der Initiierung der UL-Daten befindet.

Die Zeitspanne in der die betreffenden Daten übertragen werden müssen überlappt mit der Zeitspanne in der die DL-Daten übertragen werden. Die IE-S-zu-IE-S-Daten haben jedoch einen größeren Freiraum und können vor und nach den DL-Daten übertragen werden.

3.3.2 Übertragung der SVC- und BE-Daten

In den IEs ergibt sich die Lage und Dauer des UC-Kanals, in dem BE-Daten übertragen werden können, aus der Position der jeweiligen DL- und UL-Telegramme. Im PSN beginnt dieses Intervall nach den DL-Telegrammen und erstreckt sich über den GSP hinweg bis zum ersten UL-Telegramm. In den SSNs beginnt es dagegen nach den UL-Telegrammen und erstreckt sich über die Zyklusgrenze hinweg bis zum ersten DL-Telegramm. Die SVC-Daten befinden sich in den DL- und UL-Telegrammen.

Diese beiden Datentypen besitzen aber weder (strikte) zeitliche Anforderungen noch ein deterministisches Auftreten. Zudem haben sie keine höhere Priorität als einer der anderen Datentypen. Es ist daher möglich diese Daten in einer Queue zu puffern und nur bei Gelegenheit in den Ressourcen des Backbones zu übertragen, die nach dem vollständigen Scheduling der anderen Daten übrig bleiben. SVC-Daten bekommen wegen ihrer Systemrelevanz dabei eine höhere Priorität zugewiesen als BE-Daten.

Genau diese Problemstellung wird in [22] untersucht. Die Autoren des Papers haben dazu einen prioritätsbasierten Scheduling-Algorithmus für azyklische Daten in deterministischen, zyklischen Netzwerken entworfen und untersucht. Ihr Algorithmus erzielt dabei deutlich bessere Ergebnisse als einfaches zyklisches Polling-Schema, nutzt die Kanalkapazitäten effizienter aus als ein Round-Robin-Schema und kommt zugleich ohne ein statistisches Modell des Datenaufkommens aus. Weil das Paper im Hinblick auf ein PSSS-Systems verfasst wurde und alle Grundannahmen aus dem Paper auch in dieser Arbeit gelten, könnte der Algorithmus direkt übernommen werden.

3.3.3 Aufteilung des Ressourcenraums

Aus der Anforderungsbeschreibung wird deutlich, dass eine der wichtigsten Eigenschaften des Backbones eine geringe Latenz sein muss. Um diese zu erreichen, eignet sich besonders die sequentielle Anordnung der Ressourcenblöcke, wie in Abbildung 2.5 dargestellt. Weil sie darüber hinaus keiner anderen Anforderung entgegen läuft, wird ausschließlich die sequentielle Anordnung im Backbone genutzt. Diese Einheitlichkeit vereinfacht mitunter die Implementierung deutlich.

Die PHY-Daten, also Präambeln und TPC/TA-Daten, treten in regelmäßigen Abständen auf und werden unmittelbar übertragen. Ihre Platzierung folgt also direkt aus dem PSSS-Zyklus von DL und UL. Als nächstes haben die RT-Daten Priorität. Sie können jeweils zu beliebigen Zeitpunkten zwischen ihren Releasezeiten und Deadlines übertragen werden. Aufgrund des Netzwerkaufbaus ist die Annahme gerechtfertigt, dass die RT-Übertragungen im PSN und in

den SSNs nicht überlappen. Damit befinden sich auch die späteste Releasezeit noch vor der frühesten Deadline. Das genaue Scheduling – also wann die RT-Daten in welcher Reihenfolge übertragen werden – wird in Abschnitt 3.4 ausführlich behandelt. Ein Ergebnis sei jedoch bereits vorweg genommen: wie in Abschnitt 3.4.4 gezeigt wird, ist es unter der genannten Annahme immer möglich die RT-Daten in einem zusammenhängenden Block im Backbone zu senden. Es müssen keine Pausen in der RT-Übertragung eingefügt werden, abgesehen von Unterbrechungen durch PHY-Daten, die von außen erzwungen werden. Dazu ist es allerdings unumgänglich, dass die RT-Übertragung im Allgemeinen weder sofort zur ersten Releasezeit beginnt, noch punktgenau zu einer Deadline endet. Der tatsächliche Zeitraum ergibt sich erst durch Anwendung des in Abschnitt 3.4 beschriebenen Algorithmus. Er liegt in einem Intervall zwischen der ersten Releasezeit und letzten Deadline. Die verbleibenden Datentypen können alle übrigen Ressourcen nutzen. Auf ihr genaues Scheduling wird in dieser Arbeit nicht weiter eingegangen.

3.4 Scheduling der RT-Daten

Im Folgenden wird der Algorithmus hergeleitet, der zum Scheduling der RT-Daten genutzt werden kann. Seine Aufgaben bestehen darin

1. die Reihenfolge, in der die Daten gesendet werden, festzulegen,
2. den Zeitpunkt, ab dem die Daten gesendet werden, festzulegen,
3. die Daten in Blöcke einzuteilen, die per FEC-Blockcode geschützt werden und
4. zusätzlich zu den RT-Daten ebenfalls die Übertragung der FEC-Daten und einer CRC-Prüfsumme einzuplanen.

Dabei muss der Algorithmus die im Backbone zur Verfügung stehenden Ressourcen beachten. Er benötigt somit einerseits Kenntnis über die Größe, Releasezeit und Deadline jedes RT-Datums und andererseits über die den Zustand des Kanals zu jedem Zeitpunkt.

Gesucht ist dabei ein vollständiger Algorithmus, der einen zulässigen, minimalen Schedule berechnet. Die genannten Kriterien bedeuten im Detail:

- Der Algorithmus ist genau dann vollständig, wenn er garantiert eine Lösung findet, wenn das Problem lösbar ist.
- Der Schedule ist genau dann zulässig, wenn alle folgenden Bedingungen erfüllt sind.
  a) Jedes Datum wird erst nach seiner Releasezeit übertragen,

b) jedes Datum trifft vor seiner Deadline ein,

c) alle FEC- und CRC-Informationen, die zu einem Datum gehören, treffen ebenfalls vor dessen Deadline ein und

d) alle Daten sind per FEC und CRC abgesichert.

- Der Schedule ist genau dann minimal, wenn kein anderer zulässiger Schedule existiert, der weniger Ressourcen verbraucht. Da die Menge der Nutzdaten festgelegt ist, entspricht das einem Schedule mit minimalem Overhead.

Obwohl die Forderung nach einem minimalen Schedule intuitiv erst einmal plausibel wirkt, ist sie diskussionswürdig. Denn Overhead entsteht auf zwei Arten: durch die FEC- und CRC-Daten. Werden diese reduziert, kann das eine Verschlechterung der Performance und somit eine Verletzung der primären Anforderungen an das System bedeuten. Diese Implikationen werden in Abschnitt 3.4.5 genauer thematisiert.

Bevor der Algorithmus diskutiert wird, werden noch die Rahmenbedingungen beleuchtet, denen das Scheduling unterliegt.

3.4.1 Präemptives Scheduling

Eine willkommene Eigenschaft der Problemstellung ist, dass die Übertragung eines Datums jederzeit unterbrochen und zu einem späteren Zeitpunkt fortgesetzt werden kann, ohne dass dadurch signifikante zusätzliche Kosten entstehen. Es entsteht pro Unterbrechung zwar ein geringer Mehraufwand beim Sender und Empfänger, dieser ist aber bei den gegebenen Datenmengen und wenigen Unterbrechungen vernachlässigbar.

Die temporäre Unterbrechung von Übertragungen, um beispielsweise in der Zwischenzeit eine andere Übertragung mit höherer Priorität durchzuführen, nennt sich Präemption und erlaubt in vielen Fällen ein deutlich effizienteres Scheduling und einfachere Scheduling-Algorithmen. Ohne Präemption würde die Verwendung von Blockcodes zum Beispiel ein komplexes Partitionierungsproblem ergeben.

3.4.2 Abbildung auf eine lineare Zeitbasis

Betrachtet man die RT-Datenübertragung als einen Bitstrom, fällt auf, dass die Zeit aus Sicht der Daten nicht linear vergeht. Während zwischen der Übertragung von Bits im gleichen PSSS-Symbol keinerlei Zeit vergeht, liegt zwischen Bits in aufeinanderfolgenden Symbolen ein Zeitsprung. Dieser ist durch Übertragungspausen – beispielsweise aufgrund von Präambeln –

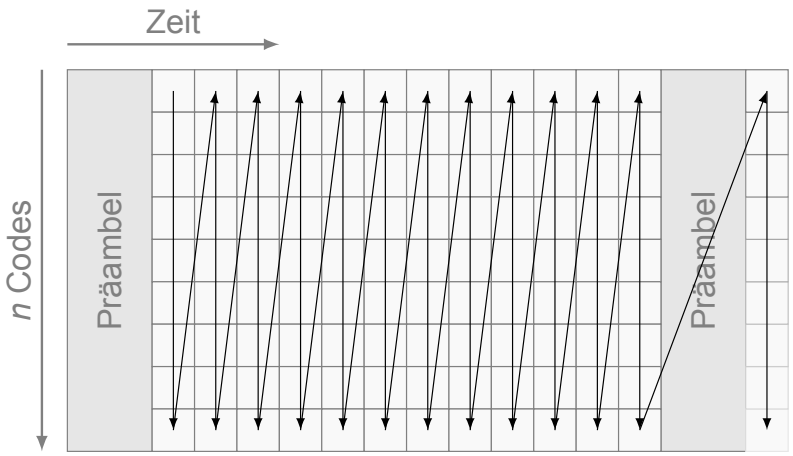


Abbildung 3.2: Die Serialisierung des Ressourcenraums.

nicht einmal von konstanter Dauer. Das ist für Scheduling-Algorithmen eine schwierige Ausgangslage.

Um die Beschreibung und Berechnung der Schedules zu vereinfachen, wird daher eine neue Scheduling-Zeitbasis eingeführt. Durch eine nichtlineare Abbildung zur Realzeit, vergeht aus Datensicht in der künstlichen Zeitbasis die Zeit linear. Dazu muss die Abbildung folgende Punkte leisten:

- Die Serialisierung des Ressourcenraums,
- das Ausblenden von Übertragungsunterbrechungen und
- optional die Normalisierung der Datenrate auf eine simple Größe.

Wie schon in Abschnitt 3.3 besprochen, soll ausschließlich die sequentielle Anordnung von Ressourcenblöcken verwendet werden. Dadurch ist die Serialisierung des Ressourcenraum trivial, indem wie in Abbildung 3.2 serialisiert wird. Zeitabschnitte, die nicht zur Übertragung von RT-Daten genutzt werden können, werden bei der Serialisierung schlicht ignoriert. Das ist insbesondere möglich, weil diese Lücken durch Präemption der Übertragung ohne zusätzlichen Aufwand übersprungen werden können. Es ist nur zu beachten, dass die Releasezeiten und Deadlines immer auf Symbolgrenzen abgebildet werden, damit keine Auswirkungen auf das Scheduling entstehen.

Zuletzt kann die neue Zeitbasis noch so normalisiert werden, dass ein simpler Zusammenhang zwischen Datenvolumen und Übertragungsdauer eines Datums entsteht. Da die Zeitbasis keiner realen Zeit entspricht, ist die Wahl der Zeiteinheit sehr offen. Der simpelste Zusammenhang entsteht, indem als Zeiteinheit die Größe „1 bit” gewählt wird. Viele Berechnungen werden dadurch trivial. Zum Beispiel berechnet sich das Datenvolumen, das zwischen zwei Zeitpunkten a und b übertragen werden kann, als $|a - b|$ ohne zusätzliche Einheitenkonvertierung. Die

Übertragungsdauer eines Datums ist beispielsweise wiederum identisch mit seiner Größe in Bit.

Jedes Datum wird im folgenden durch seine Releasezeit r_i, seine Deadline d_i und seine Größe p_i charakterisiert. Der Name p_i kann als Abkürzung für *„processing time"* oder *„processing size"* interpretiert werden und orientiert sich an der üblichen Notation bei Scheduling-Problemen. Diese Größen sind – ebenso wie alle anderen in den folgenden Abschnitten verwendeten Größen – auf die gerade eingeführte Zeitbasis bezogen. Die praktische Umrechnung von und zur Realzeit wird nicht weiter behandelt.

3.4.3 Festlegung auf systematische Blockcodes

Diese Arbeit soll sich vor allem die Nutzung von linearen Blockcodes für die FEC konzentrieren, da diese auch in den bisher veröffentlichten Arbeiten zu PSSS-Systemen eingesetzt wurden. Jeder Block besitze dabei ein festes Nutzdaten-Fassungsvermögen p_{blk} und eine feste Größe der Redundanzdaten p_{fec}.

Da sich – wie in [23, Kap. 3.2] gezeigt wird – jeder nicht-systematische lineare Code in einen äquivalenten systematischen Code mit den gleichen Eigenschaften umwandeln lässt, stellt eine Beschränkung auf systematische Codes keine zusätzliche Einschränkung dar. Zusätzlich sollen hier nur solche Codes betrachtet werden, bei denen die Nutzdaten am Anfang des Codeworts stehen. Das eröffnet die Möglichkeit die ersten Daten eines Blocks bereits zu übertragen, wenn die restlichen Daten zu diesem Zeitpunkt noch gar nicht bekannt sind. So kann jede verfügbare Übertragung sofort stattfinden und muss nicht auf die Releasezeiten der folgenden Daten warten.

Weil Blockcodes nur auf Blöcke mit genau p_{blk} Bits angewendet werden können, aber die Daten üblicherweise eine andere Größe besitzen, sind Aggregation, Splitting und Padding der Nutzdaten notwendig. Bei der Aggregation werden mehrere kleine Daten in einem Block gebündelt und gemeinsam codiert. Beim Splitting wird dagegen ein übergroßes Datum auf mehrere Blöcke aufgeteilt und separat codiert. Beides zusammen erlauben eine sehr flexible Aufteilung der Daten auf FEC-Blöcke. Beim Padding wird schließlich eine feste und bekannte Zeichenfolge an die Nutzdaten angehängt um die nötige Blockgröße zu erreichen. Da das Padding allen Empfängern bekannt ist, muss es selbst nicht mit übertragen werden. Ein Block mit Padding wird daher als verkürzter FEC-Block bezeichnet.

In Abschnitt 3.4.6 werden nochmals einige Verbesserungen oder Alternativen zur Verwendung von Blockcodes diskutiert. Sie werden in dieser Arbeit aber nicht zum Einsatz kommen.

3.4.4 Scheduling des Downlinks

Nachdem alle relevanten Umstände erklärt wurden, wird nun der optimale Scheduling-Algorithmus für den RT-Downlink beschrieben. Charakteristisch für den Downlink ist, dass das PGW als einziger Sender fungiert und daher nicht um den Kanalzugriff konkurrieren muss. Außerdem muss der Scheduler im Downlink keine Unterscheidung der Daten anhand ihrer Zielnetzwerke vornehmen, weil die SGWs alle Daten empfangen und dann auf Anwendungsebene filtern können.

Trotzdem ist der Algorithmus zu komplex um ihn ohne Weiteres verständlich zu beschreiben. Stattdessen werden zuerst zwei Spezialfälle betrachtet, für die gut verständliche Algorithmen hergeleitet werden können. Zuerst wird ein Algorithmus beschrieben, der nur bei einer einzigen Releasezeit funktioniert, dann ein Algorithmus, der nur bei einer einzigen Deadline funktioniert. Zum Schluss wird auf dieser Grundlage der vollständige Algorithmus eingeführt.

Einheitliche Releasezeit

Als erstes wird der Fall betrachtet, dass alle Datenpakete eine gemeinsame Releasezeit $r_i = r$ besitzen. Erst wird die Herleitung des Algorithmus anhand von Beispielen beschrieben und dann der vollständige Algorithmus zusammengefasst.

Die Grundidee des Algorithmus besteht darin die Menge an Nutzdaten, die bis zu jeder Deadline übertragen wird, zu maximieren, wenn nichts anderes durch die Randbedingungen des Problems erzwungen wird. Dadurch wird garantiert, dass ein Datum nur dann seine Deadline verpasst, wenn der maximale Datendurchsatz des Backbones überschritten wird oder das Problem keine Lösung zulässt. In beiden Fall ist es nicht möglich die RT-Daten entsprechend der Problemstellung zu senden.

Sendepausen Als erstes wird bewiesen, dass es ausreicht wenn der Algorithmus nur solche Schedules erzeugt, die keine Sendepausen enthalten. Dazu wird das in Abbildung 3.3 gezeigte Beispiel verwendet.

Es sei ein zufälliger und beliebiger Schedule gegeben, der Sendepausen enthalte. Das ist durch die erste Zeile der Abbildung visualisiert. Tatsächlich wird hierbei keine Annahme darüber getroffen wie die Daten geschedult sind oder welche Eigenschaften sie besitzen. Dadurch ist dies ein generelles und allgemeingültiges Modell.

