UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

FAKULTÄT FÜR
ELEKTROTECHNIK,
INFORMATIK UND
MATHEMATIK

Adaptive Echtzeitkommunikationsnetze

Zur Erlangung des akademischen Grades

DOKTORINGENIEUR (Dr.-Ing.)

der Fakultät für Elektrotechnik, Informatik und Mathematik
der Universität Paderborn
vorgelegte Dissertation
von

Dipl.-Ing. Björn Griese
aus Soest

Referent:	Prof. Dr.-Ing. Ulrich Rückert
Korreferent:	Prof. Dr.-Ing. Joachim Böcker

Tag der mündlichen Prüfung: 17.12.2008

Paderborn, den 13.02.2009

Diss. EIM-E/246

Inhaltsverzeichnis

Kapitel 1

Einleitung

Heutige Automatisierungssysteme bestehen aus komplexen verteilten Rechensystemen mit bis zu hundert und mehr intelligenten Knoten, die über ausgeklügelte Echtzeitkommunikationsnetze miteinander verbunden sind. Eingesetzt werden Automatisierungssysteme in der Gebäudeautomatisierung, zur Steuerung und Regelung von industriellen Anlagen sowie zur Steuerung und Regelung in Fahrzeugen. Zu den vernetzten Einheiten zählen die Überwachung, die Regelung und die Steuerung von Systemzuständen und die Interaktion mit dem Benutzer. Durch die große Anzahl an Knoten und die hohen Anforderungen an die Echtzeitfähigkeit und die Zuverlässigkeit wird das Echtzeitkommunikationssystem zu einer entscheidenden Komponente, die die Leistungsfähigkeit, die Zuverlässigkeit und die Kosten des gesamten Automatisierungssystems maßgeblich beeinflusst.

Mit der Einführung der Feldbussysteme wurden erstmals Echtzeitkommunikationsnetze in Automatisierungssystemen eingesetzt. Die Feldbusse ersetzten die einzelnen Verbindungsleitungen, die sternförmig von der zentralen Steuereinheit zu jedem Sensor und Aktor gelegt werden mussten. Durch die Einführung der Feldbussysteme konnte dieser Verkabelungsaufwand und damit auch die Kosten erheblich gesenkt werden. Die limitierte Bandbreite der Feldbussysteme konnte aber der steigenden Anzahl von Knoten sowie der Forderung nach umfangreicheren Diagnosemöglichkeiten und einer Vernetzung vom Büro bis in die Anlage nicht mehr gerecht werden. Es wurden neuartige Echtzeitprotokolle entwickelt, die aufgrund der hohen Verfügbarkeit und der geringen Kosten zu einem Großteil auf Ethernet basieren. Da Ethernet bereits seit Jahrzehnten zur Vernetzung von Personal Computern (PC) in der Bürowelt verwendet wird, kann die bereits vorhandene Infrastruktur genutzt werden, um die Informationen aus der Anlage bis ins Büro weiterzuleiten.

Die mit der Anzahl der Knoten und den Echtzeitanforderungen gestiegene Komplexität eines Echtzeitkommunikationsnetzes erfordert eine zeitintensive und somit auch eine kostenintensive Planung. Aufwendige Planungswerkzeuge werden verwendet, um das Kommunikationssystem zu projektieren und um die auftretenden Latenzen und die benötigte Bandbreite zu evaluieren und zu optimieren. Mögliche Fehlerfälle und die dazu benötigte Redundanz müssen ebenso in der Planungsphase mit berücksichtigt werden, wie die Bandbreite, die für die Übertragung von Daten ohne Echtzeitanforde-

rungen notwendig ist. Durch die vorherige Planung der gesamten Kommunikation ist die Flexibilität dieser Systeme sehr stark eingeschränkt. Eine eigenständige Anpassung der Echtzeitkommunikation an Anforderungen, die sich während des Betriebs ändern, ist in diesem Fall nicht durchführbar.

Möglich wird eine größere Flexibilität mit adaptiven Echtzeitkommunikationsnetzen, die eine eigenständige Anpassung der Echtzeitkommunikation im Betrieb erlauben. Dadurch wird nicht nur der Planungsaufwand für die Kommunikation reduziert, sondern es werden verteilte selbstoptimierende Systeme möglich, die sich selbstständig an ihre Umgebung anpassen. Selbstoptimierende Systeme sind die Motivation für die in der vorliegenden Arbeit entwickelte adaptive Echtzeitkommunikation.

1.1 Selbstoptimierende Systeme

Ein selbstoptimierendes System ist in der Lage, sich eigenständig an sich verändernde Anforderungen von Benutzer und der Umwelt anzupassen. Im Sonderforschungsbereich 614 *Selbstoptimierende Systeme des Maschinenbaus* wird die Selbstoptimierung als das Zusammenwirken der drei folgenden wiederkehrenden Aktionen definiert [FGK+04]:

- Analyse der Ist-Situation
- Bestimmung der Systemziele
- Anpassung des Systemverhaltens

Ziel eines selbstoptimierenden Systems ist nicht nur die Adaption an die Anforderungen, sondern das System versucht sich der Situation entsprechend zu optimieren. Dies beinhaltet nicht nur eine Optimierung des Verhaltens, sondern auch die Optimierung der benötigten Ressourcen. Neben der Energie zählt zu den Ressourcen auch die Rechenleistung, die zur Steuerung und Regelung aller Applikationen benötigt wird. Dazu ist eine Informationsverarbeitung notwendig, die ihre Ressourcen den Anforderungen entsprechend zur Verfügung stellt. Damit die Informationsverarbeitung auch bei einem Ausfall von Teilkomponenten ihre Aufgaben noch erfüllen kann, sollte sie über redundante Ressourcen verfügen. Mithilfe einer verteilten Informationsverarbeitung, die über genügend Ressourcen verfügt, können Ausfälle von Teilen des Systems kompensiert werden. Eine verteilte Informationsverarbeitung besteht aus vielen einzelnen Rechenknoten, die über ein Echtzeitkommunikationsnetz miteinander verbunden sind.

In einem selbstoptimierenden System werden auf einer verteilten Informationsverarbeitung nur die Applikationen ausgeführt, die momentan benötigt werden. Um den benötigten Applikationen möglichst viel Rechenleistung zur Verfügung zu stellen, wird

auch die Verteilung der Applikationen auf der Informationsverarbeitung optimiert. Die Verteilung der Applikationen auf die vorhandenen Ressourcen wird im Sonderforschungsbereich 614 von dem Echtzeitbetriebssystem DREAMS übernommen, welches in [Dit99] beschrieben wird. Das Betriebssystem verwaltet auch die Anforderungen der Applikationen bezüglich des Ressourcenbedarfs und der Echtzeitkommunikation. Ändern sich die Anforderungen an das System, bedingt durch eine Veränderung der Umwelt, und wird aus diesem Grund eine neue Applikation ausgeführt, verändert sich auch die Kommunikation zwischen den Knoten. In einem Anwendungsbeispiel des Sonderforschungsbereichs 614 werden in einem autonomen Bahnshuttle unterschiedliche Regler für die Federung bei der Geradeausfahrt und bei der Kurvenfahrt verwendet. Fährt das Shuttle durch eine Kurve, wird der eine Regler von einem Knoten entfernt und der andere Regler auf einen anderen Knoten geladen, weil dieser z. B. mehr Ressourcen zur Verfügung stellt. Die Sensor- und Aktordaten werden in diesem Fall zu einem anderen Knoten gesendet. Die Kommunikation sowie die Anforderungen an die Kommunikation verändern sich auch, wenn z. B. ein Knoten ausfällt und dessen Aufgaben von anderen Knoten übernommen werden. Fällt z. B. der Knoten aus, auf dem die Regelung der Federung ausgeführt wird, so übernimmt ein anderer Knoten diese Aufgabe. Die Sensor- und Aktordaten müssen wieder zu einem anderen Knoten gesendet werden, der vorher gar nicht oder mit geringeren Echtzeitanforderungen kommuniziert hat. Im Kommunikationsnetz muss in diesem Fall eine neue Verbindung für den echtzeitfähigen Datenverkehr zwischen den Kommunikationspartnern aufgebaut werden. Durch eine neue Kommunikationsverbindung verändern sich nicht nur die Echtzeitanforderungen für die neue Übertragungsstrecke, sondern auch die Kommunikationslast. Ein Echtzeitkommunikationsnetz, bei dem die Kommunikation im Voraus geplant und festgelegt wurde, kann sich an solche dynamische Veränderungen nicht anpassen. Aus diesem Grund wird ein adaptives Echtzeitkommunikationsnetz benötigt, das auf sich ändernde Kommunikationsanforderungen reagiert und sich anpasst.

1.2 Zielsetzung dieser Arbeit

Zielsetzung der vorliegenden Arbeit ist die Entwicklung eines adaptiven Echtzeitkommunikationsnetzes, das sich an dynamische Veränderungen der Kommunikation und der Echtzeitanforderungen anpasst und das in der Lage ist, harte Echtzeitanforderungen zu garantieren. Im Wesentlichen sollen Methoden und Verfahren ausgearbeitet werden, die eine Adaption eines Echtzeitkommunikationsnetzes im Betrieb ermöglicht. Dabei ist zu berücksichtigen, dass eine Adaption innerhalb des Echtzeitkommunikationsnetzes nicht zum Verlust von Paketen oder zur Verletzung von Echtzeitanforderungen führt.

In der vorliegenden Arbeit wird die Adaption des Echtzeitkommunikationsnetzes auf der Ebene der Netzwerkkomponente und der Protokollebene untersucht.

Adaption der Netzwerkkomponente: Zielsetzung einer adaptiven Netzwerkkomponente ist es, den eigenen Ressourcenbedarf der Netzwerkkomponente an die Kommunikationsanforderungen anzupassen. Umgesetzt wurde die adaptive Netzwerkkomponente mithilfe eines rekonfigurierbaren Ethernet-Switches. Wenn die Kommunikationslast gering ist und der Switch nur wenige Pakete übertragen muss und gleichzeitig auch die Anforderungen bezüglich Jitter und Latenz gering sind, soll der Switch nur wenige Ressourcen benötigen. Diese eingesparten Ressourcen stehen in diesem Fall anderen Applikationen zur Verfügung. Eine andere Applikation könnte z. B. ein Optimierungsalgorithmus sein, der die Regelung der Federung des Shuttles optimiert und so den Fahrkomfort erhöht. Steigt die Kommunikationslast oder wird eine geringe Latenz und ein geringer Jitter benötigt, erhöht sich auch der Ressourcenbedarf des Switches. Die Ressourcen reichen nun für den Optimierungsalgorithmus nicht mehr aus und die Regelung für die Federung kann nicht mehr ganz so komfortabel arbeiten. Auf diese Weise verhält sich die adaptive Netzwerkkomponente wie alle Applikationen im selbstoptimierenden System. Sie fordert nur die Ressourcen an, die sie tatsächlich benötigt. Eines der wichtigsten Ziele bei der Adaption einer Netzwerkkomponente ist es, Methoden zu entwickeln, die eine Anpassung des Ressourcenbedarfs ermöglichen, ohne dabei einen Abriss des Paketstroms bzw. Verlust von Paketen zu verursachen.

Adaption auf Protokollebene: Ziel bei der Adaption auf Protokollebene ist die Entwicklung eines Echtzeitprotokolls, das harte Echtzeitanforderungen erfüllt, ohne die Kommunikation vollständig zu planen und festzulegen. Unter dieser Voraussetzung soll die Zykluszeit, in der ein Knoten seine Echtzeitdaten sendet, an die Anforderungen der Applikationen angepasst werden. Bei einer sehr großen Zykluszeit eines Knotens soll dieser die ihm zugeteilte Bandbreite anderen Knoten und damit anderen Applikationen zur Verfügung stellen. Die freigegebene Bandbreite kann z. B. von anderen Knoten zur Übertragung von nicht echtzeitrelevanten Daten verwendet werden. Benötigt der Knoten, z. B. aufgrund einer neuen Applikation, wieder eine geringere Zykluszeit, muss er die freigegebene Bandbreite wieder zurückerhalten, um die geringere Zykluszeit garantieren zu können. Bei der Adaption auf Protokollebene stellt sich nicht nur die Frage nach den Methoden, die für die Entwicklung eines adaptiven Echtzeitprotokolls notwendig sind, sondern es stellt sich auch die Frage, wie garantiert werden kann, dass die Adaption der Zykluszeit eines Knotens die Echtzeitanforderungen der anderen Knoten nicht beeinflusst. Nur auf diese Weise können bei einem adaptiven Echtzeitprotokoll harte Echtzeitanforderungen zu jedem Zeitpunkt garantiert werden.

1.3 Aufbau der Arbeit

Nach dieser Einleitung werden in Kapitel 2 die Grundlagen der Kommunikation und der Echtzeitkommunikation in Netzwerken beschrieben. Die Protokolle, die für eine Kommunikation oder eine Echtzeitkommunikation notwendig sind, werden in Schichten unterteilt, um die Komplexität der Kommunikation implementieren zu können. Einen Schwerpunkt setzt dieses Kapitel auf das Ethernet-Protokoll. Der Ethernet-Standard ist die Basis für die meisten aktuellen Echtzeitprotokolle, von denen einige in diesem Kapitel exemplarisch beschrieben und miteinander verglichen werden. Der Vergleich dieser Echtzeitprotokolle dient als Referenz für das in dieser Arbeit entwickelte adaptive Echtzeitprotokoll.

Bei der in Kapitel 3 beschriebenen adaptiven Netzwerkkomponente handelt es sich um einen rekonfigurierbaren Ethernet-Switch. Switches zählen zu den wichtigsten Netzwerkkomponenten bei den Ethernet-basierten Echtzeitprotokollen. Zur eigenständigen Anpassung seines Ressourcenbedarfs in Abhängigkeit der Anforderungen verwendet der rekonfigurierbare Ethernet-Switch die partielle dynamische Rekonfiguration eines FPGAs (Field-Programmable Gate Arrays). Die rekonfigurierbare Hardware des FPGAs erlaubt es dem Ethernet-Switch, seinen Ressourcenbedarf bezüglich der benötigten Hardware zu variieren. Nicht benötigte Hardwareressourcen des FPGAs stehen im Sinne der Selbstoptimierung anderen Applikationen zur Verfügung. Für den rekonfigurierbaren Ethernet-Switch wurden spezielle Rekonfigurationsstrategien entwickelt und analysiert, die die Rekonfiguration des Switches ohne einen Verlust von Paketen ermöglichen. Mithilfe einer prototypischen Implementierung des rekonfigurierbaren Ethernet-Switches wird die Leistungsfähigkeit des rekonfigurierbaren Switches bei geringem und bei hohem Ressourcenbedarf untersucht. Neben der Leistungsfähigkeit des rekonfigurierbaren Switches wird auch der Aufwand, den eine partielle dynamische Rekonfiguration eines Switches erfordert, diskutiert.

Das Kapitel 4 befasst sich mit der Adaption eines Echtzeitkommunikationsnetzes auf der Protokollebene. Dazu wird ein adaptives selbstsynchronisierendes Echtzeitprotokoll für eine virtuelle Ringtopologie vorgestellt, das weder einen zentralen Knoten noch eine explizite Uhrensynchronisation benötigt, um harte Echtzeitanforderungen zu garantieren. Dieses selbstsynchronisierende Echtzeitprotokoll trägt den Namen Self*S*. Die Adaption erfolgt beim Self*S*-Protokoll durch eine Variation der Zykluszeit von beliebigen Knoten zur Laufzeit. Die Einstellung der Zykluszeit kann für jeden Knoten unabhängig als ein beliebiges Vielfaches der Paketumlaufzeit gewählt werden. Das Protokoll garantiert die gewählten Zykluszeiten und ermöglicht eine gleichzeitige Übertragung von nicht echtzeitkritischen Daten. Auf der Basis eines theoretischen Modells wird die Performance des Self*S*-Protokolls analysiert und das deterministische Verhalten von Self*S* nachgewiesen. Im Detail erfolgt eine Untersuchung des verteilten

Selbstsynchronisierungsprozesses, der die Echtzeiteigenschaften von Self*S* maßgeblich beeinflusst. Das Protokoll und dessen theoretische Analysen sind unabhängig von der physikalischen Infrastruktur, basieren aber auf einer Switch-Technologie, die auch vom rekonfigurierbaren Switch verwendet wird.

Eine Portierung des Self*S*-Protokolls auf den Ethernet-Standard erfolgt in Kapitel 5. Die für die Portierung von Self*S* auf Ethernet notwendigen Anpassungen werden zunächst beschrieben, bevor die Leistungsfähigkeit des Ethernet-basierten Self*S*-Protokolls mithilfe einer rechnergestützten Netzwerksimulation evaluiert wird. Für die Netzwerksimulation wurde ein Simulationsmodell erstellt, dessen Netzwerkkomponenten auf der Grundlage des rekonfigurierbaren Ethernet-Switches modelliert wurden. Mithilfe der Netzwerksimulation können so der Einfluss von Ethernet und der Einfluss realer Netzwerkkomponenten untersucht werden. Schwerpunkt der Analysen bilden neben der Selbstsynchronisation die Echtzeiteigenschaften des Protokolls bezüglich der minimalen Zykluszeit, der Latenzzeit und des Jitters. Die Korrektheit der simulierten Ergebnisse wird exemplarisch mithilfe einer an das Self*S*-Protokoll angepassten prototypischen Implementierung des rekonfigurierbaren Ethernet-Switches bestätigt. Zusätzlich werden die Echtzeiteigenschaften bzw. die Leistungsfähigkeit des Self*S*-Protokolls den Ergebnissen des Vergleichs der Ethernet-basierten Echtzeitprotokolle aus dem Grundlagenkapitel gegenübergestellt. Letztendlich werden in diesem Kapitel noch einige Erweiterungen von Self*S* diskutiert, die die Variabilität der Zykluszeit und die Fehlertoleranz erhöhen.

Die wesentlichen Ergebnisse der vorliegenden Arbeit werden in Kapitel 6 abschließend zusammengefasst und bewertet. Zusätzlich erfolgt ein Ausblick auf mögliche Erweiterungen des adaptiven Echtzeitkommunikationsnetzes.

Kapitel 2

Echtzeitkommunikationsnetze

Im Internet und bei der Kommunikation im Bürobereich soll über das Kommunikationsnetz ein möglichst großer Durchsatz erzielt werden. Mit Durchsatz ist die Datenmenge gemeint, die pro Zeiteinheit über das Netzwerk übertragen wird. Die Zeit, die ein Paket für die Übertragung von der Quelle bis zum Ziel benötigt, sollte zwar möglichst kurz sein, sie kann aber bei einer wiederholten Übertragung von derselben Quelle bis zum selben Ziel sehr stark schwanken. Im Gegensatz dazu wird in einem Echtzeitkommunikationsnetz die Zeit, in der ein Paket sein Ziel erreicht, garantiert. In diesem Kapitel erfolgt eine Einführung in die Echtzeitkommunikation in einem Netzwerk.

Nach einer kurzen Erläuterung der Echtzeit in Kapitel 2.1 sowie der in diesem Bereich verwendeten Begriffe folgt in Kapitel 2.2 eine Einführung in die netzwerkbasierte Kommunikation. Die in diesem Abschnitt erläuterten Verfahren bilden nicht nur die Grundlage für die Kommunikation im Internet und im Bürobereich, sondern sie sind auch die Basis, auf der die Echtzeitkommunikation aufbaut. Nach einer kurzen Erörterung von allgemeinen Methoden der Echtzeitkommunikation in Kapitel 2.3 folgt in Kapitel 2.3.3 eine Beschreibung der Feldbusse. In der Geschichte der Entwicklung der Echtzeitkommunikation sind die Feldbusse die ersten Echtzeitkommunikationsnetze, die im industriellen Bereich eingesetzt werden. Höhere Echtzeitanforderungen und eine gestiegene Anzahl von Knoten führten zur Entwicklung einer neuen Generation von Echtzeitkommunikationsnetzen, die auf dem im Bürobereich verwendeten nicht echtzeitfähigem Ethernet-Standard basieren. Kapitel 2.4 beschreibt die Verfahren, die Ethernet vom Kommunikationsnetz im Büro zum Echtzeitkommunikationsnetz erweitern. Exemplarisch werden in Kapitel 2.5 einige Ethernet-basierte Echtzeitprotokolle vorgestellt, die für den industriellen Einsatz standardisiert wurden. Schließlich wird die Leistungsfähigkeit dieser Protokolle in Kapitel 2.6 bewertet und verglichen.

2.1 Echtzeit

Der Begriff *Echtzeit* (engl.: Real-Time (RT)) wird verwendet, um zum Ausdruck zu bringen, dass ein Ergebnis innerhalb einer definierten Zeit vorliegt. Im Deutschen wird neben der Echtzeit auch der Begriff *Realzeit* verwendet, um den Bezug zum Englischen

hervorzuheben. Trotzdem findet der Begriff *Echtzeit* eine größere Verbreitung und wird auch hier verwendet. Häufig suggeriert die Echtzeit, dass ein System seine Ergebnisse besonders schnell generiert. Genau das Gegenteil ist der Fall. Die Mechanismen, die die Einhaltung von Zeitschranken garantieren, kosten zusätzliche Ressourcen und verringern die Effizienz des Systems. Deutlich werden die Eigenschaften der Echtzeit auch bei der folgenden Beschreibung der Korrektheit eines Echtzeitsystem:

> *The correctness of the system depends on both the logical results and the time at which those results appear.* [MZ95]

Anders ausgedrückt ist nach diesem Zitat ein Echtzeitsystem korrekt, wenn seine Ergebnisse innerhalb einer definierten Zeit bzw. innerhalb einer Zeitschranke generiert werden. Generell trifft diese Definition auf jedes System zu, wenn die Zeit nur groß genug gewählt wird [KS97]. Ein Geldautomat z. B., der garantiert innerhalb von zwei Stunden mit der Auszahlung beginnt, wird den Benutzer nicht zufriedenstellen, ist aber nach der Definition ein korrektes Echtzeitsystem. Allerdings werden Zeitschranken normalerweise von der Umgebung vorgegeben, in der sich das Echtzeitsystem befindet. Bei einem Computerspiel muss z. B. 25mal in der Sekunde ein neues Bild berechnet und angezeigt werden, damit das menschliche Gehirn alle Bewegungen als natürlich und fließend erkennt. Bilder, die zu spät berechnet werden, führen zu einem Ruckeln der Bewegung. In einem kritischen System, wie z. B. in einem Bremssystem im Auto, kann ein zu spät generiertes Ergebnis viel schwerwiegendere Folgen haben als nur ein Ruckeln der Bewegung. Daher wird in Abhängigkeit des Nutzens, der beim Überschreiten einer Echtzeitschranke eintritt, die Echtzeit häufig in hart und weich unterteilt:

- **Harte Echtzeit:** Ein Überschreiten der Zeitschranke ist kritisch und der Nutzen für das System verschwindet oder ist sogar negativ.
- **Weiche Echtzeit:** Ein gelegentliches Überschreiten der Zeitschranke beeinflusst nicht das korrekte Verhalten eines Systems. Der Nutzen für das System verringert sich mit steigender Verspätung.

Dargestellt ist eine abstrakte Nutzenfunktion für harte und weiche Echtzeit in Abbildung 2.1. Sehr gut zu erkennen ist die Verschlechterung des Nutzens bei steigender Verspätung, der auch bei weicher Echtzeit negativ werden kann. Typische Beispiele für weiche Echtzeit sind Multimediaanwendungen, wie z. B. Internetfernsehen und Voice-over-IP. Beispiele für harte Echtzeitanwendungen sind die Steuerung von Industrierobotern in einer Fertigungsstraße oder digitale Regelungen in kritischen Systemen, z. B. im Auto und Flugzeug. Eine dritte Echtzeitkategorie wird bei Burns und Willings als *firm* oder als *fest* bezeichnet [BW01]. Im Gegensatz zu der harten Echtzeit

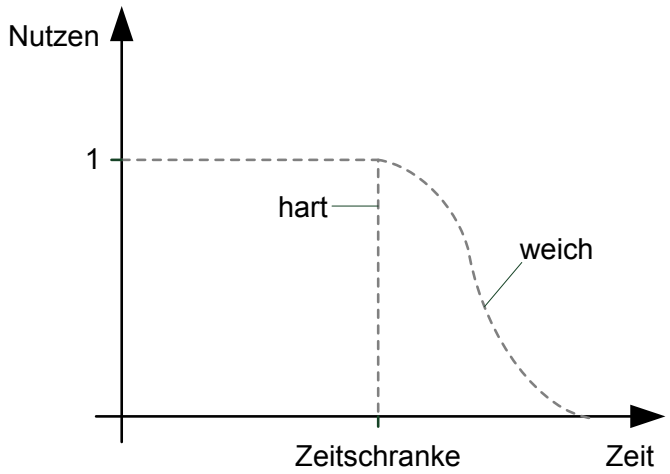


Abbildung 2.1: Nutzenfunktion bei harter und weicher Echtzeit

kann hierbei die Zeitschranke gelegentlich überschritten werden, ohne das Systemverhalten zu stören. Das Ergebnis hat aber auch in diesem Fall keinen Nutzen mehr. Ein Beispiel für ein festes Echtzeitsystem sind robuste Regler, die z. B. bei [Cha96] beschrieben werden. Gelegentlich verspätete Daten werden von einem robusten Regler toleriert, aber nicht mehr ausgewertet. Im weiteren Verlauf wird nur zwischen weicher und harter Echtzeit unterschieden und *firm* als Teil der weichen Echtzeit gewertet.

Des Weiteren wird bei harten und weichen Echtzeitsystemen die Art ihrer Validierung unterschieden [Liu00]. Für ein weiches Echtzeitsystem ist es ausreichend zu zeigen, dass das System statistisch seine Anforderungen erfüllt, indem z. B. die durchschnittliche Anzahl an überschrittenen Zeitschranken in der Minute ermittelt wird. Bei einem harten Echtzeitsystem muss durch einen korrekten Beweis oder durch intensives Simulieren und Testen nachgewiesen werden, dass das System immer seine Zeitschranken einhält.

2.1.1 Begriffserläuterungen

Einige Begriffe, wie z. B. Zeitschranke und weiche und harte Echtzeit, sind bereits im letzten Abschnitt beschrieben worden. In diesem Abschnitt werden weitere Begriffe aus dem Echtzeitumfeld eingeführt. Einige Begriffe werden in ähnlicher Form auch bei Buttazzo beschrieben [But04].

- **Echtzeitanforderungen** sind Anforderungen, die in Abhängigkeit der Anwendung die vom System einzuhaltenden Zeitschranken und deren Genauigkeit definieren. Echtzeitanforderungen definieren z. B. die Ausführungszeit, die Zykluszeit, die Latenzzeit und den Jitter, die vom System eingehalten werden müssen. Als Synonym wird in dieser Arbeit auch der Begriff *Echtzeitkriterien* verwendet.

- **Die Ankunftszeit** beschreibt den Zeitpunkt, an dem das System von der Anwendung aufgefordert wird, eine Aufgabe zu bearbeiten. In einem periodischen System wiederholt sich die Ankunftszeit mit der Zykluszeit.

- **Die Zykluszeit** ist die konstante Zeitperiode T_{Cyc}, in der das System die Anforderung zur Bearbeitung einer identischen Aufgabe erhält. In einem Kommunikationssystem besteht die Aufgabe darin, Pakete periodisch zu übertragen.

- **Die Ausführungszeit** ist die Zeit, die das System benötigt, um das Ergebnis zu generieren.

- **Die Latenzzeit** T_{Lat} ist die Verzögerung von der Ankunftszeit einer Aufgabe bis zu ihrer vollständigen Ausführung. Sie wird auch als *Antwortzeit* des Systems bezeichnet. In dieser Arbeit ist unter Latenz die Verzögerung bei der Übertragung von Daten von der Quelle bis zum Ziel zu verstehen.

- **Der Jitter** ist die Schwankung eines periodisch auftretenden Ereignisses. In der Kommunikation ist ein solches Ereignis z. B. der periodische Empfang oder das periodische Senden eines Paketes. In dieser Arbeit wird der Jitter des vorgestellten Kommunikationssystems näher analysiert und bedarf daher einer genaueren Definition. Im Allgemeinen wird bei der Definition eines Jitters zwischen periodischem Jitter, *Cycle-to-Cyle*-Jitter und akkumuliertem Jitter unterschieden [PL04, IDT07].

  Der periodische Jitter wird definiert als

$$J_{\mathrm{Per}} = T_k - T_{\mathrm{Cyc}}, \tag{2.1}$$

  wobei T_k die tatsächliche Zykluszeit nach dem k-ten Auftreten des Ereignisses ist und T_{Cyc} der erwarteten Zykluszeit entspricht.

  Der Cycle-to-Cyle-Jitter ist die maximale Abweichung zwischen zwei aufeinanderfolgenden tatsächlichen Zyklen über einen Zeitraum von K Zyklen und wird definiert als

$$J_{\mathrm{Cyc}} = \max(T_k - T_{k+1}) \qquad k \in \{1, \cdots, K-1\}\,. \tag{2.2}$$

  Der akkumulierte Jitter wird definiert als

$$J_{\mathrm{Akk}} = t_k - k \cdot T_{\mathrm{Cyc}}, \tag{2.3}$$

wobei t_k die Ankunftszeit des k-ten Ereignisses ist.

Messungen des Jitters, z. B. mit einem Oszilloskop, resultieren meistens in dem periodischen Jitter. Die Analysen in dieser Arbeit beziehen sich auf den periodischen Jitter, der im weiteren Verlauf mit J symbolisiert wird. Sowohl bei den Messungen in Kapitel 3.3.2 als auch bei den Simulationen in Kapitel 5.4 ist die erwartete Zykluszeit unbekannt. Stattdessen wird der arithmetische Mittelwert der Zykluszeit $\overline{T}_{\mathrm{Cyc}}$ zur Berechnung des Jitters verwendet und aus der Gleichung (2.1) folgt:

$$J_k = T_k - \overline{T}_{\mathrm{Cyc}} \tag{2.4}$$

Des Weiteren wird bei der Angabe eines Jitters über eine Menge von Ereignissen K zwischen durchschnittlichem Jitter und maximalen Jitter unterschieden. Der durchschnittliche Jitter ist die mittlere Abweichung von $\overline{T}_{\mathrm{Cyc}}$ und wird bestimmt durch

$$\overline{J} = \frac{\sum_{k=1}^{K} |J_k|}{K}. \tag{2.5}$$

Der maximale Jitter ist die maximale Differenz zwischen zwei tatsächlichen Zykluszeiten T_k und T_j und wird bestimmt durch

$$J_{\mathrm{Max}} = \max |T_k - T_j| \qquad k,\, j \in \{1, \cdots, K\},\, k \neq j. \tag{2.6}$$

- **Die Zeitschranke**, im englischen *Deadline*, beschreibt den Zeitpunkt, bis zu dem das Ergebnis nach der Ankunft der Aufgabe vorliegen muss.

- **Echtzeitfähig** ist ein System, wenn es alle von der Anwendung gestellten Echtzeitanforderungen erfüllt.

Einige der oben erklärten Begriffe werden in der Abbildung 2.2 am Beispiel einer periodischen Aufgabe illustriert. Bei einer aperiodischen Echtzeitaufgabe, wie z. B. die Reaktion eines Systems auf das Betätigen eines Notausschalters, wird die Ankunftszeit nicht durch die Zykluszeit, sondern durch das Auftreten des Ereignisses bestimmt.

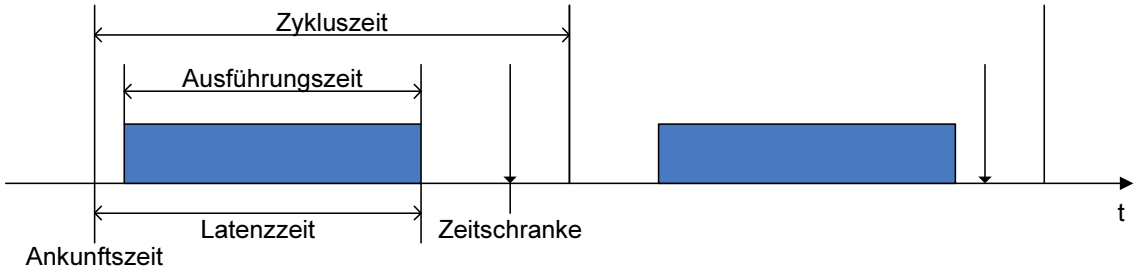


Abbildung 2.2: Periodische Aufgaben in einem Echtzeitsystem (vgl. [But04])

2.2 Kommunikationsnetze

Der schnelle weltweite Austausch von Informationen ist eine der wichtigsten Errungenschaften der letzten Jahrzehnte und ist aus unserem Alltag nicht mehr wegzudenken. Ob beim Telefonieren, beim Versenden einer E-Mail oder beim Surfen im Internet, alle Informationen bzw. Daten werden über Kommunikationsnetze ausgetauscht. Sie sind die Infrastruktur, die Computer und andere technische Systeme miteinander verbinden. Ihre Aufgabe ist es, den Transfer von Informationen zu steuern und den Empfang der Informationen an ihrem Ziel zu gewährleisten. Die Übertragung in einem Kommunikationsnetz erfolgt sowohl über drahtgebundene Medien, wie Kupfer- und Glasfaserkabel, als auch über drahtlose Medien, z. B. über elektromagnetische Wellen. Die Verarbeitung der Informationen von der Anwendung bis zur Übertragung über das physikalische Medium wird in einem Schichtenmodell beschrieben, welches nach einer Erläuterung der wichtigsten Begriffe beschrieben wird.

2.2.1 Begriffserläuterungen

In diesem Abschnitt werden Begriffe aus dem Bereich der Kommunikationsnetze erläutert, mit denen die Leistungsfähigkeit der Kommunikation beschrieben wird.

- **Die Datenrate** oder Übertragungsrate r beschreibt die Anzahl der Bits, die in einer Sekunde von einer Netzwerkkomponente übertragen werden können.
- **Die Bitzeit** ist die Zeit, die eine Netzwerkkomponente benötigt, um ein Bit bei einer bestimmten Datenrate zu übertragen.
- **Die Ausbreitungsgeschwindigkeit** ist die Geschwindigkeit ν, mit der sich ein Signal auf dem Medium ausbreitet. Die Ausbreitungsgeschwindigkeit errechnet sich aus Lichtgeschwindigkeit c und dem Ausbreitungskoeffizienten γ. In der Tabelle 2.1 sind als Beispiel die im Ethernet-Standard [IEE05] genannten Ausbreitungskoeffizienten für verschiedene Ethernet-Medien aufgelistet.

Ethernet-Medium	Ausbreitungskoeffizient γ
10BASE-5 (Koax)	0,77
100BASE-T (Twistet Pair Cat 5)	0,60
100Base-FX (Glasfaser)	0,67

Tabelle 2.1: Ausbreitungskoeffizienten bei Ethernetmedien

$$\nu = \gamma \cdot c \tag{2.7}$$

- **Die Übertragungsverzögerung** ist die Zeit T_{Trn}, die ein Sender für die Übertragung eines Paketes der Länge l bei einer Datenrate r benötigt.

$$T_{\text{Trn}} = \frac{l}{r} \tag{2.8}$$

- **Die Ausbreitungsverzögerung** ist die Zeit T_{Prd}, die ein Bit für den Transport über eine Leitung der Länge d benötigt.

$$T_{\text{Prd}} = \frac{d}{\nu} \tag{2.9}$$

- **Die Übertragungszeit** ist die Zeit T_{Trt}, die ein Paket von Beginn der Übertragung über eine Verbindung bis zum vollständigen Empfang benötigt.

$$T_{\text{Trt}} = T_{\text{Prd}} + T_{\text{Trn}} \tag{2.10}$$

2.2.2 ISO/OSI-Referenzmodell

Das ISO/OSI-Referenzmodell [ISO94] (OSI - *Open Systems Interconnection*) wurde von der *International Standards Organisation* (ISO) zur Standardisierung von verschiedenen Protokollen erarbeitet. Abbildung 2.3 zeigt die sieben Schichten des Referenzmodells, in denen Übertragungsdienste nach ihrem Abstraktionsgrad aufgeteilt sind. Die Schichten stellen der nächsthöheren Schicht festgelegte Dienste zur Verfügung und kommunizieren miteinander über definierte Schnittstellen. Auf der gleichen Schichtebene wird zwischen zwei Teilnehmern über ein Protokoll kommuniziert. Die Protokolle definieren das Format und die Bedeutung der ausgetauschten Rahmen.

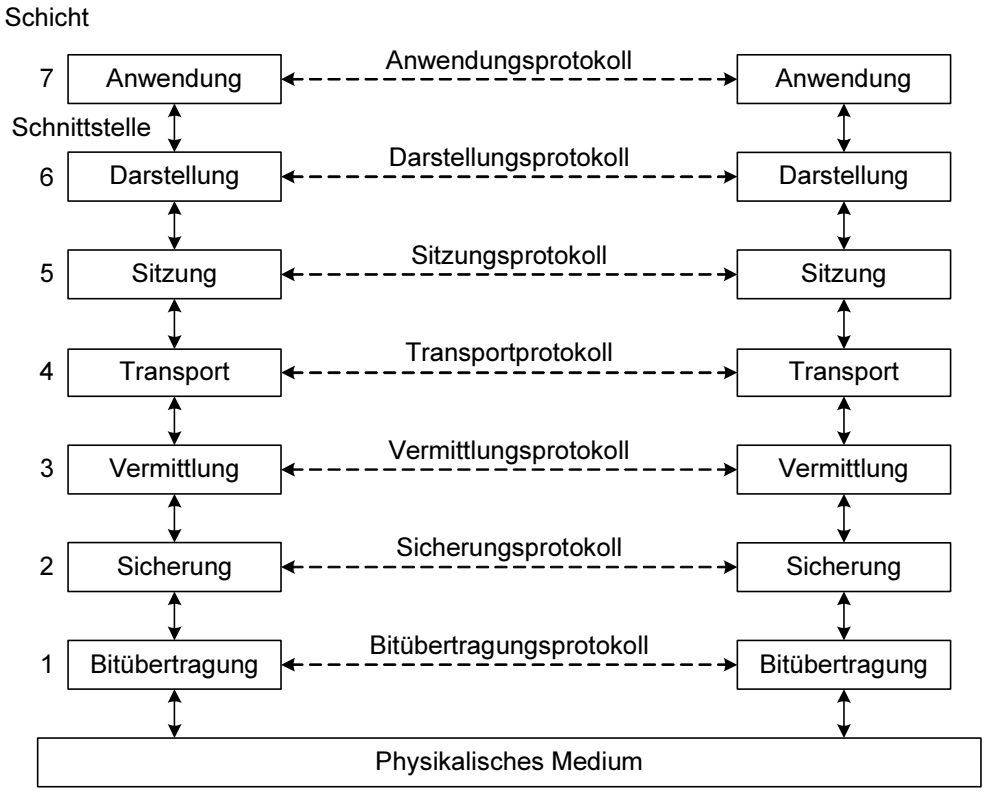


Abbildung 2.3: Das ISO/OSI-Referenzmodell [Tan03]

Durch die Aufteilung in Dienste, Schnittstellen und Protokolle können einzelne Schichten ausgetauscht werden, ohne dass eine Anpassung der anderen Schichten notwendig ist. Im Folgenden werden die einzelnen Schichten von unten nach oben kurz beschrieben [Tan03]:

- **Die Bitübertragungsschicht** überträgt einzelne Bits über ein vorhandenes Medium und spezifiziert die elektrischen und mechanischen Eigenschaften (Spannungen, Kabeltypen, Stecher usw.)

- **Die Sicherungsschicht** wird in die beiden Teilschichten *Logical Link Control* (LLC) und *Medium Access Control* (MAC) unterteilt. Die LLC-Teilschicht bettet die Eingangsdaten in einen Rahmen ein und gewährleistet durch Fehlererkennung und Behebung eine fehlerfreie Übertragung über die Leitung. Bei *Broadcast*-Netzen wird über die MAC-Teilschicht der Medienzugriff geregelt.

- **Die Vermittlungsschicht** ermöglicht die logische Verbindung zwischen Endsystemen in unterschiedlichen Netzwerksegmenten. Ihre wichtigste Aufgabe ist die Suche von Paketrouten.

- **Die Transportschicht** trennt die unteren transportorientierten Schichten von den oberen anwendungsorientierten Schichten. Die zugrunde liegende Netztechnologie ist für die Transportschicht transparent. Die Transportschicht hat einen Ende-zu-Ende-Charakter und verbindet Anwendungsprozesse miteinander.

- **Die Sitzungsschicht** steuert und regelt über Sendeberichtigungen die Prozess-zu-Prozess-Kommunikation.

- **Die Darstellungsschicht** regelt die Syntax und Semantik der zu übertragenden Informationen, um ein gegenseitiges Verstehen der Kommunikationspartner zu ermöglichen.

- **Die Anwendungsschicht** ist die Schnittstelle zum Anwendungsprozess und bietet den Dienst an, über den die Anwendungsprozesse kommunizieren. Ein Beispiel für ein Anwendungsprotokoll ist das *File Transfer Protocol* (FTP). Über dessen Schnittstelle können Anwendungen verschiedener Hersteller Dateien austauschen.

Nicht alle Schichten sind bei den implementierten Kommunikationsarchitekturen berücksichtigt worden. Die Internetprotokolle kommen ohne die Schichten 5 und 6 aus und die meisten Echtzeitkommunikationsprotokolle setzen direkt auf der Sicherungsschicht auf. Besondere Bedeutung bei der Echtzeitkommunikation haben die Medienzugriffsverfahren, die im nächsten Abschnitt beschrieben werden.

2.2.3 Medienzugriffsverfahren

Eingeteilt werden die Medienzugriffsverfahren in die zwei Kategorien *zufällig* und *deterministisch*. Für die Echtzeitkommunikation spielt diese Einteilung eine besondere Rolle. Bei den zufälligen Verfahren erfolgt der Zugriff durch einen Teilnehmer völlig wahlfrei. Teilen sich mehrere Teilnehmer das Medium, führt dies zwangsläufig zu Kollisionen. Der Rahmen bzw. das Paket ist nicht mehr lesbar und muss erneut übertragen werden. Die Dauer einer erfolgreichen Paketübertragung kann daher nicht absolut bestimmt werden. In Abhängigkeit von der Auslastung des Netzwerkes können nur Durchschnittswerte für die Übertragungsdauer angeben werden. Harte Echtzeitanforderungen können in diesem Fall nicht garantiert werden. Daher darf ein zufälliges Medienzugriffsverfahren nur in Kommunikationsnetzen mit weichen Echtzeitanforderungen eingesetzt werden.

Ein sehr weit verbreitetes zufälliges Zugriffsverfahren ist das CSMA/CD-Verfahren (*Carrier Sense Multiple Access with Collision Detection*), welches unter IEEE 802.3 [IEE05] den Standard für Ethernet bildet. Alle CSMA-Verfahren haben gemeinsam,

dass sie zunächst den Kanal abhören und erst senden, wenn der Kanal frei ist. Zusätzlich wird beim CSMA/CD eine Kollision von den Teilnehmern erkannt und nach dem Senden eines Störsignals brechen die Teilnehmer ihre Übertragung ab. Ein erneuter Sendeversuch erfolgt nach dem *Binary Exponential Backhoff (BEB) Algorithmus*. Damit die Wahrscheinlichkeit für eine erneute Kollision beim nächsten Sendeversuch verringert wird, gibt der Algorithmus ein Intervall vor, in dem die Wartezeit bis zum erneuten Senden zufällig bestimmt wird. Nach jeder Kollision wird das Intervall vergrößert und somit die Wahrscheinlichkeit verringert, dass das Paket beim nächsten Sendeversuch erneut kollidiert. Die Größe des Intervalls beim BEB ist also abhängig von der Anzahl der Kollisionen i und liegt zwischen 0 und $2^i - 1$ Zeitschlitzen (ein Zeitschlitz bei Ethernet mit einer Datenrate bis zu 100 MBit/s entspricht 512 Bitzeiten). Nach zehn Kollisionen wird die Intervallgröße nicht mehr weiter erhöht. Letztendlich verworfen wird das Paket nach 16 Kollisionen. Durch den BEB-Algorithmus sind die Wartezeiten nach einer Kollision nicht vorhersagbar. Daher ist das CSMA/CD für eine harte Echtzeitkommunikation nicht geeignet. In Kapitel 2.4 wird das nicht deterministische Verhalten von Ethernet näher beschrieben.