Nun werden aus dem gegebenen Schedule alle Sendepausen entfernt, indem die nachfolgenden Daten entsprechend früher gesendet werden. In der Abbildung ist das in der zweiten Zeile

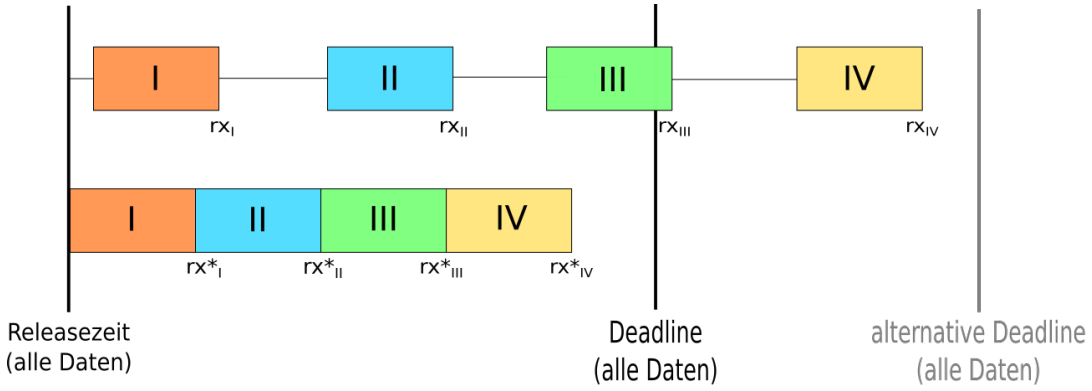


Abbildung 3.3: Unterschied beim Scheduling mit und ohne Pausen. Abgebildet sind zwei alternative Schedules mit den gleichen vier Datenblöcken, aber unterschiedlichen Sendezeitpunkten. rx_i und rx_i^* geben jeweils den Empfangszeitpunkt des i. Datums an.

dargestellt. Das Verschieben der Daten ist ohne Einschränkung möglich, da sich alle Daten weiterhin nach ihrer Releasezeit befinden und somit „bekannt" sind. Wie klar zu erkennen ist, führt das Entfernen der Pausen dazu, dass im neuen Schedule alle Daten zu einem früheren Zeitpunkt beim Empfänger eintreffen. Allgemein formuliert gilt:

$$rx_i^* \leq rx_i \tag{3.7}$$

wobei rx_i die Empfangszeitpunkte im ursprünglichen Schedule und rx_i^* die Empfangszeitpunkte nach Entfernen der Pausen angibt.

Es ist daher möglich, dass ein Schedule, der die Deadline nicht einhalten konnte, durch Entfernen der Pausen zu einem zulässigen Schedule wird. Das ist klar am Beispiel in der Abbildung ersichtlich. Umgekehrt kann ein Schedule, der bereits mit Pausen die Deadline erfüllt, durch Entfernen der Pausen niemals eine Deadline verletzten, da vollkommen allgemein für alle Empfangszeitpunkte $rx_i^* \leq rx_i$ gilt. Das lässt sich anhand der Abbildung erkennen, wenn die alternative Deadline betrachtet wird. Zusammenfassend lässt sich somit feststellen, dass zu jedem Schedule mit Sendepausen mindestens ein gleich guter oder besserer Schedule ohne Sendepausen existiert. Folgerichtig wird der Algorithmus daher so gestaltet, dass er nur Schedules ohne Sendepausen erzeugt.

Sendereihenfolge Als nächstes wird die Sendereihenfolge der Daten betrachtet. Es wird bewiesen, dass der Algorithmus darauf basieren darf, dass alle Daten in der Reihenfolge ihrer Deadlines gesendet werden. Dazu wird das Beispiel aus Abbildung 3.4 verwendet.

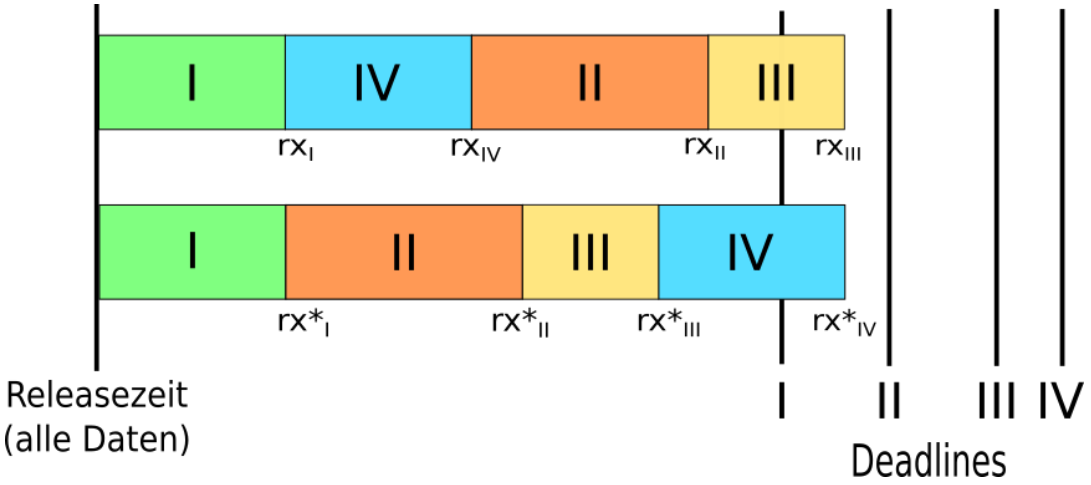


Abbildung 3.4: Zwei Schedules mit verschiedenen Sendereihenfolgen. Die Sortierung im ersten Schedule ist zufällig, während die Daten im zweiten Schedule nach ihren Deadlines sortiert sind. Beide Schedules sind zulässig.

Es sei ein beliebiger zulässiger Schedule gegeben, dessen Sendereihenfolge nicht der Reihenfolge der Deadlines entspricht. Er wird in der Abbildung durch den Schedule in der ersten Zeile repräsentiert. In einem solchen gibt es mindestens ein Datum, das vor einem Datum mit früherer Deadline geschedult ist. Im Beispiel ist dies das Datum *IV*.

Indem dieses Datum so weit nach hinten verschoben wird, dass danach nur noch Daten mit späterer Deadline folgen, wird nun ein neuer Schedule erzeugt. Alle Daten dazwischen werden entsprechend weit nach vorne verschoben. Das Ergebnis ist in der zweiten Zeile der Abbildung zu sehen; Datum *IV* wurde ans Ende des Schedules verschoben. Das letzte Datum, das dabei nach vorne geschoben wurde, nimmt dabei eine Sonderrolle ein; im Beispiel also Datum *III*. Es erfüllt in jedem Fall die Bedingung, dass seine Deadline vor der Deadline des nach hinten geschobenen Datums liegt. Ansonsten hätte das Verschieben bereits vor diesem Datum geendet.

Werden die Empfangszeitpunkte und Deadlines der vertauschten Blöcke vor der Vertauschung betrachtet, gilt also ganz allgemein

$$rx_{IV} < rx_{III} \qquad rx_i \leq d_i \quad \forall i \qquad d_i \leq d_{i+1} \quad \forall i \quad . \tag{3.8}$$

Dass die erste Ungleichung gilt, wurde eben erläutert und liegt daran, dass *IV* und *III* entsprechend ausgewählt wurden. Die zweite Ungleichung folgt daraus, dass es ein zulässiger Schedule ist. Und die dritte Ungleichung gilt per Definition. Die Verwendung der Indizes aus dem Beispiel stellt dabei keine Beschränkung der Allgemeinheit dar. Beim Verschieben ändern sich

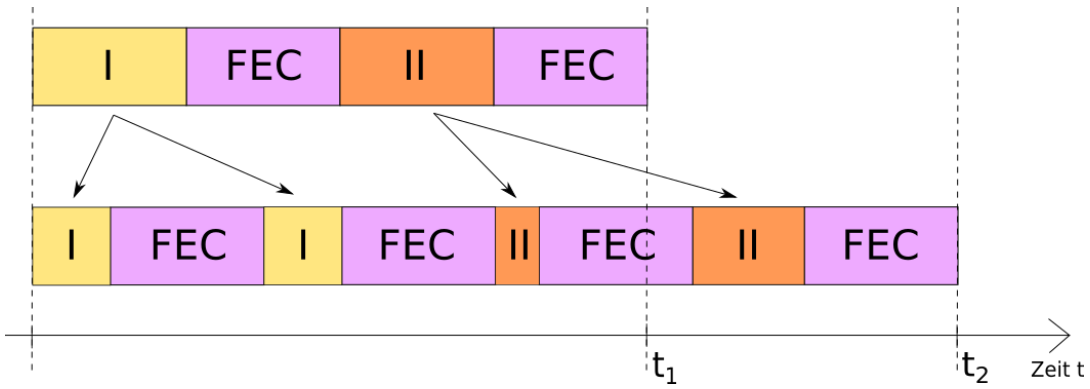


Abbildung 3.5: Unterschied beim Scheduling mit vollen oder verkürzten FEC-Blöcken.

die Empfangszeitpunkt so, dass danach

$$rx_i^* \leq rx_i \quad \forall i \neq IV \qquad\qquad rx_{IV}^* = rx_{III} \tag{3.9}$$

gilt. Kombiniert man alle Aussagen miteinander, ergibt sich

$$rx_i^* \leq rx_i \leq d_i \quad \forall i \neq IV \qquad\qquad rx_{IV}^* = rx_{III} \leq d_{III} \leq d_{IV} \tag{3.10}$$

und somit

$$rx_i^* \leq d_i \quad \forall i \quad . \tag{3.11}$$

Damit ist bewiesen, dass ein zuvor zulässiger Schedule nach dem Verschieben weiterhin zulässig bleibt. Da durch wiederholtes Verschieben in dieser Weise irgendwann ein nach Deadline sortierter Schedule entsteht, lässt sich jeder unsortierte zulässige Schedule in einen sortierten zulässigen Schedule umwandeln. Anders gesagt existiert immer ein sortierter zulässiger Schedule, wenn das Problem lösbar ist. Der Beweis ist [24, Kap. 3.2] entlehnt.

Da bereits nur pausenfreie Schedules betrachtet werden, hängt die Gesamtdauer des Schedules ausschließlich von der enthaltenen Datenmenge ab. Weil diese sich durch Verschiebung nicht ändert, behält der Schedule zudem seine Gesamtdauer bei.

Der Algorithmus wird daher so gestaltet, dass er immer Schedules erzeugt, die nach Deadline sortiert sind. Das reduziert die Komplexität des Algorithmus deutlich, weil sich Reihenfolge der Daten leicht durch jeden beliebigen Sortieralgorithmus bestimmen lässt.

Aufteilung auf FEC-Blöcke Die nächste Eigenschaft, die der Algorithmus erfüllen soll, ist die Vermeidung von verkürzten FEC-Blöcken. Abbildung 3.5 enthält ein Beispiel dazu. Das Beispiel ist eine Vereinfachung, da die Releasezeit, die Deadlines und der CRC-Code nicht

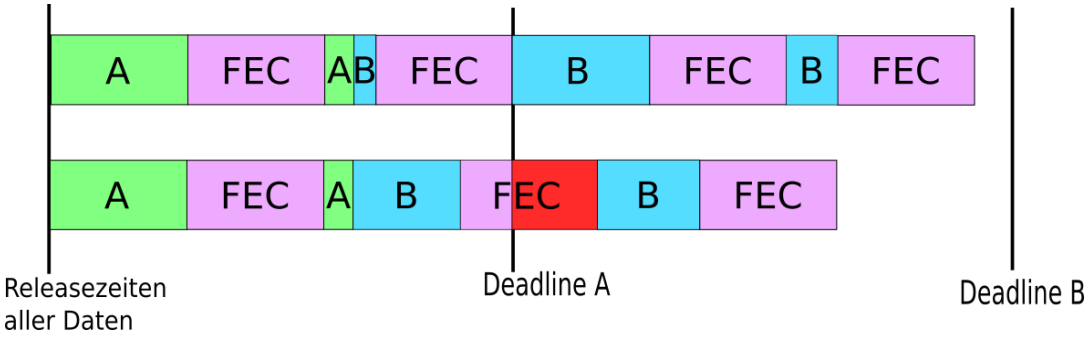


Abbildung 3.6: Ein Beispiel für die erzwungene Verkürzung eines FEC-Blocks. Im unteren Schedule wurde der zweite FEC-Block nicht verkürzt, sodass Datum *A* seine Deadline verpasst.

dargestellt werden. Es werden außerdem im Folgenden keine vollständigen Beweise mehr geführt, weil diese aufgrund ihrer Komplexität der Verständlichkeit abträglich sind. Stattdessen wird versucht die Designentscheidungen einfach zu begründen und leicht nachvollziehbar zu machen.

Die erste Zeile der Abbildung zeigt die beiden Daten *I* und *II*, die in maximal gefüllten FEC-Blöcken untergebracht sind. In der zweiten Zeile werden beide Daten jeweils auf zwei Blöcke aufgeteilt. Wie zu erkennen ist, benötigt der zweite Schedule mehr Zeit, weil ein FEC-Block immer eine gleichbleibende Menge an Redundanzdaten besitzt, egal wie stark er verkürzt wurde. Durch die Aufteilung muss mehr Overhead übertragen werden. Das widerspricht der Grundidee, dass vor jeder Deadline die maximal mögliche Menge an Nutzdaten übertragen werden soll. Im Beispiel ist passend dazu auch zu erkennen, dass je nachdem wo die Deadline liegt, nur der erste Schedule diese erfüllen könnte und somit zulässig wäre. Der Algorithmus versucht daher alle Datenblöcke vollständig zu füllen.

Die einzige Ausnahme ist, wenn die Kürzung eines Blocks für die Einhaltung einer Deadline absolut notwendig ist. Ein Beispiel dafür ist in Abbildung 3.6 zu finden. Es ist unverkennbar, dass der erste Schedule länger braucht als der zweite Schedule, da der erste einen verkürzten Block enthält. Der zweite Schedule ist im Gegensatz zum ersten aber nicht zulässig. Das Problem liegt im rot markierten Teil im zweiten Schedule. Der gesamte FEC müsste noch vor der Deadline *A* übertragen worden sein, damit der Schedule zulässig ist. Es muss also zwangsläufig ein verkürzter Block wie im ersten Schedule genutzt werden. Auch der letzte Block muss natürlich verkürzt werden, da keine weiteren Daten vorhanden sind, die ihm noch hinzugefügt werden könnten.

Deshalb wird der Algorithmus so gestaltet, dass er zwar immer versucht volle Blöcke zu senden, aber wenn er durch eine Deadline dazu gezwungen wird einen verkürzten Block einzufügen, dann tut er das stattdessen. Dieser Fall tritt genau dann ein, wenn ein Block, der eine Deadline schneidet noch Daten enthält, die dieser Deadline zugeordnet werden. Würde der

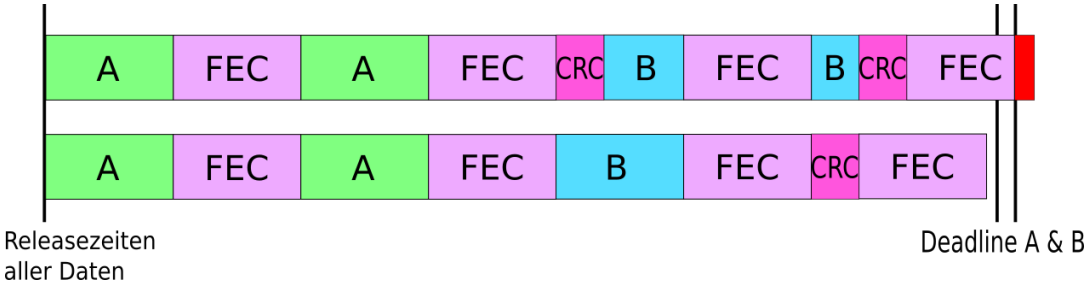


Abbildung 3.7: Zwei Beispiele zum Einfügen von CRCs.

zweite Block im Beispiel keine Daten von *A* enthalten, würde er entsprechend also auch nicht verkürzt werden. Dass er dann wie im zweiten Schedule die Deadline kreuzt ist unproblematisch, da alle Daten für *A* schon vorhanden sind.

Ist es trotz Verkürzung nicht möglich alle Daten vor der entsprechenden Deadline zu senden, dann existiert kein zulässiger Schedule für das Problem und der Algorithmus muss mit einer Fehlermeldung abbrechen. Der Beweis dazu kann induktiv erfolgen, indem jeweils das Scheduling bis zur k-ten Deadline durchgeführt wird und dann die $(k + 1)$-te Deadline betrachtet wird.

CRC Platzierung Es verbleibt noch die Frage, wann ein CRC gesendet werden soll. In Abbildung 3.7 sind zwei denkbare Beispiele gezeigt. Im ersten wird ein CRC nach jedem Datum eingefügt, während im zweiten nur unmittelbar vor einer Deadline ein CRC gesendet wird. Daneben sind natürlich auch andere Varianten denkbar, aber zur Veranschaulichung reichen diese beiden aus.

Es ist unmittelbar ersichtlich, dass im ersten Beispiel zwar Deadline *A* erfüllt aber Deadline *B* verletzt wird. Das zweite Beispiel nutzt hingegen nur minimalen Overhead und schafft es daher beide Deadlines zu erfüllen. Es ist überflüssig einen zweiten CRC einzuführen, der *A* schützt, weil ein gemeinsamer CRC offensichtlich nicht nur ausreichend, sondern besser ist.

Unterm Strich wird der Algorithmus deshalb so gestaltet, dass vor jeder Deadline genau ein einziger CRC einfügt wird; und zwar so spät wie möglich, um so viele Daten wie möglich damit abzusichern. Für die Deadlines, für die seit dem letzten CRC gar keine weiteren Daten hinzugekommen sind, wird allerdings wiederum kein zusätzlicher CRC hinzugefügt.

Algorithmus Mit diesen Designentscheidungen sind alle Teile des Algorithmus festgelegt. Es entsteht insgesamt der folgende Algorithmus für ein Problem mit einer einheitlichen Releasezeit.

Beginne mit der Initialisierung:

1. Sortiere die Datenpakete aufsteigend nach ihrer Deadline.
2. Beginne zur gemeinsamen Releasezeit r mit der Datenübertragung und füge niemals Sendepausen ein.

Führe die nächsten Schritte iterativ aus, bis alle Daten gesendet sind:

3. Übertrage die Daten – entsprechend ihrer Sortierung – in voll gefüllten FEC-Blöcken, bis der nächste eingefügte Block eine Deadline schneiden würde. Ignoriere dabei jede Deadline, zu der keine Daten mehr fällig sind, weil diese bereits vorher übertragen und per CRC geprüft wurden.
4. Sobald der nächste eingefügte Block eine Deadline schneiden würde, probiere einen CRC am Ende des aktuellen Blocks zu platzieren. Das schlägt genau dann fehl, wenn zuvor noch nicht alle Daten übertragen wurden, die zu dieser Deadline fällig sind.
5. Falls der 4. Schritt fehlgeschlagen ist, sende als nächstes einen verkürzten Block, der genau zur Deadline endet. Platziere den CRC am Ende dieses Blocks. Wenn dennoch vorher nicht alle Daten für diese Deadline übertragen wurden, beende den Algorithmus mit einer Fehlermeldung, dass kein kein zulässiger Schedule existiert.
6. Wenn noch weitere Daten vorhanden sind, fahre ab Schritt 3 weiter fort.

Sobald alle Daten abgearbeitet wurden, ist das Scheduling beendet.

Einheitliche Deadline

Als nächstes wird der Fall betrachtet, dass alle Datenpakete eine gemeinsame Deadline $d_i = d$ besitzen. Auch hierzu wird ein Algorithmus beschrieben, der in der Lage ist einen minimalen, zulässigen Schedule zu berechnen, wenn ein solcher existiert.

Herleitung Im Prinzip unterscheidet sich dieser Fall kaum vom vorherigen Fall, obwohl sie auf den ersten Blick sehr verschieden wirken. Das vorliegende Problem lässt sich aber einfach in umgekehrter zeitlicher Richtung betrachten. Releasezeiten und Deadlines tauschen dabei ihre Rolle und das Problem wird äquivalent zum Problem mit einheitlicher Releasezeit. Der einzige Unterschied besteht darin wie FEC und CRC behandelt werden müssen. Der Schedule wird daher gedanklich von Hinten nach Vorne aufgebaut. Die Daten werden also beginnend an der Deadline immer voreinander gesetzt.

Zuerst wird ein FEC ans Ende gesetzt und direkt davor der CRC platziert. Da nur eine Deadline vorliegt, genügt ein einziger CRC für alle Daten mit der selben Argumentation wie im vorherigen

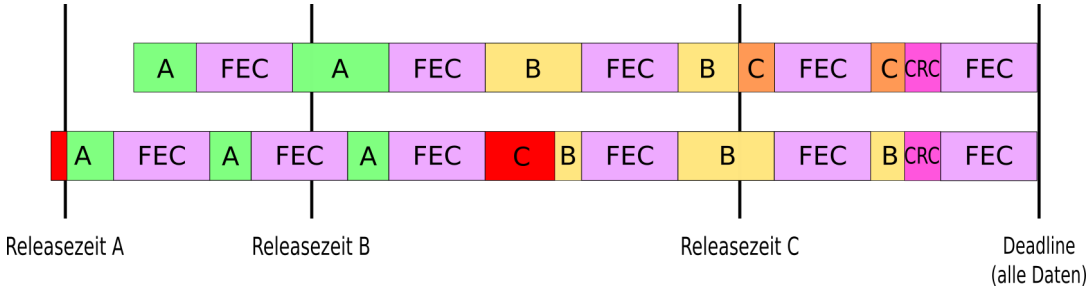


Abbildung 3.8: Vergleich zwischen zwei Schedules für Daten mit einer gemeinsamen Deadline.

Algorithmus. Die Sendereihenfolge der Daten soll der Reihenfolge ihrer Releasezeiten entsprechen, was analog zu zuvor gezeigt werden kann. Im Wechsel mit Daten werden dabei wie gehabt die FECs gesetzt. Alle Blöcke bis auf den vordersten Block werden vollständig gefüllt. Da die Einhaltung einer Releasezeit in keiner Weise einen FEC notwendig macht, treten dabei auch nicht die Probleme mit verkürzten Blöcken auf wie beim vorhergehenden Algorithmus.

In Abbildung 3.8 sind dazu zwei Beispiele dargestellt. Die erste Zeile zeigt einen Schedule wie er vom erklärten Algorithmus erzeugt wird; die Daten befinden sich in der Reihenfolge ihrer Releasezeiten, alle bis auf den vordersten Block sind voll gefüllt und es gibt nur einen CRC direkt vor der Deadline. An Releasezeit *C* ist gut zu erkennen, dass ein Release keinen verkürzten FEC-Block notwendig macht. Der Schedule in der zweite Zeile zeigt dagegen, dass hier die gleichen Probleme wie zuvor entstehen, wenn kürzere Blöcke als notwendig eingesetzt werden oder die Reihenfolge nicht eingehalten wird. Der vorderste Block verletzt die Releasezeit von *A*, weil die beiden anschließenden Blöcke unnötig verkürzt wurden. Zudem verletzt Datum *C* seine Releasezeit, weil es entgegen der Reihenfolge der Releasezeiten vor Datum *B* gesetzt wurde.

Wird der Algorithmus befolgt und es verletzt dennoch ein Datum seine Releasezeit, dann existiert kein zulässiger Schedule und der Algorithmus muss mit einer Fehlermeldung abbrechen.

Algorithmus Zusammengefasst ergibt sich daher dieser Algorithmus:

1. Sortiere die Datenpakete aufsteigend nach ihrer Releasezeit und füge der sortierten Liste zuletzt den CRC-Code als abschließendes Datum hinzu.

2. Beginne von der Deadline aus – zeitlich rückwärts – voll gefüllte FEC-Blöcke in den Schedule einzuplanen. Ordne jedem neuen Block – rückwärts – Daten vom Ende der obigen Liste zu. Beginne also damit den CRC am Ende des zuletzt übertragenen Blocks einzuplanen; fahre dann mit dem letzten Datum fort und plane es direkt vor dem CRC ein; fahre so immer weiter fort.

3. Wenn schließlich alle Daten eingeplant wurden und der zuerst zu übertragende Block erreicht ist, lässt sich dieser im Allgemeinen nicht vollständig füllen. Der erste Block ist daher gegebenenfalls ein verkürzter Block.

Eine unerfreuliche Eigenschaft dieses Algorithmus ist, dass er die Daten in negativer zeitlicher Richtung schedult. Das kann unter Umständen seine Implementierung erschweren. Wird der Schedule in der ersten Zeile von Abbildung 3.8 betrachtet, lässt sich aber erahnen, dass auch leicht ein alternativer Algorithmus gefunden werden kann, der in positiver zeitlicher Richtung arbeitet und denselben Schedule erzeugt. Die einzige Schwierigkeit besteht darin den Startzeitpunkt des Schedules zu bestimmen. Da die Übertragungsdauer sich aber direkt als Funktion der Nutzdatenmenge beschreiben lässt, ist es ein leichtes den nötigen Startzeitpunkt im Voraus zu berechnen. Zur Implementierung in Kapitel 4 wird der Algorithmus deshalb genau so umgesetzt.

Allgemeiner Fall

Im allgemeinen Fall liegt jedoch weder eine einheitliche Releasezeit noch eine einheitliche Deadline vor. Um diese Problemstellung zu lösen, wird das Problem in zwei Teilprobleme aufgespalten, die sich jeweils mit einem der beiden bekannten Algorithmen lösen lassen. Die Aufteilung kann dabei nicht beliebig erfolgen, sondern es muss mithilfe des hier vorgestellten Algorithmus eine optimale Aufteilung berechnet werden. Auch dieser Algorithmus wird in der Lage sein immer einen minimalen, zulässigen Schedule zu berechnen, wenn ein solcher existiert. Um den Algorithmus zu beschreiben, werden vorab die folgenden Begriffe eingeführt:

- Der Zeitpunkt $\mathcal{D}$ bezeichnet die früheste Deadline. Diese Deadline ist von besonderer Bedeutung, da alle Daten, die vor ihr übertragen werden, nach dem Algorithmus für eine einheitlichen Deadline geschedult werden und umgekehrt alle Daten, die nach ihr übertragen werden, mit dem Algorithmus für eine einheitlichen Releasezeit geschedult werden.
- Die Menge $\mathbb{S}_1$ beschreibt das erste Teilproblem und soll alle Daten enthalten, die vor $\mathcal{D}$ gesendet werden können.
- Die Menge $\mathbb{S}_2$ beschreibt das zweite Teilproblem und soll alle Daten enthalten, die nach $\mathcal{D}$ gesendet werden müssen.
- Ein Datum kann in einer oder in beiden Mengen liegen. Damit klar ist zu welchen Anteilen ein Datum in jedem der Teile liegt, wird p_i^* eingeführt. Es gibt die Anzahl der Bits an, die im ersten Teil gesendet werden sollen. Im zweiten Teil müssen entsprechend die restlichen $p_i - p_i^*$ Bits gesendet werden.

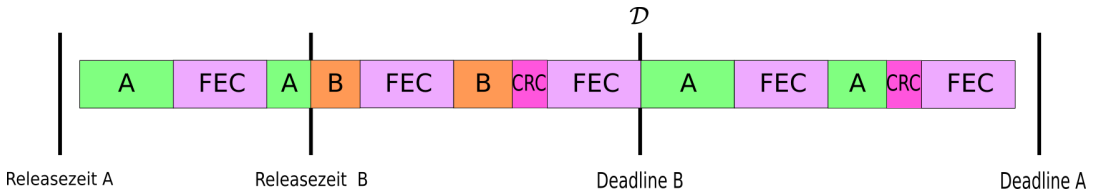


Abbildung 3.9: Ein Beispiel für das Ergebnis des vorgestellten Algorithmus. Da Deadline *B* die erste Deadline ist, liegt dort der Zeitpunkt $\mathcal{D}$. Vor $\mathcal{D}$ befinden sich die Daten *A* und *B*, weshalb $\mathbb{S}_1 = \{A, B\}$ ist. Weil hinter $\mathcal{D}$ nur Datum *A* liegt, ist $\mathbb{S}_2 = \{A\}$. Der Datenumfang von *A* beträgt $p_A = 3$ Blöcke und von *B* $p_B = 1$ Block. Weil *B* komplett vor $\mathcal{D}$ liegt ist ebenfalls $p_B^* = 1$ Block. Für *A* ist hingegen nur $p_A^* = 1{,}5$ Blöcke, weil dies der Anteil von *A* ist, der vor $\mathcal{D}$ liegt.

- Die Kapazität $capacity(t)$ wird definiert als die Menge an Nutzdaten, die zwischen dem Zeitpunkt t und der ersten Deadline $\mathcal{D}$ übertragen werden kann. Sie ist nur für $t < \mathcal{D}$ definiert und verwendet die Annahme, dass immer volle FEC-Blöcke gesendet werden und der letzte Block genau zu $\mathcal{D}$ endet.
- Sei tx_i die tatsächliche Sendezeit eines Datums in einem gegebenen Schedule. Dann beschreibt der Slack s_i des Datums die Kapazität, die zwischen der Releasezeit r_i und der Sendezeit tx_i des Datums liegt. Er lässt sich über $s_i = capacity(r_i) - capacity(tx_i)$ berechnen. Der Slack wird in Abbildung 3.10 veranschaulicht.
- Wenn mehrere Daten die gleiche Releasezeit besitzen, dann ist davon für den Algorithmus nur der Slack interessant, den das am frühesten gesendeten Datum besitzt. Er wird zur Unterscheidung mit $s[r]$ bezeichnet.

Die Größen $\mathcal{D}$, $\mathbb{S}_1$, $\mathbb{S}_2$ und p_i^* werden in Abbildung 3.9 und der zugehörigen Bildbeschreibung veranschaulicht. Der Slack wird in Abbildung 3.10 visualisiert.

Wenn zwei Daten dieselbe Releasezeit und dieselbe Deadline aufweisen, dann werden sie im Folgenden immer als ein Datum betrachtet, dessen Größe der Summe der Einzelgrößen entspricht. Das vereinfacht die Besprechung und beschleunigt den Algorithmus sogar in der Praxis, da dessen Laufzeit maßgeblich von der Anzahl der Daten abhängt.

Algorithmus Diesmal erfolgt keine Herleitung des Algorithmus. Die Idee ist einfach – wie bereits erwähnt – die maximal mögliche Menge an Nutzdaten vor der ersten Deadline zu senden. Wie viele Nutzdaten dort untergebracht werden können ermittelt der Algorithmus über den Slack der verschiedenen Releasezeiten. Nach der folgenden Beschreibung des Algorithmus wird er anhand eines ausführlichen Beispiels nachvollzogen.

Beginne den Algorithmus mit folgender Initialisierung:

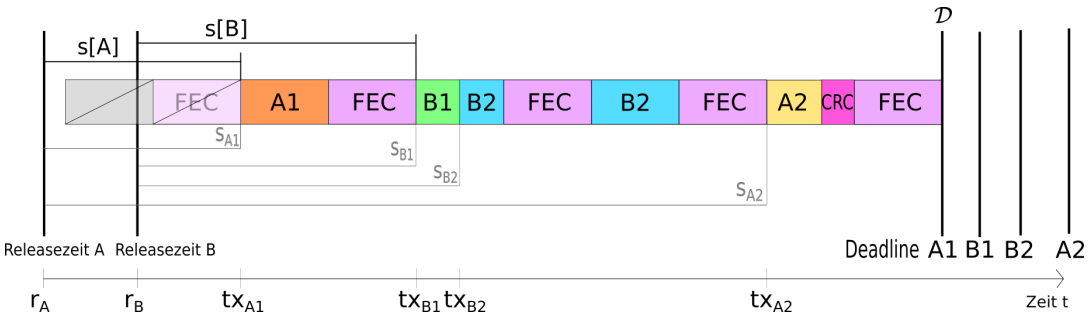


Abbildung 3.10: Eine Veranschaulichung des Slacks. Die durchgestrichenen Blöcke links sind nicht Teil des Schedules sondern nur zur Visualisierung eingefügt. Der Slack s_i eines Datums i ist die Menge der Nutzdaten, die zwischen seiner Releasezeit und seinem Sendezeitpunkt tx_i liegen oder liegen könnten. Die dargestellte Linie bezeichnet hier nicht den Slack selbst, sondern nur den Bereich der dafür betrachtet wird. So ist s_{A1} = 1 Block, s_{B1} = 2 Blöcke, s_{B2} = 2,5 Blöcke und s_{A2} = 4 Blöcke. Die Slacks $s[r]$ der Releasezeiten sind jeweils das Minimum der Slacks ab dieser Releasezeit. Deshalb sind hier $s[A] = s_{A1}$ und $s[B] = s_{B1}$.

1. Erstelle eine Liste Q aller Datenpakete. Sortiere diese primär nach aufsteigender Deadline und bei Gleichständen zusätzlich nach aufsteigender Releasezeit.
2. Setze $\mathbb{S}_1 = \emptyset$ und $\mathbb{S}_2 = \emptyset$.
3. Erstelle einen Vektor, der die Slacks $s[r]$ für alle vorkommenden Releasezeiten $r \in \bigcup_i r_i$ speichert. Initialisiere $s[r]$ jeweils mit $s[r] = \mathit{capacity}(r)$.

Führe dann die folgenden Schritte iterativ durch, bis alle Daten aus Q verarbeitet wurden:

4. Nimm das nächste Datum i aus der Liste Q.
5. Bestimme den minimalen Slack $s^* = \min_{r \leq r_i} s[r]$ über alle Releasezeiten kleiner als r_i.
6. Mache eine Fallunterscheidung bezüglich s^*:
   a) Falls $s^* = 0$ ist, setzte $p_i^* = 0$ und füge das Datum zu $\mathbb{S}_2$ hinzu.
   b) Falls $s^* \geq p_i$ ist, setzte $p_i^* = p_i$ und füge das Datum zu $\mathbb{S}_1$ hinzu.
   c) Ansonsten setzte $p_i^* = s^*$ und füge das Datum zu $\mathbb{S}_1$ und $\mathbb{S}_2$ hinzu.
7. Subtrahiere p_i^* von allen Slacks $s[r]$ mit $r \leq r_i$.

Nachdem alle Daten so verarbeitet wurden, ist die Aufteilung abgeschlossen. Beende das Scheduling wie folgt:

8. Schedule mit dem Algorithmus für eine einheitliche Deadline jeweils die ersten p_i^* Bits der Daten in $\mathbb{S}_1$. verwende als gemeinsame Deadline dabei $\mathcal{D}$.

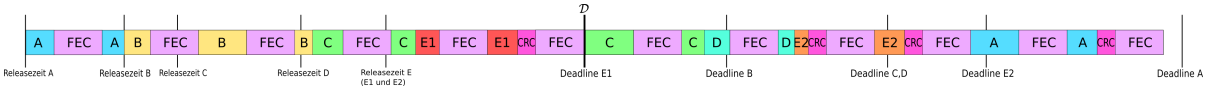


Abbildung 3.11: Ein ausführliches Beispiel für einen fertigen Schedule.

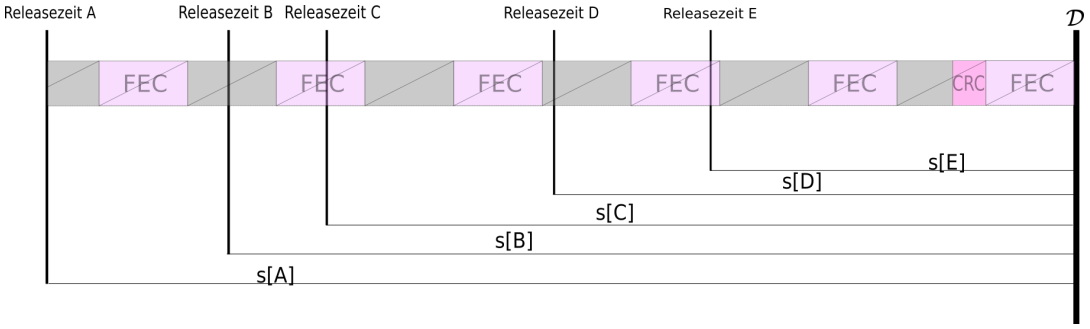


Abbildung 3.12: Der Zustand nach der Initialisierung. Die dargestellten Blöcke sind nicht tatsächlich vorhanden, sondern dienen lediglich der Veranschaulichung. Zu diesem Zeitpunkt sind nur die Releasezeiten r_i, die Deadlines d_i, die Datengrößen p_i und die Slacks $s[r]$ bekannt.