Im Gegensatz zu den zufälligen Verfahren sind die deterministischen Zugriffsverfahren kollisionsfrei. Der Zugriff auf das Medium erfolgt koordiniert, das heißt, die Teilnehmer untereinander oder eine zentrale Instanz legt fest, welcher Teilnehmer als Nächster den Zugriff auf das Medium erhält. Beim *Master-Slave*-Verfahren werden die Teilnehmer (*Slaves*) von einer zentralen Einheit (*Master*) nacheinander aufgefordert, zu senden. Diese Art der Kommunikation wird auch *Polling* genannt. Zu den verteilten Zugriffsverfahren zählt das kollisionsfreie CSMA-Verfahren, namentlich CSMA/CA (CA - Collision Avoidance), welches im Feldbusbereich z. B. bei CAN (Controller Area Network) eingesetzt wird. Über im Vorfeld zugewiesene Prioritäten erfolgt eine Arbitrierung des Bussystems. Bei der Arbitrierung erhält der Sender mit der höchsten Priorität den Bus, ohne dass sein Telegramm während der Arbitrierung beschädigt wird. Ein ebenfalls in der Echtzeitkommunikation verbreitetes Zugriffsverfahren ist das *Time Division Multiple Access* (TDMA)-Verfahren. Den Teilnehmern werden beim TDMA im Vorfeld Zeitschlitze zugeteilt, in denen sie senden dürfen. TDMA erfordert eine Uhrensynchronisation aller Teilnehmer, die über ein Synchronisationsprotokoll realisiert wird. In Kapitel 2.4.4 wird die Uhrensynchronisation am Beispiel des IEEE-Standards 1588 erläutert. Ein weiteres verteiltes Medienzugriffsverfahren ist das *Token*-Verfahren. Der *Token* ist wie eine digitale Münze, die von Teilnehmer zu Teilnehmer weiter gereicht wird. Nur der Teilnehmer, der gerade den Token besitzt, darf sein Paket senden. Nachdem das Paket übertragen wurde, muss der Token weitergegeben werden.

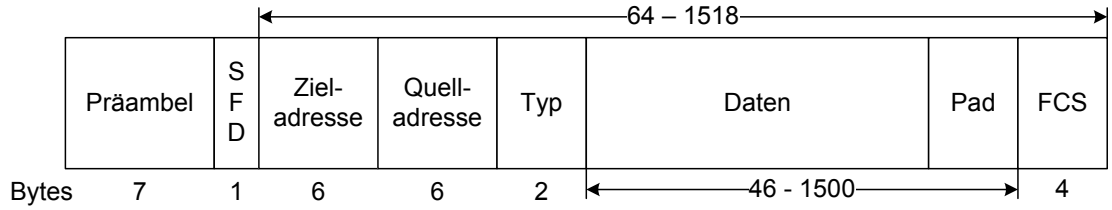
Bei allen deterministischen Zugriffsverfahren kann im Gegensatz zu den zufälligen Verfahren für Pakete mit beschränkter Länge eine obere Grenze garantiert werden, zu der ein Teilnehmer die Übertragung seines Paketes beendet hat. Aus diesem Grund werden

hauptsächlich deterministische Medienzugriffsverfahren in der Echtzeitkommunikation eingesetzt.

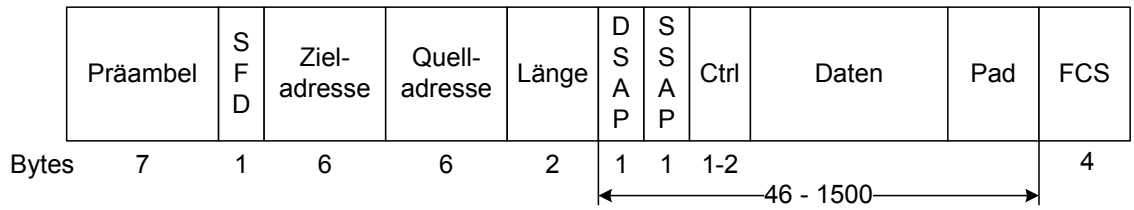
2.2.4 Netzwerktopologien

Die Topologie eines Netzwerkes beschreibt die Anordnung der Teilnehmer und ihrer Verbindungen. Dargestellt werden kann eine Topologie als Graph $G = (V,E)$ mit V als die Menge der Knoten und E als die Menge der Kanten. Die einzelnen Topologien werden unterteilt in Broadcast- und Punkt-zu-Punkt-Netzwerke. Bei einem Broadcast-Netz teilen sich mehrere Teilnehmer ein Medium, deren Zugriff über die im vorherigen Abschnitt beschriebenen Verfahren geregelt wird. Der größte Vorteil von Broadcast-Netzwerken ist, dass alle Teilnehmer ein gesendetes Paket empfangen. Es ist nicht notwendig, ein Paket mehrfach zu übertragen, um mehrere Teilnehmer zu erreichen. Die typische Topologie eines Broadcast-Netzwerkes ist der Bus. Die Teilnehmer an einem Bus sind alle in einer Linie angeordnet und mit dem Bus verbunden. Die Bustopologie ist bei den Echtzeit-Kommunikationssystemen und insbesondere bei den Feldbussen weit verbreitet (vgl. Kapitel 2.3.3).

Pakete in einem Punkt-zu-Punk-Netzwerk werden nur zwischen zwei Teilnehmern ausgetauscht. Ein Medienzugriffsverfahren wird in diesem Fall nicht benötigt und mehrere Teilnehmer können gleichzeitig senden, ohne dass Kollisionen entstehen. Typische Topologien sind der Stern und das vollvermaschte Netz. Im Stern werden alle Teilnehmer mit einer zentralen Einheit verbunden, über die die Daten vom Sender zum Empfänger geleitet werden. Bei Ethernet z. B. ist diese zentrale Einheit ein Switch. Der Nachteil der Stern-Topologie ist, dass der Ausfall der zentralen Einheit zum Ausfall des gesamten Netzwerks führt. Beim vollvermaschten Netz ist jeder Teilnehmer mit jedem anderen Teilnehmer im Netzwerk verbunden. Die vielen Verbindungen ermöglichen bei einem Ausfall, dass über redundante Routen alle Teilnehmer weiterhin erreicht werden können. Allerdings führen die vielen Verbindungen zu einem sehr hohen Verkabelungsaufwand. Für ein vollvermaschtes Netz mit n Knoten werden $n/2 \cdot (n-1)$ Verbindungen benötigt. Auch bei Punkt-zu-Punkt-Verbindungen können die Teilnehmer in einer Linie angeordnet sein. In diesem Fall wird die Netzanordnung nicht als Bus sondern als Linie bezeichnet. Werden der erste und letzte Teilnehmer zusätzlich miteinander verbunden, so ergibt sich eine Ring-Topologie. Die Ring-Topologie besitzt eine inhärente Redundanz, die auch bei dem Ausfall einer Punkt-zu-Punkt-Verbindung die Kommunikation zwischen allen Teilnehmern ermöglicht. Eine weitere Topologie ist der Baum. Dabei handelt es sich um einen Zusammenschluss mehrerer Linien oder Sterne. Diese Topologie wird heute bei der Vernetzung von Personal Computern (PC) im Bürobereich überwiegend eingesetzt.



(a) DIX-Standard



(b) IEEE 802.3-Standard

Abbildung 2.4: Ethernet Paketformat

2.2.5 Ethernet

Ethernet wurde für die Vernetzung der ersten PCs entwickelt und gilt als das erste lokale Netzwerk (LAN - Local Area Netzwork). Entworfen wurde es 1973 am Palo Alto Research Center (PARC) der Firma Xerox von Robert M. Metcalfe und David R. Boggs [MB76]. Bei der ersten Version von Ethernet wurden die Daten mit einer Geschwindigkeit von 2,94 MBit/s über ein Koaxialkabel übertragen. Ethernet war so erfolgreich, dass sich DEC und Intel Xerox anschlossen und 1980 den DIX-Standard, den ersten 10 MBit/s-Ethernet-Standard, veröffentlichten. Seit dem Jahr 1983 hat die IEEE (Institut of Electrical and Electronics Engineers) die Standardisierung von Ethernet unter der Bezeichnung IEEE 802.3 übernommen. Unter der Leitung der IEEE wird Ethernet seitdem kontinuierlich weiter entwickelt. Im Jahre 1995 folgte der Standard für das *Fast Ethernet* mit einer Übertragungsrate bis zu 100 MBit/s und 1998 wurde der Standard für die 1000 MBit/s-Variante veröffentlicht. Im Jahre 2002 folgte eine Erweiterung des Standards auf 10 GBit/s. Die aktuelle Version des Standard [IEE05] wurde 2005 veröffentlicht. Im aktuellen Standard wurde der neue Standard für Ethernet als Zugriffsnetz der ersten Meile eingeführt. Der Standard beschreibt insbesondere neue physikalische Schichten, die dem Bandbreitenaufkommen und den Kabellängen der ersten Meile gerecht werden. In der Entwicklung befindet sich zur Zeit der Ethernet-Standard IEEE P802.3ba, mit dem Übertragungsraten von 40 GBit/s und 100 GBit/s möglich werden [Eth07].

Das Ethernet ist ein verteiltes Kommunikationssystem, dessen Teilnehmer über ein Broadcast-Medium miteinander verbunden sind. Der Zugriff der Teilnehmer auf das Medium erfolgt über das in Kapitel 2.2.3 beschriebene CSMA/CD-Verfahren. Ethernet

ist also ein Protokoll der Sicherungsschicht mit den zwei Teilschichten LLC und MAC. Das Rahmen- bzw. das Paketformat von Ethernet wird für den DIX und den IEEE-Standard in Abbildung 2.4 dargestellt. Bei beiden Formaten werden vor dem Paket die Präambel und der SFD (Start-of-Frame Delimiter) übertragen. Jedes der 7 Byte der Präambel hat das Bitmuster 10101010 und dient zur Taktsynchronisation zwischen Sender und Empfänger. Das SFD kennzeichnet das Ende der Präambel und den Anfang der Paketübertragung. Jedes Paket besitzt eine Ziel- und eine Quelladresse. Die Zieladresse bestimmt entweder einen bestimmten Empfänger (*Unicast*), mehrere Empfänger (*Multicast*) oder alle Empfänger im selben Netzwerksegment (*Broadcast*). Ein Netzwerksegment ist ein Teilnetz, welches von einem Router von anderen Teilnetzen getrennt wird. Das nächste Feld im DIX-Standard (Abbildung 2.4(a)), das Typfeld, identifiziert das Vermittlungsschichtprotokoll, an welches das Paket weitergereicht wird. Im Datenfeld werden die Informationen aus den höheren Schichten eingebettet. Die Anzahl der Bytes des Datenfeldes zusammen mit dem optionalen Padfeld liegen zwischen 46-1500 Byte. Bei einer Datenmenge kleiner als 46 Byte wird das Datenfeld mit dem Padfeld aufgefüllt, um diese minimale Schranke nicht zu unterschreiten. Das letzte Feld enthält die 32 Bit breite Paketprüfsumme, die sogenannte *Frame Check Sequenze (FCS)*. Sie wird über das gesamte Paket, ohne Präambel und SFD, mit dem CRC-Verfahren (Cyclic Redundancy Check) gebildet.

Als die IEEE die Standardisierung übernahm, ersetzte sie das Typfeld durch das Längenfeld (Abbildung 2.4(b)). Das Längenfeld beschreibt die Anzahl der gültigen Bytes im Datenfeld in einem Bereich zwischen 0 und 1500 Byte. Um trotzdem mehrere Protokolle auf der Vermittlungsschicht unterscheiden zu können, wird in dem Datenfeld der LLC-Kopf integriert. Dieser besteht aus den Feldern DSAP (Destination Service Access Point) sowie SSAP (Source Service Access Point), die das Protokoll der höheren Schicht identifizieren und einem Kontrollfeld (Ctrl), das die Art des Paketes kennzeichnet. Allerdings hat sich das IEEE-Paketformat nicht durchgesetzt und die IEEE erkennt seit 1997 beide Formate an. Alle Typfeldbezeichungen liegen oberhalb von 1500 Byte. Werte kleiner oder gleich 1500 kennzeichnen dementsprechend ein Längenfeld.

Damit auch zwischen den am weitesten von einander entfernten Teilnehmern Kollisionen erkannt werden, wird bei Ethernet eine minimale Paketlänge von 64 Byte vorgeschrieben. Eine Übertragung dauert daher mindestens 512 Bitzeiten. Dieses entspricht der maximalen Laufzeit des ersten Bits für Hin- und Rückweg zwischen den entferntesten Teilnehmern einschließlich aller Signalverzögerungen durch das Medium und durch die Repeater. Damit wird sichergestellt, dass ein Sender die Übertragung nicht beendet hat, bevor er eine Kollision erkennen konnte. Über das Padfeld wird das Datenfeld auf mindestens 46 Byte aufgefüllt, falls die zu übertragenden Nutzdaten eine geringere Größe haben. Die maximale Länge eines Ethernet-Paketes beträgt

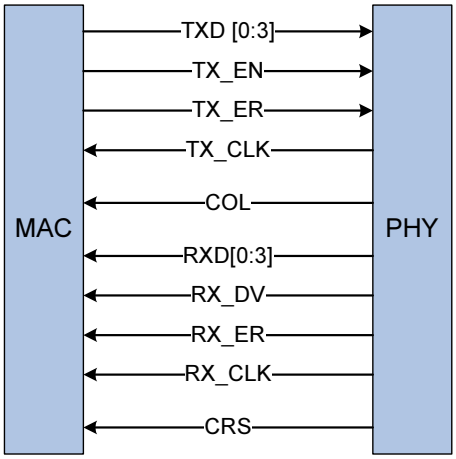


Abbildung 2.5: Signale des Media Independent Interfaces

1518 Byte. Diese Größe wurde im DIX-Standard frei definiert. Bei der Wahl der maximalen Paketlänge sollte die Größe des notwendigen Speichers für Pakete für die Sender-/Empfängereinheit begrenzt werden.

Bei der Bestimmung der Übertragungsdauer mehrerer Pakete kommt zu der Paketgröße, der Präambel und dem SFD noch eine weitere Größe hinzu, nämlich die *Inter Frame Gap (IFG)*. Die IFG ist die im Standard definierte minimale Paketlücke zwischen zwei direkt hintereinander übertragenen Paketen. Die Dauer der Paketlücke beträgt 96 Bitzeiten, d.h., bei 10 MBit/s-Ethernet beträgt die IFG 9,6 μs und bei Fast-Ethernet 0,96 μs.

Media Independent Interface: Im Ethernet-Standard [IEE05] wird unter anderem die Bitübertragungsschicht, bei Ethernet auch physikalische Schicht (kurz PHY) genannt, für unterschiedliche Medien, wie Kupfer und LWL (Lichtwellenleiter), definiert. Die Schnittstelle zwischen der Sicherungsschicht und der physikalischen Schicht wird bei 10 MBit/s und 100 MBit/s Ethernet durch das *Media Independent Interface (MII)* beschrieben. Die Signale der MII sind in der Abbildung 2.5 dargestellt und werden im Folgenden erläutert:

- **TXD (Transmit Data)** sind die Sendesignale, über die in jedem Sendetakt TX_CLK 4 Bit zum Übertragen angelegt werden.
- **TX_EN (Transmit Enable)** wird von dem MAC getrieben, wenn zu sendende Daten am MII anliegen. Das Signal kennzeichnet auch den Beginn und das Ende der Übertragung.
- **TX_ER (Transmit Coding Error)** signalisiert der PHY, einen Sendefehler zu generieren.

- **TX_CLK (Transmit Clock)** ist der Sendetakt. Die Taktrate beträgt 2,5 MHz für 10 MBit/s-Ethernet und 25 MHz für 100 MBit/s-Ethernet.

- **COL (Collision Detection)** signalisiert dem MAC eine erkannte Kollision.

- **RXD (Receive Data)** sind Empfangssignale, über die in jedem Takt RX_CLK 4 Bit empfangen werden.

- **RX_DV (Receive Data Valid)** wird von der PHY getrieben, wenn die Empfangsdaten synchronisiert zur RX_CLK an RXD anliegen. Über das Signal können auch der Beginn und das Ende eines empfangenen Paketes identifiziert werden.

- **RX_ER (Receive Error)** übermittelt von der PHY detektierte Empfangsfehler an den MAC.

- **RX_CLK (Receive Clock)** ist der Empfangstakt. Bei 10 MBit/s-Ethernet oder 100 MBit/s hat der Empfangstakt dieselbe Taktfrequenz wie der Sendetakt. Allerdings sind Sende- und Empfangstakt nicht synchron zueinander.

- **CRS (Carrier Sense)** wird von der PHY getrieben, wenn ein Trägersignal auf dem Empfangs- oder Sendemedium erkannt wird.

- **MDC (Management Data Clock)** ist der Managementtakt und dient als Zeitbasis für den Datenaustausch über das MDIO-Signal.

- **MDIO (Management Data Input/Output)** ist eine bidirektionale Verbindung zwischen PHY und MAC und wird zur Übertragung von Kontroll- und Statusinformationen verwendet.

2.2.6 TCP/UDP/IP

Die Kommunikationsverbindungen im Internet werden oftmals als TCP/IP- oder als UDP/IP-Verbindung bezeichnet. Dabei sind TCP (Transmision Control Protocol), UDP (User Datagram Protocol) und IP (Internet Protocol) eigenständige Protokolle auf den Schichten 3 und 4. Auf der Vermittlungsschicht werden mit dem IP-Protokoll mehrere Teilnetze im Internet miteinander verbunden. Die Protokolle TCP und UDP gehören zur Transportschicht und steuern den Datenfluss zwischen den Anwendungsprozessen.

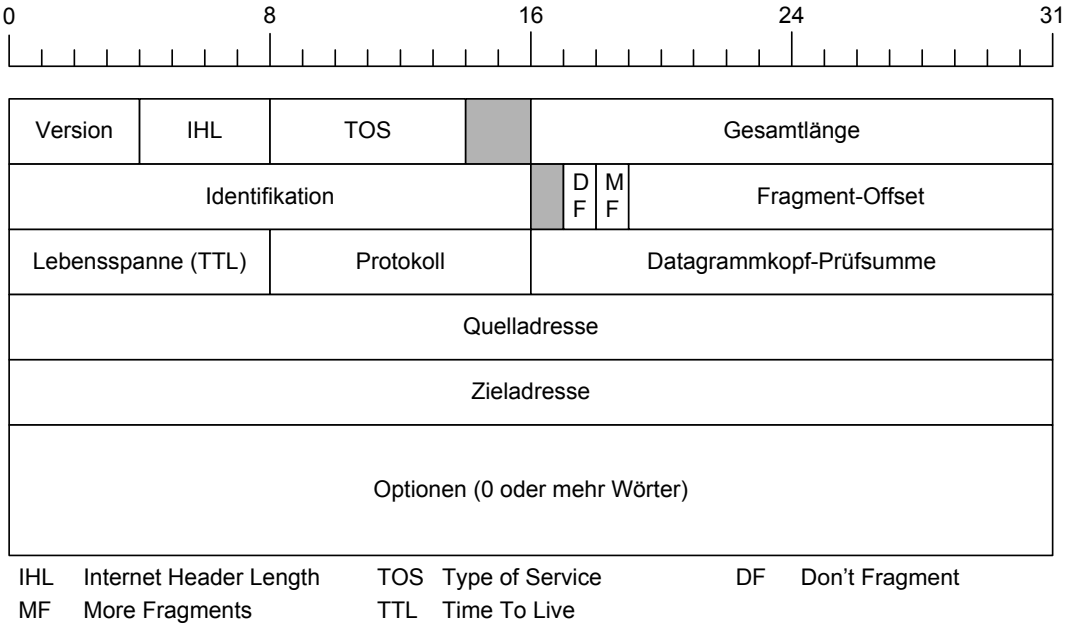


Abbildung 2.6: Der IPv4 Datagrammkopf

Das IP-Protokoll: Anhand einer 32 Bit breiten IP-Adresse (in der Version IPv4) werden beim IP-Protokoll die Teilnehmer in unterschiedlichen Netzwerksegmenten identifiziert und die Route zwischen den Netzwerksegmenten festgelegt. Die Version IPv6 ist der seit langer Zeit propagierte Nachfolger von IPv4. Mit seiner 128 Bit breiten IP-Adresse erweitert er den Adressraum des Internets und ist eine Antwort auf das ständig wachsende Netz. Allerdings lässt eine hohe Verbreitung von IPv6 im Internet noch auf sich warten. Im Folgenden wird nur die Version IPv4 betrachtet, die auch im Bereich der Echtzeitkommunikation den höchsten Verbreitungsgrad besitzt.

Ein IP-Paket wird über Ethernet eingebettet im Datenteil des Ethernet-Paketes versendet. Der Ethernet-Typ von *0x8000* kennzeichnet die darauf folgenden Daten als IP-Datagramm. Jedes IP-Datagramm besteht aus einem Datagrammkopf und einem Datenteil und hat eine maximale Länge von 65.535 Byte. Der Datagrammkopf besteht aus einem 20 Byte großen festen Abschnitt und einem Optionsfeld variabler Länge von maximal 40 Byte. Der Paketkopf ist in der Abbildung 2.6 dargestellt. Die Aufgaben der einzelnen Felder werden detailliert im [Tan03] beschrieben. Der *Type of Service (TOS)* des IP-Datagramms wird auch in der Echtzeitkommunikation verwendet, um Daten zu priorisieren. Über das TOS-Feld werden verschiedene Dienstklassen ausgewählt, die minimale Latenzen und einen maximalen Durchsatz ermöglichen.

Die Übertragung auf IP-Ebene ist völlig ungesichert, d.h., es können z. B. Pakete verloren gehen, ohne dass der Empfänger dies bemerkt. Oder Pakete erreichen den Empfänger nicht in der Reihenfolge, in der sie gesendet wurden. Für eine gesicherte Verbindung wird im Internet das TCP-Protokoll benötigt.

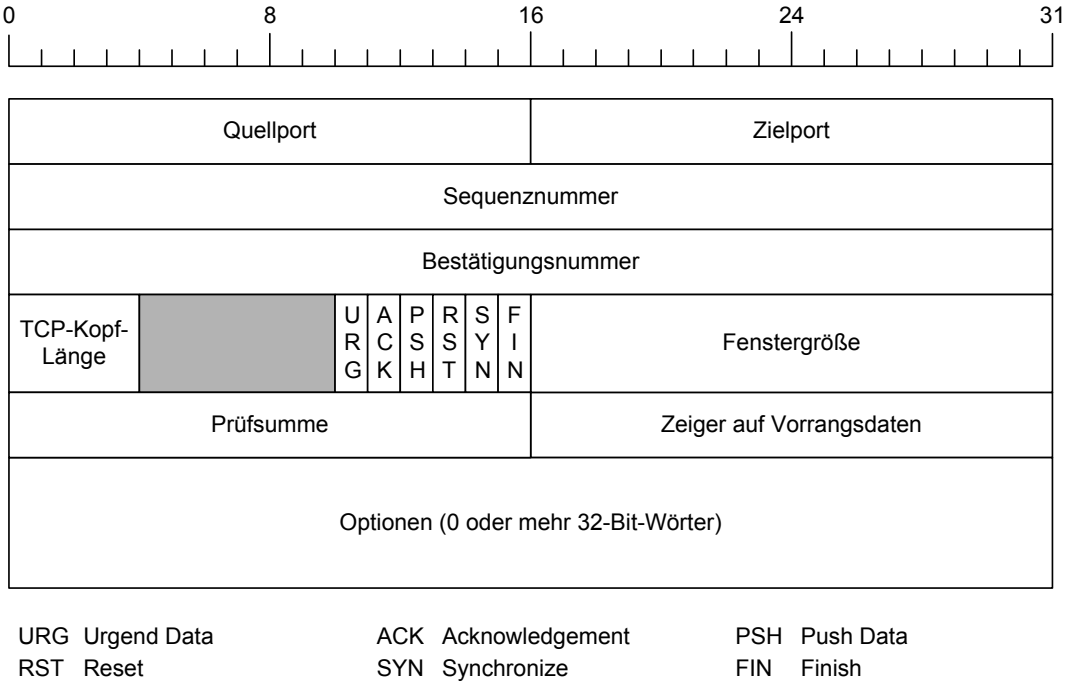


Abbildung 2.7: Der TCP-Segmentkopf

TCP-Protokoll: Auf der Basis von IP werden mit TCP sichere Verbindungen zwischen den Teilnehmern aufgebaut. Neben der Verwaltung der Verbindungen und der Überwachung der Datenreihenfolge übernimmt TCP auch die Fehler- und Flusskontrolle. Zur Fehler- und Flusskontrolle verwendet TCP einen Fenstermechanismus und bestätigt fehlerfrei empfangene Daten. Erhält der Sender keine Bestätigung, werden die Daten erneut übertragen. Die Daten werden bei TCP in Form von Segmenten ausgetauscht. Jedes TCP-Segment beginnt mit dem Segmentkopf, der einen 20 Byte großen festen Teil und einen optionalen Teil hat. Die Größe eines TCP-Segmentes ist durch die Größe des IP-Nutzdatenfeldes von 65.515 Byte begrenzt. Die einzelnen Felder des Segmentkopfes sind in Abbildung 2.7 dargestellt. Eine ausführliche Beschreibung der einzelnen Felder erfolgt in [Tan03].

Im Bereich der Echtzeitkommunikation wird mit Ausnahme von Modbus/TCP das TCP nur für die Übertragung von nicht echtzeitkritischen Daten verwendet. Im Bereich der Automatisierungsnetze werden z. B. Daten für die Parametrierung der Fertigungsanlage und des Netzwerkes sowie Statusabfragen über TCP versendet. Für die Übertragung von echtzeitkritischen Daten wird anstelle von TCP das einfachere UDP verwendet.

UDP: Ein verbindungsloses und sehr einfaches Protokoll der Transportschicht ist UDP. Der in Abbildung 2.8 gezeigte Paketkopf von UDP hat lediglich eine Länge

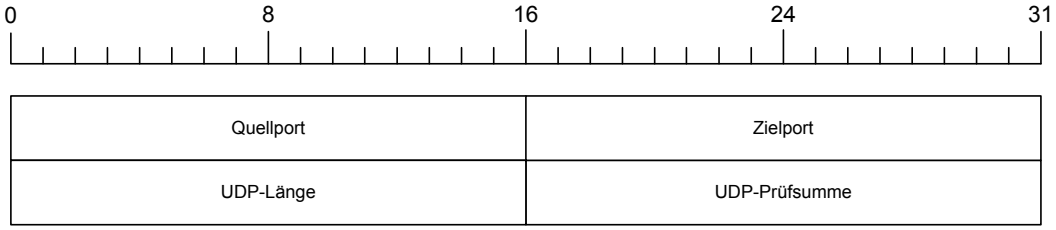


Abbildung 2.8: Der UDP-Segmentkopf

von 8 Byte. Wesentliche Aufgabe von UDP ist es, die Kommunikation zwischen zwei Prozessen über den Quell- und den Zielport eindeutig zu identifizieren. Das Protokoll ist, wie auch IP, ungesichert, d.h., es unterstützt keine Flusskontrolle und keine Überprüfung der Paketreihenfolge. Nur über die Prüfsumme kann bei UDP ein fehlerhaftes Paket erkannt werden. Da es keinen Quittierungsmechanismus gibt, werden fehlerhafte Pakete nicht erneut gesendet, sondern verworfen.

Aufgrund der Einfachheit und der fehlenden Kontrollmechanismen wird der Einsatz von UDP für die Echtzeitkommunikation gegenüber TCP bevorzugt. Der kleinere Segmentkopf von UDP erhöht die Effizienz des Protokolls. Die Protokolleffizienz ist das Verhältnis zwischen den gesendeten Nutzdaten und den tatsächlich gesendeten Daten. Insbesondere bei der Echtzeitkommunikation beträgt die Menge der Nutzdaten nur wenige Bytes. Die Größe des Paketkopfes hat daher einen größeren Einfluss auf die Protokolleffizienz. Der wesentliche Vorteil von UDP gegenüber TCP bei der Übertragung von echtzeitkritischen Daten ist die geringere Latenzzeit von UDP aufgrund der fehlenden Flusskontrolle und des fehlenden Quittierungsmechanismus.

2.3 Echtzeitkommunikation

Nach der Definition der Echtzeit in Kapitel 2.1 wird eine Aufgabe in Echtzeit dann erfüllt, wenn ihr Ergebnis innerhalb einer garantierten Zeitschranke vorliegt. Aufgabe in der Echtzeitkommunikation ist es, Daten vom Sender zum Empfänger innerhalb eines garantierten Zeitraums zu übertragen. Um den Zeitraum vom Sendewunsch bis zur Auslieferung der Daten an den Empfänger garantieren zu können, muss der Zugriff auf das Netzwerk und die Zugriffsdauer der Sender deterministisch sein. Im Bereich der Echtzeitkommunikation werden daher in den meisten Fällen keine zufälligen Medienzugriffsverfahren verwendet, sondern nur deterministische Verfahren.

Auch bei der Echtzeitkommunikation wird zwischen *weicher* und *harter* Echtzeit unterschieden. Typische Anwendungsgebiete für die weiche Echtzeitkommunikation ist die Übertragung von Video- und Sprachdaten. Die Priorisierung von Daten ist bei der weichen Echtzeitkommunikation z. B. im Internet ein weit verbreitetes Verfahren. Daten

mit einer hohen Priorität, wie z. B. Video- und Sprachdaten, werden von Netzwerkkomponenten (z. B. Router und Switch) schneller weitergeleitet als Daten mit niedriger Priorität. Normalerweise werden alle ankommenden Daten in eine Warteschlange geschrieben und der Reihe nach verarbeitet und weitergeleitet. Daten mit hoher Priorität gelangen in eine separate Warteschlange, die als Erstes von der Netzwerkkomponente abgearbeitet wird. Dadurch ist die Latenzzeit eines Datenpaketes mit hoher Priorität im Durchschnitt geringer, als die eines normalen Paketes. Anwendungsgebiete von harter Echtzeitkommunikation ist die Übertragung von Sensor- und Aktordaten in verteilten Steuerungs- und Regelungssystemen. Harte Echtzeit-Kommunikationssysteme werden z. B. zur Vernetzung der Fahrzeugelektronik oder bei der Vernetzung von Fertigungsanlagen eingesetzt. Bei der harten Echtzeitkommunikation wird der Ablauf der Kommunikation geplant, damit alle Teilnehmer ihre Echtzeitkriterien einhalten können. Dazu wird die Reihenfolge und die Dauer des Zugriffs der Teilnehmer festgelegt. Sind Reihenfolge und Zugriffsdauer bekannt, kann die maximale Dauer vom Sendewunsch bis zur Auslieferung der Daten mathematisch bestimmt werden. Somit kann auch nachgewiesen werden, dass die Echtzeitanforderungen der Teilnehmer erfüllt werden können. Allerdings gibt es Anforderungen an die Zeitschranken und an die Anzahl der Sendewünsche, die von den physikalischen Gegebenheiten des Netzwerks, d.h. von der maximal möglichen Bandbreite und von der Ausbreitungsgeschwindigkeit, nicht bewältigt werden können. Die gestellten Anforderungen sind in diesem Fall nicht planbar.

In der Praxis werden auch für harte Echtzeitanwendungen Protokolle verwendet, die nur eine Priorisierung der Daten unterstützen. Es wird davon ausgegangen, dass auch diese Protokolle harte Echtzeitanforderungen bewältigen, wenn eine ausreichende Menge an Bandbreite zur Verfügung steht [Sei00].

2.3.1 Dienstgüte und Dienstklasse

Die Dienstgüte, im Englischen auch *Quality of Service (QoS)* genannt, fasst die Anforderungen zusammen, die ein Datenstrom an das Kommunikationssystem stellt. Als die vier primären Parameter der Dienstgüte nennt [Tan03]: Zuverlässigkeit, Latenz, Jitter und Bandbreite. Die Anforderungen an die vier Parameter, d.h., ob diese hoch oder niedrig sind, bestimmt die Dienstgüte. Bei der Verwendung von QoS fordert die Anwendung eine Dienstleistung und deren Qualität im Voraus an. Das Netzwerk reserviert daraufhin eine Verbindung und die dazu benötigten Ressourcen, um die geforderte Dienstgüte zu garantieren. Die Kommunikation beim QoS erfolgt immer verbindungsorientiert. Verwechselt wird die Dienstgüte oftmals mit der Dienstklasse. Von der Dienstklasse oder dem *Class of Service (CoS)* wird gesprochen, wenn die Daten nicht mit einer garantierten Qualität übertragen werden, sondern nur aufgrund einer

höheren Priorität bevorzugt behandelt werden [Sei00]. Beim CoS werden im Gegensatz zum QoS keine Datenflüsse, sondern einzelne Pakete betrachtet. Jedes Paket wird einer Dienstklasse zugeordnet und erhält die dazu gehörige Priorität. Pakete mit der höchsten Priorität erhalten die bestmögliche Qualität, die momentan zur Verfügung steht. Der Grad der Qualität wird in diesem Fall aber nicht garantiert.

2.3.2 Ereignis- und zeitgesteuerte Echtzeitkommunikation

Bei der Echtzeitkommunikation wird zwischen der ereignisgesteuerten und zeitgesteuerten Kommunikation unterschieden. Der Sendewunsch wird bei der ereignisgesteuerten Kommunikation durch ein Ereignis ausgelöst, wie z. B. das Betätigen eines Tasters oder die Veränderung eines Sensorwertes. Die Kommunikation ist in diesem Fall azyklisch. Bei der zeitgesteuerten Echtzeitkommunikation wird der Sendewunsch nicht durch ein Ereignis, sondern durch einen periodisch wiederkehrenden Zeitpunkt ausgelöst. Diese Art der Kommunikation ist zyklisch und wird z. B. für Regler verwendet, die Sensordaten zyklisch über das Netz abfragen und Stellgrößen für die Aktoren zyklisch über das Netz versenden.

Ereignisgesteuerte und zeitgesteuerte Protokolle sind für die jeweilige Art der Echtzeitkommunikation optimiert. Ein Vergleich der Leistungsfähigkeit der beiden Protokollansätze wird in [Kop97] vorgestellt. Müssen in einer Umgebung sehr viele unregelmäßige Nachrichten zu vorher unbekannten Zeitpunkten versendet werden, sind die ereignisgesteuerten Protokolle überlegen. Ein ereignisbasiertes Protokoll reagiert umgehend auf ein Ereignis und versendet eine Nachricht. Ein weit verbreitetes ereignisgesteuertes Echtzeitprotokoll ist CAN, das bei der Vernetzung von Steuergeräten in Kraftfahrzeugen eingesetzt wird. Die zeitgesteuerten Protokolle haben bei dem periodischen Austausch von Daten einen Vorteil gegenüber den ereignisgesteuerten Protokollen. Ebenfalls können die zeitgesteuerten Protokolle Fehler, wie z. B. den Verlust eines Paketes, besser erkennen. Viele zeitgesteuerte Echtzeitprotokolle basieren auf dem TDMA, wie z. B. FlexRay und PROFINET, die in Kapitel 2.3.3 und 2.5 beschrieben werden. Eine eindeutige Einteilung eines Protokolls in ereignis- oder zeitgesteuert ist nicht immer möglich, weil viele Protokolle beide Kommunikationsformen unterstützen.

2.3.3 Feldbusse

Im industriellen Umfeld der Automatisierungstechnik gewann die Echtzeitkommunikation mit der Entwicklung der Feldbusse an Bedeutung. Feldbusse ersetzten im industriellen Bereich die sternförmigen Verkabelungen, die von der Steuerung zu jedem Sensor und Aktor gelegt werden mussten. Durch den Einsatz eines Feldbusses wird die

Anzahl der notwendigen Kabel erheblich reduziert und Kosten werden eingespart. Aufgrund einer gestiegenen Anzahl von Teilnehmern und höheren Anforderungen werden die Feldbusse heutzutage in vielen Bereichen durch die Ethernet-basierten Echtzeitprotokolle verdrängt. Die Ethernet-basierte Echtzeitkommunikation ist ein Schwerpunkt dieser Arbeit. Sie wird daher in Kapitel 2.4 in einem eigenen Abschnitt beschrieben. In diesem Abschnitt werden exemplarisch einige Feldbusse erläutert, die auch heute noch eingesetzt werden.

Feldbusse übernehmen die Kommunikation zwischen Sensoren, Aktoren und speicherprogrammierbaren Steuerungen (SPS) oder einem Industrie-PC und vernetzen diese über ein Bussystem. Die zu übertragenden Datenmengen sind bei Feldbussen sehr gering, sodass die Feldbusse mit sehr geringen Datenraten auskommen, die im Bereich von 1 Mbit/s bis 5 Mbit/s liegen [Dec01]. Typische Einsatzgebiete von Feldbussen sind die Industrie- und Gebäudeautomation sowie die Fahrzeugkommunikation.

Wie bereits erwähnt, ist CAN [Ets02] ein Feldbus, der im Bereich der Fahrzeugkommunikation sehr verbreitet ist, aber auch in der Industrieautomation als CanOpen-Protokoll [DIN03] verwendet wird. Das ereignisbasierte CAN wurde 1998 u.a. von der Firma Bosch entwickelt und verwendet das CSMA/CA-Verfahren. Die Adressierung bei CAN erfolgt nicht über die Angabe der Zieladresse, sondern mithilfe der Paketidentität, die den Inhalt der Nachricht, z. B. die Motortemperatur, identifiziert. Über die Paketidentität wird auch die Priorität für das CSMA/CA-Verfahren festgelegt. Mit seiner geringen Datenrate von maximal 1 MBit stößt CAN schon heute an seine Leistungsgrenze. Aus diesem Grunde wurde für die Kommunikation im Fahrzeug von der BMW AG und der Daimler AG zusammen mit Philips und Motorola das Echtzeitprotokoll FlexRay [Fle04] entwickelt. FlexRay kommuniziert über zwei physikalisch getrennte Leitungen mit einer Datenrate von jeweils 10 MBit/s. Die zwei Kanäle dienen hauptsächlich der redundanten und damit fehlertoleranten Übertragung von Nachrichten. Das FlexRay-Protokoll verwendet für den Medienzugriff das TDMA-Verfahren und ist somit ein zeitgesteuertes Protokoll. Für eine flexiblere Kommunikation ist das Übertragungsschema von FlexRay in ein statisches und ein dynamisches Segment aufgeteilt. Im statischen Segment werden die Echtzeitdaten in mehreren Zeitschlitzen übertragen und im dynamischen Segment erfolgt die Übertragung prioritätsgesteuert. Die Grenzen zwischen beiden Segmenten sind flexibel, womit auch ein rein statischer oder rein dynamischer Betrieb möglich ist. Dem FlexRay-Protokoll sehr ähnlich ist das TTP/C (Time-Triggered Protocol), welches an der Technischen Universität Wien entwickelt wurde [KG94]. Beide Protokolle erfordern eine explizite Synchronisation der lokalen Uhren, damit die Teilnehmer nur innerhalb ihres Zeitschlitzes Daten übertragen. Ein Vergleich der beiden Protokolle ist in [Kop01] zu finden.

Ein Feldbus aus dem Bereich der Industrieautomatisierung ist der von Phoenix Contact entwickelte Interbus [Bag98]. Die Teilnehmer beim Interbus sind ringförmig mit-

einander verbunden. Der Medienzugriff erfolgt über ein *Master-Slave*-Verfahren. Die Daten werden beim Interbus von Slave zu Slave in einem einzelnen Summenrahmen übertragen, der in jedem Zyklus erneut vom Master gesendet wird. Das Prinzip dieser Übertragung gleicht einem verteilten Schieberegister. Der Master sendet in umgekehrter Reihenfolge der Teilnehmer im Ring nacheinander die Daten an die Slaves und empfängt gleichzeitig die Sensordaten der Slaves. Die Kommunikation erfolgt also im Gegensatz zu den drei zuvor beschriebenen Protokollen in beide Richtungen. Ebenfalls wird die Kommunikation nicht von der Zeit, sondern von einer zentralen Einheit, nämlich dem Master, gesteuert. Trotzdem hat Interbus aufgrund der zyklischen Kommunikation mehr Gemeinsamkeiten mit den zeitgesteuerten Protokollen als mit den ereignisgesteuerten Protokollen. Nachteil einer zentralen Steuerung der Kommunikation ist, dass ein Ausfall des Masters zum Erliegen der gesamten Kommunikation im Netzwerk führt. Auch der Profibus [Pop00] verwendet einen Masterknoten zur Steuerung der Kommunikation. Allerdings erlaubt Profibus mehrere Masterknoten, die über ein Token-Verfahren Zugriff auf den Bus erhalten. Das Medienzugriffsverfahren bei Profibus ist daher eine Kombination aus Token- und Master-Slave-Verfahren. Bei diesem Verfahren darf nur der Master mit den Slaves kommunizieren, der den Token hält. Weitergereicht wird der Token an den Master mit der höchsten Priorität oder bei gleichen Prioritäten an den nächsten Master in dem logischen Ring. Aufgrund des hybriden Medienzugriffsverfahrens ist eine klare Einteilung des Profibus-Protokolls in zeit- oder ereignisgesteuert nicht möglich.

In der Gebäudeautomatisierung werden beispielsweise die Feldbussysteme LON (Local Operating Network) [DL98] und EIB (European Installation Bus) [DK00] eingesetzt. Das LON-Protokoll wurde von der Firma Echelon entwickelt und verwendet für den Medienzugriff das CSMA/CD-Verfahren mit Prioritäten. LON ist damit ein Beispiel für ein Echtzeitprotokoll, welches auf einem zufälligen Medienzugriffsverfahren basiert. Aufgrund der hohen Anzahl von 32.385 Knoten sind bei LON sogar alle 7 Schichten des ISO/OSI-Referenzmodells implementiert. Der EIB ist ein offener Feldbus, dessen Spezifikation von der EIB Association verwaltet wird. Mit EIB können 50.000 Endgeräte angesteuert werden. Der Medienzugriff erfolgt, wie bei CAN, nach dem CSMA/CA-Verfahren. Sowohl EIB als auch LON sind ereignisgesteuerte Protokolle.

2.4 Ethernet-basierte Echtzeitkommunikation

Durch eine gestiegene Anzahl von Knoten und höhere Leistungsanforderungen stoßen die Feldbusssysteme mit ihrer geringen Bandbreite an ihre Grenzen. Neue Aufgaben, wie die Bildverarbeitung, erweiterte Diagnosemöglichkeiten und webbasierte Statuskontrolle des Netzwerkes und jedes einzelnen Knotens erfordern breitbandige echtzeitfähige Netzwerke, um die erhöhten Datenmengen zu bewältigen. Verschiede-

ne breitbandige Netzwerke, wie z. B. das FDDI (Fibre Distributed Data Interface) und ATM (Asynchronous Transfer Mode), wurden für den Einsatz in harten Echtzeit-Kommunikationssystemen untersucht [MKZ96, EHS97, RZKJ01]. Aufgrund ihrer hohen Komplexität und den hohen Kosten für die Netzwerkkomponenten haben sich diese Protokolle im Bereich der Echtzeitkommunikation nicht durchgesetzt [Son01].

Obwohl Ethernet im LAN-Bereich sehr verbreitet ist, die Kosten für die Ethernet-Komponenten sehr günstig sind, deren Verfügbarkeit hoch ist sowie die Entwicklung von der IEEE stetig vorangetrieben wird, wurde die direkte Verwendung von Ethernet in der Echtzeitkommunikation aufgrund des undeterministischen Medienzugriffsverfahrens vermieden (siehe Kapitel 2.2.3). Das undeterministische Verhalten von Ethernet entsteht, wenn Pakete kollidieren. Die Kollisionsauflösung durch den BEB-Algorithmus resultiert in einer nicht vorhersagbaren Wartezeit bis zur tatsächlichen Übertragung eines kollidierten Paketes und kann zu einer ungerechten Bedienung verschieden Teilnehmer führen. Durch das in der Literatur als *Capture Effect* beschriebene Verhalten kann ein einzelner Teilnehmer das Netzwerk monopolisieren und direkt hintereinander Pakete senden, während andere Teilnehmer ebenfalls versuchen, auf den Bus zuzugreifen. Nach einem Beispiel aus [RY94] soll der Captur Effekt erläutert werden.