9. Schedule mit dem Algorithmus für eine einheitliche Releasezeit jeweils die letzten $p_i - p_i^*$ Bits der Daten in $\mathbb{S}_2$. verwende als gemeinsame Releasezeit dabei $\mathcal{D}$.

Bei dieser Aufteilung ist garantiert, dass der erste Teil erfolgreich geschedult werden kann. Genau dann wenn kein zulässiger Schedule für diese Problemstellung existiert, wird allerdings das Scheduling des zweiten Teils fehlschlagen, weil mindestens eine Deadline nicht eingehalten werden kann.

Beispiel Aufgrund der Komplexität des Algorithmus wird dieser nochmal anhand eines ausführlichen Beispiels durchlaufen. In Abbildung 3.11 ist der fertige Schedule zu erkennen. Das Beispiel wurde so gewählt, dass möglichst viele Sonderfälle auftreten um viele Facetten des Algorithmus aufzuzeigen. Intern arbeitet der Algorithmus nur über die Werte der Slacks und erstellt *keinen* Schedule. Alle folgenden Abbildungen für dieses Beispiel sind daher rein illustrativ.

Die Initialisierung ist in Abbildung 3.12 zu sehen. Es wird nur der Teil links der ersten Deadline $\mathcal{D}$ betrachtet, da der Algorithmus nur anhand der Slacks in diesem Teil entscheidet, welcher Job zu welchem Anteil in $\mathbb{S}_1$ und $\mathbb{S}_2$ landet. Demnach ist alles in $\mathbb{S}_2$, was nicht mehr in $\mathbb{S}_1$ passt. Zu Beginn sind nur die Releasezeiten r_i, Deadlines d_i und die Größen p_i der Daten bekannt. Aus r_i und $\mathcal{D}$ werden mittels $s[r] = \mathit{capacity}(r)$ die Slacks aller Releasezeiten initialisieren. Weiterhin wird bei der Initialisierung die Liste Q erstellt und nach der Deadline der Daten sortiert, wobei Gleichstände durch die Releasezeit aufgelöst werden. Danach ist $Q = [E1, B, C, D, E2, A]$. Weil aber noch keine Daten verarbeitet wurden, wirkt die erste Abbildung recht leer.

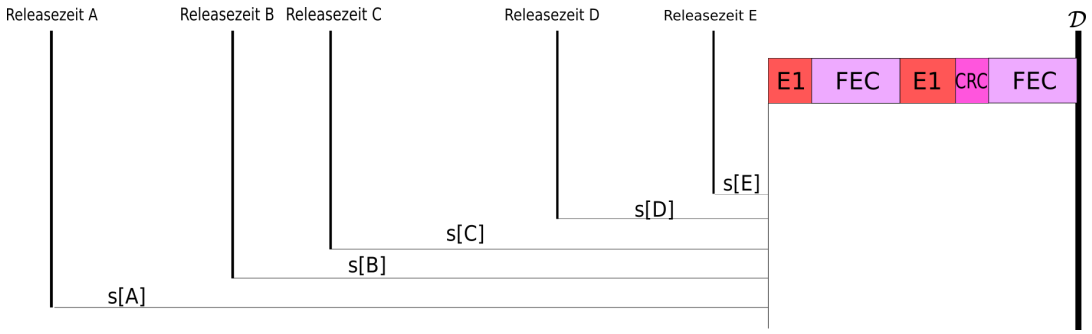


Abbildung 3.13: Der Zustand nach dem ersten Schleifenlauf. $E1$ wurde komplett vor $\mathcal{D}$ eingefügt. Die Slacks aller Daten wurden entsprechend reduziert.

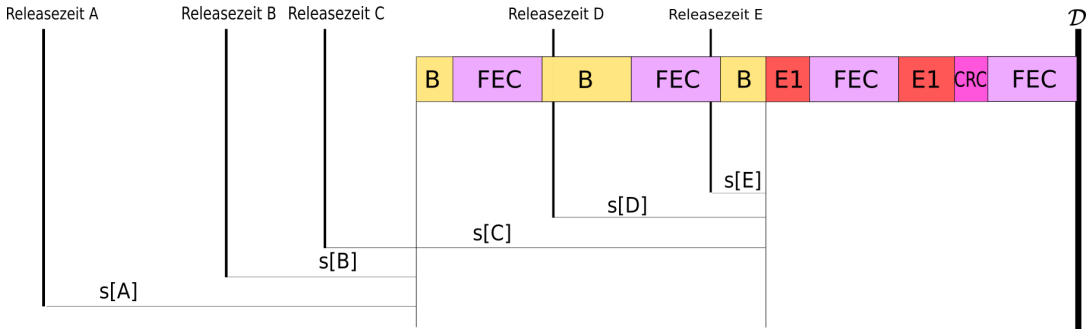


Abbildung 3.14: Der Zustand nach dem zweiten Schleifendurchlauf. B wurde vollständig vor $\mathcal{D}$ eingefügt. Die Slacks $s[A]$ und $s[B]$ wurden entsprechend reduziert.

Als nächstes werden die Schritte 4 – 7 des Algorithmus iteriert, bis Q leer ist. (Schritt 4) Entferne zuerst den ersten Eintrag aus der Liste Q, in diesem Fall $E1$. (Schritt 5) Bestimme dann den minimalen Slack $s^* = \min_{r=A,B,C,D,E} s[r]$. Es ergibt sich $s^* = s[E]$. (Schritt 6) Berechne als nächstes, ob und mit welchem Anteil die Daten von $E1$ in $\mathbb{S}_1$ liegen. Es gilt $s^* > p_{E1}$, also der zweite Fall aus Schritt 6. Daraus folgt $E1 \in \mathbb{S}_1$ und $p^*_{E1} = p_{E1}$. Das bedeutet, dass alle Daten von E vor $\mathcal{D}$ eingefügt werden. (Schritt 7) Aktualisiere zuletzt die Slacks. Da E die letzte Releasezeit ist, wird p^*_{E1} von allen Slacks abgezogen. Dadurch entsteht der Zustand aus Abbildung 3.13.

Die Iteration beginnt von Neuem. (Schritt 4) Nimm B aus der Liste Q. (Schritt 5) Bestimme wieder den minimalen Slack $s^* = \min_{r=A,B} s[i] = s[r]$. Dabei werden nur die Slacks betrachtet, deren Releasezeit kleiner oder gleich B ist, also A und B selbst. Es ergibt sich $s^* = s[B]$. (Schritt 6) Prüfe nun, welcher Fall von Schritt 6 Anwendung findet. Da $s^* > p_B$ ist und somit das gesamte B in den Schedule passt, gilt wieder der zweite Fall. Daraus folgt $B \in \mathbb{S}_1$ und $p^*_B = p_B$. (Schritt 7) Aktualisiere zuletzt die Slacks von A und B. Der aktuelle Zustand ist danach in Abbildung 3.14 zu finden.

Wiederhole nun die Schleife zum dritten Mal. (Schritt 4) Entferne C aus Q. (Schritt 5) Berechne $s^* = \min_{r=A,B,C} s[r] = s[B]$. (Schritt 6) Diesmal gilt der dritte Fall von Schritt 6, da $s^* < p_C$ ist.

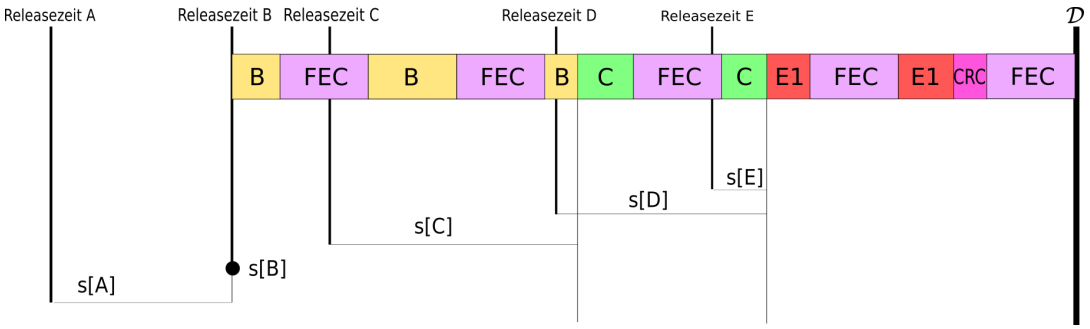


Abbildung 3.15: Der Zustand nach dem dritten, vierten und fünften Schleifendurchlauf. In der dritten Iteration wurde *C* anteilig eingefügt. So viel, dass *B* maximal weit nach vorne verschoben wurde. Der Slack von *B* ist gleich Null und wird daher als Punkt angezeigt. In der vierten und fünften Iteration ändert sich nichts, da die Daten *D* und *E*2 nicht mehr im vor $\mathcal{D}$ eingefügt werden können. Das lässt sich am Bild einerseits leicht direkt erkennen, kann aber auch daraus geschlossen werden, dass der Slack von *B* gleich Null ist und *B* somit nicht weiter nach vorn verschoben werden kann. Das wäre aber nötig, weil *D* und *E*2 der Reihenfolge entsprechend hinter *B* eingefügt werden müssten.

Satze daher $C \in \mathbb{S}_1$ und $C \in \mathbb{S}_2$. Bestimme die Aufteilung auf die beiden Teile über $p_C^* = s^* = s[B]$.

Hier lässt sich der Sinn hinter den Slacks am besten erkennen. Da *C* eine spätere Releasezeit als *B* hat, muss es nach *B* eingefügt werden. Dabei verschiebt es *B* nach vorne. Weil *C* aber mehr Daten umfasst, als hinter *B* eingefügt werden könnte ohne *B* vor seine Releasezeit zu verschieben, ist es nicht möglich *C* komplett einzufügen. Die Menge um die *B* verschoben werden kann entspricht dem Slack $s[B]$. Somit kann der Algorithmus leicht erkennen, dass nur dieser Anteil von *C* eingefügt werden kann. Weil potentiell auch die Daten *A* bereits eingeplant sein könnten, muss der Schedule auch den Slack $s[A]$ in Betracht ziehen. Ebenso wichtig ist es sicherzustellen, dass *C* seine eigene Releasezeit nicht verletzt, was am Slack $s[C]$ erkannt werden kann. Daher wird die eingefügte Menge p_C^* über $\min_{r=A,B,C} s[r]$ bestimmt. (Schritt 7) Reduziere schließlich die Slack $s[A]$, $s[B]$ und $s[C]$ um p_C^*. Danach ergibt sich der Zustand in Abbildung 3.15. Dass die Daten von *B* nicht weiter verschoben werden können, spiegelt sich darin wieder, dass $s[B]$ auf Null gefallen ist. In der Abbildung ist das mit einem Punkt gekennzeichnet.

In den nächsten beiden Iterationen passiert nur wenig. (Schritt 4) Es wird *D* und *E*2 aus der Liste *Q* entfernt. (Schritt 5) Dann werden $s^* = \min_{r=A,B,C,D} s[r]$ beziehungsweise $s^* = \min_{i=A,B,C,D,E} s[r]$ berechnet. Weil beide Male $s[B]$ enthalten ist, gilt beide Male $s^* = 0$. (Schritt 6) Somit greift der erste Fall von Schritt 6 und die Daten werden nur zu $\mathbb{S}_2$ hinzugefügt; die Anteile im ersten Schedule werden entsprechend auf $p_D^* = 0$ und $p_{E2}^* = 0$ gesetzt. Dass die Daten nicht in den betrachteten Schedule vor $\mathcal{D}$ eingefügt werden ist plausibel, da sie das nicht können ohne *B* vor seine Releasezeit zu schieben. (Schritt 7) Gleichzeitig bedeutet das

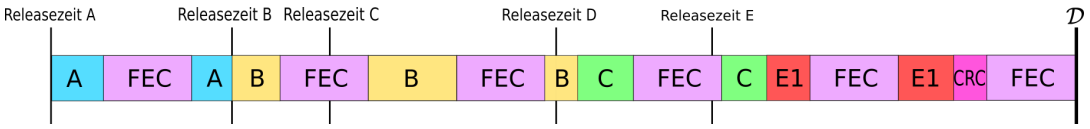


Abbildung 3.16: Der Zustand nach dem letzten Schleifendurchlauf. Es ist das selbe Ergebnis, als wenn $\mathbb{S}_1$ von dem Algorithmus mit einheitlicher Deadline geschedult wird. Er wurden offensichtlich möglichst viele Daten vor $\mathcal{D}$ untergebracht.

auch, dass sich am aktuellen Zustand nichts ändert und weiterhin Abbildung 3.15 sowie die vorherigen Slacks gelten.

In dem letzten Durchlauf wird schließlich in (Schritt 4) A aus der Liste Q entfernt. Damit ist Q leer und die Schleife wird nach diesem Durchgang beendet. (Schritt 5) Da A die erste Releasezeit besitzt, ist $s^* = \min_{r=A} s[r] = s[A]$. (Schritt 6) Es greift wieder der dritte Fall von Schritt 6, da $s^* < p_A$ ist. Also ist $p_A^* = s^*$ und $A \in \mathbb{S}_1$ sowie $A \in \mathbb{S}_2$. (Schritt 7) Bei den Slacks ändert sich nur noch $s[A] = 0$, was aber nicht mehr weiter relevant ist, da der Algorithmus abgeschlossen ist.

In Abbildung 3.16 ist der fertige Schedule vor $\mathcal{D}$ zu sehen. Die Slacks sind nicht mehr dargestellt, da sie nicht mehr gebraucht werden. Der Algorithmus hat aber – wie schon mehrmals erwähnt wurde – nicht den gezeigten Schedule berechnet, sondern lediglich die Mengen $\mathbb{S}_1$ und $\mathbb{S}_2$ sowie die Werte von p_i^* bestimmt. Das Scheduling wird daher erst durch Anwendung der beiden vorherigen Algorithmen abgeschlossen. Das Ergebnis, das sich danach ergibt, ist in Abbildung 3.11 dargestellt. Wie zu erkennen ist, entspricht der Teil vor $\mathcal{D}$ genau dem Schedule in Abbildung 3.16. Der Aufteilungsalgorithmus hat also funktioniert und die Kapazität des ersten Schedules genau ausgelastet ohne diesen dazu überhaupt explizit berechnen zu müssen.

3.4.5 Implikationen des minimierten Overheads

Es sprechen zwei Gründe für die Forderung nach einem minimalen Schedule. Erstens verbrauchen die RT-Daten dadurch am wenigsten Ressourcen, die stattdessen dann für die Übertragung der anderen Datentypen verwendet werden können. Zweitens ist der Fall, in dem nur ein Minimum an Ressourcen für die RT-Übertragung zur Verfügung steht, kein kritischer Sonderfall mehr, wenn der minimale Ressourcenverbrauch bereits eine grundlegende Eigenschaft des Algorithmus ist. Die Minimierung des Overheads ist jedoch auch kritisch zu betrachten.

Indem die Anzahl der CRC-Codes minimiert wird, wird begünstigt, dass der Inhalt sehr vieler FEC-Blöcke mit einem einzelnen CRC überprüft wird. Auch wenn die Korrektur bei nur einem oder wenigen FEC-Blöcken fehlschlägt, müssen dann alle Daten verworfen werden, weil ein

CRC nicht in der Lage ist die Fehler zu lokalisieren. Die Fehlerrate steigt somit deutlich an, was den primären Anforderungen entgegenläuft.

Es gibt zwei Alternativen zu diesem Vorgehen. Erstens könnte eine Begrenzung der Datenmenge pro CRC stattfinden. Dazu müsste aber eine Balancierung zwischen Fehlerrate und Overhead gefunden werden, was über die Möglichkeiten dieser Arbeit hinaus geht. Die Scheduling-Algorithmen sind jedoch so gestaltet, dass sie trivial um diese Nebenbedingung erweitert werden könnten.

Zweitens könnte vollständig auf den Einsatz von CRCs verzichtet werden, wenn stattdessen ein FEC-Schema mit sehr geringer U-PER eingesetzt würde. Auch der Einsatz eines CRCs garantiert ja keine U-PER von Null sondern verringert diese nur stark. Mögliche Kandidaten für geeignete FEC-Schemata werden in [25] und [26] diskutiert. Es handelt sich dabei im Prinzip immer um unvollständige FEC-Decoder, die nicht die volle Korrekturfähigkeit ausnutzen, sondern einen Teil der Redundanz zur Fehlererkennung verwenden.

Im Fall von Sercos III kommt ein weiter Aspekt hinzu. Durch die Verwendung von Summentelegrammen im IE können Daten nur mit der Granularität eines solchen Telegramms akzeptiert oder verworfen werden. Bei vier Summentelegrammen kann dadurch bestenfalls eine Verbesserung der Fehlerrate um den Faktor vier erreicht werden, was den zusätzlichen Aufwand und Verlust an Flexibilität, der dazu nötig ist, nicht rechtfertigen kann.