Angenommen wird, dass zwei Stationen gleichzeitig ein Paket im selben Zeitschlitz übertragen wollen. Beide Stationen besitzen jeweils einen Kollisionszähler i, der zu Beginn null ist. Nach der Kollision wird der Zähler i um eins erhöht. Jede Station wählt zufällig eine Wartezeit von null oder einen Zeitschlitz. Es wird der Fall angenommen, dass Station 2 eine Wartezeit von einem Zeitschlitz wählt und Station 1 direkt im nächsten Zeitschlitz überträgt. Nachdem Station 1 ein Paket erfolgreich übertragen hat, wird i für Station 1 zurückgesetzt, während der Kollisionszähler von Station 2 weiterhin eins ist. Weiter angenommen, Station 1 verfügt über weitere Pakete, die zur Übertragung bereitstehen, dann erfährt das Paket von Station 2 erneut eine Kollision. Das Intervall, aus dem der Algorithmus eine gleich verteilte Zufallszahl ermittelt, liegt nun für Station 1 zwischen 0 und 1 und für Station 2 zwischen 0 und 3. Die Wahrscheinlichkeit, dass Station 1 die Kollisionsauflösung gegen Station 2 gewinnt und erfolgreich ein Paket sendet, ist nun größer. Dieses Verhalten kann sich bis zu einem Maximum von 16 Kollisionen fortsetzen und führt zum Verwerfen des Paketes von Station 2.

Im Zusammenhang mit dem Capture Effekt steht der sogenannte *Packet Starvation Effect (PSE)*, bei dem Pakete bei einer hohen Netzwerklast sehr lange Wartezeiten erfahren oder verworfen werden. Bei hoher Netzwerklast ist es sehr wahrscheinlich, dass nach der Kollisionsauflösung wieder im gleichen Zeitschlitz ein anderer Teilnehmer ein Paket überträgt und es erneut zu einer Kollision kommt. Für Pakete, die durch mehrfache Kollisionen bereits eine hohe Wartezeit haben, ist bei einer erneuten Kollision die Wahrscheinlichkeit hoch, dass die Wartezeit bis zur erneuten Übertragung noch

größer wird. Dadurch erfahren Pakete bei PSE eine bis zu 100 mal größere Verzögerung im Vergleich zur mittleren Verzögerung oder werden sogar verworfen. Schon bei 40 % Netzwerklast können sehr hohe Latenzen entstehen, und bei 60 % können schon verworfene Pakete festgestellt werden [WSF94].

Um diese Schwachstelle von Ethernet zu beheben und damit einen deterministischen und gerechten Zugriff auf das geteilte Medium zu gewährleisten, wurden verschiedene Verfahren entwickelt. Die Verfahren basieren entweder auf einer Anpassung des CSMA/CD-Protokolls oder es wird eine zusätzliche Zugriffskontrolle auf Ethernet aufgesetzt. Im Folgenden werden einige Verfahren vorgestellt. Weitere Details zu Ethernetbasierter Echtzeitkommunikation sind beispielsweise in [Jas02, HJ03, Dec05] zu finden.

Modifizierungen des CSMA/CD-Verfahrens: Das im Ethernet-Standard definierte CSMA/CD Protokoll wird bei diesem Ansatz so angepasst, dass die Wartezeit nach einer Kollision vorherbestimmbar ist oder der Zugriff gerechter geregelt wird.

Beim **CSMA/DCR-Protokoll** (Deterministic Collision Resolution) wird der BEB-Algorithmus durch einen deterministischen Algorithmus, der auf einem binären Suchbaum basiert, ersetzt [LLR93]. Allerdings kann das Protokoll, ähnlich wie bei CAN, nur für einen Teilnehmer harte Echtzeitanforderungen garantieren, weil die Kollisionsauflösung in Abhängigkeit von der Adresse des Teilnehmers erfolgt.

Das **Virtual Time CSMA** reduziert die Wahrscheinlichkeit für das Auftreten einer Kollision. Dazu werden bei diesem CSMA-Verfahren in einem Knoten zwei Uhren verwendet, eine für die reale Zeit und eine für die virtuelle Zeit. Die reale Uhr läuft die gesamte Zeit kontinuierlich mit demselben Takt weiter. Die virtuelle Uhr läuft nur, wenn der Kanal nicht belegt ist und stoppt, sobald eine Übertragung beginnt. Eilt die virtuelle Zeit der realen Zeit hinterher, läuft sie schneller und anderenfalls mit demselben Takt wie die reale Zeit. Ein Paket wird bei dem ersten Ansatz des Virtual Time CSMA-Protokolls gesendet, sobald die virtuelle Uhr der Ankunftszeit des Paketes entspricht [MK85]. Dieser Ansatz erhöht in erster Linie die Gerechtigkeit bei der Übertragung der wartenden Pakete. Um harte Echtzeitkommunikation zu gewährleisten, wird in [ZR87] nicht aufgrund der Ankunftszeit eines Paketes übertragen, sondern aufgrund seiner Zeitschranke.

Die Behandlung von hochpriorisierten Paketen erlaubt das **CSMA/RI-Protokoll** (Reservation by Interruption) [FZ00]. Beim CSMA/RI unterbricht ein hochpriorisiertes Paket eine laufende Übertragung, um Bandbreite für dieses Paket zu reservieren. Nachdem die unterbrochene Übertragung wieder aufgenommen und beendet wurde, darf nur die Station, die reserviert hat, übertragen. Die Latenzzeit für ein Paket mit hoher Priorität kann mit dem CSMA/RI-Protokoll verbessert werden. Allerdings können harte Echtzeitanforderungen nicht garantiert werden [HJ03]. Ein Nachteil der modifi-

zierten CSMA/CD-Verfahren ist, dass sie nicht mit Standard-Ethernet-Komponenten implementiert werden können.

Zusätzliche Zugriffskontrolle auf der Basis von Ethernet: Bei der zusätzlichen Zugriffskontrolle werden die deterministischen Zugriffsverfahren Master-Slave, Token und TDMA direkt auf die Ethernet-MAC-Schicht aufgesetzt. Die Ethernet-MAC-Schicht selbst bleibt unverändert und arbeitet weiterhin nach dem CSMA/CD-Verfahren. Allerdings sorgt die zusätzliche Zugriffskontrolle dafür, dass keine Kollisionen entstehen können und der Medienzugriff deterministisch wird.

Das **RTCC-Protokoll** (Real-Time Communication Control) verwendet ein Master-Slave-Verfahren für die Ethernet-basierte Echtzeitkommunikation [WXL+00]. In einem RTCC-Netzwerk fordert der Master einen Slave auf, Daten zu senden. Mithilfe einer Instruktionstabelle, die der Master sequenziell ausführt, wird die Reihenfolge festgelegt, in der die Slaves zum Senden aufgefordert werden. Zusätzlich zur Kontrolle des Zugriffs wird bei RTCC auch die Zugriffsdauer der Slaves kontrolliert, womit das Protokoll beim Zugriff und bei der Zugriffsdauer deterministisch ist.

Auf einem *Token Passing* Verfahren basiert das 1994 an der Universität von New York entwickelte **Rether** (Real-Time Ethernet Protokoll) [VC94, Ven97]. Der Token bei Rether wird zunächst nur an die Teilnehmer gesendet, die die Übertragung eines Echtzeitpaketes reserviert haben. Anschließend wird der Token von Knoten zu Knoten weitergeleitet. In dieser zweiten Phase haben die Teilnehmer die Möglichkeit, nicht echtzeitfähige Daten zu übermitteln. Diese Phase wird abgebrochen, sobald ein Knoten wieder Echtzeitdaten übertragen muss. Das Token Passing Verfahren wird bei Rether nur im Echtzeitmodus verwendet. Liegen Reservierungen für Echtzeitdaten von keinem Knoten mehr vor, initiiert der Knoten, der als letzter Echtzeitdaten gesendet hat, den Umschaltvorgang vom Token-Modus zum CSMA/CD-Modus. Der erste Knoten, der wieder Echtzeitdaten senden möchte, sendet eine Nachricht an alle anderen Teilnehmer und initiiert so den Umschaltvorgang zum Token-Modus. Im Token-Modus erfüllt Rether harte Echtzeitanforderungen. Allerdings gibt es für die Umschaltzeit keine obere Schranke, weil diese im CSMA/CD-Mode koordiniert wird.

Ein priorisiertes Token Passing Verfahren verwendet das **RT-EP** (Real-Time Ethernet Protocol)[MHG03]. Das verwendete priorisierte Token Passing Verfahren ähnelt dem Token Ring Protokoll, das von der IEEE standardisiert wurde [IEE98]. Die Knoten bei RT-EP bilden einen logischen Ring, der vom Token durchlaufen wird. Der Token enthält die Informationen von der Station, die das Paket mit der höchsten Priorität senden möchte und die Priorität des aktuellen Paketes. Die Kommunikation beginnt mit einer Arbitrierungsphase, die vom sogenannten *Token-Master* eingeleitet wird. In der Arbitrierungsphase besucht der Token jede Station einmal. Jede einzelne Station überprüft die Priorität des Tokens und passt die Priorität an die höchste Priorität

eines eigenen Paketes an, falls diese größer ist als die Priorität des Tokens. Nachdem der Token alle Knoten besucht hat und zum Token-Master zurückgekehrt ist, sendet der Token-Master eine Nachricht zu dem Knoten, dessen Paket die höchste Priorität hat. Dieser sendet daraufhin seine eigene Nachricht. Im Anschluss daran wird diese Station der neue Token-Master. Die Prioritäten werden mittels einer Ablaufplanung festgelegt, womit auch harte Echtzeitanforderungen erfüllt werden. Sowohl RT-EP als auch das zuvor beschriebene Rether sind reine Softwareimplementierungen, die Standard-Ethernet-Komponenten verwenden.

Ein auf Ethernet aufgesetztes TDMA-Verfahren wird in [LJR02] vorgestellt. Das TDMA und die notwendige Uhrensynchronisation werden vollständig in Software implementiert und in der Umgebung des Echtzeitbetriebssystems RTLinux [YB99] ausgeführt. Die Anbindung an das physikalische Ethernet-Netzwerk erfolgt über Ethernet-Netzwerkkarten. Nachteil einer Softwareimplementierung ist die geringe Genauigkeit bei der Uhrensynchronisation.

Eine Mischung aus TDMA und Master-Slave-Verfahren ist das in [PA02] vorgestellte **FTT-Ethernet**. Der Master bei FTT-Ethernet sendet zu allen Slaves eine Trigger-Nachricht, die den Ablaufplan für das Senden des Slaves enthält und die Teilnehmer synchronisiert. Gegenüber einem reinen TDMA können in jedem Zyklus die Sender flexibel variiert werden, und Zeitschlitze bleiben nicht ungenutzt, wenn Teilnehmer in einer Runde keine Daten übertragen möchten. Die Adressierung bei FTT identifiziert, wie bei CAN, den Inhalt der Daten. Alle Knoten, die diese Daten benötigen, können die Daten vom Bus empfangen.

2.4.1 Switched-Ethernet

Den Durchbruch in der Echtzeitkommunikation gelang Ethernet aber erst mit der Einführung der Switchtechnologie und dem damit verbunden *Switched-Ethernet*. Durch den Einsatz von Switches anstelle eines Busses oder eines Hubs können bei Ethernet sämtliche Kollisionen eliminiert werden, ohne den Ethernet-Standard zu verletzen. Dadurch wurde auch in der industriellen Automatisierungstechnik Ethernet als Echtzeit-Kommunikationssystem für die unterste Ebene, in der Daten zwischen Sensoren und Aktoren ausgetauscht werden, akzeptiert. Die industrielle Kommunikation ist zur Zeit das größte Anwendungsgebiet von Switched-Ethernet im Bereich der Echtzeitkommunikation. Analysiert wurden die Anforderungen an Switched-Ethernet und dessen Leistungsfähigkeit in der industriellen Kommunikation in [JN01, Vit01, SKS02]. Eine Übersicht von sowohl drahtgebundenen als auch drahtlosen industriellen Netzen und deren Anforderungen ist in [BH06] zu finden.

Bei Switched-Ethernet wird jeweils ein Teilnehmer an einen Port des Switches angeschlossen. Wird nur ein Switch verwendet, entsteht so eine sternförmige Topologie.

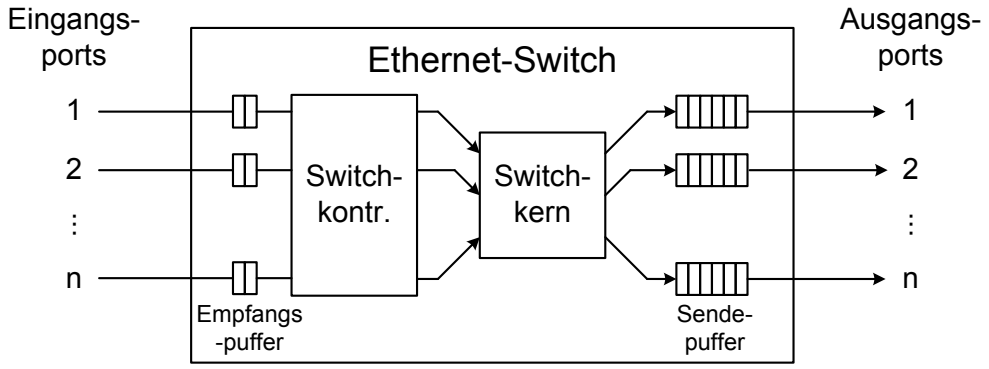


Abbildung 2.9: Schematischer Aufbau eines Ethernet-Switches

Mehrere Switches bilden einen Baum mit den Teilnehmern als Blätter. Auch an einen Hub werden alle Teilnehmer sternförmig angeschlossen. Der Hub überträgt im Gegensatz zum Switch ein empfangenes Paket an alle Teilnehmer. Der Switch überträgt das Paket nur an den Teilnehmer, für den das Paket bestimmt ist. Über einen Hub oder einen Bus kann daher immer nur ein Teilnehmer zum selben Zeitpunkt übertragen. Bei einem Switch kann jeder angeschlossene Teilnehmer parallel zueinander gleichzeitig senden und empfangen (Vollduplex). Gegenüber dem Bus oder dem Hub kann so bei gleicher Datenrate der Durchsatz im Netzwerk erhöht werden. Pakete, die von mehreren Teilnehmern an einen Teilnehmer gerichtet sind, werden im Switch gespeichert und nacheinander zum Teilnehmer übertragen. Durch den Switch wird die gemeinsame Kollisionsdomäne von Bus oder Hub in mehrere Punkt-zu-Punkt-Verbindungen aufgeteilt. Bei dem ausschließlichen Einsatz von Switches in einem Ethernet-Netzwerk können keine Kollisionen entstehen und der BEB-Algorithmus wird nicht weiter benötigt. Im Folgenden werden der Aufbau und die Architektur eines Switches beschrieben. Im Anschluss daran erfolgt eine Übersicht über die Echtzeitkommunikation mit Switched-Ethernet.

Switch-Architektur: Der generelle Aufbau einer Switcharchitektur, bestehend aus den Eingangs- und Ausgangsports, den Empfangs- und Sendepuffern, dem Switchkontroller und dem Switchkern, ist in Abbildung 2.9 dargestellt. Ein Switch ist eine Netzwerkkomponente, die von eingehenden Ethernet-Paketen auf der Ebene der Sicherungsschicht die Zieladresse des Paketes ermittelt und das Paket über den Switchkern (engl.: Switch Fabric) nur an den benötigten Ausgang weiterleitet. Im Empfangs- und Sendepuffer werden die Pakete zwischengespeichert, bis sie weitergeleitet bzw. gesendet werden können. Die meisten Switcharchitekturen verwenden Sendepuffer anstelle von Empfangspuffer. In einer reinen Empfangspufferarchitektur kann ein Paket aus einem Empfangspuffer nicht zu einem Ausgangsport weitergeleitet werden, über den bereits ein anderes Paket übertragen wird. Dieses Paket blockiert somit alle Pakete

im Empfangspuffer, auch wenn diese zu einem freien Ausgangsport übertragen werden könnten. Empfangspuffer werden daher nur zur kurzfristigen Speicherung verwendet, bis der Ausgangsport ermittelt wurde und das Weiterleiten des Paketes über den Switchkern beginnt. Die Empfangspuffer sind daher meistens kleiner als die Sendepuffer. Der Switchkontroller ermittelt mithilfe der Zieladresse den Ausgangsport, klassifiziert ein Paket z. B. aufgrund einer gesetzten Priorität und steuert den Switchkern. Für die Implementierung von Switchkontroller und Switchkern gibt es keinen Standard. Im Wesentlichen werden aber für den Switchkern die drei Strukturen Bus, geteilter Speicher und Matrix verwendet [Sei00].

Die Busarchitektur verwendet einen geteilten Hochgeschwindigkeitsbus, um Pakete zwischen den Ports auszutauschen. Der Vorteil der Busarchitektur ist die implizite Unterstützung von Multicast- oder Broadcast-Paketen. Allerdings benötigt bei dieser Architektur aufgrund des gemeinsamen Zugriffs jeder Eingangs- und Ausgangsport einen eigenen Paketpuffer. Werden Pakete von mehreren Eingangsports zu einem Ausgangsport weitergeleitet, kann es zum Überlauf des Sendepuffers kommen.

Die verbreitetste Architektur ist aufgrund der Einfachheit und der geringen Kosten der geteilte Speicher. Ankommende Pakete werden von dem Eingangsport in den geteilten Speicher geschrieben. Anschließend wird das Paket mithilfe der Zieladresse an den ermittelten Ausgangsport zur Übertragung weitergeleitet. Über den geteilten Speicher lässt sich die Größe des Sendepuffers für jeden Port anpassen, wodurch das Auftreten eines Pufferüberlaufs minimiert wird. Die Anzahl der Ports bei Bus und geteiltem Speicher ist durch die Bitrate des Busses bzw. der Speicherschnittstelle begrenzt. Bei einer großen Anzahl von Ports wird daher auf die Matrixarchitektur zurückgegriffen. Bei einer Matrixarchitektur wird für die Dauer der Übertragung eines Paketes eine Verbindung zwischen Eingangs- und Ausgangsport hergestellt. Die Architektur besteht aus einer Matrix von Schaltern, die jeden Eingangsport mit jedem Ausgangsport verbinden. Im Gegensatz zum Bus und zum geteilten Speicher ist allerdings keine implizite Übertragung von Multicast- oder Broadcast-Paketen möglich.

Des Weiteren wird beim Ethernet-Switch zwischen der *Store-and-Forward-* und der *Cut-Through-*Architektur unterschieden. Bei einem Store-and-Forward-Switch wird ein Paket erst vollständig empfangen, damit durch den CRC fehlerhafte Pakete detektiert werden können. Nur für fehlerfreie Pakete wird anschließend der Ausgangsport ermittelt und über diesen wird schließlich das Paket versendet. Die Zeit bzw. die Latenz, die für das Weiterleiten eines Paketes benötigt wird, ist in diesem Fall abhängig von der Länge des zu übertragenden Paketes. Eine Cut-Through-Architekur hat eine geringere Latenz als eine Store-and-Forward-Architektur, weil sie die Pakete direkt weiterleitet, sobald sie die Zieladresse eingelesen und den Ausgangsport identifiziert hat. Die Latenz ist nicht nur deutlich kürzer, sondern sie ist auch konstant. Allerdings werden bei einem Cut-Through-Switch aufgrund des fehlenden CRCs auch fehlerhafte Pa-

kete weitergeleitet, die das Verkehrsaufkommen im Netzwerk unnötig erhöhen. Dies betrifft nicht nur Pakete mit Bitfehlern, sondern auch die Fragmente, die nach einer Kollision übrig bleiben. Die Weiterleitung von Kollisionsfragmenten wird bei den Cut-Through-Architekturen verhindert, die die ersten 64 Byte puffern, bevor sie das Paket weiterleiten. Eine solche Architektur wird auch als *Fragment-Free Cut-Through* bezeichnet.

Echtzeitkommunikation mit Switched-Ethernet: Obwohl sämtliche Kollisionen bei Switched-Ethernet eliminiert wurden, ist Switched-Ethernet nicht vollständig deterministisch. Sind mehre zeitgleich eintreffende Pakete an denselben Ausgangsport gerichtet, entstehen Pufferwartezeiten oder sogar ein Überlauf des Sendepuffers. Die Pufferwartezeiten erhöhen die Übertragungszeit eines Paketes und sind vom Verkehrsaufkommen abhängig, wodurch sie im Voraus nicht eindeutig bestimmt werden können. Deshalb sind auch bei Switched-Ethernet weitere Kontroll- und Steuermechanismen für eine deterministische Echtzeitkommunikation notwendig.

Im Wesentlichen werden vier Ansätze verfolgt, um den Determinismus von Switched-Ethernet zu verbessern. Der erste Ansatz basiert auf einer Priorisierung der harten Echtzeitpakete, um diese ohne Verzögerung durch andere Pakete weiterzuleiten. Standardisiert wurde dieser Ansatz von der IEEE und wird in Abschnitt 2.4.3 gesondert behandelt. Die bereits beim CSMA/CD eingesetzten Verfahren, wie z. B. Master-Slave und TDMA, werden im zweiten Ansatz auf Switched-Ethernet angewendet. Bei dem dritten Ansatz werden Ablaufplanungsverfahren, die auch aus dem Bereich der Echtzeitbetriebssysteme stammen, zur deterministischen Übertragung über Swichted-Ethernet verwendet. Für die Echtzeitkommunikation werden bei dem vierten Ansatz die Protokolle aus den höheren Schichten, wie z. B. TCP/IP und UDP/IP, verwendet.

Sowohl im industriellen als auch im akademischen Bereich sind TDMA-Ansätze zur Echtzeitkommunikation über Switched-Ethernet entwickelt worden. PROFINET (Kapitel 2.5.5) und Ethernet Powerlink (Kapitel 2.5.4) sind zwei Beispiele, die bereits zum industriellen Standard wurden. Aus dem akademischen Bereich stammt das *Time-Triggered Ethernet (TTE)* [KAGS05]. Dabei handelt es sich um eine Portierung des bereits vorgestellten zeitgesteuerten TTP/C-Protokolls auf Ethernet. Bei TTE sind alle Teilnehmer sternförmig an einen Switch bzw. für eine fehlertolerante Variante an zwei Switches angeschlossen. Der Switch basiert nicht auf einer Standardkomponente, sondern ist eine eigene Entwicklung, die in [Ste06] beschrieben wird. Standard-Ethernet-Pakete werden vom Switch im Store-and-Forward-Modus und die zeitgesteuerten TT-Pakete werden nach dem Cut-Through-Verfahren weitergeleitet. Der Ablauf der zeitgesteuerten Kommunikation muss im Vorfeld geplant werden. Damit ein normales Ethernet-Paket keine TT-Pakete stört, wird die Übertragung eines Ethernet-Paketes vom Switch abgebrochen, um das TT-Paket umgehend zu übertragen. Das

Ethernet-Paket wird bei der nächsten Gelegenheit erneut übertragen. Auch die Netzwerkschnittstellen der Teilnehmer benötigen einen speziellen TT-Kontroller, um TT-Pakete verarbeiten zu können. Ein reines Master-Slave-Protokoll ist das ebenfalls in der Industrie standardisierte EtherCAT (Kapitel 2.5.6). Eine Umsetzung des zeitgesteuerten Master-Slave-Protokolls FTT-Ethernet auf Switched-Ethernet wird in [PGAB05] vorgestellt.

Ein in Echtzeitbetriebssystemen für die Ablaufplanung verwendeter Algorithmus wird in [Hoa04] zur Echtzeitkommunikation über Switched-Ethernet verwendet. In diesem Fall wird ein Netzwerk mit einem Switch betrachtet, an dem alle Knoten angeschlossen sind. Die Knoten reservieren sogenannte Echtzeitkanäle, über die sie ihre zeitkritischen Daten übertragen. Der *Earliest Deadline First (EDF)*-Algorithmus wird verwendet, um die Reihenfolge zu bestimmen, in der die Pakete vom Knoten und vom Switch gesendet werden, um so die Zeitschranken der Daten einzuhalten.

Auf die höheren Protokollschichten TCP/IP und UDP/IP setzen die industriellen Protokolle Modbus/TCP (Kapitel 2.5.2) und EtherNet/IP (Kapitel 2.5.3) auf. Auch Löser und Härtig verwenden zur Übertragung von Echtzeitdaten UDP/IP [LH04]. Eine obere Schranke für die Pufferwartezeit wird in diesem Fall über eine Glättung des Datenverkehrs (*Traffic Shaping*) bestimmt. Weiterführende Informationen zur Verwendung von Ethernet mit TCP/IP sind in [Fur03] zu finden.

Einige Erweiterungen von Switched-Ethernet sind in die IEEE Standards mit eingeflossen und werden von mehreren Echtzeitprotokollen angewendet. Die drei wichtigsten Erweiterungen werden im Folgenden vorgestellt.

2.4.2 VLAN – Ein virtuelles lokales Netz

Mit dem Konzept eines virtuellen LANs (engl.: Virtual LAN (VLAN)) wird ein physikalisches Netzwerk in mehrere logisch getrennte Netzwerksegmente aufgeteilt. In der Echtzeitkommunikation werden VLANs eingesetzt, um die Leistungsfähigkeit und die Sicherheit zu erhöhen. Ein Problem sind Broadcast-Nachrichten, die für die Dauer ihrer Übertragung das gesamte Netzwerk blockieren. In einem VLAN bleiben die Broadcast-Nachrichten in dem Netzwerksegment, in dem sie entstanden sind. In den anderen Segmenten kann in diesem Fall weiter kommuniziert werden, wodurch sich der Durchsatz im gesamten Netzwerk erhöht. Zusätzliche Sicherheit wird dadurch eingebracht, dass in einem VLAN nur die Teilnehmer miteinander kommunizieren können, die zum selben virtuellen Netz gehören.

Die formale Definition für VLAN stellt der IEEE 802.1Q Standard [IEE03] zur Verfügung. Für Ethernet folgt daraus die in Abbildung 2.10 dargestellte Erweiterung des Ethernet-Paketformats um das vier Byte große *VLAN-Tag*. Die ersten zwei Bytes beinhalten die VLAN-Protokollkennung, die immer den Wert 0x8100 besitzt. Eine

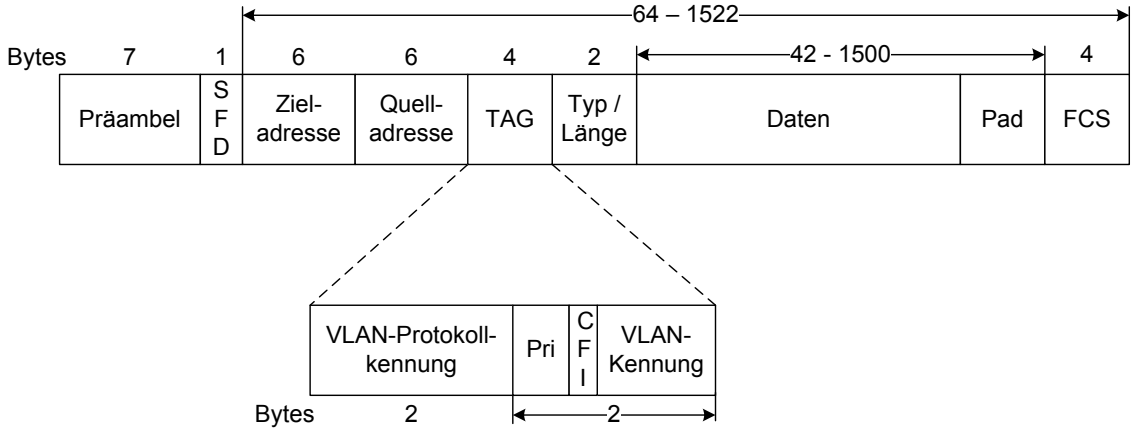


Abbildung 2.10: Ethernet-Paketformaterweiterung für VLAN

von einem Switch erkannte VLAN-Protokollkennung weist darauf hin, dass die darauf folgenden zwei Byte die VLAN-Kontrollinformationen beinhalten. Diese bestehen aus einem drei Bit großen Prioritätsfeld, welches die Priorität eines Paketes kennzeichnet. Für das VLAN hat dieses Feld jedoch keine Bedeutung. Das CFI-Bit (*Canonical Format Indicatior*) kennzeichnet ein in Ethernet eingebettetes Token-Ring-Paket, wenn es gesetzt ist. Die Zugehörigkeit zu einem bestimmten VLAN ergibt sich schließlich aus der zwölf Bit breiten VLAN-Kennung (wird auch VLAN-ID genannt). Mithilfe der VLAN-Kennung ermittelt der Switch den Port, an den er das Paket weiterleitet.

2.4.3 Priorisierung nach IEEE 802.1D

Normalerweise werden bei Switched-Ethernet alle Pakete gleich behandelt und in der Reihenfolge versendet, in der sie den Switch erreicht haben. Um die Pufferwartezeiten für Echtzeitpakete zu verringern, erhalten diese Pakete eine höhere Priorität und werden vom Switch bevorzugt weitergeleitet. Der IEEE 802.1D Standard (vorher 802.1p) [IEE04] definiert 8 Prioritätsklassen, deren empfohlener Verwendungszweck in Tabelle 2.2 aufgelistet ist. Eine gewählte Prioritätsklasse entspricht der in Kapitel 2.3.1 beschriebenen Dienstklasse bzw. der CoS eines Paketes. Bestimmt wird die Dienstklasse eines Ethernet-Paketes über das drei Bit breite Prioritätsfeld des VLAN-Tags.

Die niedrigste Prioritätsklasse 1 ist für den Hintergrunddatenverkehr bestimmt, der die Operationen der anderen Anwendungen nicht stören soll. Ohne Priorisierung wird bei Ethernet die bestmögliche (*Best Effort*) Qualität der Übertragung angeboten. Damit die Kompatibilität dazu erhalten bleibt, wird auch der Standard Priorität 0 die bestmögliche Qualität zugeordnet. Die ausgezeichnete Qualität ist etwas besser als die bestmögliche Qualität. Mit Steuerdaten sind in der vierten Prioritätsklasse keine Sensor- und Aktordaten gemeint, sondern Steuerdaten für das Netzwerk, wie z. B. ei-

Priorität	Art des Datenverkehrs
1	Hintergrund Verkehr
2	Reserviert
0 (Standard)	Bestmöglich (Best Effort)
3	Ausgezeichnet (Excellent Effort)
4	Steuerdaten
5	Videodaten
6	Sprachdaten
7	Netzwerkmanagement

Tabelle 2.2: Zuordnungsempfehlung der Prioritäten nach IEEE 802.1D

ne Zugriffskontrolle. Die Kommunikation mit Echtzeitanforderungen beginnt ab einer Priorität von 5 und ist für die Übertragung von Videodaten und ab Priorität von 6 für die Übertragung von Sprachdaten vorgesehen. Die höchste Priorität hat das Netzwerkmanagement, um sicherzustellen, dass die Nachrichten, die zur Aufrechterhaltung des Netzwerkes dienen, auf jeden Fall übertragen werden.

Zur Umsetzung der Dienstklasse in einem Switch benötigt dieser acht Sendepuffer für jeden Ausgangsport, um alle Dienstklassen zu implementieren. Ankommende Pakete werden entsprechend ihrer Dienstklasse einem Sendepuffer eines Ausgangsports zugeordnet. Eine Ablaufplanung bestimmt die Reihenfolge, in der die Pakete aus den Sendepuffern übertragen werden. Allerdings werden bei vielen Switches nicht alle notwendigen Sendepuffer pro Port implementiert, um Speicher einzusparen. Die acht Dienstklassen werden in diesem Fall auf die geringere Anzahl an Puffern aufgeteilt.

Wie bereits erwähnt, geben die Dienstklassen keine Garantie für die Qualität der Übertragung. Aber eine Verringerung der Pufferwartezeiten für Echtzeitpakete ist mit ihnen möglich. Mithilfe von Modellen von realen Fertigungssystemen wird in [Jas02] der Einsatz und die Leistungsfähigkeit von Switched-Ethernet mit CoS-Unterstützung für die Echtzeitkommunikation auf Sensor- und Aktorebene untersucht. Durch die dort durchgeführten Analysen und Simulationen konnte für hoch priorisierte Pakete eine deutliche Verringerung der oberen Zeitschranke nachgewiesen werden. Garantien für Verzögerungszeiten bei einem priorisierten Switched-Ethernet konnten simulativ für einen Switch in [FJ05] nachgewiesen werden. Bei diesem Ansatz werden, wie schon bei [Hoa04], Echtzeitkanäle etabliert, mit deren Hilfe in diesem Fall die Pufferverzögerungen bestimmt werden.

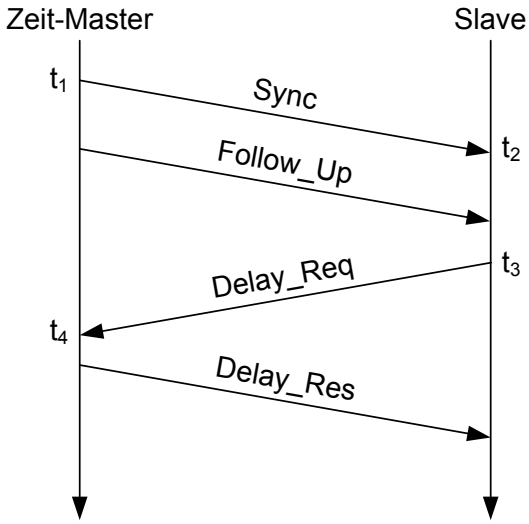


Abbildung 2.11: Ablauf der Uhrensynchronisation

2.4.4 Uhrensynchronisation nach IEEE 1588

Der IEEE 1588 Standard ist keine direkte Erweiterung von Switched-Ethernet, sondern beschreibt das *Precision Time Protocol (PTP)*, welches von vielen Ethernet-basierten Echtzeitprotokollen zur Uhrensynchronisation verwendet wird. Die Synchronisation von verteilten Uhren ist insbesondere bei zeitgesteuerten Echtzeitprotokollen notwendig, um eine gemeinsame Basis für den Beginn und das Ende eines Zeitschlitzes zu erhalten. Aber auch Anwendungen, die auf mehreren Knoten verteilt ausgeführt werden, erfordern eine Uhrensynchronisation, wenn sie z. B. bestimmte Prozesse zeitgleich starten.

Beim PTP gibt es eine Master-Slave-Beziehung, d.h., die verteilen Uhren synchronisieren sich mit einer zentralen Uhr. Als Master für die Synchronisation wird ein Knoten ausgewählt, der eine sehr genaue Uhr besitzt. Abbildung 2.11 illustriert den Ablauf der Synchronisation mit PTP. Die Synchronisation wird durch das Versenden der *Sync*-Nachricht vom Master an die Teilnehmer eingeleitet. Die Slaves merken sich die Ankunftszeit t_2 der *Sync*-Nachricht. Die Genauigkeit der Synchronisation wird über die *Follow_Up*-Nachricht erhöht. Sie enthält den genauen Zeitpunkt t_1, zu dem die *Sync*-Nachricht über die MII versendet wurde. Damit ein Slave seine Uhr genau auf die Masteruhr einstellen kann, muss die Verzögerungszeit der Verbindung zwischen Master und Slave t_{ld} zu t_1 addiert werden. Ermittelt wird t_{ld} über die *DelayReq*-Nachricht, die vom Slave zum Zeitpunkt t_3 versendet wird. Der Master antwortet dem Slave mit der *Delay_Res*-Nachricht, die den genauen Empfangszeitpunkt t_4 des *DelayReq* enthält. Unter der Annahme, dass die Verbindung in beiden Richtungen gleich lang ist, kann der Slave die Verbindungsverzögerung wie folgt berechnen:

$$t_{\mathrm{ld}} = \frac{(t_4 - t_1) - (t_3 - t_2)}{2} \tag{2.11}$$

Nachdem die Verbindungsverzögerung bekannt ist, werden vom Master in periodischen Abständen nur noch *Sync*- und *Follow_Up*-Nachrichten versendet, damit ein Auseinanderlaufen der lokalen Uhren vermieden wird. Den Korrekturfaktor t_{err} für die lokale Uhr kann der Slave nach Erhalt der *Follow_Up*-Nachricht mit $t_{\mathrm{err}} = t_2 - (t_1 + t_{\mathrm{ld}})$ bestimmen.

Die Präzision, d.h. die Genauigkeit, mit der die verteilten Uhren übereinstimmen, ist bei PTP abhängig von der Art der Implementierung des Zeitstempelns. Eine Softwarelösung erreicht eine Präzision von 10 μs, und bei einer Hardwarelösung liegt die Präzision unterhalb von einer Mikrosekunde [LL05]. Für manche Anwendungen in der Automatisierungstechnik ist noch eine höhere Präzision erforderlich. Bei PROFINET werden anstelle von *Boundary Clocks* die sogenannten *Transparent Clocks* verwendet. In einer Linien-Topologie führen die Boundary Clocks zu einer Kaskadierung der Kontrollschleife zwischen dem PTP-Master und dem PTP-Slave. Bei den transparenten Clocks entsteht nur eine Kontrollschleife zwischen dem Master und dem Slave. Hierdurch wird die Präzision erhöht [JSW04].

2.5 Standardisierte Echtzeitprotokolle

In den vorangegangenen Abschnitten sind bereits einige Ethernet-basierte Echtzeitprotokolle vorgestellt worden. Dieser Abschnitt befasst sich mit den Protokollen, die als industrieller Standard etabliert sind. Insgesamt sind bis zum Zeitpunkt der Erstellung dieser Arbeit 26 Ethernet-basierte Echtzeitprotokolle für das industrielle Umfeld entwickelt worden [Sch08]. Von den 26 Protokollen sind 11 durch die *International Electrotechnical Commision (IEC)* standardisiert worden [IEC07]. Fünf der wichtigsten Protokolle werden in diesem Unterkapitel kurz vorgestellt. Weitere Details zu den hier vorgestellten Protokollen sind in [Wil07] zu finden.

Der Aufbau der Protokolle unterscheidet sich in drei verschiedene Ansätze, die in der Abbildung 2.12 dargestellt sind. Beim ersten Ansatz erfolgt der Austausch von normalen Daten und Echtzeitdaten über TCP/IP oder UDP/IP. Das deterministische Übertragungsverhalten wird in der Anwendungsschicht implementiert. Verfolgt wird dieser Ansatz von den beiden Protokollen Modbus/TCP und EtherNet/IP. In den letzten beiden Ansätzen werden die TCP/UDP/IP Schichten übergangen und die deterministische Echtzeitübertragung direkt auf der Ethernet-Schicht aufgesetzt. Die Implementierung erfolgt bei Ethernet Powerlink und PROFINET RT nach dem zweiten Ansatz in Software und bei EtherCAT und PROFINET IRT nach dem dritten

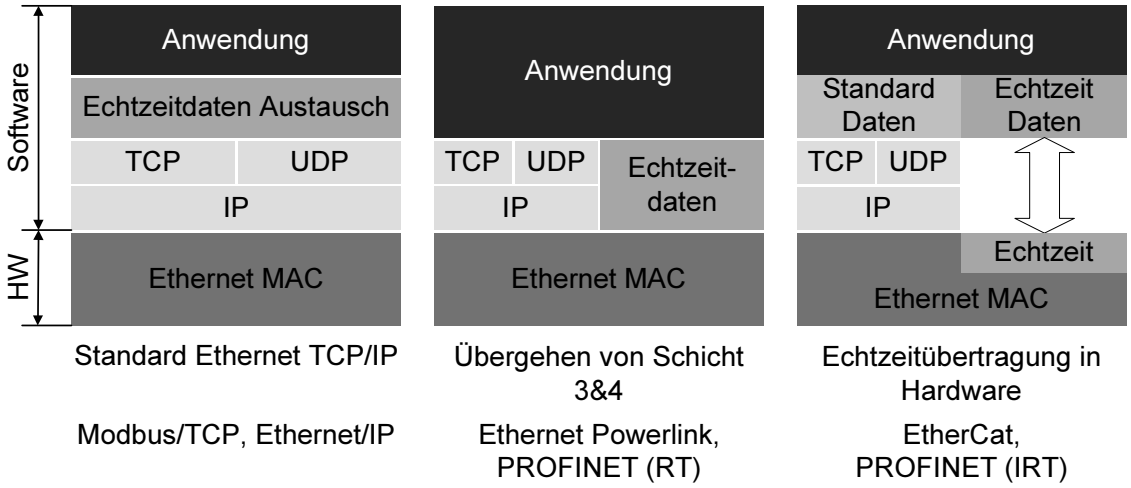


Abbildung 2.12: Ethernet-basierte Protokoll-Stapel (vgl. [LL05])

Ansatz in Hardware. Durch das Übergehen von Schicht 3 und 4 für die Echtzeitkommunikation werden die Verarbeitungszeiten durch den Protokollstapel und damit auch die Latenzzeit des gesamten Protokolls verringert. Eine Implementierung von Teilen des Protokolls in Hardware resultiert in einer weiteren Verringerung der Latenzzeit. In Kapitel 2.6 wird die Leistungsfähigkeit der hier beschriebenen Protokolle und der Einfluss des Protokollansatzes auf die Leistungsfähigkeit bewertet. Eingeteilt werden kann die Leistungsfähigkeit auch in verschiedene Echtzeitklassen, die von *Industrial Automation Open Networking Alliance (IAONA)* eingeführt wurden [LL05].

2.5.1 Leistungsklassen

Um eine Einteilung von Echtzeitnetzwerken in verschiedene Leistungsklassen hat sich die IAONA bemüht. Die in der Tabelle 2.3 aufgeführten Echtzeitklassen der IAONA sollten eine Grundlage zur einheitlichen Kennzeichnung der Leistungsfähigkeit von Echtzeitnetzen und deren Komponenten schaffen. Dazu sind insgesamt sieben Klassen definiert worden, die die Leistungsfähigkeit nach den drei Echtzeitparametern Latenz, Jitter und Bandbreite beschreiben.

Anwendungsgebiet für Netzwerke der Klasse A in allen drei Bereichen ist z. B. die Gebäudeautomatisierung. Beispielhafte Einsatzgebiete für die Klassen B bis D sind Förderanlagen und Werkzeugmaschinen. Die höchsten Anforderungen der Klassen E und F müssen Netzwerke z. B. im Bereich von synchronisierten Wellen von großen Druckstraßen erfüllen. Die meisten Echtzeitprotokolle erfüllen im Bereich der Bandbreite die Klasse B und die neuesten Weiterentwicklungen bereits die Klasse C.

Latenz (X)		**Jitter (Y)**		**Bandbreite (Z)**	
Keine Anforderungen	X	Keine Anforderungen	X		X
> 100 ms	A	> 100 ms	A	> 10 MBit/s	A
30 ms - 100 ms	B	1 ms- 10 ms	B	10 - 100 MBit/s	B
10 ms - 30 ms	C	100 μs - 1 ms	C	100 MBit/s - 1GBit/s	C
3 ms - 10 ms	D	10 μs - 100 μs	D	1 Gbit/s - 10 Gbit/s	D
1 ms - 3 ms	E	1 μs - 10 μs	E		E
300 μs - 1 ms	F		F		

Tabelle 2.3: IAONA Echtzeitklassifikation [LL05]

2.5.2 Modbus/TCP

Modbus/TCP ist ein Derivat des Modbus-Protokolls, welches bereits 1979 von der Firma Modicon für die Automatisierungstechnik entwickelt wurde. Ursprünglich für eine Kommunikation über serielle Verbindungen (RS-232, RS-485) entwickelt, basiert Modbus/TCP auf Ethernet und setzt direkt auf die TCP/IP-Schicht auf. Modbus/TCP bettet seinen eigenen Rahmen in den TCP/IP Rahmen ein (Abbildung 2.13) und nutzt die Schichten 3 und 4 für den Aufbau einer Verbindung. Die Kommunikation erfolgt nach dem *Client/Server*-Prinzip, d.h., der Client schickt immer eine Anfrage an den Server und dieser sendet das Antwort-Paket. Zwischen Client und Server werden über TCP eine oder mehrere Verbindungen aufgebaut, die bei zyklischer Kommunikation dauerhaft erhalten bleiben. Bei Modbus/TCP existiert kein Master, der die Kommunikation koordiniert. Lediglich die TCP/IP-Mechanismen werden zur Kommunikationskontrolle verwendet.