3.4.6 Verbesserungen und Alternativen zu Blockcodes

Der Einsatz von Blockcodes mit fester Länge ist eine aus Kompatibilitätsgründen getroffene Entscheidung, da bisherige Arbeiten zu ParSec ebenfalls auf dieser Annahme fußten. Diese Entscheidung bringt zwei Probleme mit sich. Erstens muss ein relativ komplizierter Scheduling-Algorithmus angewendet werden, der die Daten sinnvoll in Blöcke unterteilt. Zweitens wird durch Padding einiger Blöcke nur eine suboptimale Kanalnutzung erreicht, wie im nächsten Abschnitt erörtert wird. Daher werden im Folgenden drei ergänzende oder alternative Ansätze zum Einsatz von Blockcodes fester Länge vorgestellt.

Padding mit BE-Daten

Durch das Padding von Blöcken mit Nullen werden indirekt Ressourcen verschwendet, da die durchschnittliche Coderate sinkt. Zwar geht das mit einer verbesserten relativen Korrekturfähigkeit in diesen Blöcken einher, da die Anzahl der Bits sinkt, aber die Anzahl korrigierbarer

Fehler gleich bleibt. Doch ist die Korrekturfähigkeit des Codes idealerweise genau auf die Kanaleigenschaften abgestimmt, sodass eine unvorhersehbare und beliebige Verbesserung der Korrekturfähigkeit beim Padding keine wünschenswerte Eigenschaft ist.

Wäre es möglich die Blöcke mit anderen Nutzdaten aufzufüllen, anstatt Nullen ohne Nutzinformationen einzusetzen, dann würde dieser Nachteil verschwinden. Die Coderate würde dann ebenso wie die relative Korrekturfähigkeit konstant bleiben. Da die verkürzten Blöcke aufgrund von Ressourcenknappheit beim RT-Scheduling entstehen, kommt es allerdings nicht in Frage, dass diese anderen Nutzdaten zeitnah zum verkürzten Block gesendet werden. Stattdessen müssen sie dem Empfänger bereits im Voraus bekannt sein. Da für BE-Daten keine Echtzeitanforderungen bestehen, wäre es denkbar, dass immer eine gewisse Menge BE-Daten ohne FEC-Schutz gesendet und beim Sender und Empfänger in einer Queue gespeichert wird. Muss dann irgendwann in der RT-Phase Padding eingefügt werden, wird dieses aus den ältesten Daten in der Queue gebildet.

Auf Kosten einer höheren Latenz der BE-Daten wird damit die Kanalauslastung gesteigert. Dies kann in Szenarien wünschenswert sein, in denen das Netzwerk nahe an seiner Kapazitätsgrenze betrieben wird.

Verwendung von Faltungscodes

Die Probleme mit Blockcodes lassen sich komplett umgehen, wenn stattdessen Faltungscodes zum Einsatz kommen. Was den Scheduling-Algorithmus betrifft, verhalten sich diese genauso wie ein CRC-Code. Die Länge der vorhergehenden Nutzdaten kann beliebig gewählt werden und lediglich ganz am Ende muss einmalig ein fester Overhead eingefügt werden. Dieser Overhead besteht in der Terminierung des Codes, da ohne Terminierung die Korrekturfähigkeit für die letzten Nutzdaten stark sinken würde. Somit muss bei der Verwendung von Faltungscode niemals Overhead eingefügt werden, wenn noch keine Deadline ansteht, wodurch die Kanalkapazität sehr gut ausgenutzt und das Scheduling bedeutend vereinfacht würde. Der Nachteil von Faltungscodes besteht darin, dass eine Modellierung und Simulation deutlich aufwendiger ist als bei Blockcodes und daher eine Analyse des Systems erschwert wird.

4 Implementierung

Die Algorithmen zum Scheduling der RT-Daten sind – in der Weise wie sie im vorherigen Kapitel beschreiben wurden – noch nicht zur direkten Implementierung geeignet. Die bisherige Beschreibung ist eher auf das menschliche Verständnis ausgelegt. Daher beschäftigt sich dieses Kapitel damit die tatsächliche Implementierung der Scheduling-Algorithmen zu beschreiben. Dabei wird besonders darauf geachtet, dass auf eine gute Laufzeit optimiert wird, damit die Simulation großer Testreihen und die Umsetzung in einem echten System möglich sind.

Außerdem wird in diesem Kapitel die Implementierung eines geeigneten Kanalmodells für die Simulation beschrieben.

Das Kapitel beginnt mit einer Beschreibung der Eingabeparameter des Algorithmus. Danach wird in Abschnitt 4.2 beschrieben, wie die Abbildung des Ressourcenraums implementiert wird, die in Abschnitt 3.4.2 eingeführt wurde. In Abschnitt 4.3 folgt die Implementierung der Algorithmen in Form von Pseudocode. Zuletzt wird in Abschnitt 4.4 das implementierte Kanalmodell vorgestellt.

4.1 Eingabeparameter

Als Eingabe sollen folgende Parameter in der beschriebenen Form gegeben werden. Das Ganze wird in Tabelle 4.1 an einem Beispiel verdeutlicht.

- Ein Array `jobs[]`, das alle „Jobs” enthält, die zu schedulen sind. Jeder Job beschreibt ein zu sendendes Datum und enthält folgende Informationen:
  - Die Releasezeit `job.rel` des Datums.
  - Die Deadline `job.due` des Datums.
  - Ein Array `job.data[]`, das einen Platzhalter für den Inhalt des Datums darstellt. Jedes Feld des Arrays repräsentiert ein Bit des Datums. Da das Scheduling bereits stattfindet, bevor die echten Daten verfügbar sind, können diese hier nicht direkt verwendet werden.

Tabelle 4.1: Beispielhafte Eingabeparameter für den Scheduling-Algorithmus.

`jobs[1..N_jobs]`

#	job.rel	job.due	job.data
1	70 ps	250 ps	d_1[1..8]
2	50 ps	150 ps	d_2[1..32]
3	70 ps	250 ps	d_3[1..16]
4	50 ps	250 ps	d_4[1..32]
5	70 ps	100 ps	d_5[1..16]

`rel[1..N_rel]`

#	Wert
1	50 ps
2	70 ps

`due[1..N_due]`

#	Wert
1	100 ps
2	150 ps
3	250 ps

- Ein Array `rel[]`, das alle Releasezeiten in aufsteigend sortierter Reihenfolge enthält.
- Ein Array `due[]`, das alle Deadlines in aufsteigend sortierter Reihenfolge enthält.

4.2 Abbildung des Ressourcenraums

Bei der theoretischen Betrachtung des Schedulings wurde in Abschnitt 3.4.2 eine neue Zeitbasis eingeführt, mit der das Problem einfacher betrachtet werden konnte. Die drei Vorteile die dadurch erreicht wurden waren

1. die Serialisierung des Ressourcenraums,
2. das Ausblenden von Übertragungsunterbrechungen und
3. die Normalisierung der Datenrate auf eine simple Größe.

Die praktische Umsetzung einer Abbildung der realen Zeitbasis auf diese „lineare” Zeitbasis ist jedoch zu komplex. Stattdessen wird in der Implementierung daher nur ein Teil der obigen Punkte umgesetzt.

Da die erarbeiteten Algorithmen nur auf einen seriellen Datenfluss ausgelegt sind, ist die Umsetzung des ersten Punkts unausweichlich. Tatsächlich erleichtert das die Implementierung aber sogar, da in OMNeT++ keine native Möglichkeit besteht ein Codemultiplex zu modellieren. Das müsste stattdessen über mehrere diskrete Kanäle umgesetzt werden, die jeweils einen der Codes repräsentieren. Da eine sequentielle Anordnung der Ressourcenblöcke verwendet wird, würde das aber zur Folge haben, dass jedes Datum auf sehr viele Kanäle aufgeteilt würde. Das würde nicht nur die Komplexität des Programms erhöhen, sondern auch die Performance verringern, weil viele Nachrichten-Objekte pro Datum erzeugt werden müssten. Mithilfe der Serialisierung wird das Codemultiplex stattdessen in ein Zeitmultiplex verwandelt, das sich in OMNeT++ leicht und effizient umsetzten lässt.

Die Datenrate wird so gewählt, dass die n Bits eines PSSS-Symbols genau eine Symboldauer T_{sym} Übertragungszeit benötigen. Dadurch sind die reale Zeit und die Simulationszeit identisch in allen Größenordnungen über T_{sym}. Die Bitrate des Kanals in der Simulation ergibt sich damit zu $R_b = \frac{n}{T_{sym}}$ beziehungsweise die Bitdauer zu $T_{bit} = \frac{1}{R_b} = \frac{T_{sym}}{n}$. Punkt drei aus der oben Liste wird daher zwar nicht umgesetzt, aber durch die „virtuelle" Bitdauer lässt sich leicht zwischen Datengröße und Übertragungsdauer in der Simulation umrechnen.

Der zweite Punkt – das Ausblenden der Präambeln – ist am heikelsten. Er muss in irgendeiner Weise umgesetzt werden, da die Algorithmen nicht auf die Beachtung dieser Unterbrechungen ausgelegt sind, aber er lässt sich nicht gut durch eine transformierte Zeitbasis darstellen, weil deren Implementierung zu komplex wäre. Stattdessen wird das Problem gelöst, indem die komplexen Anteile in Subroutinen ausgelagert werden, die von den eigentlichen Algorithmen auf einfache Weise eingesetzt werden können. Wie sich herausstellt, genügt es die Übertragungsunterbrechungen in nur zwei Funktionen zu beachten:

- Die erste Funktion entspricht der in Abschnitt 3.4.4 für den allgemeinen Fall eingeführten Funktion `capacity(t1, t2)`. Sie nimmt zwei Zeitpunkte `t1` und `t2` entgegen und gibt die maximal übertragbare Menge an Nutzdaten zwischen diesen beiden Zeitpunkten zurück. Dabei beachtet sie wie schon in Abschnitt 3.4.4 den Overhead durch das FEC-Schema und nun zusätzlich auch noch den Einfluss durch Übertragungsunterbrechungen.

- Die zweite Funktion ist die neu eingeführte `schedule_at(t, data)` Funktion. Sie nimmt einen Zeitpunkt `t` und einen Daten-Platzhalter `data` entgegen und plant die Übertragung des gegebenen Datums zum Zeitpunkt `t` beginnend ein. Sie beachtet bei der Planung die Verzögerung durch Übertragungsunterbrechungen und gibt zuletzt den Zeitpunkt zurück, an dem die Übertragung komplett abgeschlossen ist. Der aufrufende Algorithmus kann den zurückgegebenen Zeitpunkt daher direkt an den nächsten Aufruf der `schedule_at` Funktion weiterreichen um das nächste Datum direkt im Anschluss an das vorhergehende zu senden, ohne genau zu wissen wie die Zeitpunkte zustande kommen.

Die `capacity` Funktion gibt den Algorithmen somit die Fähigkeit *„vorauszudenken"*, während die `schedule_at` Funktion das *„Denken"* beim letzten Schritt des Schedulings abnimmt. Da diese Subroutinen vergleichsweise zeitintensiv sind, sollte ihr Aufruf möglichst selten und nur wenn wirklich notwendig erfolgen. Ihre Implementierungen werden nicht beschrieben, da sie einerseits hochgradig vom zugrunde liegenden Simulations- oder Anwendungssystem abhängig sind und andererseits keine interessanten algorithmischen Ansätze aufweisen.

4.3 Implementierung

Die Implementierung der Algorithmen wird in Listing 4.1 bis 4.3 vorgestellt. Die grobe Arbeitsweise ist in Form von Kommentaren darin angemerkt. Die genauere Funktion ergibt sich durch Studium des Codes selbst.

4.4 Kanalmodell

Die Implementierung des Kanalmodells erfolgt unter verschiedenen Annahmen. Erstens wird vorausgesetzt, dass ein Iterativer-Symbol-Decoder genutzt wird und somit die Modellierung mithilfe der Bitfehlerwahrscheinlichkeit eines BPSK-Systems, wie sie in Formel 2.14 angegeben ist, ausreicht. Zweitens wird angenommen, dass der Abstand der Präambeln deutlich kleiner als die Kanalkohärenzzeit ist und somit während eines Frames konstante Kanaleigenschaften vorliegen. Drittens wird vorausgesetzt, das jederzeit alle Codes eines PSSS-Symbols verwendet werden. Andernfalls müsste ein zusätzlicher Term eingeführt werden, der den Spreizgewinn modelliert, worauf hier aber verzichtet wurde, weil anzunehmen ist, dass durch die sequentielle Anordnung der Ressourcenblöcke ohnehin fast alle PSSS-Symbole vollständig genutzt werden. Zuletzt müsste bei einem Multiple-Access-Szenario noch die Desynchronisation der verschiedenen Signale am Empfänger beachtet werden. Da aber sowieso nur der Downlink-Kanal implementiert wurde, stellt sich dieses Problem nicht, weil nur ein einzelner Sender existiert. Die Berechnung der Bitfehlerwahrscheinlichkeit ist in Algorithmus 4.4 dargestellt.

Algorithmus 4.1: Aufteilung des Problems und Delegation der Teilprobleme an die weiteren Algorithmen.

```
1  function schedule(jobs[1..Njobs], rel[1..Nrel], due[1..Ndue])
2    assert is_sorted(rel) and is_sorted(due)
3
4    // Falls nur eine Releasezeit oder Deadline vorliegen, kann
5    // direkt zum entsprechenden Algorithmus gesprungen werden.
6    if Nrel = 1
7      schedule_with_common_release(jobs, rel[1])
8      return
9    else if Ndue = 1
10     schedule_with_common_deadline(jobs, due[1])
11     return
12
13   // Die Job-Liste sortieren.
14   jobs ← sort_by_deadline_first_and_release_second(jobs)
15
16   // Die Slacks initialisieren.
17   slack ← new array[1..Nrel]
18   foreach k in 1..Nrel
19     slack[k] ← capacity(rel[k], due[1])
20
21   L1, L2 ← new empty arrays[]
22   foreach job in jobs
23
24     // Den minimalen Slack s* bestimmen.
25     s* ← ∞
26     foreach k in 1..Nrel where rel[k] ≤ job.rel
27       s* ← min(s*, slack[k])
28
29     // Die anteilige Zuordnung zu L1 und L2 durchfuhren.
30     if s* = 0
31       L2.append(job)
32     else if s* ≥ length(job.data)
33       s* ← length(job.data)
34       L1.append(job)
35     else
36       job1,job2 ← copies(job)
37       job1.data ← job.data[1..s*]
38       job2.data ← job.data[(s*+1)..end]
39       L1.append(job1)
40       L2.append(job2)
41
42     // Die Slacks aktualisieren.
43     foreach k in 1..Nrel where rel[k] ≤ job.rel
44       slack[k] ← slack[k]-s*
45
46   // Die passenden Algorithmen auf die beiden Teilprobleme aufrufen.
47   schedule_with_common_deadline(L1, due[1])
48   schedule_with_common_release(L2, due[1])
```

Algorithmus 4.2: Scheduling der Daten mit einheitlicher Deadline.

```
1  function schedule_with_common_deadline(jobs[1..Njobs], D)
2    sort_by_release(jobs)
3
4    // Die Gesamtgrosse der Daten berechnen.
5    data_size = ∑ length(job.data) foreach job in jobs
6    total_size = data_size + crc_size
7    first_blk_size ← total_size mod blk_size
8
9    // Die Startzeit des Schedules bestimmen. Dazu solange die
10   // Startzeit variieren bis die Kapazitat zur Datenmenge passt.
11   time ← D
12   capacity ← 0
13   while capacity ≠ total_size
14     time ← time − (total_size−capacity)·bit_duration
15     capacity ← capacity(time, D)
16
17   bits_until_fec ← first_blk_size
18   foreach job in jobs
19     while length(job.data) > 0
20
21       // Das Datum so zuschneiden, dass es in den aktuellen FEC−Block passt.
22       if length(job.data) ≤ bits_until_fec
23         data ← job.data
24         job.data ← ∅
25       else
26         data ← job.data[1..bits_until_fec]
27         job.data ← job.data[(bits_until_fec+1)..end]
28
29       // Das Datum schedulen.
30       time ← schedule_at(time, data)
31
32       // Den FEC−Block abschliessen, falls er voll ist.
33       bits_until_fec ← bits_until_fec − length(data)
34       if bits_until_fec = 0
35         time ← schedule_at(time, new FEC)
36         bits_until_fec ← blk_size
37
38   // Den CRC schedulen und den letzten FEC−Block abschliessen.
39   time ← schedule_at(time, new CRC)
40   time ← schedule_at(time, new FEC)
```

Algorithmus 4.3: Scheduling der Daten mit einheitlicher Releasezeit.

```
1  function schedule_with_common_release(jobs[1..Njobs], R)
2    sort_by_deadline(jobs)
3
4    time ← R
5    job ← first(jobs)
6    while not all jobs processed
7
8      // Die nachste Deadline und die Mindestmenge an Daten bis dahin bestimmen.
9      D ← job.due
10     due_size ← ∑ length(_job.data) foreach _job in remaining jobs where _job.due = D
11
12     // Den Slack – aka. die mogliche Nutzdatenmenge – bis zur Deadline bestimmen.
13     capacity ← capacity(time, D)
14     slack ← capacity – crc_size
15     if slack < due_size
16       abort with error
17
18     // Berechnen, ob der verkurzte Block genutzt werden muss.
19     last_blk_size ← capacity mod blk_size
20     if slack – last_blk_size ≥ due_size
21       slack ← slack – last_blk_size
22
23     bits_until_fec ← blk_size
24     while slack > 0 and not all jobs processed
25
26       // Das Datum so zuschneiden, dass es in den aktuellen FEC–Block passt.
27       size ← min(slack, bits_until_fec)
28       if length(job.data) ≤ size
29         data ← job.data
30         job ← next(jobs)
31       else
32         data ← job.data[1..size]
33         job.data ← job.data[(size+1)..end]
34
35       // Das Datum schedulen.
36       time ← schedule_at(time, data)
37
38       // Den FEC–Block abschliessen, falls er voll ist.
39       bits_until_fec ← bits_until_fec – length(data)
40       if bits_until_fec = 0
41         time ← schedule_at(time, new FEC)
42         bits_until_fec ← blk_size
43
44       // Den Slack aktualisieren.
45       slack ← slack – length(data)
46
47     // Den CRC schedulen und den letzten FEC–Block abschliessen.
48     time ← schedule_at(time, new CRC)
49     time ← schedule_at(time, new FEC)
50
51     // Weiter mit dem nachsten Datum.
```