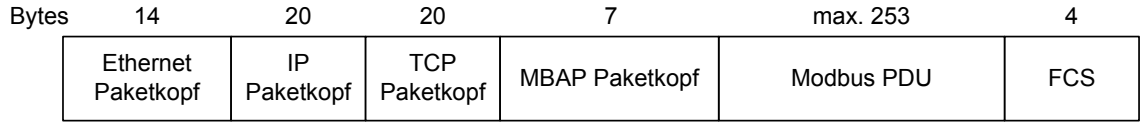


Abbildung 2.13: Aufbau eines Modbus/TCP-Paketes

Bei einem Modbus/TCP-Paket werden, wie in Abbildung 2.13 dargestellt, in den TCP/IP-Rahmen die zwei Felder *Modbus Applikation (MBAP)*-Paketkopf und der Modbus *Protocol Data Unit (PDU)* eingebettet [Mod04b]. Der MBAP-Paketkopf identifiziert den Client, das Paket als Modbus-Paket, die Länge der Daten und ordnet einer Anfrage die entsprechende Antwort zu. In der Modbus-PDU wird der vom Client angeforderte Befehl und die dazugehörigen Daten übertragen.

Modbus/TCP stellt keine besonderen Anforderungen an die Hardware und kann mit jeder Standard-Ethernet-Komponente implementiert werden. Im Vergleich mit den

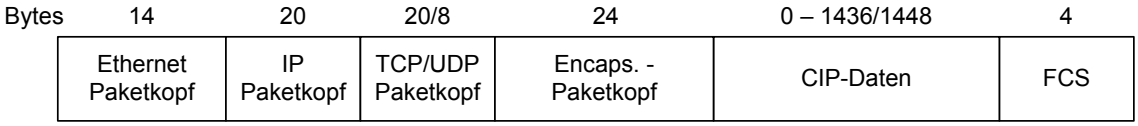


Abbildung 2.14: Aufbau eines EtherNet/IP-Paketes

hier vorgestellten fünf Echtzeitprotokollen erfüllt Modbus/TCP die geringsten Echtzeitanforderungen im Bezug auf Jitter und Latenz.

2.5.3 EtherNet/IP

Das *EtherNet Industrial Protocol*, kurz EtherNet/IP, wurde zur Anbindung von Feldbussen an das Unternehmens-Intranet über Standard-Ethernet mit TCP/UDP/IP von den drei Nutzerorganisationen *Open Device Vendor Association (ODVA)*, *ControlNet International* und der *Industrial Ethernet Association (IEA)* definiert [Eth01]. Die einfache Anbindung der Feldbusse DeviceNET und ControlNet an EtherNet/IP wird über ein gemeinsames Protokoll der Anwendungsschicht, dem *Control and Information Protocol (CIP)*, sichergestellt. Das CIP unterscheidet zwischen *expliziten* und *impliziten* Nachrichten. Bei der expliziten Nachricht werden Daten nur zwischen zwei Anwendungsprozessen ausgetauscht. Im Falle von EtherNet/IP verwenden explizite Nachrichten das TCP, um die Zielanwendung zu adressieren. Im Gegensatz dazu werden implizite Nachrichten bei Ethernet/IP über UDP versendet. Implizite Nachrichten basieren auf dem *Producer/Consumer*-Modell und werden für die zyklische Echtzeitkommunikation eingesetzt. Das Producer/Consumer-Modell ähnelt der nachrichtenorientierten Adressierung bei CAN. Der Producer sendet seine Nachricht als Broadcast an alle Teilnehmer oder als Multicast an eine Gruppe von Teilnehmern. Gelesen wird eine Nachricht nur von den Consumern, die sich für die Daten interessieren. Gegenüber dem Client-Server-Prinzip reduziert das Producer/Consumer-Modell den Kommunikationsaufwand, weil auf die explizite Anfrage der Daten verzichtet wird. Des Weiteren können ohne Kenntnisnahme des Producers weitere Consumer dem Netzwerk hinzugefügt werden [Fur03].

Eine CIP-Nachricht wird bei EtherNet/IP in den Datenteil eines TCP- oder UDP-Paketes eingebettet, woraus das in Abbildung 2.14 dargestellte Paketformat resultiert. Die eingebettete CIP-Nachricht besteht aus dem Encapsulation-Paketkopf und den CIP-Daten. Mithilfe des Paketkopfes wird die Länge der Daten identifiziert und es können Kommandos und Statusinformationen über die Ausführung eines Kommandos übertragen werden.

Ein EtherNet/IP-Netzwerk kann mit Standard-Ethernet-Komponenten aufgebaut werden. Allerdings können mit einer Priorisierung der zyklischen Echtzeitkommunikation

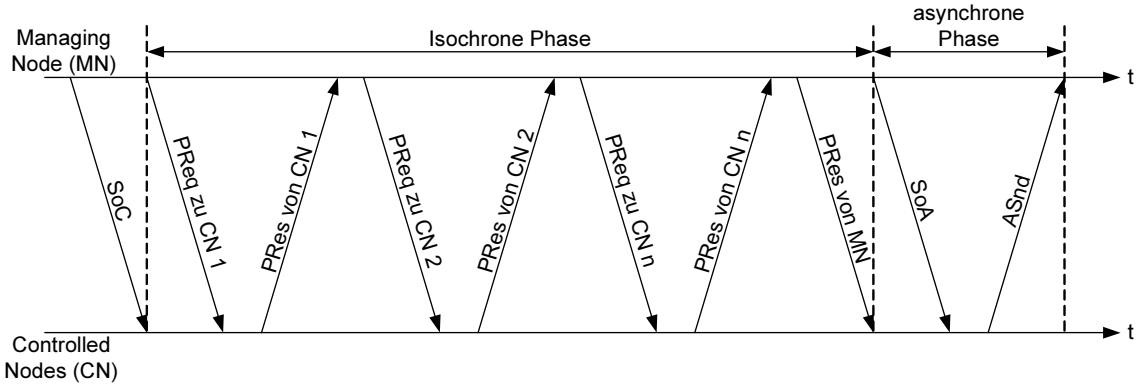


Abbildung 2.15: Ablauf eines Ethernet Powerlink-Zyklus (vgl. [EPL03])

nach IEEE 802.1D und mit der Erweiterung *CIPSync*, die eine Synchronisation der lokalen Uhren nach IEEE 1588 ermöglicht, höhere Echtzeitanforderungen erfüllt werden. Daher sollten Netzwerkschnittstellen und Switches verwendet werden, die ebenfalls die IEEE-Standards 802.1D und 1588 unterstützen.

2.5.4 Ethernet Powerlink

Ethernet Powerlink wurde ursprünglich vom Steuerungshersteller Bernecker & Reichner Industrie-Elektronik entwickelt und bis zu seiner Offenlegung im Jahre 2002 nur für die Vernetzung der eigenen Geräte verwendet. Die Spezifikation [EPL03] des Protokolls wird seitdem von der *Ethernet Powerlink Standardization Group (EPSG)* weiter entwickelt.

Im Gegensatz zu Modbus/TCP und Ethernet/IP wird die Echtzeitkommunikation bei Ethernet Powerlink nicht von der Anwendungsschicht sondern von der Sicherungsschicht aus gesteuert und das Protokoll setzt direkt auf Ethernet auf. Das Verfahren zur Steuerung der Echtzeitkommunikation heißt bei Ethernet Powerlink *Slot Communication Network Management (SCNM)*. Es handelt sich dabei um ein zeitschlitzbasiertes Verfahren, das von einem zentralen Knoten, dem *Managing Node (MN)*, gesteuert und überwacht wird. Die anderen Kommunikationsteilnehmer werden als *Contolled Node (CN)* bezeichnet. Der Ablauf der Kommunikation bei SCNM erfolgt in periodischen Buszyklen (Abbildung 2.15) mit fester Intervalllänge und wird im Vorfeld geplant. Jeder Zyklus beginnt mit der *Start of Cyclic (SoC)*-Nachricht, die vom MN als Broadcast an alle anderen Teilnehmer gesendet wird. Die CNs synchronisieren sich mithilfe der SoC-Nachricht, speichern ihre Sensorwerte und erstellen ihr Antwortpaket. In der darauf folgenden isochronen Phase werden die CNs nacheinander vom MN abgefragt. Jeder CN beantwortet die an in gestellte Anfrage (*PollRequest*) mit seinem Antwortpaket (*PollResponse*), das als Broadcast an alle Teilnehmer gesendet wird. Abgeschlos-

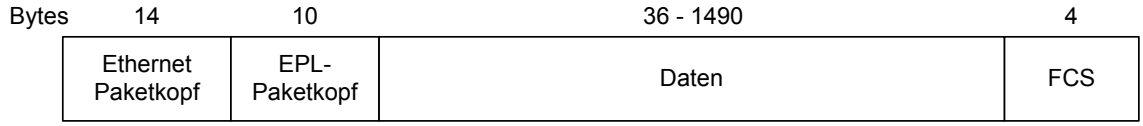


Abbildung 2.16: Aufbau eines Ethernet Powerlink-Paketes

sen wird die isochrone Phase durch eine PollResponse des MN. Die darauf folgende asynchrone Phase wird vom MN durch die *Start of Asynchronous (SoA)*-Nachricht eingeleitet. In der asynchronen Phase können die CNs nicht echtzeitkritische Daten versenden, die dieses in ihrem Antwortpaket angefordert haben. Beendet wird der Buszyklus durch die Wartephase bis zum neuen Buszyklus.

Die Powerlink-Daten werden zusammen mit einem zusätzlichen 10 Byte großen Paketkopf direkt in das Nutzdatenfeld eines Ethernet-Paketes eingebettet. Der Ethernet Powerlink-Paketkopf beschreibt den Nachrichtentyp, die Quelle, das Ziel und die Länge der Ethernet Powerlink-Nachricht. Das Paketformat eines Ethernet Powerlink-Paketes zeigt die Abbildung 2.16.

Auch ein Ethernet Powerlink-Netzwerk kann mit Standard-Ethernet-Komponenten aufgebaut werden. Empfohlen wird aber die Verwendung von Hubs, die bei Broadcast-Nachrichten zu einer Verringerung der Latenzen führen. Auch bei der Verwendung von Switches wird im Halbduplex-Modus mit einer Datenrate von 100 MBit/s übertragen. Des Weiteren unterstützt Ethernet Powerlink auch eine Uhrensynchronisation nach IEEE 1588.

2.5.5 PROFINET

PROFINET ist eine Ethernet-Lösung der *Profibus Nutzerorgansisation (PNO)* und wurde als übergeordnetes Kommunikationssystem für den Profibus entwickelt. Es wird zwischen PROFINET CBA (Component Based Automation) und PROFINET IO unterschieden. Bei PROFINET CBA wird eine Automatisierungsanlage aus vorgefertigten Komponenten mit einer festen Funktion zusammengestellt. Im Wesentlichen ist PROFINET CBA ein Ansatz für den Anlagenentwurf und den Anlagenbetrieb. Die Konzepte für die Echtzeitkommunikation auf der Feldebene werden von PROFINET IO zur Verfügung gestellt. Die folgende Betrachtung der Kommunikation von PROFINET bezieht sich daher auch auf PROFINET IO.

Für PROFINET wurden insgesamt drei Leistungsklassen entwickelt. In der ersten Klasse kommuniziert auch PROFINET über die TCP/UDP/IP-Schicht. Erst später wurde PROFINET für den Einsatz im Sensor-/Aktor-Bereich erweitert, und es entstanden zwei neue Leistungsklassen, namentlich *Real-Time (RT)* und *Isochronous Real-Time (IRT)*. Die Echtzeitkommunikation wird bei den beiden neuen Leistungs-

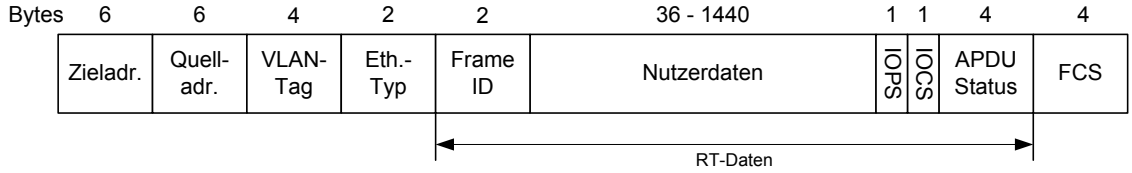


Abbildung 2.17: Aufbau eines PROFINET RT Paketes

klassen direkt von der Sicherungsschicht aus gesteuert. Bei PROFINET RT setzt die Kommunikationssteuerung direkt auf Ethernet auf, und bei PROFINET IRT wird die Steuerung von einer speziellen Hardware übernommen. In einem PROFINET-Netzwerk erfüllen die drei Leistungsklassen unterschiedliche Aufgaben. Die Kommunikation über TCP/UDP/IP wird für den nicht echtzeitkritischen Datentransfer verwendet, wie z. B. der Parametrierung der Knoten. Der zyklische Austausch von Prozessdaten erfolgt überwiegend über die softwaregestützte RT-Kommunikation. Nur bei sehr hohen Echtzeitanforderungen mit Zykluszeiten von 1 ms und einem Jitter kleiner 1 μs, wie sie z. B. bei der synchronen Steuerung mehrerer Motoren entstehen, wird der Einsatz von PROFINET IRT erforderlich [PRO06].

PROFINET RT: Die zyklischen Daten bei PROFINET RT werden verbindungsorientiert ohne Bestätigung zwischen *Providern* und *Consumern* in einem vorher festgelegten Zeitraster übertragen. Das *Provider/Consumer*-Modell ist prinzipiell mit dem *Producer/Consumer*-Modell des Ethernet/IPs identisch. Bei beiden Modellen können mehrere Consumer Nachrichten eines Providers lesen, wenn diese als Multicast oder Broadcast gesendet werden. Die Kommunikationsmodelle sind aber nicht durch dieselben Algorithmen implementiert.

Eingebettet werden die Echtzeitdaten bei PROFINET RT direkt in das Datenfeld des Ethernet-Rahmens (Abbildung 2.17). Grundsätzlich besitzt jedes RT-Paket das VLAN-Tag nach IEEE 802.Q, mit dem RT-Pakete gegenüber NRT-Paketen priorisiert werden können. Das Ethernet-Typfeld kennzeichnet ein RT-Paket und besitzt immer den Wert *0x8892*. Falls es vom Switch unterstützt wird, kann auch der Ethernet-Typ zur bevorzugten Behandlung eines RT-Paketes verwendet werden. Mit der *Frame_ID* werden die nachfolgenden Daten näher beschrieben. Im Anschluß an die Nutzdaten folgen noch die Statusfelder, *IO-Data Object Producer Status (IOPS)*, *IO-Data Object Consumer Status (IOPS)* und *Application Protocol Data Unit Status (APDU)*, die Informationen über die Aktualität und die Gültigkeit der Daten enthalten [Pop05]. In einem PROFINET RT Netzwerk können Standard-Ethernet-Komponenten eingesetzt werden, die eine Vollduplex-Übertragung mit einer Datenrate von 100 MBit/s gewährleisten und eine Priorisierung von Daten unterstützen.

PROFINET IRT: Bei RT können noch unerwartete Verzögerungen in den Switches entstehen, z. B. durch die Übertragung eines nicht echtzeitkritischen Paketes, die zu Schwankungen von bis zu 10 ms führen können [Pop05]. Bei IRT werden unerwartete Verzögerungen durch einen Zeitschlitz-basierten Buszyklus ausgeschlossen. Dieser Buszyklus besteht aus den drei Phasen Synchronisation, deterministische Kommunikation und offene Kommunikation. In der Synchronisationsphase werden die lokalen Uhren der Teilnehmer und der Switches mit dem *Precision Time Clock Protocol (PTCP)* synchronisiert. Das nach *International Electrotechnical Commision (IEC)* 61158 [IEC06] standardisierte PTCP ist eine Erweiterung des PTP und erreicht eine größere Genauigkeit bei der Uhrensynchronisation. Während der deterministischen Kommunikation dürfen nur IRT-Pakete übertragen und von den Switches weitergeleitet werden. Die Zeitschlitze, in denen ein Teilnehmer oder ein Switch senden darf, werden in der Projektierung festgelegt. In der offenen Phase können sowohl RT- als auch TCP/IP-Daten übertragen werden. Die Dauer der deterministischen und der offenen Phase kann im Voraus variabel bestimmt werden. Das in Abbildung 2.17 dargestellte Paketformat eines RT-Paketes ist nahezu identisch mit der Paketformat eines IRT-Paktes. Beim IRT-Paket entfällt im Gegensatz zum RT-Paket das Feld für den VLAN-Tag. Die Kommunikation mit IRT erfordert eine spezielle Hardwareunterstützung. Ein Beispiel für eine solche Hardwareunterstützung ist der ERTEC400 (*Enhanced Real-Time Ethernet Controller*) [Ert06]. Der ERTEC400 ist ein spezieller ASIC (Application Specific Integrated Circuit), der als PROFINET-Switch von der Firma Siemens entwickelt wurde. Eine Besonderheit ist die hybride Switcharchitektur, die IRT-Pakete nach dem Cut-Through-Verfahren weiterleitet. Alle anderen werden nach dem Store-and-Forward-Verfahren weitergeleitet.

2.5.6 EtherCAT

EtherCAT wurde von der Firma Beckhoff Industrie Elektronik entwickelt und erstmalig 2003 auf der Hannover-Messe vorgestellt [JB03a]. Um die weitere Verbreitung von EtherCAT zu unterstützen, wurde die *EtherCAT Technology Group (ETG)* gegründet, die auch für die Veröffentlichung der EtherCAT-Spezifikation [Eth04] zuständig ist.

Die Kommunikation bei EtherCAT erfolgt nach dem *Master-Slave*-Prinzip. Auf der Ethernet-Ebene treten alle EtherCAT-Slaves als ein gemeinsamer Teilnehmer auf, an den der Master seine Pakete sendet. Die gesendeten Pakete durchlaufen jeden Slave und kehren am Ende wieder zum Master zurück. Die Slaves extrahieren die für sie bestimmten Eingabewerte und schreiben ihre Ausgabewerte, während das Paket durch den Slave hindurchläuft. Die Pakete müssen nicht extra gespeichert und dann bearbeitet werden, wodurch die Paketverzögerung durch den Slave nur wenige Bitzeiten beträgt. Die Slaves werden im Allgemeinen in einer Linie verschaltet, die ein Paket

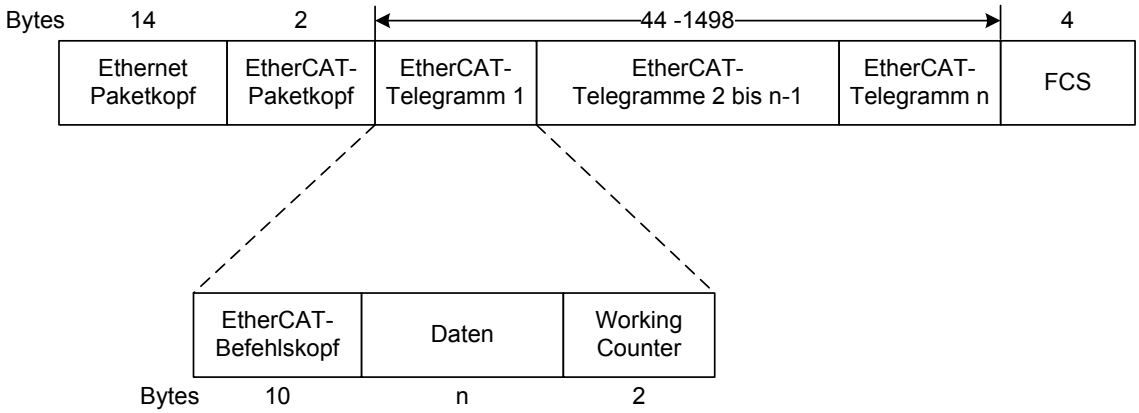


Abbildung 2.18: Aufbau eines EtherCAT-Paketes

bis zum letzten Slave durchläuft. Der letzte Slave in der Linie verbindet die Hinleitung mit der Rückleitung, sodass das Paket wieder durch alle Slaves bis zum Master zurückläuft. Bei einer Querverzweigung durchläuft das Paket zuerst die Stichleitung. Sobald das Paket das Ende der Stichleitung erreicht hat, kehrt es zurück und läuft dann weiter durch die Hauptlinie. Der Master und die Slaves bilden so immer einen virtuellen Ring und müssen in beide Richtungen kommunizieren.

Auch bei EtherCAT werden die Prozessdaten in das Netzdatenfeld eines Standard-Ethernet-Paketes eingebettet. Die Abbildung 2.18 zeigt das EtherCAT-Paketformat mit den zwei EtherCAT-spezifischen Bereichen, nämlich dem konstanten EtherCAT-Paketkopf, der die Nutzdatenlänge und den Paketkopf identifiziert und einem oder mehreren EtherCAT-Telegrammen, die die Prozessdaten enthalten. Ein EtherCAT-Telegramm setzt sich immer aus dem Befehlskopf, den Daten und einem *Working Counter* zusammen. Der Befehlskopf enthält im Wesentlichen die Adressierung der Slaves und den Befehl, der von den Slaves ausgeführt wird. Auf die eigentlichen Prozessdaten im Datenfeld folgt der Working Counter, der von jedem Slave inkrementiert wird, der das Telegramm erfolgreich bearbeitet hat.

Standard-Ethernet-Komponenten können nur für den Master eingesetzt werden. Die Bearbeitung der Pakete, während sie durch den Slave fließen, erfordert eine spezielle Hardware, wie z. B. den ET1200 [Eth08] von der Firma Beckhoff. Zur Synchronisation der lokalen Uhren verwendet EtherCAT ein eigenes Verfahren, welches die virtuelle Ringstruktur ausnutzt. Für die externe Uhrensynchronisation, z. B. zwischen unterschiedlichen Netzen, wird auch der IEEE-Standard 1588 verwendet [Eth05].

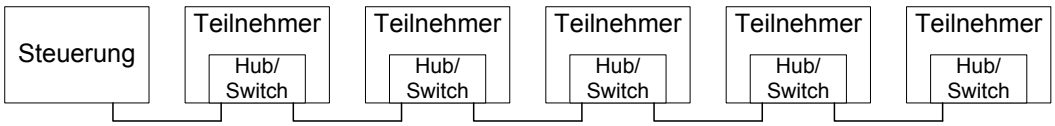


Abbildung 2.19: Untersuchte Linientopologie

2.6 Leistungsbewertung von Echtzeitprotokollen

Die Leistungsfähigkeit der auf den vorangegangenen Seiten beschriebenen industriellen Echtzeitprotokolle wird in diesem Abschnitt anhand ihrer Protokolleigenschaften bewertet. Als Maß für die Leistungsfähigkeit der Echtzeitprotokolle werden die minimalen Zykluszeiten und die Protokolleffizienz mithilfe eines Matlab-Modells untersucht. Das Modell basiert auf den in [Wil07] gemachten Arbeiten und wurde an die Anforderungen dieser Untersuchung angepasst.

2.6.1 Berechnung der Leistungsfähigkeit

Bevor die Leistungsfähigkeit der einzelnen Protokolle bewertet wird, erfolgt an dieser Stelle eine Einführung in die grundlegende Berechnung der minimalen Zykluszeit und der Protokolleffizienz. Damit die berechneten Werte der einzelnen Protokolle miteinander vergleichbar sind, wird ein einheitliches Szenario gewählt, in dem eine Steuerung Daten an alle Teilnehmer sendet. Jeder Teilnehmer antwortet der Steuerung, in dem er Daten zur Steuerung zurücksendet. Die Kommunikation erfolgt zyklisch und ein Zyklus gilt als abgeschlossen, sobald jeder Teilnehmer Daten von der Steuerung erhalten hat und alle Daten der Teilnehmer von der Steuerung empfangen wurden. Die Nutzdatenmenge, die jeder Teilnehmer empfängt und sendet, ist gleich. Es wird nur der zyklische Echtzeitdatentransfer untersucht, da die Echtzeiteigenschaften hier von Interesse sind. Des Weiteren sind zum Zeitpunkt der Untersuchung bereits alle Verbindungen aufgebaut. Die Steuerung und die Teilnehmer sind, wie in der Abbildung 2.19 dargestellt, in einer Linientopologie angeordnet. Die Kabellänge zwischen jedem Knoten beträgt 10 m.

Minimale Zykluszeit: Bei der Echtzeitkommunikation ist die minimal erreichbare Zykluszeit abhängig von der Summe der Verzögerungszeiten, die bei der Aktualisierung aller Knoten im Netzwerk entstehen. Die einzelnen Verzögerungen, die bei der Übertragung eines Paketes entstehen, sind die in Kapitel 2.2.1 beschriebene Übertragungsverzögerung und die Ausbreitungsverzögerung. Zusätzlich wird jedes Paket noch durch die Netzwerkkomponenten, die das Paket weiterleiten, und durch die Verarbeitung des Protokollstapels verzögert.

Die Verzögerung beim Weiterleiten eines Paketes hängt von der verwendeten Netzwerkkomponente ab.

- **Der Hub** hat die geringste Verarbeitungszeit, weil er das Ethernet-Signal nur verstärkt direkt an alle Ausgangsports weiterleitet. Als Verzögerungszeit des Hubs T_{Hub} wird für die Leistungsbewertung von Ethernet Powerlink ein Wert von 0,5 μs [EPL03] verwendet. Ethernet Powerlink ist das einzige Protokoll, bei dem Hubs bevorzugt verwendet werden.

- **Der Cut-Through-Switch** muss zunächst die Zieladresse des Paketes empfangen haben, bevor er das Paket an den korrekten Ausgangsport weiterleiten kann. Daher ist die Verarbeitungszeit des Cut-Through-Switches T_{Cut} höher als beim Hub. Beim ERTEC400 beträgt diese Verzögerungszeit 3 μs [Pop05]. Dieser Wert wird bei der Berechnung der Leistungsfähigkeit von PROFINET verwendet. Von den fünf Protokollen ist es das Einzige, das ausdrücklich einen Cut-Through-Switch verwendet.

- **Der Store-and-Forward-Switch** speichert beim Empfang das gesamte Paket, bevor es verarbeitet und weitergeleitet wird. Daher ist die Verzögerungszeit $T_{\text{ST\&FW}}$ abhängig von der Länge des Paketes und wird wie folgt berechnet:

$$T_{\text{ST\&FW}} = T_{\text{Trn}} + T_{\text{Swi}}. \tag{2.12}$$

  Die Zeit, die ein Switch für den Empfang eines Paketes benötigt, ist gleich der Übertragungsverzögerung T_{Trn} eines Paketes. Bei Ethernet mit einer Datenrate von 100 MBit/s beträgt die Übertragungsverzögerung von einem Paket minimaler Länge 5,76 μs und von einem Paket maximaler Länge 122,08 μs. Nach dem vollständigen Empfang eines Paketes wird angenommen, dass ein Paket erst nach einer Paketverarbeitungszeit des Store-and-Forward-Switches T_{Swi} von 2,5 μs weitergeleitet wird. Dieser Wert wurde bei einer Messung in [SZ03] für einen Store-and-Forward-Switch ermittelt. Messungen von verschiedenen Ethernet-Switches [JG07], die in der Automatisierungsindustrie eingesetzt werden, kommen zu ähnlichen Werten für die Paketverarbeitungszeit und bestätigen die hier gemachte Annahme.

Für die Verarbeitungsverzögerung eines TCP/UDP/IP-Protokollstapels T_{Stack} wird der in [LL05] angegebene Wert von 200 μs verwendet. Eine für alle Protokolle gültige Gleichung kann an dieser Stelle nicht angegeben werden. Es ist von dem jeweiligen Protokoll abhängig, wie oft Pakete übertragen werden müssen, um alle Knoten zu aktualisieren.

Protokolleffizienz: Die Protokolleffizienz S ist ein Maß für den Anteil der Nutzdaten, die tatsächlich vom Protokoll übertragen werden. Für die Leistungsbewertung wird die Protokolleffizienz definiert als das Verhältnis aus der Menge der Nutzdaten N und aus der Menge der tatsächlichen Daten D, die innerhalb eines Zyklus übertragen werden. In Prozent angegeben wird die Protokolleffizienz wie folgt bestimmt:

$$S = \frac{N}{D} \cdot 100\% \tag{2.13}$$

2.6.2 Modbus/TCP

Aufgrund des von Modbus/TCP verwendeten Client/Server-Konzeptes müssen alle Daten der Teilnehmer (Server) von der Steuerung (Client) einzeln nacheinander abgefragt werden. Im idealen Fall fordert die Steuerung alle Daten der Teilnehmer direkt nacheinander über den *Read-Request* an. Im Anschluss daran sendet die Steuerung ihre Daten zu den Teilnehmern über den *Write-Request*. Dabei wird eine Request-Nachricht im idealen Fall als Erstes an den entferntesten Teilnehmer in der Linie versendet und als Letztes an den nächstliegenden Teilnehmer. Sobald ein Teilnehmer einen *Read-Request* erhalten hat, sendet der Teilnehmer seine Daten über den *Read-Response* an den Server. Die vom Server erhaltenen Daten bestätigt der Teilnehmer, sobald er die Daten erhalten hat, mit einem *Write-Response*. Die verschiedenen Nachrichtentypen haben unterschiedliche Paketlängen. Daher wird für die Berechnung der Zykluszeit jede Übertragungsverzögerung einzeln angegeben. Die Übertragungsverzögerung von der *Read-Request*-Nachricht ist T_{RdReq}, für die *Write-Request*-Nachricht T_{WrReq} und für die *Write-Response*-Nachricht T_{WrRes}.

Für die Bestimmung der Zykluszeit von Modbus/TCP in einer Linien-Topologie werden zwei Fälle unterschieden. Im ersten Fall ist die Verarbeitungszeit des Protokollstapels größer als die Übertragungsdauer eines Paketes vom entferntesten Teilnehmer bis zur Steuerung. Die *Write-Response*-Nachricht des Teilnehmers, der sich am nächsten an der Steuerung befindet, erreicht in diesem Fall als Letztes die Steuerung. Die Zykluszeit $T_{\mathrm{Cyc_1}}$ wird für diesen Fall mit

$$T_{\mathrm{Cyc_1}} = n \cdot (T_{\mathrm{RdReq}} + T_{\mathrm{WrReq}} + 2 \cdot T_{\mathrm{Stack}}) + 2 \cdot (T_{\mathrm{Prd}} + T_{\mathrm{Swi}}) + T_{\mathrm{WrRes}} + 2 \cdot T_{\mathrm{Stack}} \tag{2.14}$$

bestimmt.

Im zweiten Fall ist die Übertragungsdauer vom entferntesten Teilnehmer größer als die Verarbeitungszeit im Protokollstapel. Als Letztes erreicht in diesem Fall die *Write-*

Response-Nachricht des entferntesten Teilnehmers die Steuerung. Die Zykluszeit $T_{\mathrm{Cyc_2}}$ wird in diesem Fall mit

$$T_{\mathrm{Cyc_2}} = n \cdot (T_{\mathrm{RdReq}} + T_{\mathrm{Stack}} + T_{\mathrm{WrReq}} + T_{\mathrm{WrRes}}) + 2 \cdot n \cdot (T_{\mathrm{Prd}} + T_{\mathrm{Swi}}) + 2 \cdot T_{\mathrm{Stack}} \quad (2.15)$$

berechnet.

Die minimale Zykluszeit von Modbus/TCP ergibt sich aus dem Maximum der Gleichungen (2.14) sowie der Gleichung (2.15) und entspricht

$$T_{\mathrm{Cyc}} = \max\left(T_{\mathrm{Cyc_1}}, T_{\mathrm{Cyc_2}}\right). \quad (2.16)$$

Im Matlab-Modell werden die minimalen Zykluszeiten von Modbus/TCP für eine variierende Anzahl von Teilnehmern und für eine variierende Nutzdatenmenge ermittelt und in der Abbildung 2.20 dargestellt. Aufgrund der Store-and-Forward-Architektur steigen die minimalen Zykluszeiten mit der Menge der Nutzdaten. Die Skala in der Abbildung 2.20 reicht nur bis zu einer maximalen Nutzdatenmenge eines Schreibzugriffs von 246 Bytes. Die 253 Bytes des Datenfeldes (siehe Abbildung 2.13) werden durch eine Adressierung im Datenfeld auf 246 Bytes verringert [Mod04a].

Die Protokolleffizienz bei Modbus wird bestimmt durch:

$$S = \frac{N_{\mathrm{In}} + N_{\mathrm{Out}}}{D_{\mathrm{RdReq}} + D_{\mathrm{RdRes}} + D_{\mathrm{WrReq}} + D_{\mathrm{WrRes}}} \quad (2.17)$$

Dabei ist die maximale Nutzdatenmenge für die Eingangsdaten N_{In} und die Ausgangsdaten N_{Out} unterschiedlich groß. Sie beträgt für N_{In} maximal 246 Bytes und für N_{Out} 250 Bytes. Die tatsächliche Datenmenge ist abhängig von dem Nachrichtentyp und wird daher einzeln aufgelistet. In der Gleichung (2.17) wird die tatsächliche Datenmenge von der *Read-Request*-Nachricht repräsentiert durch D_{RdReq}, von der *Read-Response*-Nachricht durch D_{RdRes}, von der *Write-Request*-Nachricht durch D_{WrReq} und von der *Write-Response*-Nachricht durch D_{WrRes}.

Die Abbildung 2.21 zeigt die Protokolleffizienz von Modbus/TCP. Aufgrund der geringen maximalen Nutzdatenmenge und der großen Anzahl von Bytes, die für den TCP-, IP-, Modbus- und Ethernet-Paketkopf benötigt werden, ist die Protokolleffizienz von Modbus/TCP sehr gering.

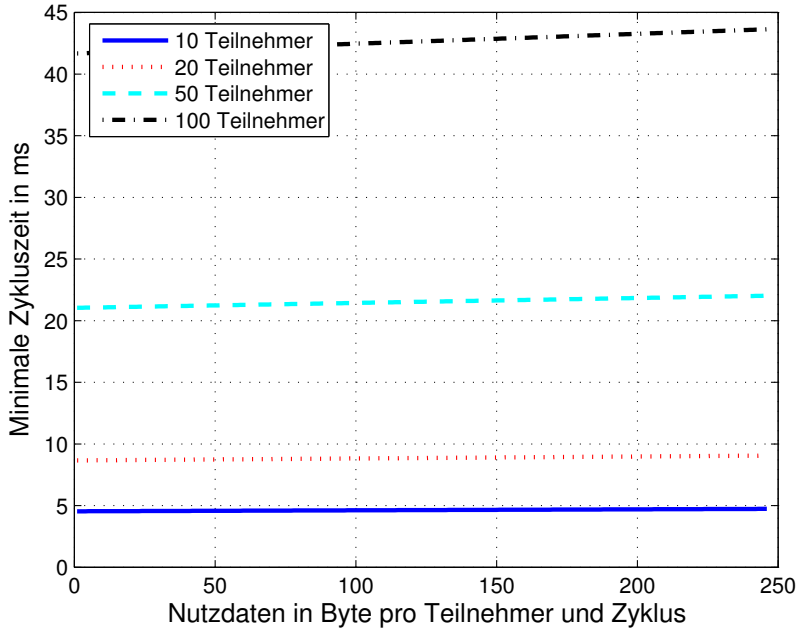


Abbildung 2.20: Berechnete minimale Zykluszeit von Modbus/TCP

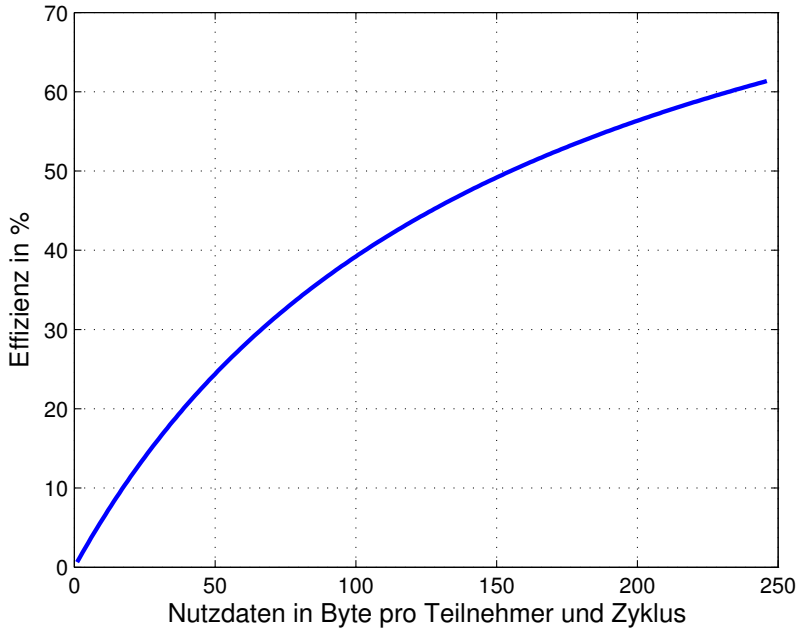


Abbildung 2.21: Berechnete Protokolleffizienz von Modbus/TCP

2.6.3 EtherNet/IP

EtherNet/IP verwendet zur Kommunikation das *Producer/Consumer*-Modell. Daher müssen die Daten nicht explizit angefragt werden, sondern die Teilnehmer (Producer) senden der Steuerung (Consumer) in periodischen Abständen eigenständig ihre Daten. Parallel dazu kann die Steuerung auch ihre Daten an jeden Teilnehmer senden. Für die Bewertung werden implizite Nachrichten mithilfe von UDP versendet.

Im idealen Fall sendet die Steuerung in einer Linie die Eingangsdaten für die Teilnehmer zuerst an den weit entferntesten Teilnehmer und zuletzt an seinen direkten Nachbar. Nachdem die Steuerung alle n Pakete übertragen hat und das letzte Paket den direkten Nachbar der Steuerung erreicht hat, sind alle Teilnehmer aktualisiert. Der Zyklus für die Übertragung der Daten von der Steuerung zu den Teilnehmern $T_{\mathrm{Cyc_{In}}}$ ist also nach

$$T_{\mathrm{Cyc_{In}}} = n \cdot (T_{\mathrm{Stapel}} + T_{\mathrm{Trn}}) + T_{\mathrm{Prd}} + T_{\mathrm{Swi}} + T_{\mathrm{Stapel}} \tag{2.18}$$

beendet.

Unter idealen Bedingungen beginnen die Teilnehmer gleichzeitig mit der Übertragung ihrer Ausgangsdaten an die Steuerung. Da bei der Leistungsbewertung der minimalen Zykluszeit die Länge der Pakete variiert wird, müssen zwei Fälle unterschieden werden. Bei einer kleinen Paketlänge erreichen die Pakete aller Teilnehmer die Steuerung, bevor die Pakete den Protokollstapel der Steuerung durchlaufen konnten. Die Zykluszeit für den Empfang der Pakete $T_{\mathrm{Cyc_{Out1}}}$ ist in diesem Fall nach

$$T_{\mathrm{Cyc_{Out1}}} = T_{\mathrm{Stapel}} + T_{\mathrm{Trn}} + T_{\mathrm{Prd}} + T_{\mathrm{Swi}} + n \cdot T_{\mathrm{Stapel}} \tag{2.19}$$

abgeschlossen. Bei großen Paketlängen ist die Übertragungszeit eines Paketes vom entferntesten Teilnehmer bis zur Steuerung größer als die Verarbeitung aller Pakete im Protokollstapel. Da alle anderen Pakete von der Steuerung vor dem weit entferntesten Paket empfangen werden, ist die Zykluszeit der Ausgangsdaten $T_{\mathrm{Cyc_{Out2}}}$ gleich:

$$T_{\mathrm{Cyc_{Out2}}} = (T_{\mathrm{Trn}} + T_{\mathrm{Prd}} + T_{\mathrm{Swi}}) \cdot n + 2 \cdot T_{\mathrm{Stapel}} + T_{\mathrm{IFG}} - T_{\mathrm{Prd}} \tag{2.20}$$

In diesem Fall begínnen alle Teilnehmer gleichzeitig mit der Übertragung ihrer Daten an den Teilnehmer. Daher muss ein Paket, das den nächsten Teilnehmer in der Linie erreicht, erst die Zeit T_{IFG} für eine IFG abwarten, bevor es von diesem Teilnehmer weitergeleitet wird.

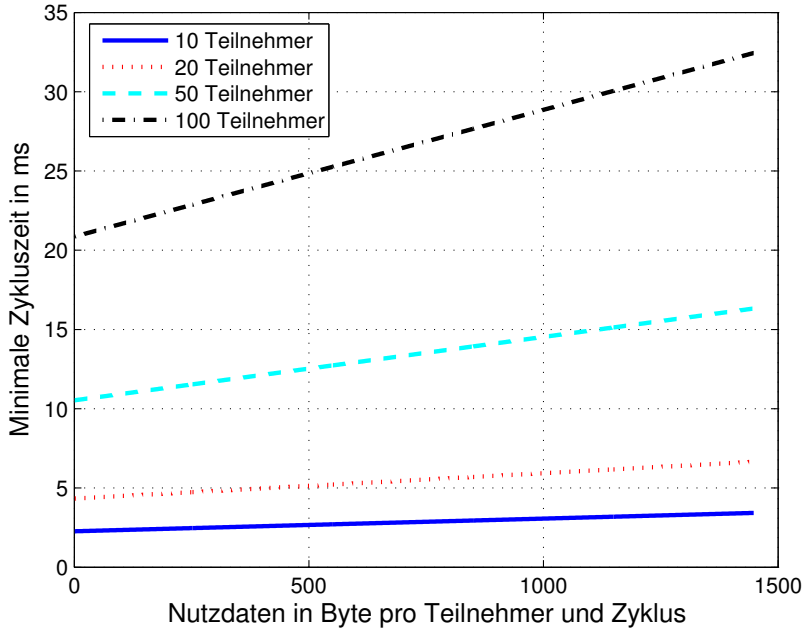


Abbildung 2.22: Berechnete minimale Zykluszeit von EtherNet/IP

Da Ausgangs- und Eingangsdaten parallel zueinander übertragen werden, ist die minimale Zykluszeit von EtherNet/IP gleich:

$$T_{\mathrm{Cyc}} = \max\left(T_{\mathrm{Cyc_{In}}}, T_{\mathrm{Cyc_{Out1}}}, T_{\mathrm{Cyc_{Out2}}}\right) \tag{2.21}$$

In dem hier untersuchten Szenario ist $T_{\mathrm{Cyc}} = T_{\mathrm{Cyc_{In}}}$, weil $T_{\mathrm{Swi}} + T_{\mathrm{Prd}} << T_{\mathrm{Stapel}}$. Die Abbildung 2.22 zeigt die berechneten minimalen Zykluszeiten für eine variierende Paketlänge und eine unterschiedliche Anzahl an Teilnehmern, wobei die maximale Paketlänge einen Wert von 1448 Bytes nicht überschreiten kann. Aufgrund der parallelen und direkten Kommunikation ohne Anfrage durch die Steuerung sind die Zykluszeiten im Vergleich zu Modbus/TCP deutlich geringer.

Die Protokolleffizienz wird bei EtherNet/IP bestimmt durch:

$$S = \frac{N_{\mathrm{In}} + N_{\mathrm{Out}}}{D_{\mathrm{In}} + D_{\mathrm{Out}}} \tag{2.22}$$

Bei Ethernet/IP müssen für die Protokolleffizienz nur die Eingangsnutzdaten N_{In}, die Ausgangsnutzdaten N_{Out}, die tatsächlichen Eingangsdaten D_{In} und die tatsächlichen Ausgangsdaten D_{Out} unterschieden werden. Die Protokolleffizienz für EtherNet/IP ist in Abbildung 2.23 dargestellt. Im Vergleich zu Modbus/TCP ist die Protokolleffizi-

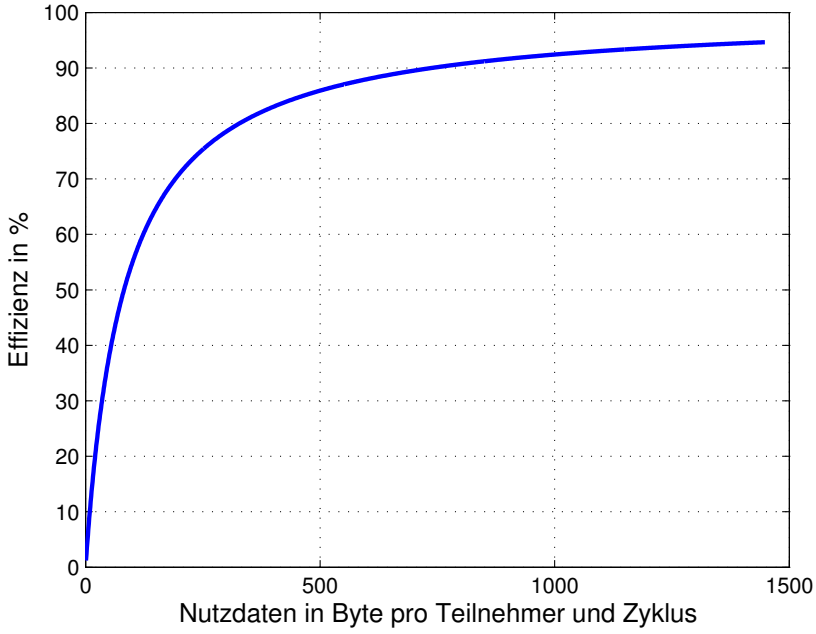


Abbildung 2.23: Berechnete Protokolleffizienz von EtherNet/IP

enz bei EtherNet/IP deutlich größer. Grund dafür ist eine insgesamt höhere Nutzdatenmenge, die in einem Paket transportiert wird sowie der Wegfall der zusätzlichen Anfrage- und Bestätigungspakete und der kleinere Paketkopf.