Algorithmus 4.4: Berechnung der Bitfehlerwahrscheinlichkeit.

```
1  // Pfadverlust berechnen.
2  path_loss_dB ← η·10·log10(distance_m) + 20·log10(frequenzy_Hz) − 147.55
3
4  // Signalstarke am Empfanger berechnen.
5  receive_power_dBm ← transmit_power_dBm − path_loss_dB − constant_attenuation_dB
6  receive_power_mW ← pow(10, receive_power_dBm/10)
7
8  // Fading−Einfluss berechnen.
9  // Parameter der Rice−Verteilung berechnen.
10 k ← 2.2 // Rice−Faktor
11 ν ← sqrt(receive_power_mW · k/(1+k)) // ν² = Leistung uber den LOS−Pfad
12 σ ← sqrt(receive_power_mW · 1/(2+2k)) // 2σ² = Leistung uber alle NLOS−Pfade
13 // Rice−verteilte Zufallsvariable erzeugen.
14 P ← poisson(ν²/2σ²)
15 X ← chi_square(2·P+2)
16 R ← sigma·sqrt(X) // R ∼ Rice(ν, σ)
17 // Empfangsleistung berechnen.
18 faded_receive_power_mW ← R²
19
20 // EbN0 berechnen.
21 effective_bit_energy_J ← faded_receive_power_mW/10³ · symbol_duration_s/bits_per_symbol
22 thermal_noise_power_spectral_density_W_per_Hz ← k_boltzmann_J_per_K · temperature_K
23 EbN0 ← effective_bit_energy_J / thermal_noise_power_spectral_density_W_per_Hz
24
25 // Bitfehlerwahrscheinlichkeit berechnen.
26 bit_error_probability ← ½erfc(sqrt(EbN0))
```

5 Simulation

Dieses Kapitel beschäftigt sich mit der Evaluation des entworfenen Konzepts. Dazu werden zuerst in Abschnitt 5.1 zwei Szenarien vorgestellt, die realen FA-Anlagen nachempfunden sind. Das erste Szenario ist ein Pick&Place-System, das zum Sortieren und Verpacken von kleinteiligen Waren geeignet ist. Das zweite Szenario ist eine Anlage zur Getränkeabfüllung in PET-Flaschen, die für einen hohen Durchsatz konstruiert ist und daher sehr hohe Anforderungen aufweist.

Anschließend wird in Abschnitt 5.2 die Implementierung des Scheduling-Algorithmus simulativ verifiziert.

5.1 Anwendungsszenarien

Anhand von zwei realistischen Anwendungsbeispielen soll die Tauglichkeit und Performance des entworfenen Systems untersucht werden. Da besonders die Probleme der Fertigungsautomation in Angriff genommen wurden, werden entsprechend auch Anwendungen aus der diskreten Fertigung betrachtet. Weil Verpackungsmaschinen die höchsten Anforderungen besitzen, wurden eben solche zur Evaluation ausgewählt.

5.1.1 Pick & Place

Das erste Szenario beschreibt ein Pick&Place-System, das ungeordnete Waren von einem Förderband greift und sie strukturiert auf einem anderen Förderband ablegt. Die Waren können dabei zum Beispiel direkt in Verpackungen platziert werden, die sich auf dem zweiten Förderband befinden. Daher ist eine sehr große Genauigkeit bei der Positionierung des Endeffektors notwendig und gleichzeitig ist für einen hohen Warendurchsatz eine hohe Arbeitsgeschwindigkeit wünschenswert. Übliche Systeme wie beispielsweise ein Bosch Paloma [27] werden vom Hersteller mit 120 Picks pro Minute pro Roboter beworben. Pro Vorgang ergibt sich damit eine Dauer von nur 500 ms bei einer zurückgelegten Strecke des Endeffektors von mindestens

30 cm. Wegen der hohen Geschwindigkeit wird eine sehr kurze IE-Zykluszeit von 1 ms angenommen.

Für diese Aufgabe kommen sehr oft sogenannte Delta-Roboter zum Einsatz. Ein Delta-Roboter wird mit seiner Basiseinheit an der Decke über seinem Arbeitsbereich montiert. er kann seine Arbeitsplattform, die am Ende dreier Arme montiert ist, parallel zum Boden verschieben, anheben und rotieren. Mehrere dieser Roboter – hier im Beispiel seien es drei – werden über zwei Förderbändern platziert. Mit Endeffektoren an ihrer Arbeitsplattform können sie Waren vom einen Förderband greifen und an einer vorgegebenen Stelle mit einer vorgegebenen Orientierung auf dem anderen Förderband ablegen. Als Endeffektoren können zum Beispiel Sauggreifer zum Einsatz kommen, die aus mehreren einzeln angesteuerten Saugnäpfen bestehen und damit eine Gruppe von Waren sehr schonend aber sicher anheben können.

Ein Delta-Roboter verfügt über drei Servomotoren zur Translation der Arbeitsplattform und einen weiteren Servomotor zur Rotation des Endeffektors. Ein Sauggreifer-Endeffektor mit mehreren Greifern wird über Druckluftsystem mit Energie versorgt und mithilfe von Druckluftventilen angesteuert. Jeder Greifer kann dabei einzeln angesprochen werden um ein aufsammeln und zusammenstellen mehrerer Waren in einem Arbeitsschritt zu ermöglichen. Damit das Greifen stabil und schonend funktioniert ist außerdem ein Druckregelventil an der Druckluftzuleitung notwendig. Alle Ventile sind in der Basiseinheit untergebracht und über Schläuche mit dem Endeffektor verbunden. Die Erzeugung der Druckluft läuft über einen zentralen Kompressor, der vereinfachend aus einem Servomotor und einem Druckregelventil bestehend, modelliert werden soll. Schlussendlich gibt es noch einen Servomotor als Antrieb für jedes der Förderbänder. Dieser befindet sich am entfernten Ende der Förderbänder von der Steuereinheit aus betrachtet. In der Praxis können die Förderbänder eine große Länge erreichen. Außerdem gibt es ein Kamerasystem, das die Position der Waren auf dem ankommenden Förderband erfasst. Das Steuergerät ist zusammen mit dem Kamerasystem in einem Gehäuse über dem ankommenden Förderband untergebracht.

Aufbau des hybriden Netzwerks Ein passender Aufbau des hybriden Netzwerks kann wie folgt realisiert werden. Das Kamerasystem wird aufgrund seiner Nähe zur Steuereinheit unmittelbar per IE mit dieser verbunden. Das PGW befindet sich ebenfalls an der Wand des Gehäuses in der Nähe der Steuereinheit. Der Kompressor und die Antriebe der Förderbänder befinden sich weiter entfernt über die Anlage verteilt und werden daher jeweils mit einem SGW ausgestattet. Ebenso wird in jedem Delta-Roboter ein SGW in seiner Basiseinheit unter der Decke platziert. Die Teilkomponenten des Roboters sind dann per IE mit dem SGW verbunden.

Abschätzung des Datenumfangs Um die Datenmenge, die pro Sensor oder Aktor anfällt abzuschätzen, werden die Datenblätter realer Bauteile betrachtet. Ein Druckluftventilsystem von Aventics aus der DDL Produktreihe [28] benötigt für die gegebene Anzahl an Ventilen in einem Endeffektor 1 Byte im Downlink und 2 Byte im Uplink. Ein Druckregelventil aus der gleichen Baureihe benötigt 2 Byte im Downlink und 3 Byte im Uplink. Bei den Servomotoren muss noch einmal zwischen verschiedenen Betriebsmodi unterschieden werden. Für die Förderbänder und den Kompressor lassen sich beispielsweise die Profile „IO-Geschwindigkeitsvorgabe" oder „FSP-Drive" eines Bosch Rexroth IndraDrive Antriebs [29] verwenden, die 2 Byte pro Zyklus im Downlink und 2 Byte pro Zyklus im Uplink benötigen. Für die Delta-Roboter muss stattdessen das Profil „Antriebsgeführtes Positionieren" verwendet werden, das 10 Byte pro Zyklus im Downlink und 14 Byte pro Zyklus im Uplink belegt.

Die Menge der TPC/TA-Daten pro SGW kann gemäß der Formeln 3.5 und 3.6 berechnet werden. Dazu müssen d_{min} und d_{max} bestimmt werden. Der minimale Abstand besteht zwischen dem PGW und dem SGW am nächstgelegenen Delta-Roboter. Abhängig von der genauen Platzierung des PGW beträgt er im Minimum $\frac{1}{2}$ m. Der maximale Abstand liegt dagegen zwischen PGW und einem der Servomotoren der Förderbänder. Er kann bis zu circa 10 m betragen. Die Chipdauer beträgt bei ParSec $T_{chip} = 50\,\text{ns}$ und für das Kanalmodell wird $\eta = 2{,}2$ angenommen. Damit werden pro SGW

$$n_{TPC,TA} = \left\lceil \log_2 \left(\eta \cdot 40 \log_{10} \left(\frac{10\,\text{m}}{\frac{1}{2}\,\text{m}} \right) \right) \right\rceil + \left\lceil \log_2 \left(20 \frac{10\,\text{m} - \frac{1}{2}\,\text{m}}{T_{chip} \cdot c_0} \right) \right\rceil = 7\,\text{bit} + 4\,\text{bit} = 11\,\text{bit}$$

benötigt. Bei insgesamt sechs SGWs ergibt sich damit eine Datenmenge von 66 bit für die TPC/TA-Daten, wobei noch die Absicherung durch einen FEC hinzu kommt.

Eine Zusammenfassung der relevanten Parameter des Szenarios findet sich in Tabelle 5.1.

5.1.2 Getränkeabfüllung

Als zweites Szenario wird eine Anlage zur Getränkeabfüllung am Beispiel der KHS Innofill Serie [30] gewählt. Dieses System ist besonders durch seine sehr hohe Arbeitsgeschwindigkeit – mit einer Abfüllleistung von 81000 Flaschen pro Stunde – und durch seine hohe Anzahl an Aktoren und Sensoren interessant. Es ist als ein rotierendes Karussell mit 170 separaten Abfülleinheiten an seiner Außenlinie aufgebaut. An einer Stelle werden leeren Flasche in die Anlage eingespeist und dann nach einem fast vollständigen Umlauf die gefüllten Flaschen wieder entnommen. Der Abfüllvorgang läuft während des Umlaufs einer Flasche im Karussell ab und wird durch verschiedene Mechanismen gesteuert. Nachdem eine leere Flasche mechanisch in eine der Abfülleinheiten geschoben wurde, wird zuerst über eine Kopplung mit der Drehbewegung

des Karussells die Flasche fixiert und gegebenenfalls ein Füllstandsensor eingeführt. Durch die rein mechanische Ausführung erfordert dieser Arbeitsschritt keine externe Steuerung. Dann wird durch ein elektromechanisches Ventilsystem anschließend der Füllvorgang gestartet. Je nachdem welches Abfüllsystem verwendet wird, sind zwei bis vier Ventile vorhanden, die aber durch eine einzelne Ventilsteuereinheit bedient werden. Während des Füllens wird die Flüssigkeitsmenge entweder durch einen Durchflusssensor oder einen Füllstandsensor überwacht. Sobald die Füllung abgeschlossen ist, werden die Ventile wieder geschlossen um den Füllvorgang zu beenden. Zum Schluss wird die Flasche mechanisch aus der Anlage entnommen. Wenn die Abfüllung bis zur Entnahme der Flasche nicht abgeschlossen wäre, müsste die Flasche halbvoll entnommen und bei der Qualitätskontrolle entfernt werden. Zusammen mit der hohen Arbeits- und Umlaufgeschwindigkeit der Anlage ist daher eine sehr präzise und zuverlässige Steuerung der Ventile notwendig. Auf die Position einer einzelnen Abfülleinheit im Karussell kann leicht durch den Drehwinkel der Anlage zurück geschlossen werden. Daher und um eine konstante Arbeitsgeschwindigkeit zu erreichen, wird das Karussell durch einen Servomotor angetrieben.

Hinzu kommen bei der Getränkeabfüllung weitere Probleme aus dem Bereich der Prozesstechnik, wie beispielsweise die gleichmäßige Kühlung und Umwälzung des Getränks. Da hierbei deutlich andere Anforderungen bezüglich Latenz und Zuverlässigkeit der Kommunikation bestehen, werden diese hier aber nicht weiter betrachtet.

Aufbau des hybriden Netzwerks Die Anlagensteuerung befindet sich am Rande der Anlage. Das PGW sollte jedoch am besten zentral oberhalb der Anlage angebracht werden um eine gute Funkstrecke zu allen SGWs zu erhalten. Die Verbindung der Steuerung mit dem PGW per IE ist dabei unproblematisch, weil beide ortsfest sind. Der Servomotor des Karussellantriebs befindet sich unterhalb der Anlage und bekommt ein eigenes SGW. Da die Anlage torodial aufgebaut ist, also innen „hohl” ist, besteht trotzdem eine Sichtverbindung zwischen dem PGW und dem SGW des Karussellantriebs. Die Anbindung der Fülleinheiten ist in verschiedenen Konfigurationen denkbar. Im einen Extremfall besitzt jede Fülleinheit ein eigenes SGW und im anderen Extremfall sind alle Fülleinheiten per IE mit einem gemeinsamen SGW verbunden. Jedes SGW bringt allerdings gewisse Hardwarekosten und einen zusätzlichen Uplink-Overhead mit sich. Somit ist die Anbindung über 170 separate SGWs unattraktiv. Gegen die Anbindung über ein einzelnes SGW sprechen keine unmittelbaren Gründe. Würde eine Anlage aber zum Beispiel aus mehreren Modulen zusammengesetzt, die jeweils 10 Fülleinheiten umfassen, könnte es sich anbieten jedes Modul mit einem eigenen SGW und internen IE auszustatten. Die resultierenden 17 SGWs wären ein Kompromiss zwischen dem Hardwareaufwand und Uplink-Overhead auf der einen und der Flexibilität auf der anderen Seite. Natürlich ist auch jede andere Gruppierung von Fülleinheiten möglich. Es ist anzunehmen, dass die SGWs

Tabelle 5.1: Zusammenfassung der wichtigsten Parameter der Simulationsszenarien. (* Die Anzahl der SGWs hängt von der Gruppierung der Fülleinheiten ab. Die angegebene Anzahl ist ein denkbares Beispiel.)

	Pick&Place	Getränkeabfüllung
Anzahl der SGWs	6	18*
Zykluszeit	1 ms	1 ms
Datenaufkommen DL	137 Byte	172 Byte
Datenaufkommen UL	198 Byte	1192 Byte
TPC/TA-Daten pro SGW	11 bit	6 bit
Minimale Distanz	$\frac{1}{2}$ m	4 m
Maximale Distanz	10 m	6 m
Maximale Geschwindigkeit	0 m/s	2,9 m/s

nahe am Rand des Karussells platziert sind und sich somit bei einem Radius der Anlage von circa 3,5 m mit 2,9 m/s bewegen.

Abschätzung des Datenumfangs Wie schon für das andere Szenario, kann das Datenaufkommen mithilfe von Datenblättern realer Komponenten abgeschätzt werden. Für die Steuerung des Servomotors reicht dabei das schon besprochene simplere Profil „IO Geschwindigkeitsvorgabe" oder „FSP-Drive" aus. Für das Ventilsystem werden hier die Daten des Druckluftventilsystems von zuvor übernommen, da sich dieses vom Steuerverhalten her nicht maßgeblich unterschieden sollte. Für den Durchflusssensor werden schließlich die Werte des Promag 100 [31] der Firma Endress+Hauser verwendet. Dieser sendet ein Datum von 5 Byte pro Zyklus im Uplink. Ein äquivalenter Füllstandsensor wird mit einem vergleichbaren Datenaufkommen angenommen.

Da das PGW unter der Decke angebracht wird, beträgt der Abstand zu den SGWs circa 4 m bis 6 m. Mit T_{chip} = 50 ns und η = 2,2, wie im anderen Anwendungsszenario, ergibt sich ein TPC/TA-Datenaufkommen von

$$n_{TPC,TA} = \left\lceil \log_2 \left(\eta \cdot 40 \log_{10} \left(\frac{6\,\text{m}}{4\,\text{m}} \right) \right) \right\rceil + \left\lceil \log_2 \left(20 \frac{6\,\text{m} - 4\,\text{m}}{T_{chip} \cdot c_0} \right) \right\rceil = 4\,\text{bit} + 2\,\text{bit} = 6\,\text{bit}$$

pro SGW. Mit der oben genannten Gruppierung von Fülleinheiten würden die insgesamt 18 SGWs somit 108 bit TPC/TA-Daten zuzüglich FEC benötigen.

Die IE-Zykluszeit wird wegen der hohen Arbeitsgeschwindigkeit und der nötigen Füllstands-Präzision mit 1 ms angenommen.

Damit sind auch alle Parameter des zweite Szenarios umrissen. Eine Zusammenfassung beider Szenarien befindet sich in Tabelle 5.1.

Tabelle 5.2: Parameter in der Simulation zur Verifizierung des Scheduling-Algorithmus.