2.6.4 Ethernet Powerlink

Wie bereits in Kapitel 2.5.4 beschrieben wurde, beginnt jeder Kommunikationszyklus mit einer Startphase, in der die isochrone Kommunikationsphase vorbereitet wird und in der die *Start of Cyclic*-Nachricht von der Steuerung gesendet wird. Im Anschluss daran werden alle Teilnehmer (CNs) nacheinander durch die Steuerung (MN) über eine *PReq*-Nachricht aufgefordert, ihre Daten mithilfe der *PRes*-Nachricht an die Steuerung zu senden. Die Übertragung der Echtzeitdaten wird durch die *PResMN*-Nachricht der Steuerung abgeschlossen. Darauf folgt der Start der Asynchronen-Phase, in der in diesem Szenario keine Daten übertragen werden.

Der Ablauf eines Ethernet Powerlink-Zyklus zeigt die Abbildung 2.15. Die Dauer der Kommunikation zwischen einem Teilnehmer und der Steuerung T_{CN} errechnet sich aus der Übertragungsdauer für die *PReq*-Nachricht $T_{\mathrm{Trn_{PReq}}}$ und für die *PRes*-Nachricht $T_{\mathrm{Trn_{PRes}}}$ sowie aus den in [EPL03] angegebenen Reaktionszeiten eines Teilnehmers $T_{\mathrm{PRq-PRs}}$ und der Steuerung $T_{\mathrm{PRs-PRq}}$ und wird als

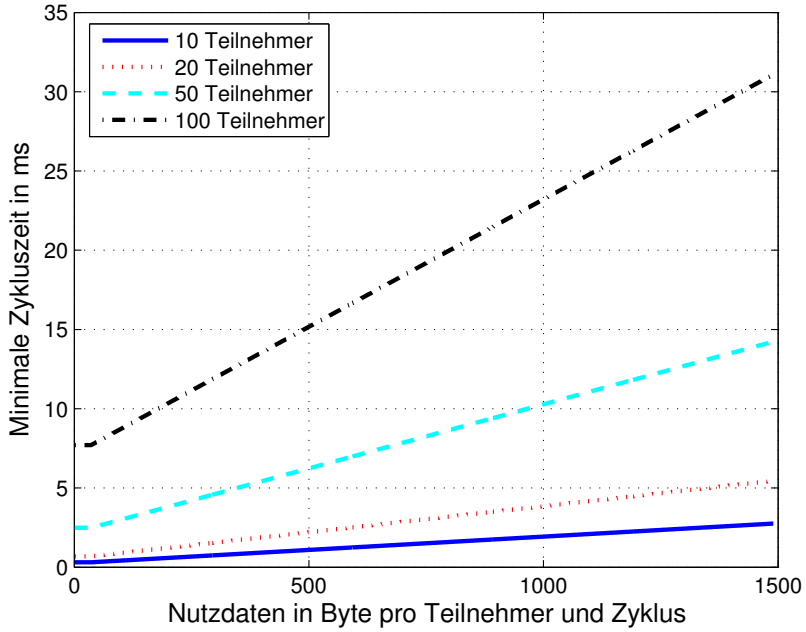


Abbildung 2.24: Berechnete minimale Zykluszeit von Ethernet Powerlink

$$T_{\mathrm{CN}} = T_{\mathrm{Trn}_{\mathrm{PReq}}} + T_{\mathrm{PRq-PRs}} + T_{\mathrm{Trn}_{\mathrm{PRes}}} + T_{\mathrm{PRs-PRq}} \tag{2.23}$$

zusammengefasst. Zusammen mit der Gleichung (2.23) ergibt die minimale Zykluszeit von Ethernet Powerlink

$$T_{\mathrm{Cyc}} = T_{\mathrm{Start}} + T_{\mathrm{Trn}_{\mathrm{PResMN}}} + n \cdot ((n+1) \cdot (T_{\mathrm{Prd}} + T_{\mathrm{Hub}}) + T_{\mathrm{CN}})\,. \tag{2.24}$$

Die Zeit für die Startphase T_{Start} umfasst einen in [EPL03] angegebenen konstanten Wert für die Vorbereitung der isochronen Phase und die Übertragungszeit für die *Start of Cyclic*-Nachricht. $T_{\mathrm{Trn}_{\mathrm{PResMN}}}$ ist die Übertragungsverzögerung der *PResMN*-Nachricht, die die Übertragung der Echtzeitdaten abschließt.

In der Abbildung 2.24 sind die errechneten minimalen Zykluszeiten bei einer variierenden Paketlänge für eine unterschiedliche Anzahl von Teilnehmern dargestellt. Bis zu einer Nutzdatenmenge von 36 Bytes ist die minimale Zykluszeit gleich groß, weil in diesem Fall das Datenfeld aufgefüllt wird, um die minimale Paketlänge von Ethernet einzuhalten.

Die minimalen Zykluszeiten von Ethernet Powerlink sind deutlich geringer als die Zykluszeiten von EtherNet/IP oder von Modbus/TCP. Der Grund ist der Verzicht auf das

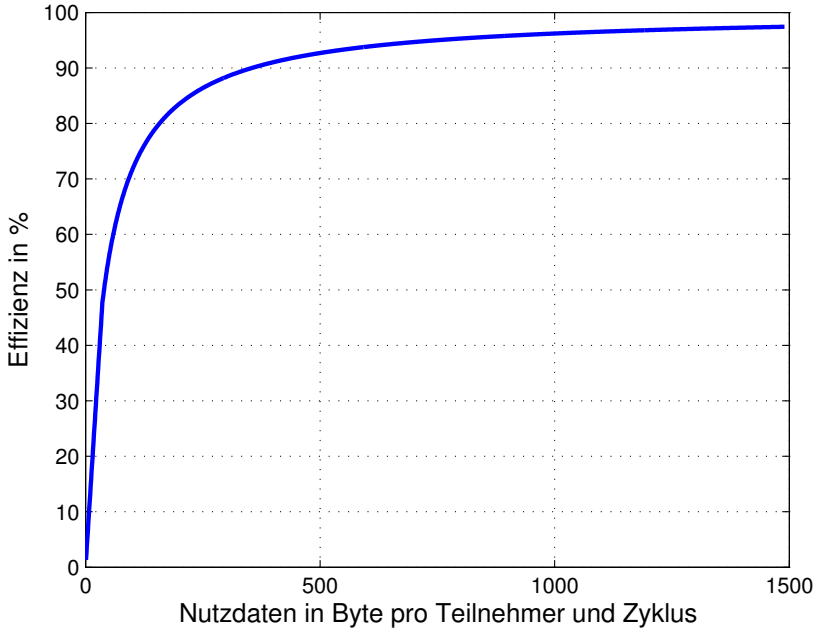


Abbildung 2.25: Berechnete Protokolleffizienz von Ethernet Powerlink

TCP/UDP/IP-Protokoll bei der Übertragung der Powerlink-Pakete. Die hohe Verarbeitungszeit des TCP/UDP/IP-Protokollstapels verursacht hauptsächlich die höheren Zykluszeiten von Modbus/TCP und Ethernet/IP.

Die Protokolleffizienz bei Ethernet Powerlink wird berechnet durch:

$$S = \frac{N_{\mathrm{PResMN}} + n \cdot (N_{\mathrm{In}} + N_{\mathrm{Out}})}{D_{\mathrm{SoC}} + D_{\mathrm{PResMN}} + n \cdot (D_{\mathrm{In}} + D_{\mathrm{Out}})} \tag{2.25}$$

D_{SoC} ist die Paketlänge der *Start of Cyclic*-Nachricht. N_{PResMN} ist die Nutzdatenmenge und D_{PResMN} die gesamte Datenmenge einer *PResMN*-Nachricht. Bei der Protokolleffizienz zahlt sich aus, dass das Protokoll direkt auf Ethernet aufsetzt. Der Paketkopf ist im Vergleich zu den vorher untersuchten Protokollen geringer. Daher ist auch die Protokolleffizienz bei Ethernet Powerlink größer als bei Modbus/TCP und Ethernet/IP.

2.6.5 PROFINET

Die Bewertung von PROFINET erfolgt nur für die höchste Leistungsklasse IRT. Die Kommunikationsmodelle von PROFINET und EtherNet/IP sind faktisch identisch. Daher gleicht der Ablauf der Kommunikation in diesem Szenario bei PROFINET dem von EtherNet/IP.

Die Übertragung der Eingangs- und Ausgangsdaten wird einzeln betrachtet, weil diese auch bei PROFINET im idealen Fall parallel abläuft. Die Eingangsdaten werden von der Steuerung direkt nacheinander an die Teilnehmer gesendet. Das erste Paket wird im idealen Fall an den weit entferntesten Teilnehmer adressiert und das letzte Paket an den direkten Nachbarn. Daher ist der Übertragungszyklus der Steuerung nach

$$T_{\mathrm{Cyc_{In}}} = n \cdot (T_{\mathrm{Trn}} + T_{\mathrm{IFG}}) + T_{\mathrm{Prd}} + T_{\mathrm{Cut}} \tag{2.26}$$

abgeschlossen [Pop05]. Die Übertragung der Teilnehmer zur Steuerung ist in derselben Zeit abgeschlossen, wenn die Teilnehmer gleichzeitig mit der Übertragung ihrer Pakete beginnen. Das Paket von dem Teilnehmer, der am weitesten von der Steuerung entfernt ist, erreicht die Steuerung in diesem Fall als letztes nach $T_{\mathrm{Cyc_{Out}}} = T_{\mathrm{Cyc_{In}}}$.

Ein PROFINET-Zyklus ist allerdings erst abgeschlossen, wenn zusätzlich zur parallelen Übertragung der Eingangs- und Ausgangsdaten noch ein Paket zur Uhrensynchronisation übertragen wurde. Die Summe aus der Übertragungsdauer für die Eingangsdaten $T_{\mathrm{Cyc_{In}}}$ und aus der Übertragungsdauer für das Synchronisationspaket $T_{\mathrm{Trn_{Syn}}}$ ergibt die minimale Zykluszeit für PROFINET und wird durch

$$T_{\mathrm{Cyc}} = T_{\mathrm{Cyc_{In}}} + T_{\mathrm{Trn_{Syn}}} \tag{2.27}$$

bestimmt. Die Übertragungsdauer für das Synchronisationspaket $T_{\mathrm{Trn_{Syn}}}$ entspricht der Übertragungsdauer eines Ethernet-Paketes minimaler Länge.

Dargestellt wird die minimale Zykluszeit für eine variierende Paketlänge und eine unterschiedliche Anzahl an Knoten in der Abbildung 2.26. Aufgrund der Hardwareunterstützung durch den ERTEC400 und dem angewendeten Cut-Through-Verfahren ist die Zykluszeit bisher am geringsten.

Für PROFINET IRT wird die Protokolleffizienz bestimmt mit

$$S = \frac{n \cdot (N_{\mathrm{In}} + N_{\mathrm{Out}})}{D_{\mathrm{Syn}} + n \cdot (D_{\mathrm{In}} + D_{\mathrm{Out}})} \tag{2.28}$$

Zusätzlich zu den Eingangs- und Ausgangsdaten muss bei PROFINET auch die tatsächliche Datenmenge des Synchronisationspaketes D_{Syn} berücksichtigt werden. Die Datenmenge D_{Syn} entspricht den 72 Byte eines vollständigen Ethernet-Paketes minimaler Länge.

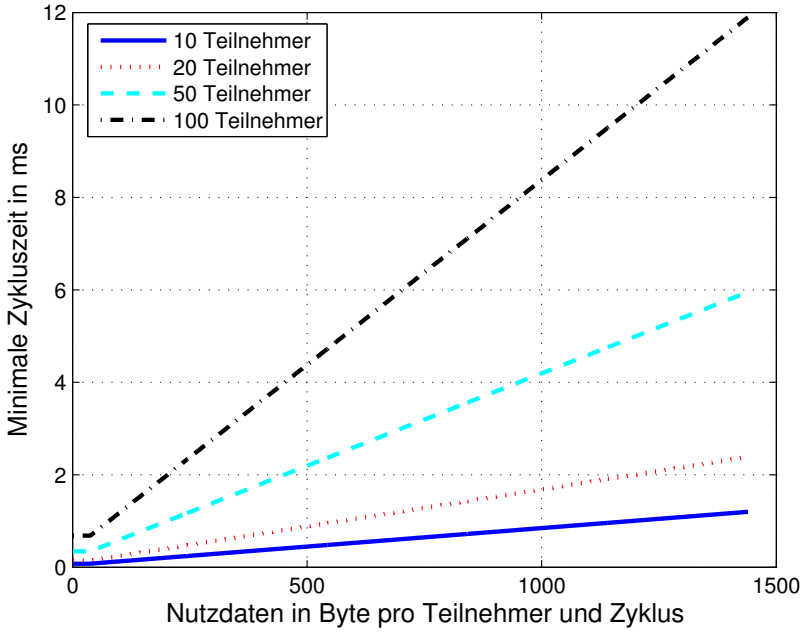


Abbildung 2.26: Berechnete minimale Zykluszeit von PROFINET

Im Vergleich zu Ethernet Powerlink ist die Protokolleffizienz von PROFINET IRT etwas größer. Grund dafür ist der etwas kleinere Paketkopf. Den Verlauf der Protokolleffizienz für eine variierende Paketlänge zeigt die Abbildung 2.27.

2.6.6 EtherCAT

Bei EtherCAT sendet die Steuerung (Master) alle seine Daten an die Teilnehmer (Slaves), die in denselben Datenstrom ihre Ausgangsdaten einbetten. Aufgrund der logischen Ringstruktur erhält die Steuerung das von ihr gesendete Paket wieder zurück. Über das EtherCAT-Telegramm, das in das Ethernet-Paket eingebettet wird, adressiert die Steuerung einen oder mehrere Teilnehmer. Für die Leistungsbewertung wird die logische Adressierung von EtherCAT verwendet, bei der nur ein EtherCAT-Telegramm notwendig ist, um alle Teilnehmer zu adressieren. Des Weiteren wird die Möglichkeit von EtherCAT genutzt, die Eingangsdaten von der Position im Telegramm zu lesen, an der auch die Ausgangsdaten geschrieben werden. Bei gleicher Größe von Eingangs- und Ausgangsdaten reduziert sich die zu transportierende Datenmenge um die Hälfte. Übersteigt die Datenmenge die maximale Paketlänge eines Ethernet-Paketes, so wird für die übrigen Daten von der Steuerung ein neues EtherCAT-Telegramm erzeugt, in ein Ethernet-Paket eingebettet und übertragen.

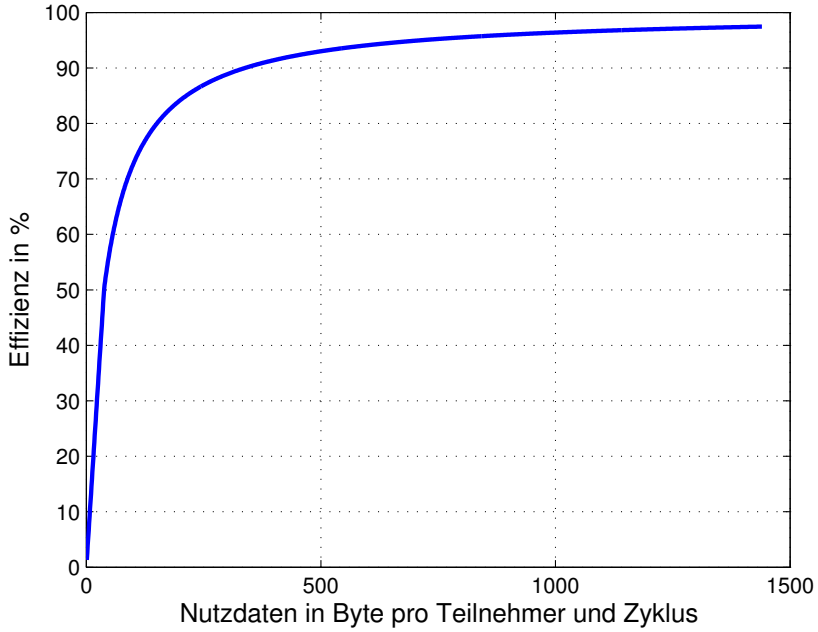


Abbildung 2.27: Berechnete Protokolleffizienz von PROFINET

Zur Bestimmung der minimalen Zykluszeit wird im Matlab-Modell anhand der zu übertragenden Eingangs- bzw. Ausgangsdaten die Anzahl n_p der von der Steuerung zu übertragenden Pakete ermittelt. Die Paketumlaufzeit wird neben der Übertragungsverzögerung T_{Trn_i} eines Paketes i auch von den Signalverzögerungen T_Prd und den Slaveverzögerungen bestimmt, die beim Durchlauf eines Paketes durch den logischen EtherCAT-Ring auf Hin- und Rückweg entstehen.

Die Slaves werden bei EtherCAT in der Regel nicht über ein Ethernet-Kabel, sondern über den EBUS, verbunden. Nur der erste Slave in der Linie ist mit dem Master über ein Ethernet-Kabel verbunden. Weil der erste Slave über Ethernet und den EBUS kommuniziert, ist die Verzögerung des ersten Slaves T_{S_1} höher als die Verzögerung der anderen Slaves. Die Pakete werden nur auf dem Hinweg verarbeitet und auf dem Rückweg weitergeleitet. Die Verzögerungszeit eines Slaves auf dem Hinweg $T_{\mathrm{S}_\mathrm{to}}$ ist daher etwas höher als die Verzögerungszeit auf dem Rückweg $T_{\mathrm{S}_\mathrm{back}}$. Die Verzögerung T_{S_1} umfasst bereits die Verzögerungen des ersten Slaves für Hin- und Rückweg. Berechnet wird die minimale Zykluszeit von EtherCAT wie folgt:

$$T_\mathrm{Cyc} = T_{\mathrm{S}_1} + (n-1) \cdot (T_{\mathrm{S}_\mathrm{to}} + T_{\mathrm{S}_\mathrm{back}}) \cdot n \cdot T_\mathrm{Prd} + \sum_{i=1}^{n_\mathrm{p}} T_{\mathrm{Trn}_i} \tag{2.29}$$

Dargestellt sind die minimalen Zykluszeiten von EtherCAT für eine variierende Anzahl von Nutzdaten und einer unterschiedlichen Anzahl von Knoten in der Abbildung 2.28.

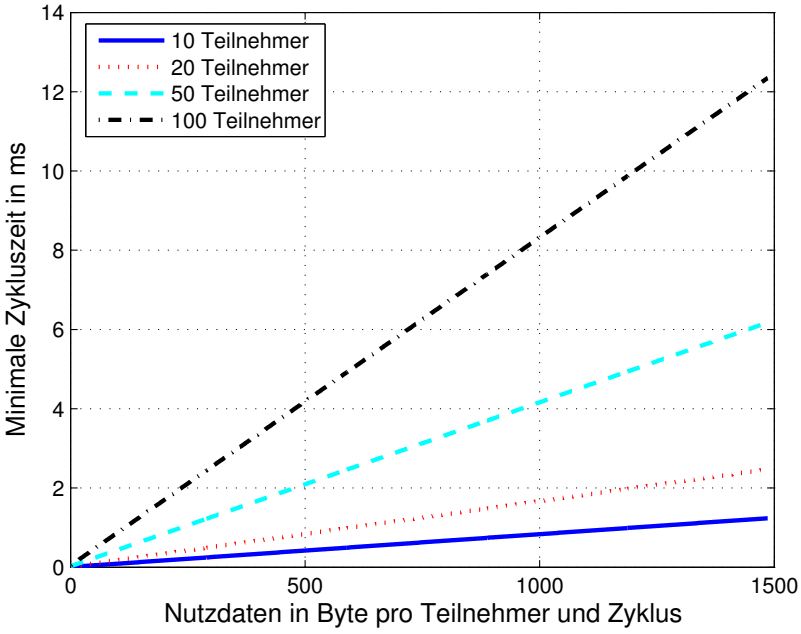


Abbildung 2.28: Berechnete minimale Zykluszeit von EtherCAT

Aufgrund der geringen Verzögerungszeiten der Slaves und weil die Ein- und Ausgangsdaten an dieselbe Position im Paket geschrieben werden, hat EtherCAT die geringste Zykluszeit.

Bei EtherCAT wird die Protokolleffizienz bestimmt durch:

$$S = \frac{n \cdot (N_{\mathrm{In}} + N_{\mathrm{Out}})}{\sum_{i=1}^{n_{\mathrm{p}}} D_i} \tag{2.30}$$

Die Integration von Ein- und Ausgangsdaten von mehreren Teilnehmern in einem Paket i mit der tatsächlichen Datenmenge D_i führt bei EtherCAT schon bei kleinen Nutzdatenmengen zu einer sehr hohen Protokolleffizienz, wie in der Abbildung 2.29 verdeutlicht wird. Die Zacken in der abgebildeten Kurve entstehen immer dann, wenn aufgrund der steigenden Nutzdatenmenge ein neues Paket erzeugt werden muss.

2.6.7 Vergleich

Abschließend werden die Ergebnisse aus der Leistungsbewertung der einzelnen Protokolle direkt miteinander verglichen. Die minimale Zykluszeit und die Protokolleffizienz werden in diesem Vergleich für eine Linie mit einer Steuerung und 50 Teilnehmern bestimmt. Der Datenaustausch wird auf die gleiche Weise geregelt, wie er in den vorangegangenen Abschnitten beschrieben wurde.

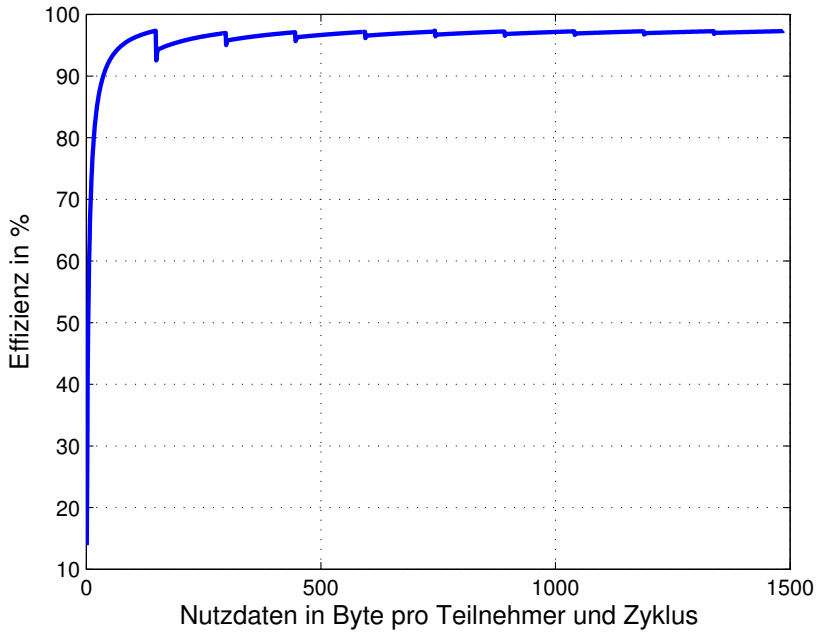
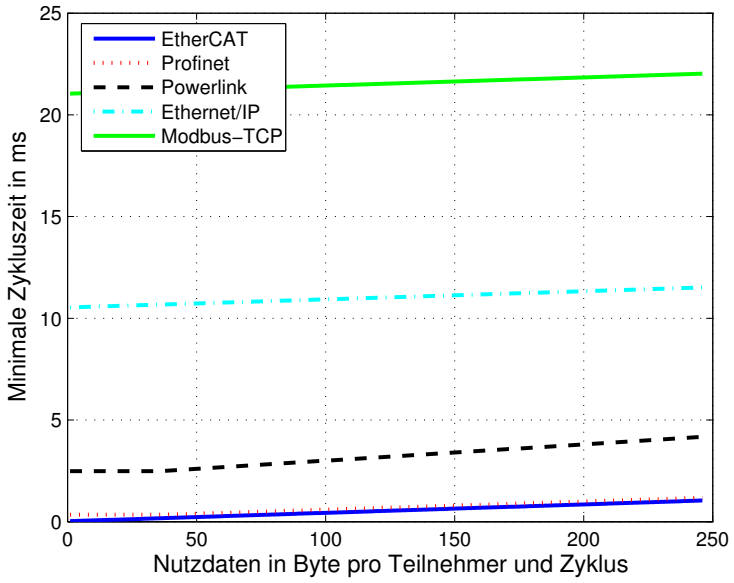
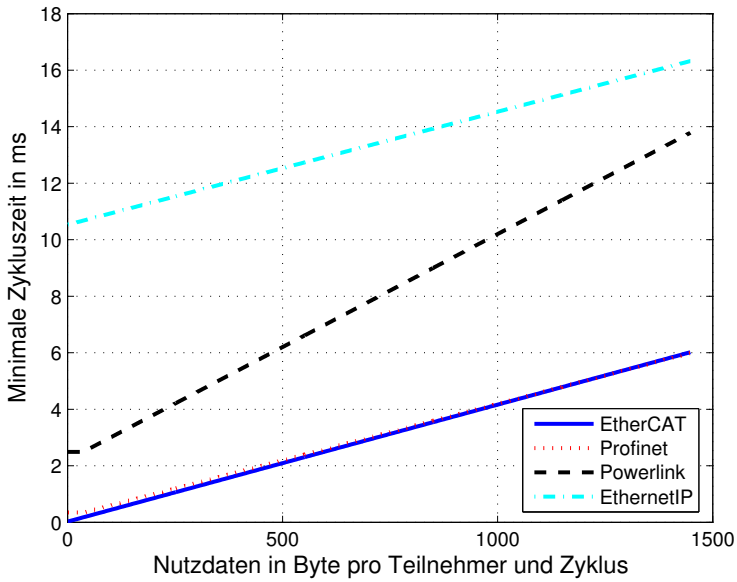


Abbildung 2.29: Berechnete Protokolleffizienz von EtherCAT

Die Ergebnisse für die minimale Zykluszeit der Echtzeitprotokolle werden in der Abbildung 2.30 zusammengefasst. Aufgrund der hohen Zykluszeit und der geringen Nutzdatenmenge sind die Zykluszeiten von Modbus/TCP nur in der Abbildung 2.30(a) dargestellt. Die Zykluszeiten von Modbus/TCP sind um ein Vielfaches größer als bei den anderen Protokollen. Hauptgrund ist die hohe Verarbeitungszeit des TCP/UDP/IP-Protokollstapels. Für einen besseren Vergleich der anderen Protokolle werden deren Zykluszeiten in Abbildung 2.30(b) nochmals zusätzlich dargestellt. Mit deutlichem Abstand zu Modbus/TCP hat Ethernet/IP in diesem Szenario die zweitgrößte Zykluszeit. Ethernet/IP hat auch eine hohe Verarbeitungszeit durch den Protokollstapel. Aber bei Modbus/TCP werden aufgrund der Client-Server-Kommunikation mehr Pakete versendet. Ethernet Powerlink benötigt keinen TCP/UDP/IP-Protokollstapel. Daher sind auch die Zykluszeiten deutlich geringer als bei Ethernet/IP und bei Modbus/TCP. Aufgrund des Kommunikationsschemas können bei Ethernet Powerlink aber nur Daten von der Steuerung zum Teilnehmer nacheinander geschrieben und gelesen werden. PROFINET IRT benötigt für die Echtzeitkommunikation auch keinen TCP/UDP/IP-Protokollstapel. Im Gegensatz zu Ethernet Powerlink profitiert PROFINET in diesem Szenario von der Möglichkeit, die Eingangs- und Ausgangsdaten parallel zu übertragen. Daher hat PROFINET auch eine geringere Zykluszeit als Ethernet Powerlink. EtherCAT hat die geringste Zykluszeit. Insbesondere bei kleinen Nutzdatenmengen, die in Automatisierungsnetzen häufig vorkommen, ist der Unterschied zwischen PROFINET und EtherCAT sehr deutlich. Bei PROFINET wird für jeden Teilnehmer ein



(a) Nutzdatenmenge bis 246 Byte

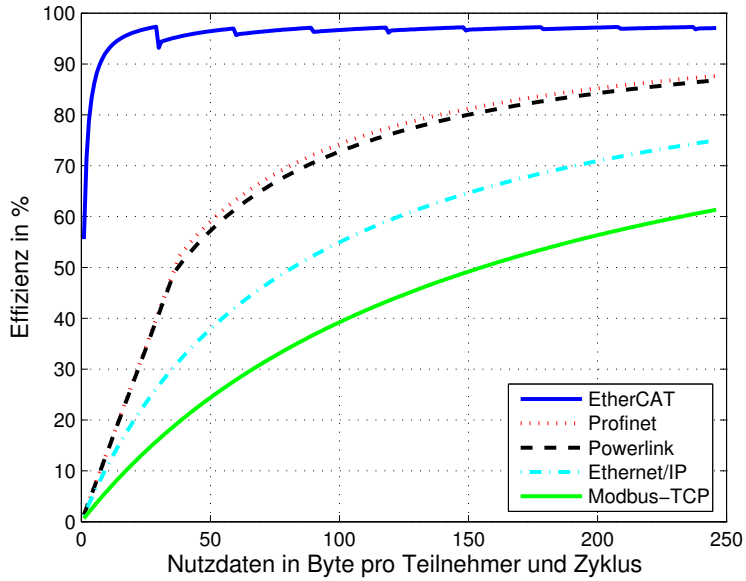


(b) Nutzdatenmenge bis 1448 Byte

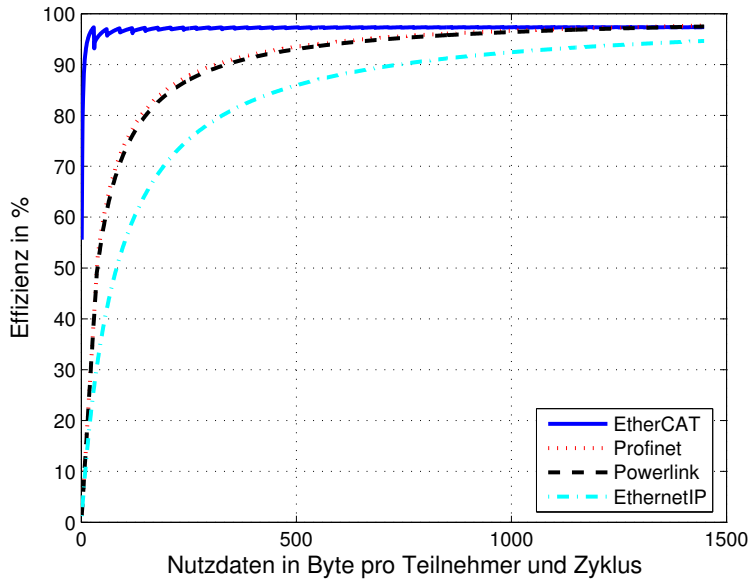
Abbildung 2.30: Vergleich der minimalen Zykluszeiten

Paket versendet, und bei EtherCAT werden die Daten mehrerer Teilnehmer in ein Paket eingebettet. Bei geringen Nutzdatenmengen werden bei EtherCAT daher weniger Pakete übertragen als bei PROFINET.

Die TCP/UDP/IP-basierten Protokolle haben aufgrund der hohen Verarbeitungszeit des Protokollstapels die höchsten Zykluszeiten. Die Protokolle, die den TCP/UDP/IP-



(a) Protokolleffizienz bis 246 Byte



(b) Protokolleffizienz bis 1448 Byte

Abbildung 2.31: Vergleich der Protokolleffizienz

Stapel umgehen und die Echtzeitkommunikation direkt über Ethernet steuern, erreichen daher eine geringere Zykluszeit. Noch geringere Zykluszeiten erreichen die Protokolle, die für die Echtzeitkommunikation eine spezielle Hardwareunterstützung benötigen.

Die Protokolleffizienz der bewerteten Protokolle ist in Abbildung 2.31 dargestellt. Die geringe Nutzdatenmenge von Modbus/TCP erfordert auch im Falle der Protokolleffizienz eine separate Darstellung mit Modbus/TCP in Abbildung 2.31(a). Die Protokolleffizienz für höhere Nutzdatenmengen zeigt die Abbildung 2.31(b). Der große Paketkopf und die geringe Nutzdatenmenge von 246 Byte führen bei Modbus/TCP zu der geringsten Protokolleffizienz von maximal 61 %. Bei derselben Nutzdatenmenge hat Ethernet/IP eine Protokolleffizienz von 75 %. Bei Ethernet Powerlink liegt die Protokolleffizienz bei 87 % und bei PROFINET IRT liegt sie bei 88 %. Nur EtherCAT hat schon bei deutlich geringeren Nutzdatenmengen eine Protokolleffizienz von 96 %. Diesen Wert können Ethernet Powerlink und PROFINET erst ab einer Nutzdatenmenge von 1000 Byte überschreiten. Die Ergebnisse bescheinigen insbesondere den Echtzeitprotokollen, die TCP/IP oder UDP/IP verwenden, eine deutlich schlechtere Protokolleffizienz.

2.7 Zusammenfassung

Gemeinsame Basis von Kommunikations- und Echtzeitkommunikationsnetzen ist die Aufteilung der Kommunikation in die Schichten des ISO/OSI-Referenzmodells. Um die Komplexität einer Kommunikation zu bewältigen, werden beim Referenzmodell die einzelnen Schritte einer Kommunikation in definierte Funktionen zusammengefasst, die den einzelnen Schichten zugeordnet werden. Schon die Wahl des Medienzugriffsverfahrens auf einer der untersten Schichten des Referenzmodells entscheidet maßgeblich über das deterministische Verhalten eines Echtzeitkommunikationsnetzes.

Die ersten Echtzeitkommunikationsnetze entstanden im industriellen Bereich mit der Entwicklung der Feldbusse, die zur Vernetzung von Sensoren und Aktoren mit einer SPS oder einem Industrie-PC verwendet werden. Durch eine gestiegene Anzahl von Knoten und höhere Anforderungen an die Kommunikation werden die Feldbusse nach und nach von den Ethernet-basierten Echtzeitprotokollen abgelöst. Ethernet ist der Standard für Kommunikationsnetze im Büro- und Heimbereich. Aufgrund seines nicht deterministischen Medienzugriffsverfahrens wurde der Einsatz von Ethernet in Echtzeitkommunikationsnetzen lange Zeit ausgeschlossen. Den Durchbruch in den Bereich der Echtzeitkommunikation erlangte Ethernet durch die Einführung der Switch-Technologie, auf der viele Ethernet-basierte Echtzeitprotokolle aufbauen.

Eine Aufteilung der Ethernet-basierten Echtzeitprotokolle erfolgt in drei Gruppen. Die erste Protokollgruppe verwendet den TCP/UDP/IP-Protokollstapel und realisiert die Echtzeiteigenschaften des Protokolls auf der Anwendungsschicht. Aufgrund des verwendeten TCP/UDP/IP-Stapels können die Protokolle auf jeder Standard-Ethernet-Komponente ausgeführt werden. Allerdings haben diese Protokolle aufgrund

von TCP/UDP/IP einen sehr großen Paketkopf, der im Vergleich mit den anderen Protokollgruppen zu einer schlechten Protokolleffizienz führt. Bei der zweiten Protokollgruppe wird bei der Echtzeitkommunikation der TCP/UDP/IP-Protokollstapel übergangen und das Protokoll setzt direkt auf der Ethernet-Schicht auf. Die geringste Zykluszeit und die größte Protokolleffizienz werden mit den hier vorgestellten Protokollen erzielt, die der dritten Gruppe angehören. Für die Echtzeitkommunikation werden bei dieser Protokollgruppe Teile des Protokolls in Hardware ausgeführt und damit die Leistungsfähigkeit der Protokolle gesteigert. Für die harte Echtzeitkommunikation sind diese Protokolle am besten geeignet. Nur sie erfüllen die höchsten Anforderungen aus den IAONA-Echtzeitklassen. Dafür benötigen sie aber spezielle ASICs und können nicht mehr auf einer Standard-Ethernet-Komponente ausgeführt werden.

Bei diesen Protokollen erfordert die Erfüllung von harten Echtzeitanforderungen eine vorherige Planung des gesamten Kommunikationsablaufs. Dazu werden komplexe Planungswerkzeuge verwendet, mit denen die Netzwerkinfrastruktur projektiert wird und mit denen die Zykluszeiten sowie die Latenzzeiten optimiert werden. Eine Adaption während des Betriebs an sich ändernde Echtzeitanforderungen ist bei solchen Echtzeitkommunikationsnetzen weder auf der Protokollebene noch auf der Ebene der Netzwerkkomponente möglich. Insbesondere die Verwendung eines ASICs erschwert eine Anpassung des Ressourcenbedarfs zur Laufzeit.

Kapitel 3

Eine rekonfigurierbare Switch-Architektur

Eine Möglichkeit, ein Echtzeit-Kommunikationssystem an veränderte Anforderungen anzupassen, ist die dynamische Adaption des Systems, das die Daten verarbeitet und weiterleitet. Für diese Aufgabe werden bei Ethernet üblicherweise Switches und Prozessoren verwendet. Die Switch-Architektur, die für das schnelle Weiterleiten von Datenpaketen zuständig ist, wird in den meisten Fällen als ASIC gefertigt. Die Verarbeitung von höheren Protokollschichten, wie z. B. TCP/IP, wird in Software realisiert und auf einem Prozessor ausgeführt.

ASICs sind für die spezielle Applikation, für die sie entworfen sind, sehr leistungsstark. Durch die Spezialisierung verfügen sie nur über eingeschränkte Möglichkeiten, sich während des Betriebes an Veränderungen anzupassen. Bei einigen ASICs können z. B. Teile des Bausteins abgeschaltet werden, wenn diese momentan nicht benötigt werden. Dadurch lässt sich der Energieverbrauch an die Anforderungen anpassen. Eine weitere Möglichkeit, Energie zu sparen, ist, die Datenrate von z. B. 1 Gbit/s auf 100 MBit/s zu reduzieren [ARSG06].

Prozessoren sind durch die freie Programmierbarkeit sehr flexibel und können durch eine Veränderung der Software oder das Laden eines neuen Programms an die unterschiedlichsten Anforderungen im Betrieb angepasst werden. Für die Verarbeitung und die Weiterleitung von Paketen müssen allerdings sehr leistungsstarke Prozessoren verwendet werden, um hohe Datenraten zu erreichen.

FPGAs (Field-Programmable Gate Arrays) bieten hier eine Alternative. Auf einem FPGA kann, wie auch auf einem Prozessor, eine beliebige Applikation abgebildet werden. Aber insbesondere bei hochgradig parallelisierbaren und datenflussorientierten Applikationen bieten FPGAs bei gleicher Flexibilität gegenüber Prozessoren eine vergleichbare oder sogar größere Leistungsfähigkeit [GNVV04]. FPGAs bestehen aus einer Matrix von konfigurierbaren Logikblöcken, die über ein Verbindungsnetzwerk miteinander verbunden sind. Ein Logikblock kann auf Bitebene jede beliebige boolesche Funktion annehmen. Sowohl die konfigurierbaren Logikblöcke als auch das Verbindungsnetzwerk sind konfigurierbar und es können beliebige komplexe digitale Schal-

tungen auf einem FPGA abgebildet werden. Die Konfiguration eines FPGAs erfolgt entweder vollständig oder es wird bei der sogenannten partiellen dynamischen Rekonfiguration nur ein Teil von dem FPGA zur Laufzeit verändert. Mit diesem Verfahren kann ein Teil der Architektur während des Betriebes an sich verändernde Anforderungen angepasst werden.

In diesem Kapitel wird eine rekonfigurierbare Switch-Architektur vorgestellt und untersucht. Mithilfe der partiellen dynamischen Rekonfiguration passt sich die Architektur des Switches an sich ändernde Kommunikations- und Echtzeitanforderungen (Latenz und Jitter) automatisch an. Bei einer hohen Datenrate und einer geringen Latenz benötigt der Switch mehr Fläche auf dem FPGA als bei einer geringeren Datenrate. In einem adaptiven *System-on-Chip* kann die freie Fläche, die bei einer geringen Datenrate entsteht, von anderen Anwendungen genutzt werden. Während der FPGA rekonfiguriert wird, dürfen keine Pakete verloren gehen. Ein Ansatz für die paketverlustfreie Rekonfiguration wird in Kapitel 3.1 vorgestellt. Weitere Teile des Kapitels 3.1 sind die Beschreibung des rekonfigurierbaren Switches und die Verfahren zur automatischen Adaption eines Switches. Die prototypische Umsetzung des rekonfigurierbaren Switches in Kapitel 3.2 befasst sich mit den Teilkomponenten des Systems und den notwendigen Besonderheiten für eine Adaption der Architektur. Abgeschlossen wird dieses Kapitel mit der Performanceevaluation der Switch-Architektur, die die Leistungsfähigkeit, den Ressourcenbedarf und den Aufwand für die Adaption des rekonfigurierbaren Switches untersucht.

3.1 Ein rekonfigurierbarer Ethernet-Switch

In diesem Abschnitt wird ein rekonfigurierbarer Ethernet-Switch [Fer05] als ein Beispiel für eine adaptive Netzwerkkomponente vorgestellt. Der Switch (Abbildung 3.1) verwendet einen FPGA als Zielplattform und besteht aus mindestens zwei Netzwerkschnittstellen, einem eingebetteten Prozessor und dem Rekonfigurationskontroller. Die Adaption dieser Netzwerkkomponente erfolgt durch eine Rekonfiguration der Netzwerkschnittstellen, die sich an veränderte Anforderungen anpassen. Das Prinzip der Rekonfiguration der Netzwerkschnittstelle ist unabhängig von der verwendeten Netzwerktechnologie. Die Verfahren werden am Beispiel von Ethernet erläutert, das im Echtzeitbereich eine immer größere Rolle spielt. Auch die prototypische Umsetzung des rekonfigurierbaren Switches basiert auf Ethernet.