(a) PSSS- und Sercos-Parameter

Parameter	Wert
Anzahl Codes (n)	255
Symboldauer (T_{sym})	14,25 µs
Präambeldauer	$5 \cdot T_{sym}$
Präambelintervall	$34 \cdot T_{sym}$
Zyklusdauer (T_{cyc})	$50 \cdot T_{sym}$

(b) Jobs in Test 1

#	Release	Deadline	Größe
1	$6 \cdot T_{sym}$	$30 \cdot T_{sym}$	255 bit
2	$8 \cdot T_{sym}$	$31 \cdot T_{sym}$	255 bit
3	$11 \cdot T_{sym}$	$33 \cdot T_{sym}$	2040 bit

(c) Jobs in Test 2

#	Release	Deadline	Größe
1	$6 \cdot T_{sym}$	$30 \cdot T_{sym}$	510 bit
2	$8 \cdot T_{sym}$	$31 \cdot T_{sym}$	510 bit
3	$11 \cdot T_{sym}$	$33 \cdot T_{sym}$	2040 bit

5.2 Verifizierung der Implementierung

Zur Verifizierung der korrekten Funktionsweise des Scheduling-Algorithmus wird im Folgenden eine simple Problemstellung simuliert. Der von der Implementierung produzierte Schedule wird anschließend mit dem erwarteten Ergebnis abgeglichen. Es werden zwei Test mit den in Tabelle 5.2 angegebenen Parametern durchgeführt. Die Parameter sind so gewählt, dass möglichst viele Sonderfälle sichtbar werden. Im ersten Test ist das Scheduling in jedem Zyklus möglich, im zweiten Test reichen die PSSS-Kapazitäten dagegen in einigen Zyklen nicht aus um alle Daten fristgerecht zu senden. Die Implementierung der Bitfehlersimulation wird hier nicht untersucht.

Die Visualisierung des Schedules in den Abbildungen 5.1 bis 5.6 erfolgt mithilfe des von OMNeT++ bereitgestellten Tool zur Darstellung einer aufgezeichneten Simulation als Sequenzdiagramm. Die mittlere Zeile der Abbildungen repräsentiert jeweils das PGW und die untere Zeile ein beliebiges SGW. Dazwischen ist der Datenverkehr im Downlink zu erkennen. Alle Datenpakete sind mit ihrem jeweiligen Datentyp beschriftet. Aufgrund von Aggregation und Splitting der RT- und CRC-Daten lassen sich die Pakete nicht direkt den Scheduling-Jobs aus Tabelle 5.2 zuordnen. Das ist erst durch einen Blick in die Log-Dateien der Simulation möglich. Deshalb wurden die Abbildungen nachträglich mit diesen Informationen annotiert. Die obere Hälfte der Abbildungen ist nicht Teil des eigentlichen Netzwerks, sondern wurde künstlich hinzugefügt um die Releasezeiten und Deadlines der Daten erkennen zu können. Diese Informationen würden ansonsten nicht rekonstruiert werden können.

Erster Test Im ersten Test werden verschiedene Situationen betrachtet, in denen erfolgreich geschedult werden kann. Kein Datum verletzt in den folgenden Beispielen seine Releasezeit oder Deadline oder wird überhaupt nicht geschedult. Außerdem sind alle Daten korrekt durch FECs und CRCs abgesichert.

In Abbildung 5.1 ist der grundlegende Fall zu sehen, dass alle Releasezeiten und Deadlines komplett zwischen zwei Präambeln liegen. Die Übertragung der Daten kann daher unterbrechungsfrei stattfinden. Zu sehen ist, dass der erzeugte Schedule daher wie erwartet keine Lücken aufweist. Außerdem lässt sich sehen, dass in gleichmäßiger Abfolge RT-Daten und FEC-Daten gesendet werden, wie es vorgesehen ist. Nach allen Daten wird zuletzt planmäßig ein einzelner CRC zu Absicherung platziert. Dann endet die Übertragung wie vorgesehen genau zur ersten Deadline.

In Abbildung 5.2 befindet sich nun zwischen den Releasezeiten und Deadlines eine Präambel. Dadurch muss die Übertragung der Daten zwischenzeitig unterbrochen werden. Auch hier ist zu erkennen, dass der erzeugt Schedule sich erwartungsgemäß lückenlos um die Präambel schmiegt. Weil es im abgebildeten Fall unmöglich ist das dritte Datum noch früher zu senden, da es bereits zu seiner Releasezeit beginnt, kann die Übertragung nicht vor der ersten Deadline abgeschlossen werden. Somit wurde im Schedule der größtmögliche Anteil der Daten vor der ersten Deadline und der Rest im Bereich zwischen den Deadlines eingeplant. Vor der ersten und letzten Deadline wurde ein CRC eingefügt; nicht aber vor der zweiten Deadline, weil dort keine neuen Daten fällig sind. Die Daten zur zweiten Deadline wurden bereits vor der ersten Deadline übertragen und mit dem dortigen CRC abgesichert. Der zweite CRC wird auf zwei FEC-Blöcke aufgeteilt, sodass am Ende zwei CRC-Fragmente in der Abbildung zu erkennen sind.

Abbildung 5.3 zeigt einen ähnlichen Fall zu Abbildung 5.2. Durch die Lage der Präambel ist hier allerdings das erste Datum durch seine Releasezeit limitiert, weshalb nicht alle Daten bis zur ersten Deadline übertragbar sind. Weil dadurch im Gegensatz zu Abbildung 5.2 der Zeitraum zwischen den Releasezeiten voll genutzt werden kann, endet der Schedule hier früher. Es verbleibt jedoch auch hier ein kleiner Rest des dritten Datums, der nach der ersten Deadline gesendet werden muss.

Zuletzt ist in Abbildung 5.4 der Fall dargestellt, dass die Präambel den gesamten Bereich der Deadlines verdeckt. Deshalb ist der Schedule gezwungen die Übertragung bereits weit vor der ersten Deadline abzuschließen. Wie zu sehen ist, kann die Implementierung auch damit erfolgreich umgehen.

Zweiter Test Im zweiten Test wurde die Datengröße der ersten beiden Jobs gegenüber dem ersten Test verdoppelt. Dadurch ist der Schedule bereits ohne Unterbrechung durch eine Prä-

ambel an der Kapazitätsgrenze des Netzwerks angelangt. Dies lässt sich in Abbildung 5.5 erkennen. Sie ist – abgesehen von den größeren Jobs 1 und 2 – äquivalent zu Abbildung 5.1 aus dem ersten Test.

In Abbildung 5.2 liegt dagegen eine Präambel zwischen den Releasezeiten und Deadlines. Der Scheduling-Algorithmus versucht zuerst so viele Daten wie möglich vor der ersten Deadline zu senden. Dadurch gelingt es ihm den ersten und zweiten Job ebenso wie den Großteil des dritten Jobs vor der ersten Deadline zu senden. Sofort danach stellt er jedoch fest, dass der verbleibende Rest des dritten Jobs nicht mehr vor der dritten Deadline übertragen werden kann. Die Übertragung bricht daher an dieser Stelle ab ohne den Rest der Daten zu senden.

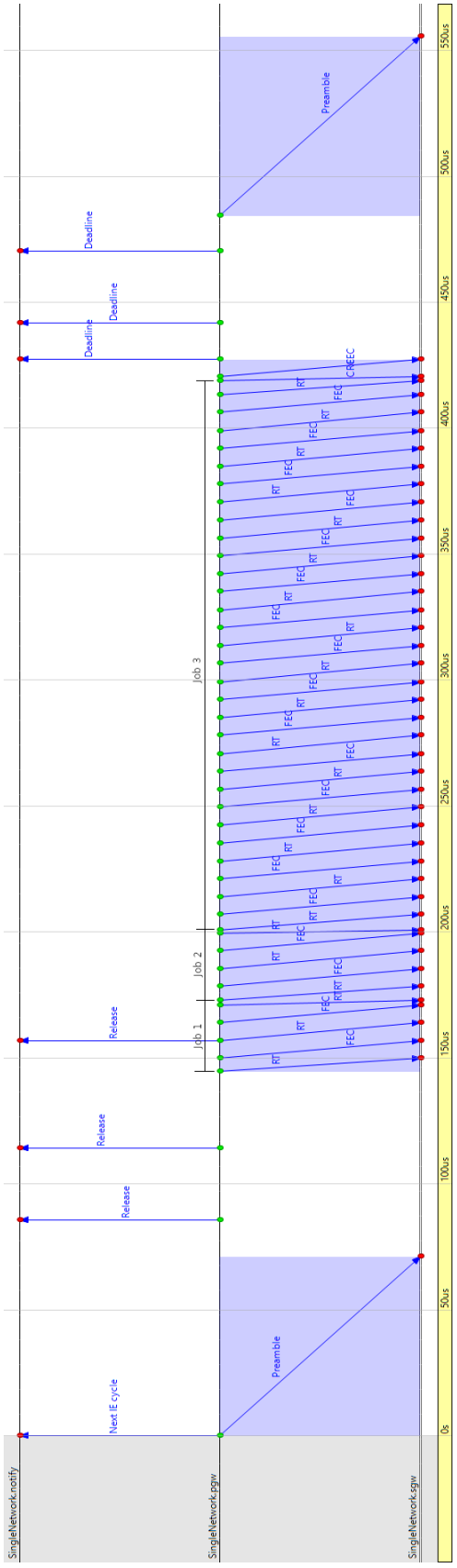


Abbildung 5.1: Erster Ausschnitt aus Test 1.

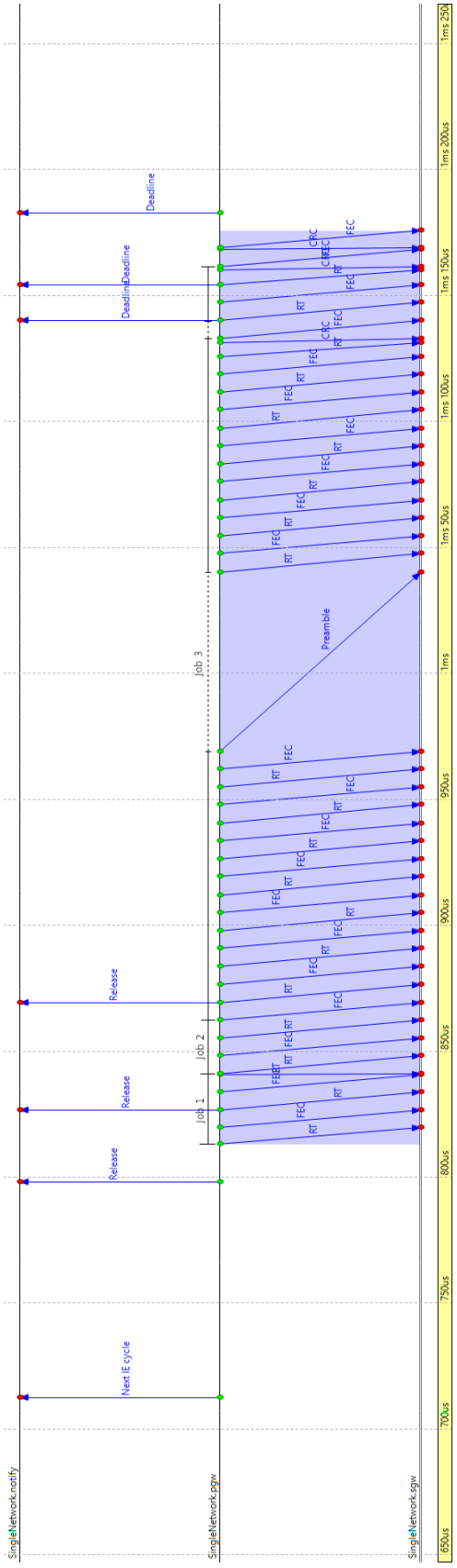


Abbildung 5.2: Zweiter Ausschnitt aus Test 1.

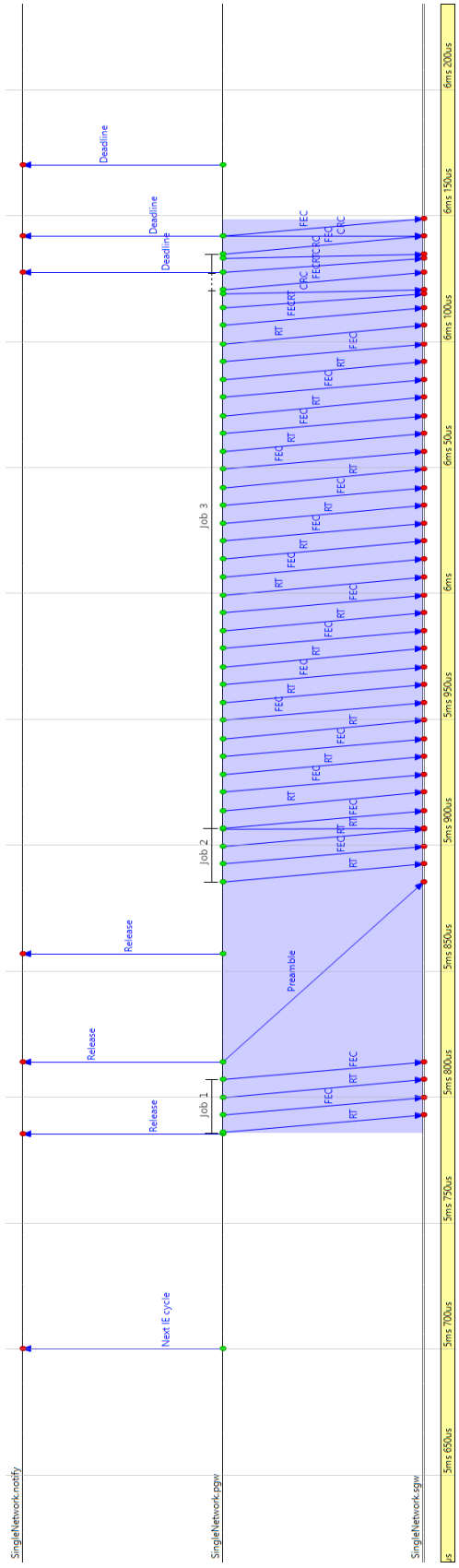


Abbildung 5.3: Dritter Ausschnitt aus Test 1.

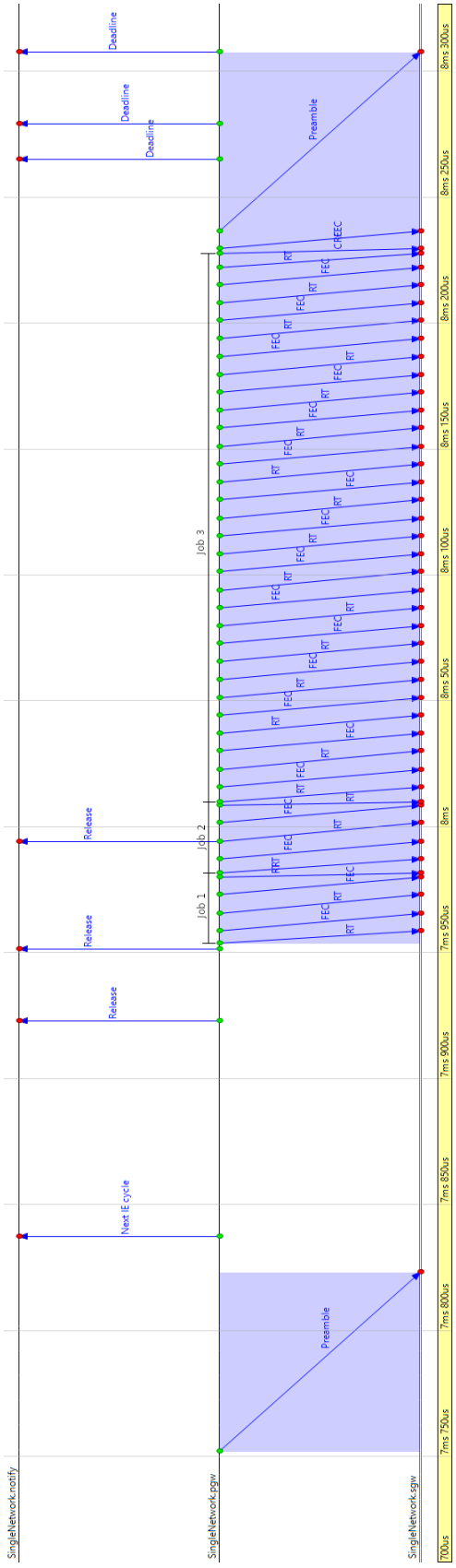


Abbildung 5.4: Vierter Ausschnitt aus Test 1.

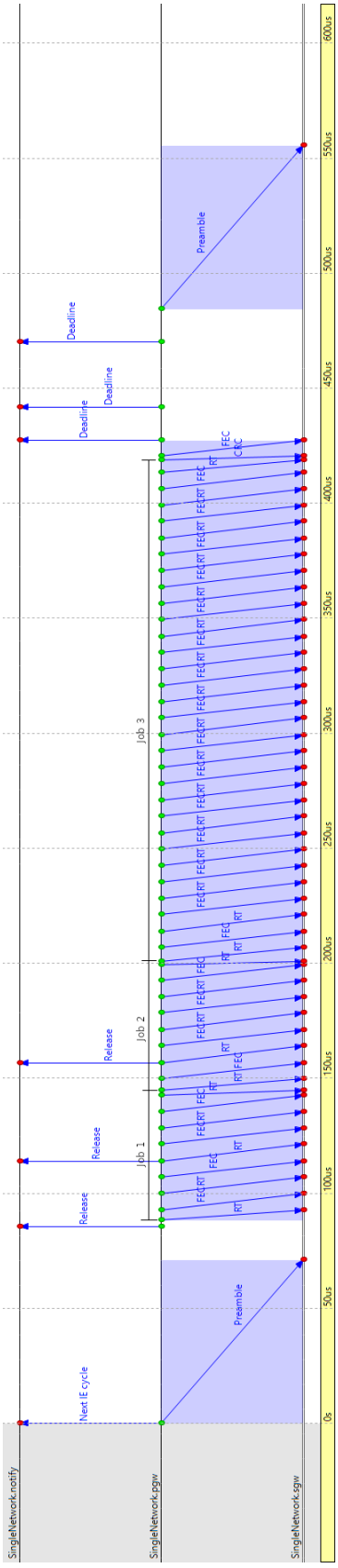


Abbildung 5.5: Erster Ausschnitt aus Test 2.