Der Ansatz von rekonfigurierbaren Netzwerkschnittstellen für Echtzeitnetzwerke wurde von uns erstmals in [VGPR04a, VGPR04b] vorgestellt und evaluiert. Die Ergebnisse zu der prototypischen Umsetzung werden in [GP06] beschrieben. Nicht nur für Echtzeitsysteme werden rekonfigurierbare Netzwerkschnittstellen auf der Basis von rekonfigurierbarer Logik verwendet, sondern beispielsweise auch für Firewalls [FN01] und

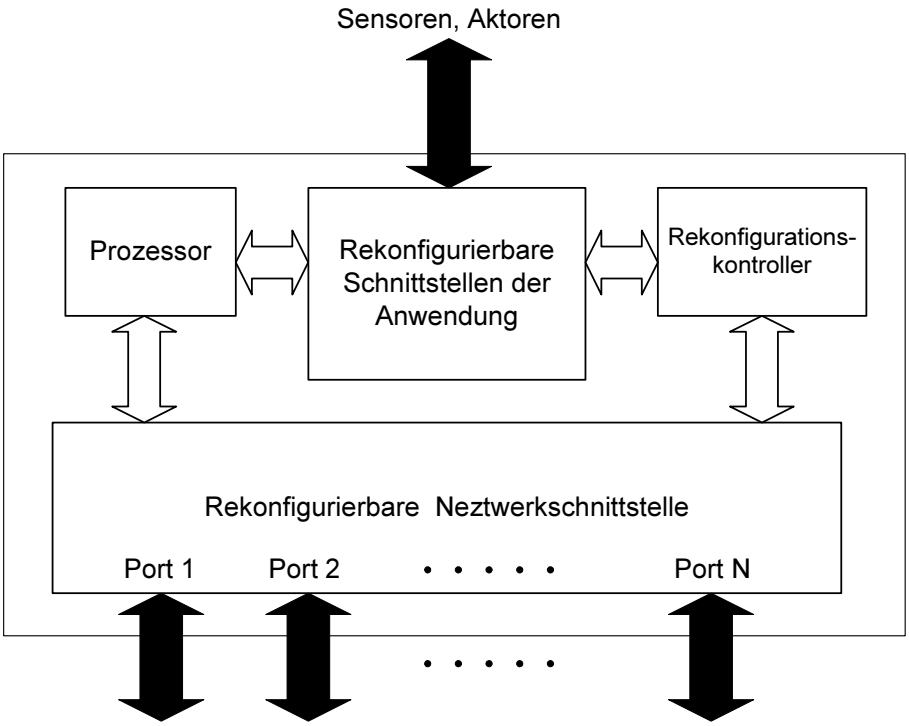


Abbildung 3.1: Aufbau der adaptiven Netzwerkkomponente

Virenschutzsysteme [LMR+03], die mit Viren infizierte Pakete blockieren. Ein Beispiel aus dem Bereich von Echtzeitsystemen sind die rekonfigurierbaren Netzwerkknoten von ReCoNets [HKT03, KSD+06]. Schwerpunkt bei ReCoNets ist die Fehlertoleranz des gesamten Netzwerks. Durch die Rekonfiguration der Knoten können bei ReCoNets die Aufgaben (engl.: Tasks) eines ausgefallenen Knotens auf die restlichen verbliebenen Knoten migriert werden. Ebenfalls werden, wie auch bei dem Ausfall einer Verbindung, neue Routen für die Kommunikationsverbindungen zwischen den Aufgaben gebildet. Obwohl das ReCoNets-Projekt für den Einsatz im Echtzeitbereich konzipiert ist, werden harte Echtzeitkommunikationsanforderungen bisher nicht betrachtet.

Der rekonfigurierbare Switch verarbeitet zwei Arten von Datenströmen. Die einen Datenströme werden vom Prozessor verarbeitet und enthalten Daten für die Sensoren oder von den Aktoren und die anderen Datenströme werden vom Switch an seine Nachbarn weitergeleitet. Bei einer geringen Netzwerklast und geringen Echtzeitanforderungen an die Latenz und den Jitter sind einfache Netzwerkschnittstellen ausreichend. Die Datenpakete werden in diesem Fall von einer Softwareimplementierung auf dem eingebetteten Prozessor von einem Port zum anderen weitergeleitet. Das Weiterleiten von Paketen verursacht eine hohe Prozessorlast und belastet ebenfalls den Systembus, der den Prozessor mit den Netzwerkschnittstellen, dem Speicher und der sonstigen Peripherie verbindet. Allerdings benötigt eine einfache Netzwerkschnittstelle nur wenige FPGA-Ressourcen und die restlichen freien FPGA-Ressourcen stehen anderen Appli-

kationen, wie z. B. einer Aktor-Regelung, zur Verfügung. Reicht die Leistungsfähigkeit der Softwarelösung nicht mehr aus oder sind höhere Echtzeitanforderungen gefordert, können die separaten Netzwerkschnittstellen durch einen Hardware-Switch während des Betriebs ersetzt werden. Dieser Switch leitet die Datenpakete eigenständig weiter und kann so ein wesentlich höheres Verkehrsaufkommen bewältigen. Jedoch benötigt der Hardware-Switch aufgrund seiner komplexeren Struktur zusätzliche FPGA-Ressourcen, die dann anderen Applikationen nicht mehr zur Verfügung stehen. Die Entscheidung über eine Rekonfiguration kann eigenständig von dem rekonfigurierbaren Switch anhand von Statistiken, wie z. B. der Netzwerk- und der Systemlast, getroffen werden. Einen ähnlichen Ansatz, FPGA-Ressourcen entweder zur Beschleunigung einer Applikation oder zur Verarbeitung der Kommunikationsdaten zu verwenden, wird bei Underwood et al. [USL01] verfolgt. Das FPGA wird in diesem Beispiel nicht zum beschleunigten Weiterleiten von Paketen verwendet, sondern übernimmt die Protokollverarbeitung ankommender Daten. Anwendungsgebiete sind im Gegensatz zu den hier betrachteten Switches einfache Netzwerkschnittstellen von Endgeräten.

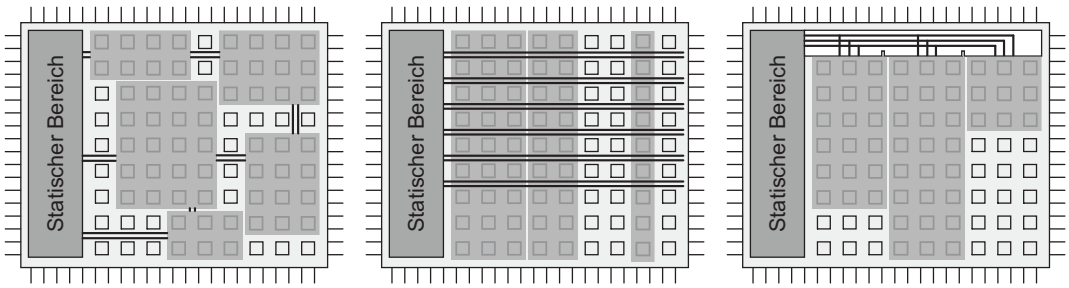
Die Adaption einer Netzwerkkomponente wird am Beispiel eines Dual-Port Ethernet-Switches untersucht. Für viele Netzwerkkomponenten im Bereich der Echtzeitkommunikation sind zwei Ports ausreichend, weil sich mit diesen die in diesem Anwendungsgebiet häufig vorkommenden Linien- und Ringtopologien realisieren lassen.

3.1.1 Die dynamische partielle Rekonfiguration eines FPGAs

Bevor im nächsten Abschnitt die Rekonfigurationsstrategien erläutert werden, die eine Rekonfiguration von einer Switchvariante zu einer anderen Switchvariante gewährleisten, soll dieser Abschnitt kurz die dynamische partielle Rekonfiguration eines FPGAs erläutern.

Bei der dynamischen partiellen Rekonfiguration werden Bereiche auf dem FPGA, unabhängig von der restlichen Konfiguration, zur Laufzeit umkonfiguriert. Das FPGA wird dazu in einen statischen und einen dynamisch partiell rekonfigurierbaren Bereich eingeteilt. Im statischen Bereich befinden sich die Systemkomponenten, die während des gesamten Betriebs benötigt werden, wie z. B. ein Prozessor und Speicher oder die in Kapitel 3.2 erläuterte Bussteuerung. In dem dynamisch partiell rekonfigurierbaren Bereich werden die rekonfigurierbaren Module zur Laufzeit beliebig platziert. Rekonfigurierbare Module sind z. B. der Hardware- und der Software-Switch.

Ziel der partiellen dynamischen Rekonfiguration ist es, die Ressourcen eines FPGAs effizienter zu nutzen. Rekonfigurierbare Module werden nur geladen, wenn sie von den Applikationen benötigt werden. Die Konfigurationsdaten von momentan nicht benötigten Modulen werden in einem Modulspeicher abgelegt. Die Fläche von dem FPGA wird so mehrfach belegt, und die Funktionalität von dem FPGA wird bei



(a) Zweidimensional (b) Eindimensional (c) Feste Modulplätze

Abbildung 3.2: Drei Platzierungsstrategien für rekonfigurierbare Module

gleicher Fläche erhöht. Voraussetzung dafür ist, dass nicht alle Funktionen gleichzeitig benötigt werden. Der Begriff Fläche eines FPGAs oder FPGA-Fläche wird äquivalent zu dem Begriff FPGA-Ressourcen verwendet.

Eine Voraussetzung für die Umsetzung eines dynamisch partiell rekonfigurierbaren Systems ist eine Kommunikationsinfrastruktur, die die Kommunikation zwischen den Modulen und dem statischen Bereich während und nach einer partiellen Rekonfiguration gewährleistet. Auch für die prototypische Umsetzung in Kapitel 3.2 spielt die Kommunikationsinfrastruktur eine große Rolle und wird daher an dieser Stelle näher untersucht.

Die Umsetzung der Kommunikationsinfrastruktur ist abhängig von der Platzierungsstrategie. Abbildung 3.2 zeigt drei mögliche Platzierungsstrategien, die in [KKK+05, GKP06] im Detail beschrieben werden. Die Abbildung 3.2(a) zeigt einen zweidimensionalen Ansatz, bei dem die Module eine beliebige Höhe und Weite annehmen können. Zur Laufzeit können diese Module an jede XY-Position platziert werden. Diese Art der Platzierung erfordert eine Kommunikationsinfrastruktur, die im Betrieb angepasst werden muss. Für eine statische Lösung stehen nicht genug Verbindungsstrukturen auf dem FPGA zur Verfügung, damit von jeder Position die Kommunikation zwischen den Modulen und zum statischen Bereich ermöglicht wird. Eine dynamische Anpassung der Kommunikationsstruktur ist ein sehr komplexes Problem, deren effiziente Realisierung auf heute verfügbaren FPGAs schwierig ist. Ein dynamisches Routen während des Betriebes, wie bei Hübner et al. beschrieben [HSB06], würde z. B. auf einem eingebetteten System zu einer viel zu hohen Rechenzeit führen.

Bei der eindimensionalen Platzierungsstrategie (Abbildung 3.2(b)) verwenden die Module die volle Höhe des Bausteins und die Weite der Module bleibt flexibel. Diese Platzierungsstrategie benötigt eine homogene Kommunikationsinfrastruktur, wie sie in [HKKP07b] vorgestellt wurde. Implementiert wird diese homogene Kommunikationsinfrastruktur aus sogenannten eingebetteten Makros, die sich horizontal über den gesamten dynamisch partiell rekonfigurierbaren Bereich erstrecken und auch während der

Rekonfiguration eines Moduls ihre Verbindung aufrechterhalten. Eingebettete Makros basieren entweder auf den *Tri-State*-Leitungen oder einer *Slice*[1]-basierten Kommunikationsstruktur. Der wesentliche Vorteil einer homogenen Kommunikationsinfrastruktur ist die Platzierung desselben Moduls an beliebige Positionen entlang der Horizontalen. Mit einer Manipulation des *Bitstreams* (Konfigurationsdatei für das FPGA) zur Laufzeit [KP06] wird die X-Position des Moduls verändert. Bei einer inhomogenen Kommunikationsinfrastruktur ist eine Verschiebung mit einer Bitstreammanipulation nicht möglich. Die Verbindungspunkte der Module mit der Infrastruktur befinden sich für beliebige Positionen in diesem Fall an unterschiedlichen Stellen des Moduls und somit muss für jede Position eine eigene Konfigurationsdatei erstellt werden. Die Flexibilität wird sich hier durch einen hohen Bedarf an Speicher für die Konfigurationsdateien erkauft.

Die in Abbildung 3.2(c) dargestellte Platzierungsstrategie basiert auf festen Modulplätzen mit einer festen Weite und Höhe. Ein Modulplatz kann zu einem Zeitpunkt immer nur ein rekonfigurierbares Modul beinhalten. Damit ist die Größe der rekonfigurierbaren Module an die Größe der Modulplätze gebunden. Auch kleine Module besetzen einen gesamten Modulplatz und belegen wesentlich mehr FPGA-Ressourcen, als sie benötigen. Die Anforderungen an die Kommunikationsinfrastruktur sind geringer als bei den anderen Platzierungsstrategien. Daher ist die Implementierung eines rekonfigurierbaren Systems mit festen Modulplätzen am einfachsten.

Der Einsatz verschiedener FPGA-Verbindungsstrukturen, die auch auf zukünftigen FPGAs eine Kommunikationsinfrastruktur für die partielle dynamische Rekonfiguration erlauben, wird in [HKKP07a] betrachtet. Die Ergebnisse aus [HKKP07a] bestätigen ebenfalls, dass die für die prototypische Umsetzung (Kapitel 3.2) verwendete Kommunikationsinfrastruktur sich für das rekonfigurierbare System des Switches am besten eignet.

3.1.2 Rekonfigurationsstrategien

Damit bei der Rekonfiguration von der aktiven Switchvariante zur neuen Variante keine Pakete verloren gehen, wurden verschiedene Rekonfigurationsstrategien entwickelt, die im Verlauf dieses Abschnitts beschrieben werden. Ein Paketverlust entsteht, wenn sich im aktiven Switch noch Pakete in den Puffern befinden und der Switch durch eine andere Variante einfach ersetzt wird. Insbesondere bei der Echtzeitkommunikation sollte ein Paketverlust vermieden werden, weil die meisten Echtzeitprotokolle auf eine wiederholte Übertragung desselben Paketes verzichten. Die erneute Übertragung eines Echtzeitpaketes kann dazu führen, dass andere Echtzeitdaten gestört werden und diese ihre Echtzeitkriterien nicht einhalten können. Des Weiteren erreichen die

[1] Logikblock eines Xilinx-FPGAs

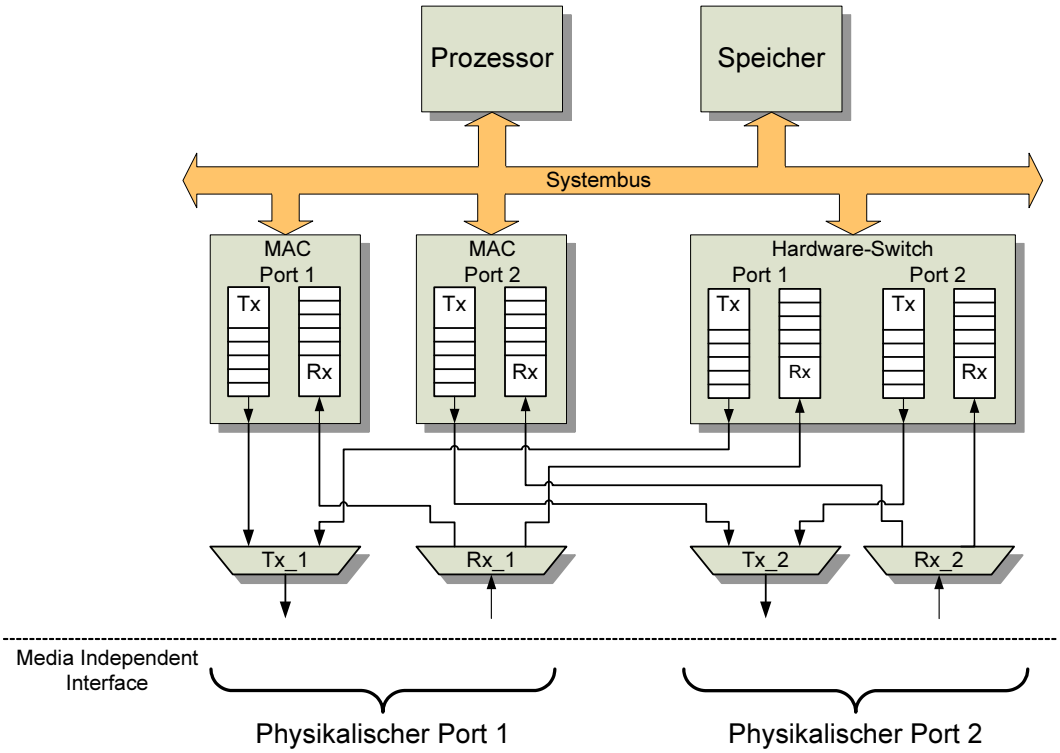


Abbildung 3.3: Architektur des rekonfigurierbaren Switches

Echtzeitdaten, wenn sie ein zweites Mal gesendet werden, meistens nicht mehr rechtzeitig ihr Ziel, und die Daten sind bereits veraltet, wenn sie z. B. von der Anwendung verarbeitet werden.

Voraussetzung für eine paketverlustfreie dynamische Rekonfiguration ist, dass während der Rekonfiguration der Software- und der Hardware-Switch, wie in der Abbildung 3.3 dargestellt, parallel zueinander auf dem FPGA geladen wurden. Auf der rechten Seite der Abbildung befinden sich die zwei MACs (Medium Access Controller) des Software-Switches und links daneben der Hardware-Switch. Der Prozessor, der Speicher, der Systembus und die vier Multiplexer sind feste Bestandteile des Systems und werden durch die Rekonfiguration des FPGAs nicht verändert. Die vier Multiplexer steuern den Zugriff der beiden Ethernet-Komponenten auf die gemeinsamen MII-Schnittstellen (Medien Independent Interface) und ermöglichen es, den Empfangs- und den Sendeprozess unabhängig voneinander umzuschalten.

Die Rekonfigurationsstrategien unterscheiden sich im Umgang mit den Paketen, die sich zu Beginn der Rekonfiguration noch in den Paketpuffern der zu ersetzenden Switchvariante befinden. Aus den Variationsmöglichkeiten sind drei Rekonfigurationsstrategien entstanden, die im Folgenden beschrieben und analysiert werden. Untersucht werden die Dauer des gesamten Rekonfigurationsprozesses und die Aktivierungszeit der neuen Switchvariante. Alle drei Rekonfigurationsstrategien können in beide Rich-

tungen vom Software- zum Hardware-Switch oder umgekehrt gleichermaßen verwendet werden. Erläutert werden sie am Beispiel der Rekonfiguration vom Software- zum Hardware-Switch.

Rekonfigurationsstrategie 1: Bei der ersten Strategie werden zuerst die Empfangssignale der beiden Ethernet-Ports von den Multiplexern Rx_1 und Rx_2 zum Hardware-Switch umgeschaltet. Damit dabei kein Paket zerstört wird, wird ein Paket, das beim Start der Rekonfiguration empfangen wird, vollständig in den entsprechenden Empfangspuffer (RX) des Software-Switches abgelegt und anschließend werden die dazugehörigen Empfangssignale vom Multiplexer innerhalb der im Ethernet-Standard definierten Paketlücke *Inter Frame Gap (IFG)* zum Hardware-Switch umgeschaltet. Fällt der Start der Rekonfiguration zufällig in die IFG, können die Signale für den entsprechenden Port direkt umgeschaltet werden. Das Umschalten erfolgt durch die Multiplexer eigenständig. Nachdem sie das Signal zur Rekonfigurationseinleitung erhalten haben, warten sie auf die nächste IFG und schalten dann um. Sind Rx_1 und Rx_2 umgeschaltet, werden neu ankommende Pakete in die Empfangspuffer des Hardware-Switches abgelegt. Pakete, die sich noch in den Empfangspuffern des Software-Switches befinden, werden vom Prozessor zu den Sendepuffern (TX) des Hardware-Switches umkopiert. Mit Abschluss des Kopiervorgangs ist die Rekonfiguration des Empfangsprozesses abgeschlossen.

Bei der Rekonfiguration des Sendeprozesses werden keine Pakete umkopiert, sondern die Multiplexer Tx_1 und Tx_2 schalten erst dann um, wenn alle Pakete, die sich in den Sendepuffern vom Software-Switch befinden, vollständig übertragen sind. Dazu müssen die MACs den beiden Multiplexern mitteilen, wenn die Puffer leer sind. Auch während der Rekonfiguration muss der Sendeprozess eine minimale IFG zwischen zwei Paketen einhalten. Daher muss nach der letzten Übertragung die Zeit für eine minimale IFG abgewartet werden, bevor die Sendesignale zum Hardware-Switch umgeschaltet werden. Die Rekonfiguration des Sende- und Empfangsprozesses ist unabhängig voneinander und kann parallel erfolgen. Ist die Rekonfiguration beider Prozesse abgeschlossen, kann der Software-Switch vom FPGA entfernt werden.

Der Vorteil dieser Methode ist eine sehr schnelle Aktivierung des Hardware-Switches. Er erhält die vollständige Kontrolle über beide physikalische Ports, sobald alle Pakete aus dem Sendepuffer des Software-Switches übertragen sind und die Zeit T_{IFG} für eine IFG abgelaufen ist. Damit entspricht die Rekonfigurationszeit des Sendeprozesses der Zeit bis zur Aktivierung des Hardware-Switches und ist gleich

$$T_{\mathrm{TX_1}} = n_{\mathrm{TX}} \cdot \frac{l+h}{r} + T_{\mathrm{IFG}} \tag{3.1}$$

unter der Voraussetzung, dass n_{TX} die Anzahl der Pakete im Sendepuffer des Software-Switches ist, die Paketlänge l für alle Pakete gleich groß ist und r der Datenrate des Netzwerkes entspricht. Die Paketlänge l besteht in diesem Fall aus den Nutzdaten eines Ethernet-Paketes, den 12 Bytes für Quell- und Zieladresse und den 2 Bytes für das Typfeld. Es ist dieselbe Datenmenge, die beim Empfangsprozess vom Prozessor kopiert wird. Bei der Übertragung eines Ethernet-Paketes über das Netzwerk erhöht sich die Paketlänge noch um den Faktor h, der mit insgesamt 24 Byte aus der Präambel, dem SFD, der FCS und der IFG besteht. Die Rekonfigurationszeit des Empfangsprozesses ist

$$T_{\mathrm{RX_1}} = n_{\mathrm{RX}} \cdot \frac{l}{g}, \tag{3.2}$$

mit der Anzahl der Pakete im Empfangspuffer n_{RX}, der Datenrate g, in der der Prozessor die Pakete über den Systembus in den Sendepuffer des Hardware-Switches schreibt. Da die Rekonfiguration des Empfangsprozesses und des Sendeprozesses parallel zueinander abläuft, ist der gesamte Umschaltvorgang nach $T_{\mathrm{Um_1}} = \max(T_{\mathrm{TX_1}}, T_{\mathrm{RX_1}})$ abgeschlossen. In der prototypischen Implementierung ist die Rekonfigurationszeit des Empfangsprozesses für die Gesamtzeit des Umschaltvorgangs ausschlaggebend, weil die Datenrate g deutlich geringer ist als r.

Ein Nachteil der schnellen Aktivierung ist eine mögliche Vertauschung der Paketreihenfolge. Sie entsteht, wenn ein vom Hardware-Switch empfangenes Paket weitergeleitet wird, bevor die restlichen Pakete aus den Empfangspuffern des Software-Switches zum Hardware-Switch kopiert sind. Eine Vertauschung der Paketreihenfolge entsteht nicht, wenn innerhalb des Zeitfensters $T_{\mathrm{w}} = (l+h)/r$, in dem der Hardware-Switch ein Paket empfangen kann, alle Pakete aus den Empfangspuffern kopiert werden. Wird das Weiterleiten der Pakete im Hardware-Switch gestoppt, bis die Rekonfiguration des Empfangsprozesses abgeschlossen ist, wird eine Vertauschung der Paketreihenfolge ebenfalls vermieden. Allerdings erhöht sich für $T_{\mathrm{RX_1}} > T_{\mathrm{TX_1}}$ die Aktivierungszeit des Hardware-Switches und die Empfangspuffer der Hardwarevariante können überlaufen. Ein Überlauf des Empfangspuffers führt auch zu einem Paketverlust. Daher benötigt der Hardware-Switch je nach Anforderung entweder größere Puffer, um einen Paketverlust möglichst zu vermeiden, oder eine Vertauschung der Paketreihenfolge wird toleriert.

Rekonfigurationsstrategie 2: Die zweite Rekonfigurationsstrategie besitzt im Vergleich zur ersten Rekonfigurationsstrategie eine veränderte Rekonfiguration des Empfangsprozesses. Nachdem die Empfangssignale zum Hardware-Switch umgeschaltet sind, werden die restlichen Pakete aus den beiden Empfangspuffern des Software-

Switches nicht zum Hardware-Switch kopiert, sondern zu den entsprechenden Sendepuffern des Software-Switches. In diesem Fall leitet der Software-Switch alle in seinen Empfangspuffern verbliebenen Pakete eigenständig weiter. Die Sendesignale werden erst umgeschaltet, wenn beide Puffer leer sind. Der Hardware-Switch wird bei dieser Strategie erst nach Abschluss des gesamten Umschaltvorgangs vollständig aktiviert.

Für die Zeit des gesamten Umschaltvorgangs müssen die zwei Fälle $g > r$ und $g < r$ unterschieden werden. Für den Fall $g > r$ bestimmt die Dauer des Sendeprozesses die Zeit, die für den Umschaltvorgang benötigt wird. Im Gegensatz zur Strategie 1 müssen in diesem Fall zusätzlich zu der Anzahl der Pakete n_{TX}, die sich im Sendepuffer befinden, noch die Anzahl der Pakete aus dem Empfangspuffer n_{RX} übertragen werden, womit

$$T_{\mathrm{TX}_2} = \frac{(n_{\mathrm{TX}} + n_{\mathrm{RX}}) \cdot (l + h)}{r} + T_{\mathrm{IFG}} \tag{3.3}$$

ergibt. Für $g < r$ ist der Kopiervorgang und die Übertragung des letzten Paketes aus dem Sendepuffer ausschlaggebend für den gesamten Umschaltvorgang und wird durch

$$T_{\mathrm{RX}_2} = n_{\mathrm{RX}} \cdot \frac{l}{g} + \frac{l + h}{r} + T_{\mathrm{IFG}} \tag{3.4}$$

beschrieben. Die Gleichung (3.4) ist die Summe aus der Zeit, die zum Kopieren der restlichen empfangenen Pakete notwendig ist und aus der Zeit, die zur Übertragung des letzten Paketes aus dem Sendepuffer benötigt wird. Die vorangegangenen Pakete werden parallel zum Kopiervorgang übertragen. Allerdings kann das letzte Paket erst dann gesendet werden, wenn es vollständig in den Puffer kopiert wurde. Deshalb dauert der gesamte Umschaltvorgang für den Fall $g < r$ etwas länger als bei der ersten Strategie. Die Gefahr eines Pufferüberlaufs ist daher größer und kann durch einen größeren Puffer im Hardware-Switch kompensiert werden. Angaben zu den notwendigen Puffergrößen in Abhängigkeit des Umschaltzeitpunkts erfolgen in Kapitel 3.3.5. Die Zeit für den gesamten Umschaltvorgang der Strategie 2 T_{Um_2} wird aus dem Maximum der Gleichungen (3.3) und (3.4) mit $T_{\mathrm{Um}_2} = \max(T_{\mathrm{TX}_2}, T_{\mathrm{RX}_2})$ bestimmt.

Bei der zweiten Strategie entsteht keine Vertauschung der Paketreihenfolge. Im Gegensatz zu Strategie 1 muss dazu der Weiterleitungsprozess im Hardware-Switch nicht gestoppt werden.

Rekonfigurationsstrategie 3: Bei der dritten Rekonfigurationsstrategie werden die Empfangs- und Sendesignale unabhängig von der Anzahl der Pakete, die sich noch in den Puffern des Software-Switches befinden, zum Hardware-Switch umgeschaltet.

Auch bei dieser Strategie schalten die Multiplexer Rx_1 und Rx_2 nur innerhalb einer IFG die Signale der physikalischen Schnittstelle zum Hardware-Switch. Das Gleiche gilt ebenso für die Multiplexer Tx_1 und Tx_2. Auch sie schalten erst um, nachdem das Senden des aktuellen Paketes abgeschlossen ist. Der Inhalt der Sende- und Empfangspuffer wird anschließend vom Prozessor über den Systembus in den entsprechenden Sendepuffer des Hardware-Switches umkopiert. Damit ist der gesamte Umschaltvorgang abgeschlossen.

Bei diesem Verfahren wird der Hardware-Switch am schnellsten aktiviert und kann Pakete weiterleiten. Das zusätzliche Kopieren der Pakete aus dem Sendepuffer führt jedoch zur Erhöhung der Zeit für den gesamten Umschaltvorgang auf

$$T_{\mathrm{Um}_3} = (n_{\mathrm{RX}} + n_{\mathrm{TX}}) \cdot \frac{l}{g}. \tag{3.5}$$

Können nicht alle Pakete in dem Zeitfenster T_{w} kopiert werden, kommt es auch bei diesem Verfahren zu einer Vertauschung der Paketreihenfolge. Wie bei der ersten Strategie kann der Weiterleitungsprozess des Hardware-Switches gestoppt werden, bis alle Pakete kopiert wurden. Eine Vertauschung der Paketreihenfolge ist somit ausgeschlossen. Der Hardware-Switch wird dann erst nach T_{Um_3} aktiviert. Hierbei handelt es sich um die bisher größte Aktivierungszeit. Das Risiko eines Pufferüberlaufs und damit auch das Risiko eines Paketverlustes ist in diesem Fall am größten. Im Gegensatz zu den Strategien 1 und 2 benötigen die Sendemultiplexer keine Statusinformation über leere Sendepuffer von den MACs. Diese Strategie kann eingesetzt werden, wenn MACs verwendet werden, die diesen Pufferstatus nicht nach außen weitergeben.

Vergleich: Einen Vergleich der Aktivierungszeit und der Umschaltzeit der drei Rekonfigurationsstrategien für den Fall $g < r$ zeigt die Tabelle 3.1 und für den Fall $g > r$ die Tabelle 3.2. Die Werte wurden unter der Annahme ermittelt, dass zum Startzeitpunkt des Umschaltvorgangs kein Paket empfangen oder übertragen wird, und die Übertragungsrate r den 100 MBit/s von FastEthernet entspricht. Für den Fall $g < r$ wird eine Kopierrate g von 20 MBit/s gewählt, die mit der Kopierrate der Implementierung des Software-Switches übereinstimmt. Im Fall $g > r$ wird eine Kopierrate von 400 MBit/s angenommen, die unter optimalen Bedingungen vom Software-Switch erreicht werden könnte. Beide Tabellen enthalten Aktivierungs- und Umschaltzeiten für die minimale und die maximale Paketgröße sowie für verschiedene Füllstände des Empfangs- und des Sendepuffers. Die Zeiten werden in beiden Tabellen in Mikrosekunden und ohne Nachkommastellen dargestellt.

Die Anzahl der Pakete im Empfangspuffer variiert in Tabelle 3.1 zwischen null und vier Paketen. Die vier Pakete entsprechen dabei der Größe der Paketpuffer der por-

	Anzahl der Paket im		Aktivierungszeit (μs) für Strategie			Umschaltvorgang (μs) für Strategie		
Fall	**RX-Puffer**	**TX-Puffer**	**1**	**2**	**3**	**1**	**2**	**3**
a	0	1	7	7	1	0	7	18
a	1	0	1	25	1	18	25	18
a	4	0	1	80	1	74	80	74
a	4	1	7	80	1	74	80	92
b	0	1	123	123	1	0	123	600
b	1	0	1	723	1	600	723	600
b	4	0	1	2523	1	2400	2523	2400
b	4	1	123	2523	1	2400	2523	3000

a: Nutzdaten 46 Bytes; b: Nutzdaten 1500 Bytes

Tabelle 3.1: Vergleich der Rekonfigurationsstrategien für $g < r$

totypischen Implementierung des rekonfigurierbaren Switches. Im Sendepuffer kann sich nur maximal ein Paket befinden, weil die Pakete schneller vom Sendepuffer aus übertragen werden, als sie in diese hineinkopiert werden.

Alle drei Strategien erreichen die kürzeste Umschaltzeit, falls sich nur ein Paket minimaler Länge im Sendepuffer und kein Paket im Empfangspuffer befindet. Für die Strategie 2 ergibt sich in diesem Fall auch die kürzeste Aktivierungszeit. Die Strategie 1 erlangt die kürzeste Aktivierungszeit von 1 μs, wenn sich kein Paket im Sendepuffer befindet. Die Aktivierungszeit von Strategie 3 ist unabhängig von der Paketlänge und der Anzahl der Pakete in den Puffern. Sie entspricht unter der oben genannten Annahme immer der Zeit einer IFG. Die höchste Aktivierungszeit von 123 μs erhält die Strategie 1, wenn sich ein Paket maximaler Länge im Sendepuffer befindet. Diese ist aber noch deutlich geringer als die höchste Aktivierungszeit von Strategie 2. Sie beträgt 2523 μs und entsteht, wenn sich vier Pakete maximaler Länge im Empfangspuffer befinden. In diesem Fall dauert auch der Umschaltvorgang von Strategie 1 und 2 am längsten. Die höchste Umschaltzeit wird von der Strategie 3 erreicht, wenn sich neben den vier Paketen im Empfangspuffer noch ein Paket im Sendepuffer befindet. Haben alle Pakete die maximale Paketlänge, dauert der Umschaltvorgang in diesem Fall 3000 μs.

Die Strategien 1 und 3 ermöglichen für den Fall $g < r$ eine sehr schnelle Aktivierung des Hardware-Switches, können aber eine Vertauschung der Paketreihenfolge verursachen. Eine Vertauschung der Paketreihenfolge kann durch eine Nummerierung der Pakete, wie sie z. B. bei TCP/IP verwendet wird, im Empfänger wieder rückgängig gemacht werden. Bedingt durch die schnelle Aktivierung sind Pufferüberläufe im Hardware-Switch bei diesen beiden Strategien nicht möglich. Auch der gesamte Um-

schaltvorgang wird bei der ersten Strategie in allen Fällen am schnellsten oder wenigstens gleichschnell abgeschlossen.

Bei Strategie 2 dauert der gesamte Umschaltvorgang etwas länger als bei Strategie 1, weil nach dem Kopiervorgang noch das letzte Paket aus dem Sendepuffer übertragen werden muss, bevor zum Hardware-Switch umgeschaltet wird. Der Umschaltvorgang ist bei der Strategie 2 erst dann beendet, wenn der Hardware-Switch aktiviert ist. Die Aktivierungszeit ist aus diesem Grunde auch größer als bei den anderen Strategien. Daher existiert bei dieser Strategie das Risiko, dass die Puffer überlaufen und Pakete verloren gehen. Dieses Risiko wird dadurch minimiert, dass der Start der automatischen Rekonfiguration (siehe Abschnitt 3.1.3) erfolgt, bevor die Netzwerklast zu groß wird. Der Software-Switch kann nur bis zu einer bestimmten Netzwerklast alle Pakete ohne Verlust weiterleiten. Während des Umschaltvorgangs leitet der Software-Switch ebenfalls die restlichen Pakete weiter. Wird die Netzwerklast für den Software-Switch während des Umschaltens nicht überschritten, laufen auch die Puffer des Hardware-Switches nicht über.

Die dritte Strategie kann direkt zum Hardware-Switch umschalten, wenn keine Pakete gesendet oder empfangen werden. Der gesamte Umschaltvorgang dauert jedoch im schlechtesten Fall von allen drei Strategien am längsten. Im Gegensatz zu den anderen Strategien muss gegebenenfalls noch zusätzlich ein Paket aus dem Sendepuffer mit der geringen Kopierrate des Prozessors in die Puffer des Hardware-Switches übertragen werden.

Im Fall $g > r$ werden alle Pakete schneller vom Empfangs- in den Sendepuffer kopiert, als sie überhaupt empfangen werden können. Aus diesem Grund kann im Empfangspuffer kein Paketstau entstehen. Zum Start der Rekonfiguration befindet sich im Puffer daher nur das kurz zuvor empfangene Paket. Im Sendepuffer ist ein Paketstau ebenfalls nicht möglich, weil die Pakete mit der gleichen Datenrate empfangen und gesendet werden. Daher befindet sich auch im Sendepuffer maximal das zuletzt kopierte Paket. Aus den genannten Gründen variieren die Pufferstände in der Tabelle 3.2 nur zwischen 0 und 1.

Die geringste Aktivierungs- und Umschaltzeit haben die Strategien 1 und 3, wenn sich ein Paket minimaler Länge im Empfangspuffer befindet. Bei der Strategie 2 muss in diesem Fall das Paket noch in den Empfangspuffer kopiert werden, bevor es übertragen wird. Daher ist die Dauer der Aktivierung und der Umschaltung am geringsten, wenn sich nur ein Paket minimaler Länge im Sendepuffer befindet. Die Aktivierungszeit von Strategie 3 wird auch im Fall $g > r$ durch die IFG bestimmt. Strategie 1 hat seine höchste Aktivierungs- und Umschaltzeit, sobald sich ein Paket maximaler Länge im Empfangspuffer befindet. Strategie 2 erreicht seine maximale Aktivierungs- und Übertragungszeit, wenn sich in beiden Puffern ein Paket maximaler Länge befindet.

	Anzahl der Paket im		Aktivierungszeit (μs) für Strategie			Umschaltvorgang (μs) für Strategie		
Fall	**RX-Puffer**	**TX-Puffer**	**1**	**2**	**3**	**1**	**2**	**3**
a	0	1	7	7	1	7	7	1
a	1	0	1	8	1	1	8	1
a	1	1	7	12	1	7	12	2
b	0	1	123	123	1	123	123	30
b	1	0	1	153	1	1	153	30
b	1	1	123	245	1	123	245	60

a: Nutzdaten 46 Bytes; b: Nutzdaten 1500 Bytes

Tabelle 3.2: Vergleich der Rekonfigurationsstrategien für $g > r$

In diesem Fall erreicht auch die Strategie 3, die am meisten von der hohen Kopierrate g profitiert, ihre maximale Umschaltzeit.

Die prototypische Umsetzung des rekonfigurierbaren Ethernet-Switches verwendet die Rekonfigurationsstrategie 1 für die Rekonfiguration vom Software- zum Hardware-Switch und die Strategie 2 für die Rekonfiguration vom Hardware- zum Software-Switch. Rekonfigurationsstrategie 3 eignet sich insbesondere für Implementierungen, die keine Statusinformationen über die Pufferstände erfassen oder diese nicht nach außen weitergeben. Die prototypische Umsetzung (Kapitel 3.2) liefert die notwendigen Statusinformationen an die Multiplexer, und der Einsatz der Rekonfigurationsstrategie 3 ist nicht notwendig.

3.1.3 Automatische Rekonfiguration des Switches

Damit sich der rekonfigurierbare Switch an sich ändernde Kommunikationslasten und sich ändernde Echtzeitanforderungen selbstständig durch eine Rekonfiguration ohne Paketverlust anpassen kann, müssen diese Veränderungen vom Switch detektiert werden. Die Rekonfiguration muss daraufhin automatisch erfolgen. Mögliche Auslöser für den automatischen Start einer Rekonfiguration sind die Prozessorauslastung, die Netzwerklast und die Anzahl der Pakete, die sich in den Empfangspuffern befinden. Diese drei Parameter sind Indikatoren für eine Rekonfiguration vom Software- zum Hardware-Switch und umgekehrt und werden daher vom rekonfigurierbaren Switch ermittelt und ausgewertet. Bei einer zu hohen Prozessorauslastung oder bei einer zu hohen Netzwerklast können z. B. nicht mehr alle Pakete mit der Geschwindigkeit vom Software-Switch weitergeleitet werden, mit der sie ihn erreichen. Die Puffer der Switches laufen dann über. Wird eine Erhöhung rechtzeitig erkannt und eine Rekonfiguration zum Hardware-Switch eingeleitet, passt sich das System an die höhere Last an und ein Paketverlust wird vermieden. Im umgekehrten Fall wird bei einer geringen

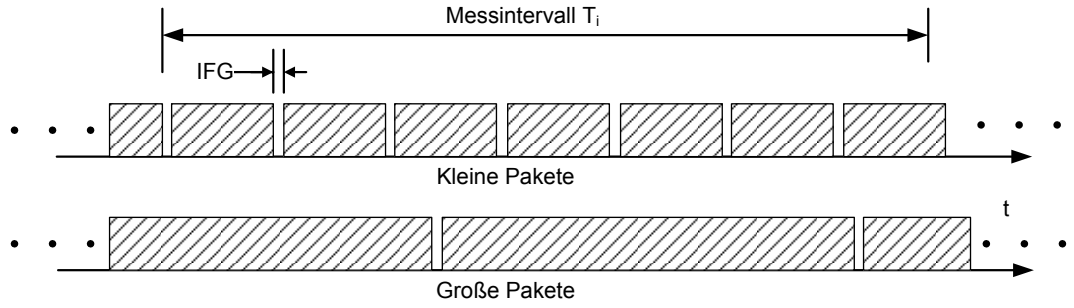


Abbildung 3.4: Bestimmung der Netzwerklast

Prozessorauslastung und einer geringen Netzwerklast der Hardware-Switch nicht weiter benötigt und kann durch den Software-Switch ersetzt werden, um Ressourcen für andere Applikationen auf dem FPGA frei zu machen.

Die Netzwerklast wird direkt in der Hardwareimplementierung des MAC ermittelt, nicht nur um den Prozessor von einer weiteren Aufgabe zu entlasten, sondern auch weil eine Hardwareimplementierung genauere Messergebnisse liefert. Weil die Länge von Ethernet-Paketen (mit Präambel und SFD) zwischen 72 und 1526 Bytes variiert, ist die Paketrate für die Bestimmung der Netzlast ungeeignet. Die Paketrate liefert innerhalb eines definierten Zeitintervalls unterschiedliche Ergebnisse für verschieden große Pakete. In Abbildung 3.4 wird deutlich, dass in dem skizzierten Messintervall mehr kleinere als größere Pakete empfangen werden. Anstelle der Paketrate wird bei dem rekonfigurierbaren Ethernet-Switch die Zeit T_{DV} in einem Intervall T_{I} gemessen, in der der MAC Pakete empfängt. Zur exakten Zeitmessung von T_{DV} wird zu Beginn des Empfangs eines jeden Paketes die Messung gestartet und zum Empfangsende wieder gestoppt. In der Implementierung werden Paketanfang und Paketende über das *Data-Valid*-Signal (RX_DV) der MII-Schnittstelle detektiert. Die Netzwerklast L_{Netz} errechnet sich dann aus:

$$L_{\mathrm{Netz}} = \frac{n_{\mathrm{IFG}} \cdot T_{\mathrm{IFG}} + T_{\mathrm{DV}}}{T_{\mathrm{I}}} \tag{3.6}$$

Eine Netzwerklast L_{Netz} von 1 entspricht der maximalen Datenrate. Bei Ethernet muss die IFG bei der maximalen Datenrate zwischen jedem Paket eingehalten werden. Daher wird die Anzahl der IFG im Messintervall T_{I} detektiert und durch den Faktor n_{IFG} repräsentiert. T_{IFG} entspricht der Zeit einer minimalen IFG des angewandten Ethernet-Protokolls (0,96 μs für 100 MBit/s).

Zur Bestimmung der Prozessorauslastung wird ein Verfahren genutzt, welches die Anzahl der Schleifendurchläufe n_{Last} im Leerlauf-Task ermittelt [Tra04]. Zur Initialisierung dieser Messmethode wird die Anzahl der Schleifendurchläufe n_{Leer} in einem vor-

definierten Intervall T_I im lastfreien Fall bestimmt. Als Prozessorauslastung L_CPU wird die Zeit definiert, die der Prozessor innerhalb von T_I nicht im Leerlauf-Task verbringt:

$$L_\mathrm{CPU} = 1 - \frac{n_\mathrm{Last}}{n_\mathrm{Leer}} \tag{3.7}$$

Eine weitere Möglichkeit, eine automatische Rekonfiguration einzuleiten, ist die Überwachung der Füllstände der Empfangspuffer. Übersteigt die Anzahl der Pakete in einem Empfangspuffer eine gewisse Anzahl, dann leitet der Prozessor eine Rekonfiguration vom Software- zum Hardware-Switch ein. Um einen Paketverlust zu vermeiden, sollte die Einleitung der Rekonfiguration zu einem Zeitpunkt erfolgen, in dem die Empfangspuffer noch genügend Platz für Pakete bieten, die während der Konfiguration des Hardware-Switches auf den FPGA empfangen werden.

Trigger für eine automatische Rekonfiguration sind auch sich ändernde Echtzeitanforderungen (Latenz und Jitter). Diese Anforderungen kommen von Applikationen oder von benachbarten Knoten, die z. B. für die Kommunikation über diesen Knoten eine geringe Latenz und einen geringen Jitter benötigen. Können diese Anforderungen vom Software-Switch nicht erfüllt werden, muss der Hardware-Switch auf das FPGA konfiguriert werden. Auch eine höhere Datenrate kann von einem Nachbarknoten angefordert werden und eine Rekonfiguration einleiten. Verarbeitet und ausgewertet werden diese Anforderungen von einem Echtzeitbetriebssystem, welches dann die Entscheidung zur Rekonfiguration trifft. Ebenfalls werden vom Echtzeitbetriebssystem die gemessene Netzwerklast und die ermittelte Prozessorlast ausgewertet. Übersteigt sie mehrfach hintereinander einen bestimmten Schwellwert, leitet das Echtzeitbetriebssystem die automatische Rekonfiguration vom Software- zum Hardware-Switch ein. Die Auswertung mehrerer Messungen verhindert, dass das System nur aufgrund einer einzelnen Lastspitze rekonfiguriert wird. In die umgekehrte Richtung wird konfiguriert, wenn der Schwellwert für längere Zeit unterschritten wird.