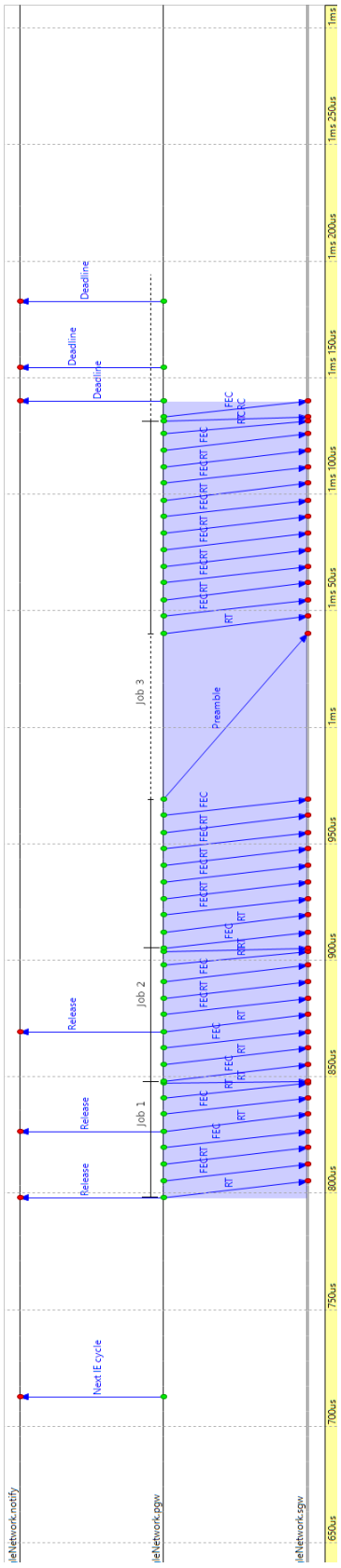


Abbildung 5.6: Zweiter Ausschnitt aus Test 2.

6 Fazit

Im Rahmen dieser Arbeit wurde ein Ressourcenzuweisungsverfahren für hybride Netzwerke in der Fabrikautomation entworfen und implementiert.

In Abschnitt 3.1 wurde die grundlegende Topologie des hybriden Netzwerks ausgewählt. Das Augenmerk lag darauf den in Kapitel 1 und Abschnitt 2.2 erklärten Anforderungen zu genügen. Allen voran wurde eine Topologie gewählt, die abwärtskompatibel mit bestehenden IE-Netzwerken ist und die eine geringe Latenz ermöglicht.

Aus dem in Abschnitt 2.3 recherchierten Aufbau von Sercos-Netzwerken wurden in Abschnitt 3.2 Anforderungsbeschreibungen der verschiedenen Datentypen des IE erstellt. Darüber hinaus wurde hergeleitet, welche Anforderungen die TPC/TA-Daten zur Aufrechterhaltung des PSSS-Uplinks besitzen.

Aus den Anforderungen der Datentypen und dem Aufbau des IEs wurde dann in Abschnitt 3.3 ein geeigneter Kommunikationsablauf im hybriden Netzwerk abgeleitet. Es hat sich herausgestellt, dass das optimale Scheduling der RT-Daten die größte Herausforderung darstellt.

Daher wurde in Abschnitt 3.4 ein neuer Scheduling-Algorithmus für die RT-Daten entwickelt, der die in Abschnitt 3.3 herausgearbeiteten Anforderungen erfüllt. Dazu wurden verschiedene Annahmen getroffen, deren Folgen und Alternativen in den Abschnitten 3.4.5 und 3.4.6 diskutiert wurden. Der Algorithmus selbst wurde in Abschnitt 3.4.4 ausführlich hergeleitet und anhand eines Beispiels verdeutlicht.

Die Beschreibung des Algorithmus in Kapitel 3 war noch nicht geeignet um daraus eine Implementierung zu erstellen. Daher wird in Kapitel 4 eine Umsetzung des Scheduling-Algorithmus in Pseudocode präsentiert. Es wurde eine Implementierung dieses Ansatzes in OMNeT++ erstellt, die neben dem Scheduling-Algorithmus außerdem das in Abschnitt 4.4 vorgestellte Kanalmodell umfasst. Mithilfe der Erkenntnisse aus Abschnitt 2.4.5 ist damit die Simulation von Bitfehlern in der PSSS-Übertragung möglich.

Um den implementierten Scheduling-Algorithmus zu verifizieren wurde in Abschnitt 5.2 ein simples Schedulingproblem simuliert. Die Ergebnisse zeigen das erwartete, korrekte Verhalten. Darüber hinaus wurden zwei realistische Anwendungsszenarien aus der Industrie recher-

chiert und in Abschnitt 5.1 vorgestellt, die zur Evaluation des Systems eingesetzt werden können.

Der nächste Schritt bestünde darin, eine Strategie zur Analyse der Systemperformance zu erarbeiten und diese anhand einer Simulation der präsentierten Szenarien durchzuführen. Um eine ganzheitliche Betrachtung zu erreichen, müsste zudem das Scheduling weiter Datentypen implementiert werden und ein klar definiertes, vollständiges MAC-Protokoll entworfen werden.

Während dieser Arbeit sind einige Fragen und Anregungen aufgekommen, die eine weitere Untersuchung motivieren könnten. Zum einen wurde die Verwendung linearer Blockcodes ausgewählt um mit vorhergehenden Arbeiten konform zu bleiben und eine einfache Fehlermodellierung umsetzen zu können. Der Scheduling-Algorithmus könnte aber durch den Einsatz von Faltungscodes signifikant vereinfacht werden. Ähnliches gilt für die Verwendung der CRC-Codes. Es existiert ein nicht-triviales Zusammenspiel zwischen den Fehlerprüfverfahren der Gateways und denen des verwendeten IEs. Dieser Aspekt benötigt eine tiefer gehende Untersuchung.

Abkürzungsverzeichnis

ARQ	Automatische Wiederholanfrage (Automatic Repeat reQuest)
AT	Sercos III Acknowledge Telegram
AWGN	Additives weißes gaussches Rauschen (Additive White Gaussian Noise)
BE	Best Effort
BEP	Bitfehlerwahrscheinlichkeit (Bit Error Probability)
BER	Bitfehlerrate (Bit Error Rate)
BPSK	Zweiphasenumtastung (Binary Phase Shift Keying)
CDD	Code Divison Duplex
CDMA	Codemultiplex (Code Division Multiple Access)
CM	Zustandsüberwachung (Condition Monitoring)
CP	Zyklisches Präfix (Cyclic Prefix)
CRC	Cyclic Redundancy Check
CSMA/CD	Collision Sensing Multiple Access / Collision Detection
DL	Downlink
DLL	Sicherungsschicht (Data Link Layer)
DSSS	Direct Sequence Spread Spectrum
FA	Fabrikautomation
FDD	Frequency Division Duplex
FDMA	Frequenzmultiplex (Frequency Division Multiple Access)
FEC	Vorwärtsfehlerkorrektur (Forward Error Correction)
FN	Fusioniertes Netzwerk
GSP	Global Sampling Point
HN	Hybrides Netzwerk
IDN	Sercos III Identification Number
IE	Industrial Ethernet
IE-M	Industrial Ethernet Master

IE-S	Industrial Ethernet Slave
IEEE	Institute of Electrical and Electronics Engineers
IFS	Inter-Frame Space
IO	Input/Output
ISI	Inter-Symbol-Interferenz
LFSR	Lineare Schieberegister (Linear Feedback Shift Registers)
LOS	Sichtlinie (Line-of-Sight)
MAC	Medienzugriffsschicht (Media Access Control)
MDT	Sercos III Master Data Telegram
MLS	Folge maximaler Länge (Maximum Length Sequence) aka. m-Sequenz
NLOS	Abseits der Sichtlinie (Non-Line-of-Sight)
PA	Prozessautomation
PER	Paketfehlerrate (Packet Error Rate)
PGW	Primäres Gateway; das Gateway im PN
PHY	Physikalische Schicht (PHYsical layer)
PN	Pseudozufallsrauschen (Pseudo-random Noise)
PS-M	PSSS-Master
PS-S	PSSS-Slave
PSN	Primäres Subnetzwerk
PSSS	Parallel Sequence Spread Spectrum
RB	Ressourcenblock
RE	Ressourcenelement
RT	Echtzeit (Real Time)
SGW	Sekundäres Gateway; das Gateway in einem SN
SN	Subnetzwerk; bezeichnet einen IE im HN
SNR	Signal-zu-Rausch Verhältnis (Signal-to-Noise Ratio)
SPS	Speicherprogrammierbare Steuerung
SSN	Sekundäres Subnetzwerk
SVC	Sercos III Service Channel
TA	Zeitversatz (Timing Advance)
TCP	Transmission Control Protocol
TDMA	Zeitmultiplex (Time Division Multiple Access)

TP	Sendeleistung (Transmit Power)
TPC	Sendeleistungsanpassung (Transmit Power Control)
UC	Sercos III Unified Communication
UL	Uplink
WLAN	Wireless Local Area Network
WSAN	Wireless Sensor and Actuator Network

Symbolverzeichnis

n	Anzahl der Bit in der m-Sequenz und somit Anzahl der PSSS-Codes
m_i	i-tes Bit der m-Sequenz
$\mathbf{m}_{\rightarrow i}$	i-te zyklische Rotation der m-Sequenz
$\mathbf{E}$	Encodermatrix
$\mathbf{D}$	Decodermatrix
$\mathbf{d}$	Nutzdaten
$\mathbf{s}$	Gesendetes PSSS-Symbol
$\mathbf{s}'$	Empfangenes PSSS-Symbol
$\mathbf{c}$	Decodiertes PSSS-Symbol
$\mathbf{h}$	Kanalimpulsantwort
τ	Signallaufzeit
E_b	Bitenergie
N_0	Rauschleistungsdichte
T_{bit}	Bitdauer
T_{chip}	Chipdauer
T_{sym}	Symboldauer
T_{cyc}	Sercos-Zykluszeit
R_b	Bitrate
n_{TPC}	Anzahl der Bits eines TPC-Datums
n_{TA}	Anzahl der Bits eines TA-Datums
c_0	Lichtgeschwindigkeit
$d_{max/min}$	Maximale/minimale Länge des Ausbreitungspfads
r_i	Releasezeit des i-ten Datums
d_i	Deadline des i-ten Datums
p_i	Anzahl der Bits des i-ten Datums
p_{fec}	Anzahl der Redundanzbits in einem FEC-Block

p_{blk}	Anzahl der Nutzbits in einem FEC-Block
p_{crc}	Anzahl der Bits eines CRC-Codes
rx_i	Empfangszeitpunkt des i-ten Datums
tx_i	Sendezeitpunkt des i-ten Datums
$\mathcal{D}$	Früheste Deadline
s_i	Slack des i-ten Datums
$s[r]$	Slack der Releasezeit r
s^*	Minimaler Slack vor der betrachteten Releasezeit
$\mathbb{S}_1$	Menge der Daten im ersten Teilproblem
$\mathbb{S}_2$	Menge der Daten im zweiten Teilproblem

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] *ParSec - Ein paralleles (Par) zuverlässiges und sicheres (Sec) Funksystem zur latenzoptimierten Fabrikautomatisierung*. Adresse: `http://www.parsec-projekt.de/` (besucht am 30. 09. 2018).

[2] S. Dietrich, G. May, O. Wetter, H. Heeren und G. Fohler, „Performance indicators and use case analysis for wireless networks in factory automation“, in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, S. 1–8.

[3] *ZigBee Specification, Document 053474r20*, ZigBee Alliance, 2012.

[4] *Bluetooth specification version 4.2*, Bluetooth SIG, 2014.

[5] *WSAN Air Interface Specification Technical Specification, Version 1.0*, PNO, 2012.

[6] *IEC 62591:2016: Industrial communication networks - Wireless communication network and communication profiles - WirelessHART*, IEC, 2016.

[7] M. Wollschlaeger, T. Sauter und J. Jasperneite, „The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0“, *IEEE Industrial Electronics Magazine*, Jg. 11, Nr. 1, S. 17–27, 2017.

[8] G. Fettweis und S. Alamouti, „5G: Personal mobile internet beyond what cellular did to telephony“, *IEEE Communications Magazine*, Jg. 52, Nr. 2, S. 140–145, 2014.

[9] *IEEE Standard 802.3-2015 (Ethernet)*, IEEE, 2016.

[10] *sercos III - Communication Specification*, Version 1.3.1-1.12, Sercos International e.V., Dez. 2013. Adresse: `https://www.sercos.de/downloads/technische-spezifikationen/`.

[11] H. Schwetlick und A. Wolf, „PSSS - parallel sequence spread spectrum a physical layer for RF communication“, in *IEEE International Symposium on Consumer Electronics, 2004*, 2004, S. 262–265.

[12] A. Wolf, „Verfahren zum Übertragen eines Daten-Worts“, Patentnummer DE2003101250, 2004.

[13] *IEEE Standard 802.15.4-2011 (WPAN)*, IEEE, 2011.

[14] L. Underberg, R. Croonenbroeck, R. Kays und R. Kraemer, „ParSec: Wireless industrial communication first PSSS measurements in industrial environment“, in *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*, 2017, S. 1–8.

[15] L. Underberg, R. Kays, S. Dietrich und G. Fohler, „Towards hybrid wired-wireless networks in industrial applications“, in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, 2018, S. 768–773.

[16] R. Kraemer, M. Methfessel, R. Kays, L. Underberg und A. C. Wolf, „ParSec: A PSSS approach to industrial radio with very low and very flexible cycle timing“, in *2016 24th European Signal Processing Conference (EUSIPCO)*, 2016, S. 1222–1226.

[17] L. Underberg, A. Wulf, R. Croonenbroeck, W. Endemann und R. Kays, „Parallel Sequence Spread Spectrum: Analytical and simulative approach for determination of bit error probability“, in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, S. 1–8.

[18] L. Underberg, R. Croonenbroeck, A. Wulf, W. Endemann und R. Kays, „A PSSS Approach for Wireless Industrial Communication Applying Iterative Symbol Detection“, *IEEE Transactions on Industrial Informatics*, Jg. 14, Nr. 5, S. 2108–2119, 2018.

[19] J. von Hoyningen-Huene, A. Mueller, S. Dietrich und G. May, „Comparison of wireless gateway concepts for industrial real-time-communication“, in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, S. 1–4.

[20] *AD9361, RF Agile Transceiver*, Rev. F, Analog Devices, Inc., Nov. 2016. Adresse: `http://www.analog.com/en/products/ad9361.html` (besucht am 30. 09. 2018).

[21] Z. Fernandez, C. Cruces, I. Val und M. Mendicute, „Deterministic MAC access control scheme for industrial hybrid IEEE 802.3/IEEE 802.11 networks“, in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, S. 1–8.

[22] J. von Hoyningen-Huene und A. Müller, „Scheduling acyclic traffic in cyclic and deterministic communication systems“, in *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*, 2017, S. 1–9.

[23] R. E. Blahut, *Algebraic Codes for Data Transmission*. Cambridge University Press, 2003.

[24] M. L. Pinedo, *Scheduling, Theory, Algorithms, and Systems*, 5. Aufl. Springer International Publishing, 2016.

[25] G. Liva, L. Gaudio, T. Ninacs und T. Jerkovits, „Code Design for Short Blocks: A Survey“, 2016. arXiv: `1610.00873`.

[26] G. Liva und F. Steiner, *Channel Codes for Short Blocks: A Survey*, Workshop at the 11th International ITG Conference on Systems, Communications and Coding, Feb. 2017. Adresse: http://www.wirelesscoding.org/SCC_Tutorial.pdf (besucht am 30. 09. 2018).

[27] *Paloma Produktbescheibung*, Robert Bosch Packaging Technology GmbH. Adresse: https://www.boschpackaging.com/en/pa/products/industries/pd/product-detail/paloma-12225.php (besucht am 30. 09. 2018).

[28] *Systembeschreibung Drive & Diagnostic Link*, Referenz R499050030/2016-12, AVENTICS GmbH, Dez. 2016.

[29] *IndraDrive MPx-20 Funktionen*, Referenz DOK-INDRV*-MP*-20VRS**-AP01-DE-P, Bosch Rexroth AG, 2017.

[30] *PET-Füller Innofill PET DRV*, KHS GmbH. Adresse: https://www.khs.com/produkte/detail/pet-fueller-innofill-pet-drv/ (besucht am 30. 09. 2018).

[31] *Proline Promag H 100 PROFINET*, Endress+Hauser Messtechnik GmbH + Co. KG, 2015. Adresse: https://www.de.endress.com/de/messgeraete-fuer-die-prozesstechnik/durchflussmessung-produktuebersicht/Magnetisch-induktives-Durchflussmessger%C3%A4t-Promag-H100 (besucht am 30. 09. 2018).

Eidesstattliche Versicherung

Timmer, Calvin	157911
Name, Vorname	Matr.-Nr.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Masterarbeit mit dem Titel

Optimierte Ressourcenzuteilung für hybride kaskadierte Netze in der Industrieautomation

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Dortmund, 23. Oktober 2018	
Ort, Datum	Unterschrift

Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -)

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Dortmund, 23. Oktober 2018	
Ort, Datum	Unterschrift