Die Zusammenarbeit von dem Echtzeitbetriebssystem DREAMS mit der prototypischen Implementierung des rekonfigurierbaren Ethernet-Switches wird in [OBG05] beschrieben. Das Betriebssystem verwendet einen flexiblen Ressourcenmanager, der den Hardware- und den Software-Switch als Profile verwaltet. In einem Profil sind die Datenrate des jeweiligen Switches und der Ressourcenbedarf eines Switches hinterlegt. Steigt die Kommunikationslast oder fordert eine Anwendung eine höhere Datenrate an, erkennt der Ressourcenmanager anhand des Profils, dass der Hardware-Switch geladen werden muss. Verringert sich die Kommunikationslast, wird vom Ressourcenmanager der Software-Switch geladen. Die freien Ressourcen stellt der Ressourcenmanager anderen Applikationen zur Verfügung. In einer Fallstudie für ein selbstoptimierendes System wurden beim aktiven Software-Switch die freien Ressourcen für eine Fließkommaeinheit verwendet, die Optimierungsalgorithmen beschleunigt [GOP05].

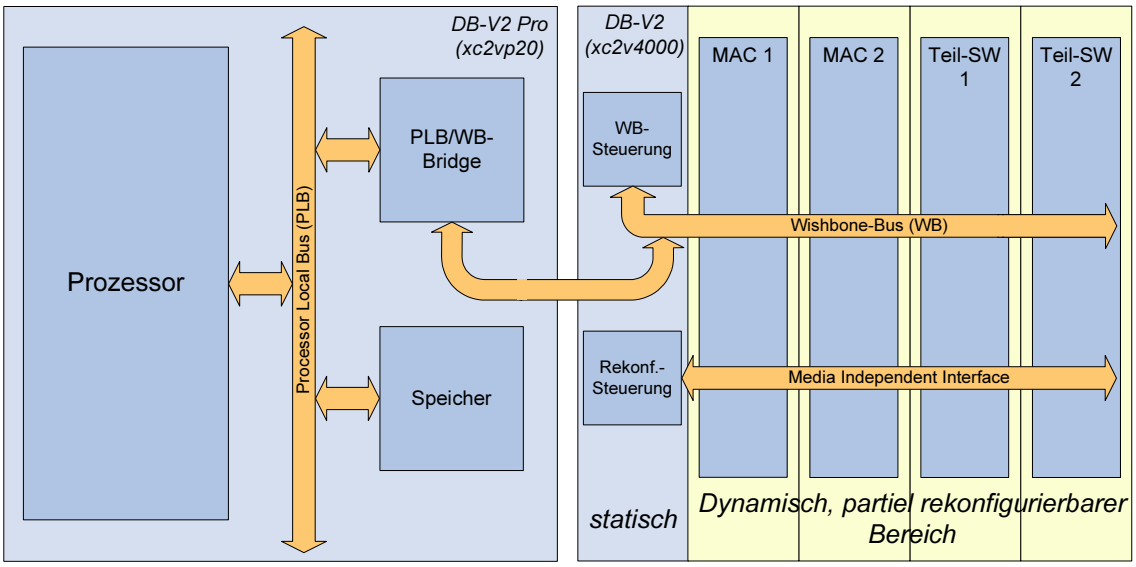


Abbildung 3.5: Aufbau des rekonfigurierbaren Gesamtsystems

Der Prozess der dynamischen partiellen Rekonfiguration eines FPGAs wird nur vom Betriebssystem eingeleitet. Gesteuert wird die partielle dynamische Rekonfiguration von einem Rekonfigurationsmanager, der in [GVPR04] beschrieben wird. Der Rekonfigurationsmanager lädt den zu konfigurierenden Bitstrom aus dem Speicher in den FPGA, überwacht den Prozess der Rekonfiguration des FPGAs und sorgt im Fehlerfall dafür, dass ein fehlerhaft konfiguriertes Modul nicht aktiviert werden kann.

3.2 Prototypische Umsetzung

Bei der prototypischen Umsetzung des rekonfigurierbaren Ethernet-Switches wurde das Gesamtsystem, wie in Abbildung 3.5 dargestellt, auf zwei FPGAs aufgeteilt [Sch05]. Das Prozessorsystem wurde auf einen Xilinx Virtex-II Pro FPGA mit eingebettetem PowerPC-Prozessor implementiert. Der dynamisch rekonfigurierbare Bereich befindet sich auf einem Xilinx Virtex-II FPGA. Die homogene Struktur des Virtex-II ermöglicht es, wie bereits in Abschnitt 3.1.1 erwähnt, rekonfigurierbare Module an verschiedenen Positionen des FPGAs zu platzieren. In der prototypischen Umsetzung wird der dynamisch rekonfigurierbare Bereich genutzt, um die Module des Hardware- und des Software-Switches unterzubringen. Abbildung 3.5 zeigt eine Konfiguration, bei der beide Switches nebeneinander positioniert sind. Implementiert wurde das Gesamtsystem auf der an der Universität Paderborn im Fachgebiet Schaltungstechnik entwickelten Rapid-Prototyping-Plattform RAPTOR2000 [KPR02].

Das Prozessorsystem auf dem ersten FPGA und der rekonfigurierbare Bereich sind über einen Wishbone-Bus miteinander verbunden. Im rekonfigurierbaren Bereich be-

steht der Wishbone-Bus aus den in Kapitel 3.1.1 beschriebenen *Tri-State*-Leitungen, die auch während der dynamischen Rekonfiguration eines Moduls die Kommunikation zwischen den anderen Modulen gewährleisten. Neben der Wishbone-Steuerung befindet sich im statischen Bereich des Virtex-II auch noch die Rekonfigurationssteuerung für den rekonfigurierbaren Switch. Über sie wird der Zugriff von Hardware- und Software-Switch auf die physikalischen Ethernet-Schnittstellen gesteuert. Die Rekonfigurationssteuerung ist mit den Signalleitungen der MII-Schnittstellen von Hardware- und Software-Switch ebenfalls über *Tri-State*-Leitungen verbunden.

Die Umsetzungen des Software- und des Hardware-Switches, der Rekonfigurationssteuerung und des Wishbone-Busses werden in den nächsten Abschnitten beschrieben. Die Beschreibung konzentriert sich auf die Besonderheiten, die zur Umsetzung eines partiell dynamisch rekonfigurierbaren Systems und der paketverlustfreien Rekonfiguration notwendig sind. Eine detaillierte Beschreibung aller Teilkomponenten des Prototyps können in [Fer05, Sch05] nachgelesen werden.

3.2.1 Software-Switch

Der Software-Switch besteht aus zwei für die Anforderungen der dynamischen Rekonfiguration entwickelten *Medium Access Controllern* (MAC). Jeder MAC lässt sich in eine Empfangs- und eine Sendesteuerung mit einem Empfangs- und einem Sendepuffer unterteilen. Die Empfangssteuerung überprüft das Rahmenformat der ankommenden Pakete und schreibt die Daten in den Empfangspuffer. Nachdem ein Paket vollständig in den Puffer geschrieben wurde und die Empfangssteuerung keine Paketfehler festgestellt hat, kann das Paket über die Wishbone-Busschnittstelle vom Prozessor ausgelesen und weitergeleitet werden. Die Sendesteuerung kapselt die Daten, die vom Prozessor in den Sendepuffer geschrieben werden, in das Ethernet-Paketformat und sendet das Ethernet-Paket über die MII-Schnittstelle an den physikalischen Ethernet-Port.

Für eine Rekonfiguration ohne Paketverlust und zur Anwendung der Rekonfigurationsstrategien wurde dieser MAC gegenüber einer Standardimplementierung erweitert. Von der Rekonfigurationssteuerung aus kann die Sendesteuerung angehalten werden. Dadurch wird verhindert, dass Daten aus dem Sendepuffer übertragen werden, bevor die Rekonfigurationssteuerung die Sendesignale zum MAC umgeschaltet hat. Auch die Einhaltung der IFG zwischen zwei Paketen beim Umschalten wird über diese Steuermöglichkeit gewährleistet. Des Weiteren übermittelt der MAC der Rekonfigurationssteuerung, dass der Sendepuffer keine Pakete mehr enthält. Daraufhin schaltet der Hardware-Switch die Sendesignale um. Die automatische Rekonfiguration (Kapitel 3.1.3) unterstützt der MAC, indem er die Netzwerklast ermittelt und dem Prozessor

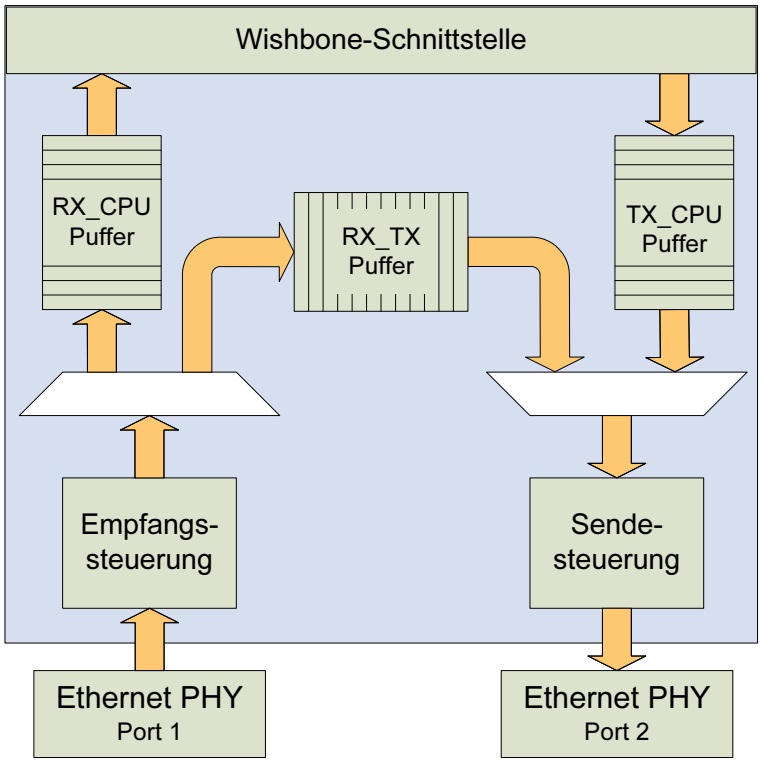


Abbildung 3.6: Aufbau einer Teilswitchkomponente

Statusinformationen über die Füllstände von Empfangs- und Sendepuffer zur Verfügung stellt.

3.2.2 Hardware-Switch

Sowohl der Hardware-Switch als auch der Software-Switch leiten Pakete nach dem *Store-and-Forward-Switching*-Verfahren weiter. Bei diesem Verfahren werden Pakete erst weitergeleitet, nachdem sie vollständig empfangen wurden. Auf diese Weise kann das komplette Paket auf Fehler überprüft werden und es wird verhindert, dass fehlerhafte Pakete weitergeleitet werden.

Der Switch besteht aus zwei gleichartigen Teilswitchkomponenten, die unabhängig voneinander die Pakete in jeweils eine Richtung weiterleiten. Jeder Teilswitch ist mit dem Wishbone-Bus verbunden, damit der Prozessor auch bei einem aktiven Hardware-Switch Pakete empfangen und senden kann. Die Abbildung 3.6 zeigt den internen Aufbau einer Teilswitchkomponente. Der Teilswitch ist eine Erweiterung des MAC des Software-Switches und besitzt einen weiteren Puffer, in den alle ankommenden Pakete, die weitergeleitet werden müssen, geschrieben werden. Pakete, die an den Prozessor gerichtet sind, werden, wie bei dem MAC, in den Empfangspuffer geschrieben, der mit dem Wishbone-Bus verbunden ist. Der Zugriff von der Empfangssteuerung auf die

beiden Puffer erfolgt über einen Multiplexer. Ein weiterer Multiplexer verbindet den Sende- und Weiterleitungspuffer mit der Sendesteuerung.

Aufgrund der einheitlichen Struktur von MAC und Teilswitch kann derselbe Softwaretreiber verwendet werden, um auf den MAC oder auf den Teilswitch zu zugreifen. Gegenüber den Applikationen verhalten sich beide Switches völlig transparent und bei einer Rekonfiguration von einer Switchvariante zur anderen können alle Applikationen weiter über den Switch kommunizieren, ohne dass eine Anpassung der Software notwendig wäre. Zur Unterstützung der dynamischen Rekonfiguration ermittelt der Teilswitch ebenfalls die Netzwerklast und auch die Sendesteuerung lässt sich für die Rekonfiguration anhalten.

3.2.3 Rekonfigurationssteuerung

Die Rekonfigurationssteuerung überwacht und regelt den Zugriff von Software- und Hardware-Switch auf die MII-Schnittstellen der zwei physikalischen Ethernet-Ports. Die Rekonfigurationssteuerung ermöglicht die in Kapitel 3.1.2 vorgestellten Verfahren zur paketverlustfreien dynamischen Rekonfiguration eines Switches. Hauptbestandteil der Rekonfigurationssteuerung sind Multiplexer, die die Empfangs- und Sendesignale der MII-Schnittstellen separat umschalten können. Auch die Steuerung der Multiplexer ist in die zwei Bereiche Empfang und Senden aufgeteilt.

Nachdem der Prozessor den Umschaltvorgang eingeleitet hat, überprüft die Steuerung des Empfangsmultiplexers zunächst, ob das RX_DV-Signal der MII getrieben wird und wartet gegebenenfalls bis zum Auftreten der IFG. Innerhalb der nächsten detektierten IFG schaltet die Steuerung über den Empfangsmultiplexer die Empfangssignale von dem noch aktiven Switch auf die andere Switchvariante um. Die neue Switchvariante empfängt nun alle neu ankommenden Pakete. Sie wird aber erst vollständig aktiviert, wenn auch die Sendesignale umgeschaltet sind.

Kontrolliert wird das Umschalten der Sendesignale von der Steuerung des Sendemultiplexers. Sie überwacht, nachdem der Umschaltvorgang eingeleitet wurde, den Sendepuffer des momentan aktiven Switches. Erst wenn das letzte Paket aus dem Sendepuffer übertragen wurde und dieser leer ist, wartet die Steuerung für die Dauer einer IFG, bevor sie den Sendemultiplexer die Sendesignale zur neuen Switchvariante umschalten lässt. Auf diese Weise wird garantiert, dass auch beim Umschalten zwischen jedem Paket die IFG eingehalten wird. Nach dem Umschalten aktiviert die Steuerung den Sendeprozess der neuen Switchvariante, der zu Beginn des Umschaltvorgangs von der Steuerung deaktiviert wurde. Ein während des Umschaltvorgangs aktivierter Sendeprozess würde Pakete, die in den Sendepuffer geschrieben werden, direkt übertragen, ohne dass die Pakete den physikalischen Port erreichen können.

3.2.4 Bus-Architektur

Ein partiell dynamisch rekonfigurierbares System benötigt, wie bereits in Kapitel 3.1.1 beschrieben, eine Kommunikationsinfrastruktur, die die Kommunikation zwischen den Modulen und dem statischen Bereich während und auch nach einer Rekonfiguration sicherstellt. Auf gar keinen Fall sollte eine Rekonfiguration zu einer Unterbrechung der Verbindung führen. Des Weiteren sollte die Kommunikationsinfrastruktur homogen sein, damit ein rekonfigurierbares Modul möglichst von jeder Position aus mit der Kommunikationsinfrastruktur verbunden werden kann. Aus diesem Grund werden Verbindungsstrukturen des FPGAs, die sogenannten *Tri-State*-Leitungen, verwendet, die sich horizontal über den gesamten FPGA ausbreiten und von der Rekonfiguration nicht beeinflusst werden. Für den Aufbau der Kommunikationsinfrastruktur mit diesen Verbindungsstrukturen eignen sich insbesondere *On-Chip*-Busarchitekturen.

Im Allgemeinen wird bei Bus-Architekturen, die sich auf einem Mikrochip oder FPGA befinden, zwischen *Tri-State*-Bussen und *Multiplexed*-Bussen unterschieden. Bei einem Tri-State-basierten Bus teilen sich die Master und die Slaves den Bus und können nur nacheinander auf den Bus zugreifen. Ein mit Multiplexern realisierter Bus ermöglicht die gleichzeitige Kommunikation von mehreren *Mastern* mit verschiedenen *Slaves*. Am Beispiel der zwei Multiplexer-basierten Busse *Processor Local Bus (PLB)* und *On-Chip Peripheral Bus (OPB)* sowie einem Tri-State basierten Wishbone-Bus haben wir in [GKP06] die Einsatzmöglichkeit von Multiplexer- und Tri-State-basierten Busarchitekturen für ein partiell dynamisch rekonfigurierbares System evaluiert.

Evaluierung von Bus-Architekturen für die dynamische Rekonfiguration: PLB und OPB sind Teil der CoreConnect-Bus-Architektur von IBM und werden auf Xilinx-FPGAs zur Verbindung von Prozessor und *IP-Cores* [2]. Der PLB wird auf Xilinx-FPGAs mit einer Breite von 64 Bit und einem Adressbereich von 32 Bit eingesetzt, um den Prozessor mit den Komponenten zu verbinden, die eine hohe Bandbreite erfordern. Komponenten mit geringen Bandbreitenanforderungen, wie z. B. eine serielle Schnittstelle, werden an den OPB angeschlossen, um den PLB nicht auszubremsen. Der OPB hat eine Breite von 32 Bit und ist mit einem Adressbereich von ebenfalls 32 Bit ausgestattet. Die Wishbone-Bus-Architektur wurde als einheitliche *On-Chip*-Verbindungsstruktur für *IP-Cores* entwickelt. Die freie Spezifikation [Wis02] der Wishbone-Bus-Architektur erlaubt Datenbusse mit einer Breite zwischen 8 Bit und 64 Bit und einem Adressbereich von bis zu 64 Bit. Die Spezifikation definiert lediglich Signale und deren Gebrauch durch den *Master* und den *Slave*. Die Implementierung der Verbindungsstruktur wird nicht vorgeschrieben und erlaubt sowohl eine Realisierung

[2] Ein IP-Core ist eine fertig entwickelte Hardwarekomponente, wie z. B. ein Speicherkontroller, die einem System hinzugefügt werden kann. IP steht hier für *Intellectual Property* und sollte nicht mit dem IP-Protokoll verwechselt werden.

als Multiplexer als auch eine *Tri-State*-basierte Lösung. Für die partielle dynamische Rekonfiguration sind der Aufbau der Busarchitektur und die Anzahl der benötigten Verbindungsstrukturen entscheidend.

Der PLB verwendet einen zentralen Multiplexerkern, an dem alle Komponenten angeschlossen werden [PLB04]. Damit die Verbindung zwischen dem Prozessor und den rekonfigurierbaren Modulen bei einer partiellen dynamischen Rekonfiguration nicht abreißt, müssen auch diese Verbindungen über Tri-State-Leitungen realisiert werden. Die Anzahl der möglichen *Tri-State*-basierten Verbindungen auf einem FPGA ist nicht unbegrenzt. Um einen PLB-Master an den Multiplexer anzuschließen, werden 210 Verbindungen und für jeden weiteren Master 203 Verbindungen benötigt. Der Anschluss eines Slaves kostet 226 Verbindungen und 95 weitere Verbindungen für jeden weiteren Slave. Auf einen Xilinx XC2V4000 FPGA könnte der Prozessor lediglich mit einem rekonfigurierbaren Modul verbunden werden, welches 226 der möglichen 320 Tri-State-Leitungen verwendet. In einem partiell dynamisch rekonfigurierbaren System soll aber mehr als ein Modul rekonfiguriert werden. Schon der rekonfigurierbare Switch benötigt zwei rekonfigurierbare Modulplätze. Der PLB benötigt zu viele Verbindungsressourcen, sodass eine effiziente Implementierung des rekonfigurierbaren Systems nicht möglich ist.

Bei dem kleineren OPB werden die Verbindungen über verteilte Multiplexer realisiert [OPB04]. Die Anbindung eines Masters benötigt 110 Verbindungen und 73 weitere Verbindungen für jeden weiteren Master. Ein Slave-Modul belegt 107 Verbindungen und jeder weitere Slave 37 Verbindungen. Wird der OPB verwendet, um den Prozessor mit den rekonfigurierbaren Modulen zu verbinden, können auf einem XC2V4000 5 Module an den Prozessor angeschlossen werden. In diesem Beispiel benötigt das System 287 der 320 möglichen Verbindungen. Zwar können beim OPB mehr Module an den Prozessor angeschlossen werden, die verteilte Multiplexerstruktur des OPB führt aber zu einer inhomogenen Kommunikationsinfrastruktur. Ein freies Platzieren der Module ist dann nicht möglich.

Ein Tri-State-basierter Wishbone-Bus mit einer Breite von 32 Bit und einem Adressbereich von ebenfalls 32 Bit besteht hauptsächlich aus den Adress- und Datenleitungen, an die alle Masters und alle Slaves angeschlossen werden. In dieser Konfiguration benötigt der Wishbone-Bus für einen Master und einen Slave 70 Tri-State-Leitungen. Jeder weitere Slave, der an den Bus angeschlossen wird, benötigt nur eine zusätzliche Verbindung und bei einem zusätzlichen Master sind es drei Verbindungen. Theoretisch können damit 250 rekonfigurierbare Module an den Bus angeschlossen werden. Wegen der geringeren Anzahl an Verbindungen und seiner homogeneren Struktur ist der Wishbone-Bus für ein partiell dynamisch rekonfigurierbares System besser geeignet und wird aus diesem Grund für die prototypische Umsetzung verwendet.

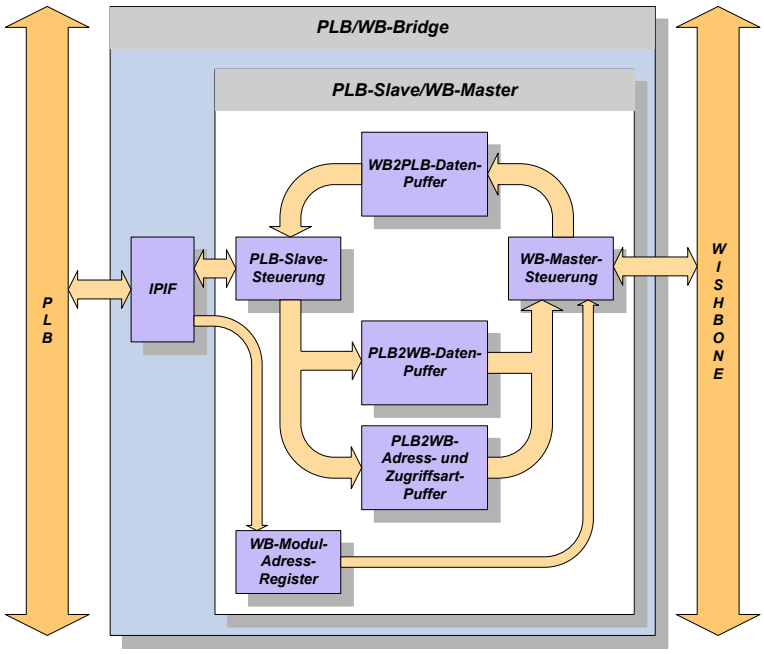


Abbildung 3.7: Aufbau der PLB/WB-Bridge

Implementierte Wishbone-Architektur: Die in der prototypischen Umsetzung implementierte Wishbone-Architektur hat ebenfalls eine Datenbreite und einen Adressbereich von jeweils 32 Bit. Der Aufbau des rekonfigurierbaren Gesamtsystems in Abbildung 3.5 zeigt auch die Teilkomponente der Wishbone-Architektur, die neben der Kommunikationsinfrastruktur, also den Busleitungen, aus der PLB-WB-Bridge und der Wishbone-Steuerung besteht.

Die PLB/WB-Bridge ist die Schnittstelle zwischen PLB und Wishbone-Bus. Abbildung 3.7 zeigt den internen Aufbau der Bridge. Sie ist Slave-Teilnehmer am PLB und Master am Wishbone-Bus und übersetzt die Datenübertragungsformate durch die PLB-Slave- bzw. WB-Master-Steuerung von einem zum anderen Bus. Dazu werden die Daten und Adressen vom Prozessor in getrennte Puffer gespeichert und anschließend auf den Wishbone-Bus übertragen. Bei einem Lesezugriff werden die von rekonfigurierbaren Modulen übertragenen Daten ebenfalls gepuffert und an den PLB weitergeleitet. Über die PLB/WB-Bridge werden auch die Adressbereiche der rekonfigurierbaren Module dynamisch vom Prozessor festgelegt. Damit lässt sich ein rekonfigurierbares Modul unabhängig von der Position des Moduls adressieren. IPIF (*Intellectual-Property-Interface*) ist eine Schnittstelle, mit der eigene IP-Cores an den PLB angeschlossen werden.

Die Wishbone-Steuerung im statischen Bereich enthält den Busarbiter und den Adressdekoder für die rekonfigurierbaren Module. Der Arbiter wird für ein System mit mehreren Mastern benötigt und regelt den Zugriff der Master auf den Bus. Der Arbiter

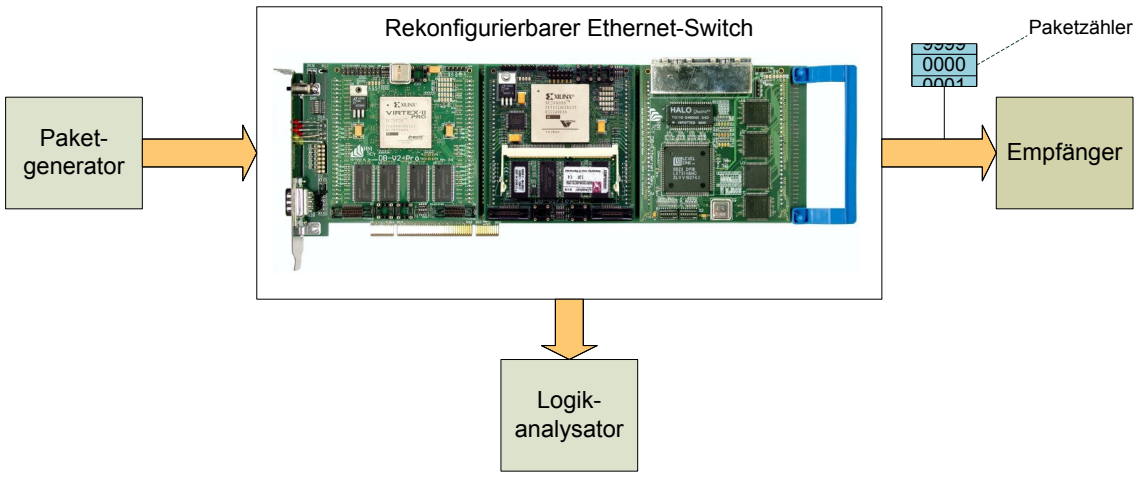


Abbildung 3.8: Messanordnung des rekonfigurierbaren Ethernet-Switches

wurde für zukünftige Erweiterungen des rekonfigurierbaren Gesamtsystems implementiert, ist aber für die prototypische Umsetzung des rekonfigurierbaren Switches nicht notwendig, weil die PLB/WB-Bridge der einzige Master am Wishbone-Bus ist. Im Adressdekoder werden von der PLB/WB-Bridge die einzelnen Adressbereiche der rekonfigurierbaren Module hinterlegt. Zur Adressdekodierung wird überprüft, ob die angelegte Adresse in einen der gespeicherten Adressbereiche fällt und das entsprechende Modul wird ausgewählt.

3.3 Performanceevaluation

3.3.1 Messaufbau

Der Messaufbau zur Bestimmung von Latenzzeit und Jitter des rekonfigurierbaren Switches und zur Verifizierung einer Übertragung ohne Paketverlust während einer Rekonfiguration wird in der Abbildung 3.8 gezeigt. Mit dem Paketgenerator werden standardkonforme Ethernet-Pakete erzeugt und versendet. Die Anzahl der Pakete, die Paketgröße und die Datenrate können in den Grenzen des Ethernet-Standards frei gewählt werden. Angeschlossen ist der Paketgenerator an das Erweiterungsmodul DB-Ethernet, auf dem sich die physikalischen Ethernet-Schnittstellen befinden. Auf dem Virtex-II-Pro befindet sich das Prozessorsystem und auf dem Virtex-II befindet sich der rekonfigurierbare Ethernet-Switch (Abbildung 3.5). Der Prozessor, ein PowerPC 405, wird bei den Messungen mit 300 MHz getaktet und der PLB hat eine Taktfrequenz von 100 MHz. Der Wishbone-Bus und der rekonfigurierbare Switch auf dem Virtex-II werden mit einem Takt von 25 MHz betrieben. Der Paketzähler befindet sich auf einem zweiten RAPTOR2000-System mit einem Virtex-II Pro Modul und einem DB-

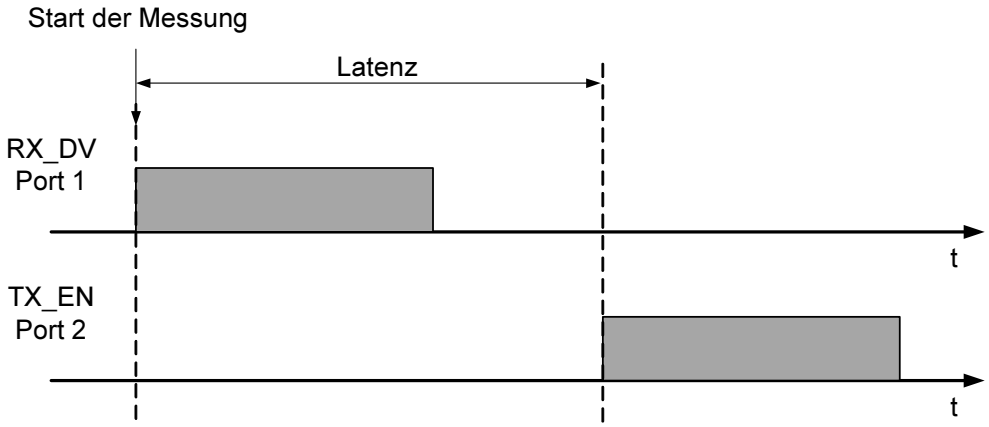


Abbildung 3.9: Messung der Latenzzeit

Ethernet. Er zählt die Anzahl der Pakete, die vom Ethernet-Switch weitergeleitet werden und zeigt diese auf einem LCD an. Mit dem Paketzähler wird überprüft, ob alle vom Paketgenerator gesendeten Pakete weitergeleitet wurden. Stimmt die Anzahl der gezählten Pakete nicht mit der Anzahl der gesendeten Pakete überein, sind Pakete verloren gegangen.

In der Messanordnung sind die Signale des MII vom rekonfigurierbaren Switch an einen *HP 16500B* Logikanalysator angeschlossen. Mit dem Logikanalysator wurde der Umschaltvorgang vom Software- zum Hardware-Switch analysiert.

Mit einer digitalen Messschaltung werden die Latenzzeit und der Jitter der beiden Switchvarianten bestimmt. Dazu wurde die Messschaltung zusätzlich zwischen MII und Rekonfigurationssteuerung in den rekonfigurierbaren Switch integriert. Die Schaltung überwacht die MII-Schnittstelle und misst die Latenz, wie in Abbildung 3.9 dargestellt, beginnend mit der Ankunft des ersten Bits am Eingangsport bis zu der Übertragung des ersten Bits desselben Paketes am Ausgangsport. Als Trigger für die Messungen wurde das Signal RX_DV für den Eingangsport und das Signal TX_EN für den Ausgangsport verwendet. Beide Signale werden synchron zum ersten Nibble[3] der Präambel getrieben und bleiben solange gesetzt, bis das letzte Nibbel von der physikalischen Ethernet-Schnittstelle übertragen wurde. Von der Messschaltung wird nur der Pegel der beiden Signale untersucht und es werden keine Signale oder Daten gepuffert. Der Paketstrom fließt direkt durch die Messschaltung hindurch und wird nicht von der Messung beeinflusst.

[3] Ein Nibble sind 4 Bit

3.3.2 Latenzzeiten und Jitter

Die Latenzzeit eines Switches ist die Verzögerungszeit, die bei der Übertragung eines Paketes über einen Switch entsteht. Die Latenzzeit von Software- und Hardware-Switches entspricht der Verzögerungszeit eines Store-and-Forward-Switches und wird durch $T_{\mathrm{ST\&FW}}$ symbolisiert. Der Jitter wird in Kapitel 2.1.1 als die Schwankung eines periodisch auftretenden Ereignisses beschrieben. Beim Switch entspricht die aktuelle Zykluszeit T_K eines Ereignisses der gemessenen Latenzzeit $T_{\mathrm{ST\&FW}}$ und der arithmetische Mittelwert der Zykluszeit $\overline{T}_{\mathrm{Cyk}}$ der mittleren Latenzzeit $\overline{T_{\mathrm{ST\&FW}}}$. Eingesetzt in die Gleichung (2.4) folgt daraus der Jitter eines Switches mit

$$J_{\mathrm{Swi}} = T_{\mathrm{ST\&FW}} - \overline{T_{\mathrm{ST\&FW}}}. \tag{3.8}$$

Gemessen wurden die Latenzzeiten für unterschiedliche Paketgrößen mit der zuvor beschriebenen Messschaltung. Um die Genauigkeit bei der Jittermessung zu erhöhen, wurde der Jitter mit einem *Textronix TLA7012* Logikanalysator und einem *Tektronix DPO 704004* digitalen Oszilloskop bestimmt. Für die Latenz- und Jittermessung wurde mit dem Ethernet-Paketgenerator mit einer gleich bleibenden Periode von 500 μs Ethernet-Pakete zum Switch gesendet. Die Messung erfolgte für unterschiedliche Paketgrößen, um die Abhängigkeit der Latenz bei einer *Store-and-Forward*-Architektur zu verdeutlichen. Für jede Paketgröße wurde die Latenzmessung für beide Switches jeweils 50 mal wiederholt und aus den Ergebnissen der Mittelwert gebildet. Abbildung 3.10 zeigt die Ergebnisse der Messung der Latenzzeit für den Software- und für den Hardware-Switch. Die dort angegebene Paketgröße entspricht der Nutzdatenmenge eines Ethernet-Paketes. Die vollständige Paketlänge kann berechnet werden, indem 26 Bytes für den Paketkopf (mit Präambel und SFD) addiert werden.

Bei beiden Switches steigt aufgrund der *Store-and-Forward*-Architektur die Latenz linear mit der Paketlänge. Die Steigung der Latenz des Software-Switches ist jedoch 3,5 mal größer als die Steigung der Latenz des Hardware-Switches. Der Hauptgrund für die höhere Steigung sind die Kopieroperationen des Prozessors, deren Dauer ebenfalls von der Paketlänge abhängt. Beim Weiterleiten eines Ethernet-Paketes kopiert der Prozessor das Paket erst in den externen Speicher und anschließend von dem Speicher in den Sendepuffer des Ausgangsports für dieses Paket. Infolge der höheren Latenz kann der Software-Switch die Pakete nicht mit der vollen Datenrate von *Fast Ethernet* (100 MBit/s) übertragen. Die Messungen haben ergeben, dass der Software-Switch im schlechtesten Fall (bei kleinen Paketlängen) eine maximale Datenrate von 18 MBit/s erreicht. Bis zu dieser maximalen Rate gehen keine Pakete verloren. Für diese Messungen wurde die minimale Paketlänge von Ethernet verwendet. Das Auslesen des Statusregisters des Switches und die Verarbeitung der Zieladresse durch den Prozes-

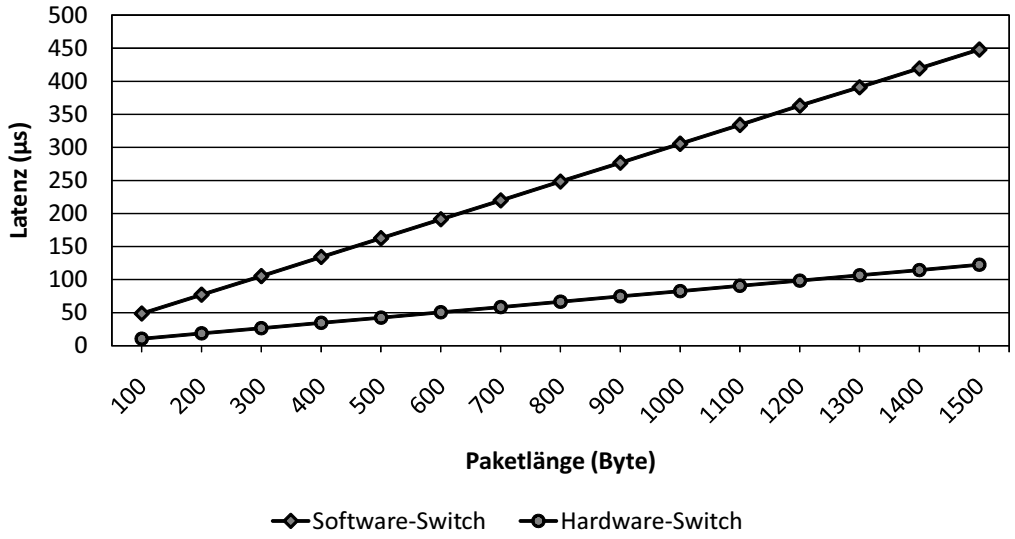
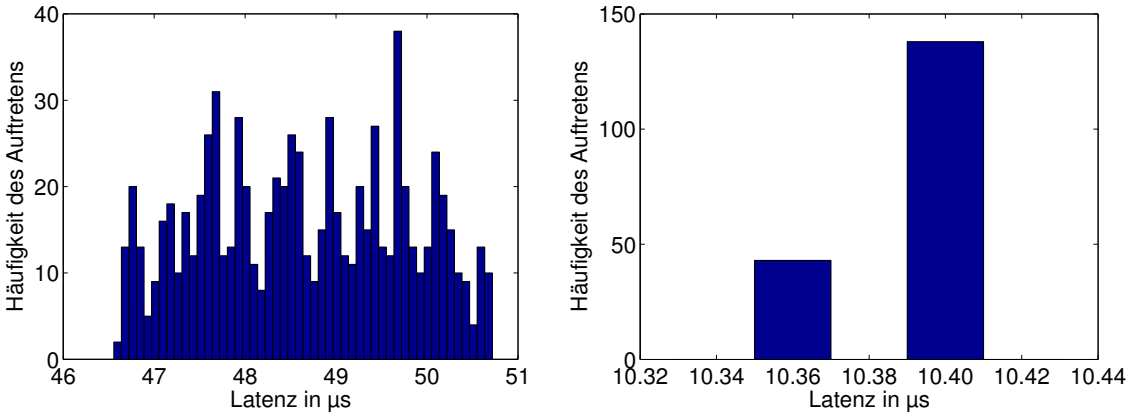


Abbildung 3.10: Gemessene mittlere Latenz von Software- und Hardware-Switch

sor haben bei kleineren Paketen einen größeren Einfluss auf die Datenrate. Bei der maximalen Paketlänge erreicht der Software-Switch daher eine höhere Datenrate von 27 MBit/s.

Im Gegensatz zum Software-Switch unterstützt der Hardware-Switch die volle Datenrate von 100 MBit/s. Seine Latenz ergibt sich zum größten Teil aus der Zeit, die er benötigt, um das Paket in seinen Empfangspuffer zu schreiben. Diese Empfangszeit des Switches ist abhängig von der Paketlänge und der Datenrate und entspricht der Übertragungsverzögerung eines Paketes und kann mit der Gleichung (2.8) bestimmt werden. Zusätzlich zu der Empfangszeit benötigt der Hardware-Switch noch 280 ns, bis das erste Bit auf dem anderen Port gesendet wird. Der Anteil der Paketverarbeitung und Weiterleitung beträgt gegenüber der gesamten Latenz bei minimaler Paketlänge nur 4,6 % und bei maximaler Paketlänge nur 0,2 %. Beim Software-Switch ist dieser Anteil wesentlich größer und beträgt bei minimaler Paketlänge 81,4 % und bei maximaler Paketlänge 72,7 %.

In der Abbildung 3.11 sind die bei der Jittermessung ermittelten unterschiedlichen Latenzzeiten sowie die Häufigkeit, mit der eine gemessene Latenzzeit auftritt, für eine Nutzdatenmenge von 100 Bytes dargestellt. Mit dem Oszilloskop wurde der Jitter des Hardware- und Software-Switches aus über 10.000 Werten bestimmt. Für die Darstellung des Jitters des Software-Switches in der Abbildung 3.11(a) wurden 800 Messwerte ausgewertet, die in mehreren Momentaufnahmen mit dem Logikanalysator ermittelt wurden. Aufgrund der geringen Streuung beim Hardware-Switch basiert die Darstellung des Jitters des Hardware-Switches in Abbildung 3.11(b) nur auf 180 Messwerte. Jittermessungen mit unterschiedlichen Nutzdatenmengen haben ergeben,



(a) Software-Switch (b) Hardware-Switch

Abbildung 3.11: Messergebnisse für den Jitter

dass der maximale Jitter im Gegensatz zur Latenz nicht von der Länge eines Paketes abhängig ist. Die maximale Schwankung der Latenzzeit bzw. der maximale Jitter beträgt beim Software-Switch 4 μs und die Standardabweichung beträgt 1,1 μs. Der Jitter wird hauptsächlich während des Kopiervorgangs durch zusätzliche Prozessoroperationen verursacht, die eine unterschiedliche Anzahl von Taktzyklen benötigen, wie z. B. das Nachladen von Instruktionen aus dem Speicher oder dem Instruktionscache. Beim Hardware-Switch beträgt der Jitter nur 40 ns. Bei einer Datenrate von 100 MBit/s entspricht dies genau einem Sende- bzw. Empfangstakt des Switches. Die Standardabweichung beim Hardware-Switch beträgt 2 ns. Die Schwankung um 40 ns ist beim Hardware-Switch implementierungsbedingt und wird durch die sequenzielle taktweise Abfrage des Sendepuffers für den Prozessor (TX_CPU) und des Sendepuffers zur Weiterleitung von Paketen (RX_TX) verursacht (siehe Abbildung 3.6). Überwacht der Hardware-Switch den TX_CPU-Puffer, während ein Paket im RX_TX-Puffer zur Weiterleitung bereitsteht, erhöht sich die Latenz um einen Takt.

Die Ergebnisse verdeutlichen, dass der Hardware-Switch bei hohen Datenraten und bei harten Echtzeitanforderungen wesentlich besser geeignet ist als der Software-Switch. Insbesondere infolge des hohen Jitters darf der Software-Switch nach den IAONA-Echtzeitklassen (Kapitel 2.5.1) beim Austausch von Sensor- und Aktordaten in einem verteilten mechatronischen System mit mehr als drei Knoten nicht eingesetzt werden.

3.3.3 Ressourcenverbrauch des rekonfigurierbaren Switches

Wie erwartet, hat der Hardware-Switch infolge seiner höheren Leistungsfähigkeit im Bezug auf Jitter und Latenz einen höheren Flächenbedarf als der Software-Switch. Der Ressourcenbedarf von Software- und Hardware-Switch sowie von beiden zusammen auf

Ressourcen	Software-Switch		Hardware-Switch		Beide Switches	
Slices	2027 / 23040	9%	4480 / 23040	19%	6507 / 23040	28%
BRAM	12 /120	10%	20 /120	16%	32 / 120	27%

Tabelle 3.3: Ressourcenbedarf der Switch-Implementierungen

einem Xilinx Virtex-II XC2V4000 FPGA sind in Tabelle 3.3 zusammengestellt. Diese Angaben beziehen sich nur auf den Teil des Gesamtsystems, der sich im rekonfigurierbaren Bereich befindet. Für den Software-Switch sind dies die beiden Ethernet-MACs und für den Hardware-Switch die beiden Teilswitches. Zwar funktioniert der Software-Switch nur zusammen mit dem Prozessor-System auf dem Virtex-II Pro. Aber die Angaben aus der Tabelle 3.3 geben Aufschluss darüber, wie viele Ressourcen beim Einsatz des Software-Switches gegenüber dem Hardware-Switch eingespart werden können. Gemessen an den belegten FPGA-Ressourcen (Slices) beträgt diese Ersparnis etwas mehr als die Hälfte. Bei geringen Anforderungen an die Bandbreite können in diesem Beispiel 10 % mehr FPGA-Ressourcen z. B. zur Beschleunigung einer anderen Applikation verwendet werden. Deutlicher fällt der Größenunterschied zwischen Software- und Hardware-Switch bei einer Implementierung auf einem kleineren FPGA auf. So benötigt der Software-Switch auf einem Virtex-II Pro XCVP20 27 % der vorhandenen Slices und der Hardware-Switch benötigt 71 %. Der rekonfigurierbare Ethernet-Switch eignet sich also besser für kleinere Systeme, die Platz für beide Switchvarianten bieten, und die bei geringeren Kommunikationsanforderungen eine zusätzliche Leistungsfähigkeit nutzen können.

Für eine Rekonfiguration ohne Paketverlust müssen während der Rekonfiguration, wie bereits in Kapitel 3.1.2 erwähnt, beide Switchvarianten parallel auf dem FPGA ausgeführt werden. In einer solchen Situation wird die Summe der Ressourcen von beiden Switches benötigt. Auf dem Virtex-II sind dies ca. 30 % der vorhandenen Ressourcen und auf dem Virtex-II Pro nahezu 100 %. In einem Szenario, in dem andere Applikationen die freien FPGA-Flächen nutzen, müssten diese Applikationen die für die Rekonfiguration zusätzlich benötigten Ressourcen erst frei geben. Während einer Rekonfiguration stehen den anderen Applikationen am wenigsten Ressourcen zur Verfügung.

3.3.4 Verifikation der Rekonfiguration ohne Paketverlust

Zur Verifikation der Rekonfiguration eines Switches ohne Paketverlust wurden vom Paketgenerator 2 Millionen Pakete versendet. Während der gesamten Sendedauer wurde mehrfach zwischen Software- und Hardware-Switch umgeschaltet. Der Versuch wurde mehrfach mit unterschiedlichen Paketgrößen wiederholt. Ein Paketverlust wurde

mithilfe des Paketzählers erst nach einer Erhöhung der Datenrate auf über 18 MBit/s festgestellt. Verursacht wurde der Paketverlust durch einen Pufferüberlauf während der Aktivität des Software-Switches oder beim Umkopieren der Pakete vom Software- zum Hardware-Switch. Dieses wird auch durch die Ergebnisse der Latenzzeitmessungen in Kapitel 3.3.2 bestätigt. Auch ohne eine Rekonfiguration kann der Software-Switch aufgrund der gemessenen Latenzen keine Pakete mit einer Datenrate größer als 18 MBit/s weiterleiten, ohne dass seine Puffer überlaufen. Die hohe Latenz, die die Datenrate des Software-Switches begrenzt, verzögert auch das Umkopieren der Pakete während des Umschaltvorgangs. Durch die Rekonfiguration entsteht kein Paketverlust, wohl aber durch zu hohe Datenraten bei einem aktiven Software-Switch. Durch eine rechtzeitige automatische Rekonfiguration (siehe Kapitel 3.1.3) wird auch dieses Problem behoben.

Bei der Verifizierung der automatischen Rekonfiguration wurde die Datenrate innerhalb der Sendedauer modifiziert. Vom Software- zum Hardware-Switch wurde aufgrund der Prozessorlast, der Netzwerklast oder der Pufferstände umgeschaltet. Die Rekonfiguration in die andere Richtung erfolgte nur in Abhängigkeit von der Netzwerklast. Die Prozessorlast und die Pufferstände sind im Testszenario für eine Rekonfiguration vom Hardware- zum Software-Switch ungeeignet. Im Testszenario existieren neben der Switch-Applikation keine weiteren Applikationen, und die Prozessorlast ist bei einem aktiven Hardware-Switch konstant. Ein leerer Puffer des Hardware-Switches ist kein Auslöser für eine Rekonfiguration, weil bei dieser geringen Latenz die Puffer auch bei einer Datenrate von 50 MBit/s zeitweise leer sind. Über die gemessene Netzwerklast kann sicherer bestimmt werden, ab wann der Software-Switch Pakete ohne Paketverlust weiterleiten kann.

3.3.5 Rekonfigurationszeit

Mit der Rekonfigurationszeit wird die Zeit angegeben, die das System nach Anfrage zur Rekonfiguration benötigt, bis die neue Switchvariante aktiviert ist. Die Rekonfigurationszeit T_{Rek} setzt sich zusammen aus der Zeit T_{FPGA}, die für die partielle Rekonfiguration benötigt wird und aus der Zeit T_{Um}, die der Umschaltvorgang in Anspruch nimmt:

$$T_{\mathrm{Rek}} = T_{\mathrm{FPGA}} + T_{\mathrm{Um}} \tag{3.9}$$

Die partielle Rekonfiguration eines Switch-Moduls dauert bei der Testanordnung ca. 10 ms. Werden die Rekonfigurationsstrategien aus Kapitel 3.1.2 nicht verwendet und der Software-Switch wird z. B. einfach vom Hardware-Switch überschrieben, dann ist die gesamte Rekonfiguration in 10 ms abgeschlossen. Allerdings ist in diesem Fall während der gesamten partiellen Rekonfiguration keine Kommunikation über den Switch

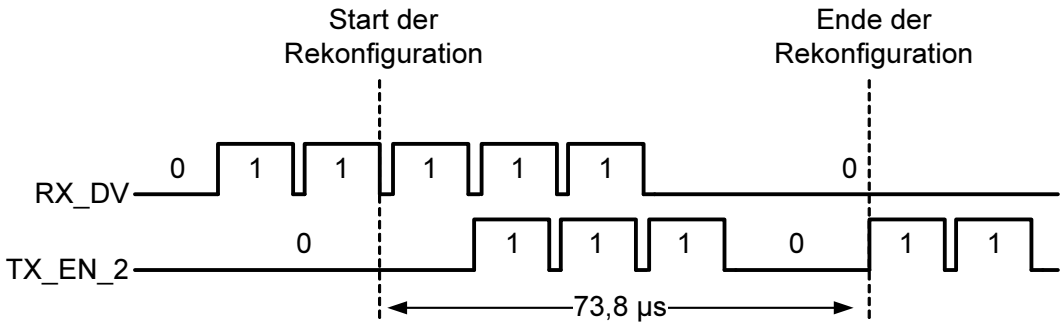


Abbildung 3.12: Illustration des Paketflusses während der Rekonfiguration

möglich. Im schlechtesten Fall, d.h., Pakete mit minimaler Länge werden mit der vollen Datenrate von 100 MBit/s zum Switch gesendet, führt dies zu einem Paketverlust von 1488 Paketen. Bei maximaler Paketlänge gehen im selben Fall noch 81 Pakete verloren. Bleibt der Software-Switch aktiv, während der Hardware-Switch neben dem Software-Switch geladen wird, können bei der maximalen Datenrate ebenfalls Pakete verloren gehen, bevor zum Hardware-Switch umgeschaltet werden kann. Aufgrund der hohen Latenz des Software-Switches gehen in diesem Fall bei minimaler Paketlänge noch 1220 Pakete verloren und bei maximaler Paketlänge beträgt der Paketverlust 59 Pakete. Mithilfe eines Ethernet-Pause-Paketes könnte der Software-Switch die Datenrate des Senders drosseln, sodass keine Pakete verloren gehen. Allerdings ist es sinnvoller, den Hardware-Switch zu laden, bevor die maximale Datenrate des Software-Switches überschritten wird. Nur in dem Fall muss das Netzwerk nicht ausgebremst werden. Fordert z. B. der Nachbarknoten die höhere Datenrate ca. 10 ms vorher an, steht der Hardware-Switch rechtzeitig zur Verfügung. Befinden sich beide Switches schließlich auf dem FPGA, führt ein direktes Umschalten zum Verlust aller Pakete, die sich im Empfangs- und im Sendepuffer des Switches befinden. Im schlechtesten Fall können 8 Pakete verloren gehen. Dieser Paketverlust kann durch die drei Rekonfigurationsstrategien vermieden werden.

Am Beispiel der ersten Rekonfigurationsstrategie wurde die Umschaltzeit mit dem Logikanalysator bestimmt. Dazu befanden sich beide Switches parallel zueinander auf dem FPGA und der Umschaltvorgang vom Software- zum Hardware-Switch wurde automatisch durch einen drohenden Paketüberlauf initiiert. In diesem Szenario wurde der Umschaltvorgang automatisch eingeleitet, sobald zwei Pakete sich im Empfangspuffer des Software-Switches befinden. Insgesamt fünf Pakete mit einer Nutzdatenmenge von 100 Bytes werden an den Switch gesendet. Abbildung 3.12 skizziert den auf dem Logikanalysator gemessenen Ablauf des Umschaltvorgangs zwischen Software- und Hardware-Switch. RX_DV kennzeichnet die an Port 1 ankommenden Pakete und TX_EN2 die an Port 2 ausgehenden Pakete. Die senkrechten gestrichelten Linien markieren den Anfang und das Ende des Umschaltprozesses, der der Zeit für das

Nutzdaten	Umschaltzeit (μs)			Empfangene Pakete		
(Byte)	Strat. 1	Strat. 2	Strat. 3	Strat. 1	Strat. 2	Strat. 3
100	77,14	88,18	115,72	7	8	11
500	240,92	283,96	361,39	6	7	9
1000	446,68	529,72	670,02	6	7	9
1500	651,76	774,80	977,63	6	7	8

Tabelle 3.4: Umschaltzeiten vom Software-Switch zum Hardware-Switch

Umkopieren der zwei Pakete entspricht. Für die Messung mit dem Logikanalysator wurde die Software zum Umkopieren der Pakete angepasst, sodass die Pakete aus dem TX_CPU-Puffer (vgl. Abbildung 3.6) erst übertragen werden, wenn der Kopiervorgang abgeschlossen ist. Daher ist das Ende der Rekonfiguration gleichzeitig der Beginn der Übertragung des ersten Paketes, welches noch vom Software-Switch empfangen wurde. Die Umschaltzeit $T_{\mathrm{Um_1}}$ beträgt für dieses Beispiel 73,8 μs. Demnach ist der gesamte Rekonfigurationsprozess T_{Rek} nach 10,07 ms abgeschlossen.

Anhand von Abbildung 3.12 lässt sich auch eine Vertauschung der Paketreihenfolge feststellen. Die drei Pakete, die nach dem Start der Rekonfiguration abgebildet sind, werden direkt vom Hardware-Switch empfangen und mit einer geringen Verzögerung umgehend gesendet. Die zwei Pakete aus dem Software-Switch werden erst nach dem Ende der Rekonfiguration gesendet. Die Messung bestätigt, dass der Hardware-Switch bei der ersten Rekonfigurationsstrategie sehr schnell aktiviert wird und umgehend die ersten empfangenen Pakete überträgt. Bei maximaler Datenrate können in der Umschaltzeit von 73,8 μs sieben Pakete mit derselben Paketlänge empfangen werden. Sie müssen bei dieser Strategie nicht alle vom Hardware-Switch gespeichert werden, sondern werden direkt übertragen. Ein Paketverlust während des Umschaltens ist ausgeschlossen. Tabelle 3.4 listet die Ergebnisse für die Dauer des Umschaltprozesses T_{Um} der drei Rekonfigurationsstrategien für unterschiedliche Paketlängen auf. Diese Ergebnisse basieren auf dem oben beschriebenen Beispiel und auf den Messungen der Latenzzeit $T_{\mathrm{ST\&FW}}$ des Software-Switches der jeweiligen Paketlänge. Sie werden für Strategie 1 und 3 wie folgt berechnet:

$$T_{\mathrm{Um_{1,3}}} = \left(T_{\mathrm{ST\&FW}} - \frac{l}{r}\right) \cdot n_{\mathrm{p}} \tag{3.10}$$

Von der gemessenen Latenz wird in der Gleichung (3.10) die Verzögerung beim Übertragen eines Paketes subtrahiert, die sich aus Paketlänge l einschließlich Paketkopf, Präambel und SFD und der Datenrate r errechnet. Das Ergebnis der Subtraktion ist die Zeit, die der Software-Switch zum Kopieren eines Paketes benötigt. Das Symbol n_{p} steht für die Anzahl der Pakete, die umkopiert werden.

Nutzdaten (Byte)	Umschaltzeit (μs)	Empfangene Pakete
100	22,08	2
500	86,08	2
1000	166,08	2
1500	246,08	2

Tabelle 3.5: Umschaltzeiten vom Hardware-Switch zum Software-Switch

Bei der zweiten Rekonfigurationsstrategie wird die Dauer des Umschaltprozesses mit

$$T_{\mathrm{Um_2}} = \left(T_{\mathrm{ST\&FW}} - \frac{l}{r}\right) \cdot (n_{\mathrm{p}} - 1) + T_{\mathrm{ST\&FW}} + T_{\mathrm{IFG}} \tag{3.11}$$

bestimmt. Wie bereits in Kapitel 3.1.2 erwähnt, wird bei der Strategie 2, nachdem alle Pakete in den Sendepuffer kopiert sind, noch das letzte Paket übertragen und die Dauer einer IFG abgewartet, bevor die Sendesignale zum Hardware-Switch umgeschaltet werden. In dem zuvor beschriebenen Beispiel ergibt sich nach der Gleichung (3.11) eine Umschaltzeit von 88,1 μs für die Strategie 2. Im Gegensatz zu Strategie 1 und 3 müssen in dieser Zeit alle empfangenen Pakete erst im Hardware-Switch gespeichert werden, um einen Paketverlust zu vermeiden. Damit auch bei maximaler Paketgröße bei Strategie 2 kein Pufferüberlauf auftritt, muss der Weiterleitungspuffer (RX_TX-Puffer in Abbildung 3.6) eines Teilswitches so dimensioniert werden, dass er mindestens 10,6 KByte an Daten aufnehmen kann. Diese Puffergröße entspricht der Datenmenge von 7 Paketen mit maximaler Paketlänge. Wird der Umschaltvorgang erst bei einer Anzahl von drei Paketen eingeleitet, erhöht sich die notwendige Puffergröße für den Hardware-Switch von 10,6 KByte auf 13,6 KByte. Bestimmt wird die Puffergröße B mit der folgenden Gleichung:

$$B = \frac{T_{\mathrm{Um_2}}}{\frac{l}{r} + T_{\mathrm{IFG}}} \cdot (l - 64\,Bit) \tag{3.12}$$

In der Gleichung (3.12) werden für die Präambel und den SFD 64 Bit von der Paketlänge l subtrahiert, weil diese im Puffer nicht gespeichert werden.

Die dritte Rekonfigurationsstrategie liefert in dem hier beschriebenen Beispielszenario dieselben Ergebnisse wie die Strategie 1, weil keine Pakete aus dem Sendepuffer des Software-Switches kopiert werden müssen. Für die Ergebnisse der Strategie 3 in Tabelle 3.4 wurde daher angenommen, dass sich beim Start der Rekonfiguration ein Paket in dem Sendepuffer des Software-Switches befindet. Diese Annahme hat auf die Ergebnisse der anderen beiden Rekonfigurationsstrategien keinen Einfluss, weil diese

Komponnente	Slices
WB-Steuerung	278
WB-Verbindungsstruktur	580
PLB/WB-Bridge	414
Rekonfigurationssteuerung	78
MII-Verbindungsstruktur	336
Gesamt:	1686

Tabelle 3.6: Ressourcenbedarf des rekonfigurierbaren Systems

Pakete aus dem Sendepuffer parallel zu der Kopieroperation senden. Nur die dritte Strategie muss in diesem Fall anstelle von zwei Paketen drei Pakete übertragen.

Für eine lückenlose Betrachtung der Rekonfigurationszeit wird diese auch für die Rekonfiguration vom Hardware-Switch zum Software-Switch erläutert. In diese Richtung wird nur die Rekonfigurationsstrategie 2 eingesetzt. Alle anderen Strategien erfordern auch beim Hardware-Switch ein Umkopieren der Pakete durch den Prozessor und sind aufgrund der hohen Latenz wesentlich langsamer. Für die Ergebnisse aus Tabelle 3.5 wird ebenfalls angenommen, dass sich zwei Pakete beim Start der Rekonfiguration im Weiterleitungspuffer des Hardware-Switches befinden. Sie wurden für unterschiedliche Paketgrößen nach der Formel (3.3) bestimmt.

3.3.6 Leistung und Ressourcenbedarf des rekonfigurierbaren Systems

Die Möglichkeiten, die ein partiell dynamisch rekonfigurierbares System in Form des rekonfigurierbaren Switches bietet, sind bereits geschildert worden. Dieser Abschnitt beschäftigt sich mit den zusätzlichen Ressourcen, die zur Realisierung der partiellen Rekonfiguration benötigt werden und mit der Leistungsfähigkeit des rekonfigurierbaren Systems am Beispiel des rekonfigurierbaren Switches.

Wesentlicher Bestandteil des rekonfigurierbaren Systems ist die Kommunikationsinfrastruktur, bestehend aus der WB-Steuerung, der WB-Verbindungsstruktur und der PLB/WB-Bridge, die sich auf dem Virtex-II Pro befindet. Das restliche Prozessorsystem (siehe Abbildung 3.5) musste nicht zusätzlich für die Realisierung der partiellen dynamischen Rekonfiguration entwickelt werden und wäre auch bei einem nicht rekonfigurierbaren Switch Bestandteil des Systems. Zusätzlich werden aber für den rekonfigurierbaren Switch noch die Rekonfigurationssteuerung und die dazugehörige MII-Kommunikationsinfrastruktur benötigt. Tabelle 3.6 listet den Ressourcenbedarf für die Komponenten der Kommunikationsinfrastruktur auf.

Insgesamt benötigt die Implementierung der partiellen dynamischen Rekonfiguration zusätzlich 1686 Slices. Das Ziel der partiellen dynamischen Rekonfiguration ist aber eine Flächenersparnis gegenüber einer statischen Lösung. Bei einer statischen Lösung müssten alle für die Anwendung benötigten Funktionsblöcke auf dem FPGA konfiguriert werden. Ein partiell dynamisch rekonfigurierbares System kommt dann mit weniger Ressourcen aus, wenn Funktionsblöcke, z. B. unterschiedliche Filter für die Bilderkennung, nacheinander auf den FPGA geladen werden. Der Ablauf, in der die Funktionsblöcke benötigt werden, ist in diesem Fall schon während des Entwurfs bekannt. Eine weitere Möglichkeit ist es, nur die Funktionsblöcke zu laden, die momentan von dem Benutzer oder von dem System angefordert werden. Für ein solches System wurde auch der rekonfigurierbare Switch entwickelt, der seinen Ressourcenbedarf an die Kommunikationsanforderungen anpasst.

An dieser Stelle sei auch erwähnt, dass rekonfigurierbare Module nicht alle Ressourcen bei der 1D-Platzierungsstrategie (vgl. Kapitel 3.1.1) verwenden, die in den Modulgrenzen zur Verfügung stehen. Für diese zusätzliche Belegung von eigentlich nicht benötigten Ressourcen wurde in [KPR05] der Begriff interne Defragmentierung eingeführt und deren Auftreten bei den unterschiedlichen Platzierungsstrategien untersucht. Die Abbildung 3.13 stellt die vom Software- und Hardware-Switch belegten Ressourcen in dem jeweiligen Bereich dar. Der Bereich des rekonfigurierbaren Moduls für den Software- und den Hardware-Switch ist in diesem Beispiel gleich groß. In der Abbildung 3.13 ist deutlich zu erkennen, dass die interne Defragmentierung beim rekonfigurierbaren Modul des Software-Switches sehr viel größer ist als beim Hardware-Switch.

Auch die Leistung des Software-Switches wird durch das partiell rekonfigurierbare System beeinflusst. Aufgrund der Tri-State-Signale, die sich zur Anbindung von Software- und Hardware-Switch fast über den gesamten FPGA erstrecken, kann die Wishbone-Bus-Struktur nur mit 25 MHz betrieben werden. Als statisches Design, ohne Tri-State-Signale, sind 50 MHz möglich. Dadurch verringert sich die Datenrate, mit der die Daten zwischen Prozessor und Switch ausgetauscht werden.

Ein Nachteil ist die durch ein partiell rekonfigurierbares System verringerte Leistung nur dann, wenn sie auch tatsächlich vom geplanten System benötigt wird. Für den rekonfigurierbaren Switch bedeutet eine geringere Leistung und damit eine Erhöhung der Latenz, dass der Bereich, in dem der Software-Switch ohne Paketverlust eingesetzt werden kann, ebenfalls geringer ist. In einem System, in dem nur ein Hardware-Switch statisch implementiert wurde, können jedoch bei einer geringen Kommunikationslast durch eine Verwendung des Software-Switches keine zusätzlichen FPGA-Ressourcen anderen Anwendungen zur Verfügung gestellt werden.

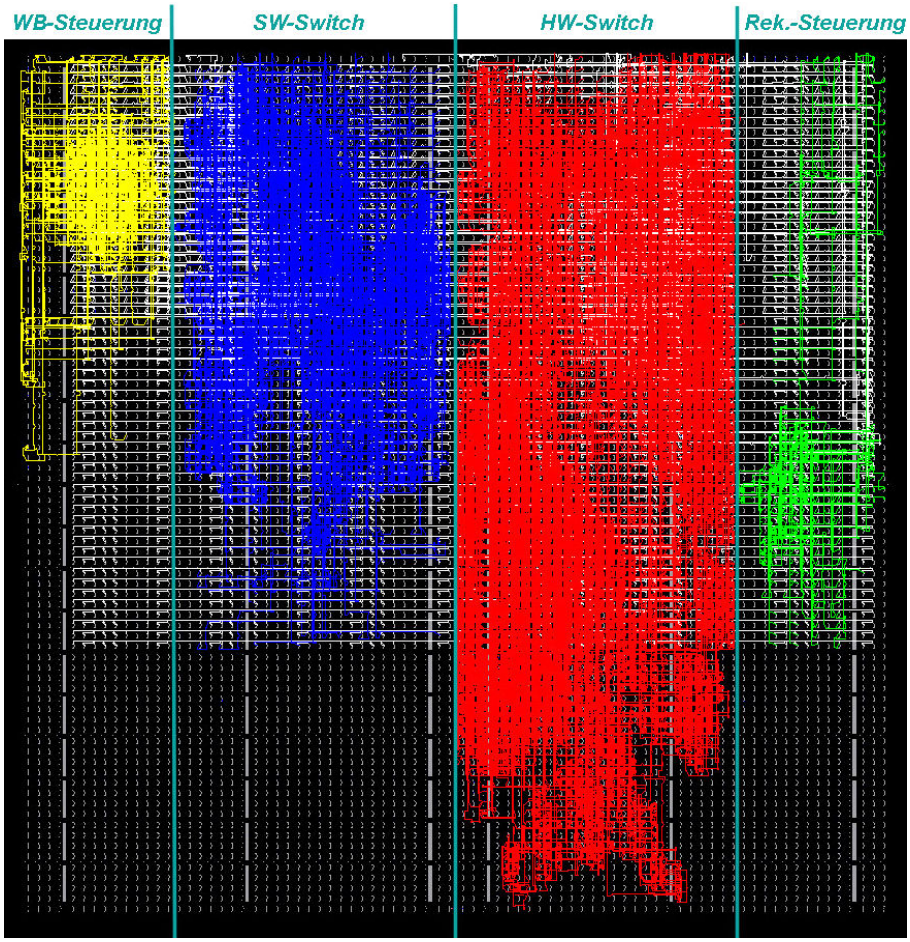


Abbildung 3.13: Ressourcenbelegung des rekonfigurierbaren Switches

3.4 Zusammenfassung

Die partielle dynamische Rekonfiguration eines FPGAs ermöglicht die Anpassung des Hardware-Ressourcenbedarfs zur Laufzeit, ohne die Verarbeitung anderer Hardware-Module auf dem FPGA zu beeinflussen. Unter Ausnutzung der partiellen dynamischen Rekonfiguration ist ein rekonfigurierbarer Ethernet-Switch entstanden, der seinen Ressourcenbedarf in Abhängigkeit der Kommunikations- und Echtzeitanforderungen anpasst. Dazu wurden zwei Varianten eines Ethernet-Switches implementiert, zwischen denen rekonfiguriert werden kann. Bei der einen Variante erfolgt das Weiterleiten von Paketen softwaregesteuert, wodurch dieser Software-Switch nur wenige Ressourcen auf dem FPGA benötigt. Bei der anderen Variante erfolgt das Weiterleiten von Paketen vollständig in Hardware. Dieser Hardware-Switch benötigt in der prototypischen Umsetzung etwas mehr als die doppelte Menge an FPGA-Ressourcen. Aber im Gegensatz zum Software-Switch erreicht der Hardware-Switch die volle Datenrate von Ethernet und hat eine wesentlich geringere Latenz und einen geringeren Jitter. Bei geringer

Kommunikationslast sowie geringen Echtzeitanforderungen bezüglich Latenz und Jitter verwendet der rekonfigurierbare Ethernet-Switch anstelle des Hardware-Switches den Software-Switch. Die in diesem Fall nicht benötigten Ressourcen stehen dann anderen Applikationen zur Verfügung. Erkennt der rekonfigurierbare Switch eine steigende Kommunikationslast, wechselt er automatisch zum Hardware-Switch. In diesem Fall stehen aufgrund höherer Kommunikationsanforderungen anderen Applikationen weniger FPGA-Ressourcen zur Verfügung.

Ein einfaches Ersetzen der Software-Switches durch den Hardware-Switch auf dem FPGA ist nicht möglich, weil ansonsten die Pakete, die sich noch in den Puffern befinden oder gerade übertragen werden, verloren gehen. Daher wurden spezielle Rekonfigurationsstrategien entwickelt, die einen Paketverlust während einer Rekonfiguration vermeiden. Die drei Strategien wurden analysiert und unterschiedliche Anwendungsgebiete wurden aufgezeigt.

In der Beschreibung der prototypischen Umsetzung des rekonfigurierbaren Switches wurden die Erweiterungen verdeutlicht, die gegenüber einer gewöhnlichen Switch-Architektur notwendig sind, um ein partiell dynamisch rekonfigurierbares System zu erhalten, das eine Adaption der Netzwerkkomponente ohne Paketverlust ermöglicht. Die prototypische Umsetzung bildet die Grundlage für die Performanceanalyse des rekonfigurierbaren Ethernet-Switches. Die Analysen bewerten die Leistungsfähigkeit und den Ressourcenbedarf des Software- und des Hardware-Switches. Sie bestätigen eine höhere Latenz und einen höheren Jitter sowie einen geringeren Ressourcenbedarf des Software-Switch gegenüber dem Hardware-Switch. Zusätzlich zu den beiden Switches wurden auch die Rekonfigurationszeit und die FPGA-Ressourcen untersucht, die für die partielle dynamische Rekonfiguration benötigt werden. Obwohl das rekonfigurierbare System ständig FPGA-Ressourcen belegt, würde ein nicht rekonfigurierbarer Hardware-Switch mehr Ressourcen beanspruchen als ein Software-Switch zusammen mit dem rekonfigurierbaren System. Allerdings haben die Analysen auch ergeben, dass sich der Einsatz eines rekonfigurierbaren Switches nur bei FPGAs rentiert, die nur eine begrenzte Menge an Ressourcen zur Verfügung stellen. Mit steigender Anzahl der zur Verfügung stehen Logikressourcen eines FPGAs verringert sich die prozentuale Differenz der verwendeten FPGA-Ressourcen zwischen Hardware- und Software-Switch, sodass sich der Aufwand für die Adaption des Switches nicht mehr lohnt. In diesem Fall ist ein fester Hardware-Switch die bessere Alternative. Bei kleineren FPGAs, in denen der Hardware-Switch schon einen großen Teil der Ressourcen belegt, ist eine Rekonfiguration zum Software-Switch bei geringem Kommunikationsaufkommen eine sinnvolle Möglichkeit, um anderen Applikationen mehr FPGA-Ressourcen zur Verfügung zu stellen.

Kapitel 4

Ein selbstsynchronisierendes Echtzeitprotokoll

Im letzten Kapitel wurden adaptive Netzwerkkomponenten zur Verarbeitung von Echtzeitdaten vorgestellt. Die Adaption der Netzwerkkomponente ist aber nicht die einzige Möglichkeit, ein Echtzeitnetzwerk im Betrieb an sich ändernde Anforderungen anzupassen. Eine weitere Möglichkeit sind adaptive Echtzeitprotokolle, die auf der Protokollebene eine Anpassung an unterschiedliche Anforderungen ermöglichen. Die meisten im industriellen Bereich eingesetzten Echtzeitprotokolle stellen eine solche Adaptivität nicht zur Verfügung. Diese Protokolle benötigen komplexe Planungswerkzeuge, um die Netzwerkinfrastruktur zu projektieren und Zykluszeiten sowie Latenzen zu optimieren. Nur mithilfe der Planung und Festlegung der Kommunikation können harte Echtzeitanforderungen erfüllt werden. Eine Veränderung von Zykluszeiten oder der Aufbau einer neuen Echtzeitverbindung zwischen zwei Teilnehmern ist im Betrieb nicht möglich und erfordert eine neue Planung und Analyse der Kommunikation. Um diese Planungsphase zu vermeiden und eine Adaption der Kommunikation an sich ändernde Anforderungen zu ermöglichen, wird ein adaptives Echtzeitprotokoll benötigt. Zusätzlich sollte dieses Protokoll ohne eine zentrale Steuerung (Master) für die Kommunikation auskommen. Die zentrale Steuerung für die Kommunikation hat den Nachteil, dass ein Ausfall der zentralen Steuerung zum Ausfall der gesamten Kommunikation führt.

Ein Beispiel für ein verteiltes adaptives Echtzeitprotokoll ist RTNet [HMJ06], das auf dem von Grow eingeführten *Timed-Token*-Protokoll [Gro82] basiert. Im Gegensatz zu dem *Timed-Token*-Protokoll wird der Token bei RTNet nicht von Knoten zu Knoten in einem Ring weitergegeben, sondern der Besitzer des Tokens gibt diesen an den Knoten mit der höchsten Priorität weiter. Ermittelt wird der Knoten mit der höchsten Priorität über einen dynamisch berechneten Ablaufplan (engl.: Schedule). Den Ablaufplan berechnet der Knoten über einen Echtzeit-Scheduling-Algorithmus, wie z. B. EDF (Earliest Deadline First) oder RM (Rate Monotonic). Die Informationen, die für die Berechnung benötigt werden, sind im Token enthalten. Der Knoten, der den Token besitzt, kann mit den Informationen aus dem Token, unabhängig von den anderen Knoten, den Knoten mit der höchsten Priorität ermitteln. Dazu benötigen alle Knoten

eine gemeinsame Zeitbasis und somit erfordert das Protokoll die Synchronisation der Uhren. Für den Aufbau einer neuen Echtzeitverbindung muss im Betrieb ein Akzeptanztest durchgeführt werden, der überprüft, ob für diese Verbindung noch genügend Bandbreite zur Verfügung steht. Aufsetzen lässt sich RTnet auf jedes Netzwerk mit *Broadcast*-Funktionalität.

Ein weiteres verteiltes Echtzeitprotokoll ist TrailCable [FR05], das ebenfalls einen Standard-Echtzeit-Scheduler verwendet, um das Echtzeitverhalten zu garantieren. Im Gegensatz zu RTnet agieren die TrailCable-Knoten als Router in einem Punkt-zu-Punkt-Netzwerk. Echtzeitverbindungen zwischen zwei Teilnehmern werden über sogenannte logische Echtzeitkanäle aufgebaut. Zum Etablieren eines neuen Echtzeitkanals muss ein Akzeptanztest durchgeführt werden, der überprüft, ob der neue Echtzeitkanal andere Kanäle, die dieselben Verbindungen nutzen, beeinflusst. Die Knoten verwenden einen präemptiven EDF-Scheduler, um die Reihenfolge festzulegen, in der die Echtzeitnachrichten von verschiedenen Echtzeitkanälen über eine Verbindung übertragen werden.

Mithilfe des Akzeptanztests verlagern beide Protokolle die Planungsphase in die Betriebsphase. Jede Änderung der Anforderungen für eine bestehende Kommunikationsbeziehung oder eine neu hinzugefügte Kommunikationsbeziehung führt zu einer erneuten Ausführung des Akzeptanztests. Dieses bedeutet eine hohe Belastung für die vorhandenen Rechenressourcen, die nur durch die Kommunikation verursacht wird.

Ein selbstsynchronisierendes Protokoll im Bereich der Echtzeitkommunikation wurde in [CC97] vorgestellt. Das Protokoll verwendet einen rückgekoppelten Synchronisationsmechanismus, der Informationen der Flusskontrolle und selbstsynchronisierende Zeitschlitze verwendet, um eine globale Zeit zu erzeugen. Das Protokoll wurde für die Vernetzung von Rechencluster entwickelt und erfüllt nicht die für Steuerungen und Regelungen erforderlichen harten Echtzeitanforderungen.

In diesem Kapitel wird ein verteiltes selbstsynchronisierendes adaptives Echtzeitprotokoll für virtuelle Ringtopologien beschrieben und analysiert. Das Protokoll benötigt weder einen Master-Knoten noch eine explizite Synchronisation der Uhren, um einen periodischen Austausch von Daten unter harten Echtzeitbedingungen zu unterstützen und wurde erstmals in [GBP08] vorgestellt. Das Protokoll trägt den Namen Self$S$ und bietet ein hohes Maß an Adaptivität, die die meisten Echtzeitprotokolle aus dem industriellen Bereich nicht zur Verfügung stellen. Beispielsweise lässt sich die Zykluszeit von jedem Knoten beim selbstsynchronisierenden Echtzeitprotokoll im Betrieb modifizieren, ohne die Kommunikation neu planen zu müssen. Der geringe Planungsaufwand macht auch einen aufwendigen Algorithmus überflüssig, der bei jeder Adaption erneut ausgeführt werden müsste. Für jeden Knoten muss lediglich die Zykluszeit als ein Vielfaches der Paketumlaufzeit gewählt werden. Self$S$ garantiert die gewählten Zykluszeiten und ermöglicht eine gleichzeitige Übertragung von nicht echtzeitkritischen

Daten. Eine Adaption der Zykluszeit für einen Knoten hat keinen Einfluss auf die Zykluszeiten der übrigen Knoten, sondern nur auf die Bandbreite, die für die nicht echtzeitrelevanten Daten zur Verfügung steht. Das Self*S*-Protokoll und die in diesem Kapitel durchgeführten Analysen sind unabhängig von der physikalischen Infrastruktur, basieren aber auf einer Switch-Technologie, wie sie auch vom rekonfigurierbaren Switch verwendet wird.

Nachdem in Kapitel 4.1 das Modell, das für die Analyse des Echtzeitprotokolls verwendet wird, vorgestellt wurde, folgt in Abschnitt 4.2.1 die Erläuterung des Prozesses der Selbstsynchronisation für den einfachen Fall der Ringtopologie. Die Ringtopologie ist der Startpunkt für die Analyse des Protokolls, weil diese aufgrund ihrer einfachen Verkabelung, der leichten Wartbarkeit und der inhärenten Redundanz im industriellen Bereich sehr verbreitet ist. In Abschnitt 4.2.2 wird aufgezeigt, dass sich nicht echtzeitrelevanter Datenverkehr effizient mit einbeziehen lässt, ohne dabei die Synchronisationszeit oder den Echtzeitdatenverkehr zu beeinflussen. Die dort vorgeschlagene Protokollerweiterung ermöglicht für jeden Knoten variable Zykluszeiten als ein beliebiges Vielfaches der Paketumlaufzeit. Frei wählbare Zykluszeiten werden in Abschnitt 4.2.3 analysiert und es wird gezeigt, dass es nicht möglich ist, diese in einem Ring zu verwenden, ohne einen sehr hohen Jitter zu akzeptieren, der sogar im Falle einer geringen Auslastung des Netzwerkes entstehen kann.

Nach den Analysen im Falle der einfachen Ringtopologie folgen im Abschnitt 4.3 Untersuchungen der Performance und der Selbstsynchronisation auf der Basis von weiteren Topologien. Zunächst werden in Abschnitt 4.3.1 mehrere Ringe mit einer geteilten Verbindung betrachtet. Darauf folgt in Abschnitt 4.3.2 eine Erweiterung des Protokolls auf beliebige Topologien. Dazu wird der Ring in eine Baumtopologie eingebunden. Baumstrukturen in einem Netzwerk sind von besonderem Interesse, weil sie typischerweise mithilfe von *Spanning-Tree*-Protokollen im Bereich von LANs (*Local Area Networks*) gebildet werden, um einen eindeutigen Pfad in beliebig miteinander verbundenen Netzwerken zu erhalten. Eine zusätzliche Erweiterung wird in Abschnitt 4.4 vorgestellt, die eine Einbindung von ereignisbasierten Nachrichten ermöglicht.

4.1 Modell

Im Modell wird ein Netzwerk mit n Knoten durch einen Graphen $G = (V,E)$ repräsentiert. $V = \{v_1,v_2,\ldots,v_n\}$ steht für die Menge der n Knoten und $E = \{e_1,e_2,\ldots,e_m\}$ für die Menge der m bidirektionalen Kanten zwischen den Knoten. Die untersuchten Netzwerke sind in einer Ringtopologie oder in einer Baumtopologie mit virtuellem Ring angeordnet. In beiden Topologien werden Pakete von ihrem Quellknoten aus gesendet und durchlaufen einmal den (virtuellen) Ring, bevor sie zu ihrem Quellknoten

zurückkehren. Bei mehreren Ringen oder bei einem virtuellen Ring unterscheidet sich die Anzahl der Knoten im Netzwerk n von der Anzahl der Knoten im Ring n_w.

Ohne Beschränkung der Allgemeinheit wird angenommen, dass alle Pakete die gleiche Paketlänge besitzen und jedes Paket über eine Zieladresse, eine Quelladresse, ein Typfeld und eine Paketidentifizierungsnummer (kurz Paket-ID) verfügt. Die Ziel- und die Quelladresse identifizieren den Ziel- und den Quellknoten der Nutzdaten des jeweiligen Paketes. Das Typfeld kennzeichnet ein Paket als ein Echtzeitpaket (engl. RT: Real-Time), als ein Nicht-Echtzeitpaket (NRT: Non-Real-Time) oder als ein leeres Paket. RT-Pakete sind für den Transport von echtzeitkritischen Daten vorgesehen und werden in festen Zyklen vom Quellknoten aus übertragen. Im Gegensatz zu den RT-Paketen werden NRT-Pakete nicht in festen Zyklen übertragen, sondern es erfolgt nur eine Übertragung, wenn keine RT-Pakete gesendet werden. In diesen Zeiträumen werden die NRT-Pakete so schnell wie möglich von einem Knoten zum anderen Knoten übertragen. Diese Art der Dienstklasse wird im Englischen als *Best Effort* bezeichnet. Ein leeres Paket wird von seinem Quellknoten ohne Inhalt versendet und kann die NRT-Daten von anderen Knoten aus dem Ring aufnehmen. In diesem Zusammenhang kennzeichnet die Paket-ID den ursprünglichen Sender des leeren Paketes, da Quell- und Zieladresse zu den NRT-Nutzdaten gehören und vom neuen Sender überschrieben werden. Die Verarbeitung der verschiedenen Pakettypen wird in Kapitel 4.2.2 näher beschrieben.

Dieses Modell baut auf einer *Store-and-Forward*-Architektur auf, in der empfangene Pakete zunächst vollständig im Eingangspuffer gespeichert werden, bevor sie vom Switch zum Zielport weitergeleitet werden. Die Größe der Paketpuffer ist in diesem Modell nicht limitiert, wodurch auch ein Paketverlust aufgrund von überlaufenden Puffern ausgeschlossen wird. Ohne Beschränkung der Allgemeinheit werden in diesem Modell keine netzwerk- oder switchspezifischen Eigenschaften, wie z. B. die im Ethernet-Standard spezifizierte Paketlücke (IFG), die Verarbeitungsverzögerung des Switches oder der durch die Ethernet-Ports verursachte Jitter, betrachtet. Aus diesem Grund kann davon ausgegangen werden, dass zwei Pakete, die im Puffer gespeichert sind, ohne Lücke zwischen den Paketen übertragen werden können. Die Auswirkungen der netzwerk- und switchspezifischen Eigenschaften werden in der simulativen Analyse des Self*S*-Protokolls in Kapitel 5 untersucht.

Jeder Knoten im Modell besteht aus einem Prozessor und einem Store-and-Forward-Switch mit mindestens 2 Ports. Neben der Applikation werden vom Prozessor eines Knotens i alle eingehenden Pakete mit der Zieladresse von i verarbeitet. Ebenfalls werden vom Prozessor Daten für neue Pakete generiert. Die Zeit wird an dieser Stelle in sogenannte Zeitschlitze unterteilt. Dabei entspricht jeder Zeitschlitz der Zeit, die für das Versenden eines einzelnen Paketes benötigt wird. Die Aufgabe des Switches eines Knotens i ist es, alle Pakete weiterzuleiten, die eine Paket-ID haben, die nicht

mit der Paket-ID von i übereinstimmen. Pakete mit der Paket-ID von i werden zur Verarbeitung an den Prozessor weitergeleitet. Ohne eine Lücke im Paketstrom zu erzeugen, wird ein neues Paket mit dieser Paket-ID vom Prozessor erstellt. Der Switch ist in der Lage, in beide Richtungen Pakete weiterzuleiten (Vollduplexbetrieb). Um die Erklärungen in den folgenden Teilen dieses Kapitels so klar wie möglich zu halten, beziehen sich die Erläuterungen auf eine Kommunikation in eine Richtung, wenn die Kommunikation in die andere Richtung in gleicher Weise behandelt werden kann. Sollte dieses nicht zutreffen, wird darauf im Einzelnen hingewiesen.

Mithilfe der Store-and-Forward-Architektur können alle n Knoten gleichzeitig ein Paket zu ihrem Nachbarknoten senden, ohne Paketkollisionen zu verursachen. Folglich kann jeder Knoten mindestens ein Paket pro Zeitschlitz senden. Aufgrund der fehlenden Kollisionen kann das Übertragungsverhalten mithilfe der Switch-Architektur als deterministisch betrachtet werden. Mit dem deterministischen Übertragungsverhalten erfüllt die Architektur eine Voraussetzung für den Einsatz unter harten Echtzeitbedingungen. Des Weiteren können unvorhersagbare Pufferverzögerungszeiten ausgeschlossen werden, wenn die n Knoten miteinander synchronisiert sind, d. h., sie beginnen zur selben Zeit mit der Übertragung ihres Paketes. Im synchronisierten Zustand werden keine Verzögerungszeiten durch Pakete verursacht, die auf die Übertragung des vorangegangenen Paketes warten müssen. Bei gleicher Paketlänge ist das vorangegangene Paket bereits übertragen, wenn das nächste Paket vollständig empfangen wurde. Unter dieser Voraussetzung kann die Verzögerungszeit des Store-and-Forward-Switches mit der Übertragungszeit T_{Trt} gleichgesetzt werden.

Die Übertragungszeit $T_{\mathrm{Trt}} = T_{\mathrm{Prd}} + T_{\mathrm{Trn}}$ ergibt sich aus der Summe der Ausbreitungsverzögerung T_{Prd} und der Übertragungsverzögerung T_{Trn} über eine Kante. Die Ausbreitungsverzögerung steht in Abhängigkeit zu der Geschwindigkeit, mit der ein einzelnes Bit über das gewählte Medium übertragen werden kann und der Länge der Verbindung zwischen zwei Knoten. Die Übertragungsverzögerung ergibt sich aus der Division von Paketlänge und Datenrate. Die Datenrate und das Medium werden von der Spezifikation des gewählten Protokollstandards vorgegeben und können für ein Netzwerk als feste Größen angenommen werden. Ohne Beschränkung der Allgemeinheit werden in diesem Modell die Paketlänge und die Länge der Verbindung zwischen zwei Knoten als konstante Größen definiert. Daraus folgt, dass die Übertragungszeit T_{Trt} über jede Verbindung und für jedes Paket konstant ist. Die Übertragungszeit T_{Trt} entspricht einem Zeitschritt im Modell. Bei einer konstanten Übertragungszeit ist die minimale Paketumlaufzeit (eng. Round-Trip Time) T_{RTT} nur abhängig von der Übertragungszeit T_{Trt} und der Anzahl Knoten in einem (virtuellen) Ring n_w und wird nach Gleichung (4.1) bestimmt.

$$T_{\mathrm{RTT}} = T_{\mathrm{Trt}} \cdot n_w \tag{4.1}$$

Mit der Einführung eines Zeitschrittes entspricht die Paketumlaufzeit n_w Zeitschritten. Der Fokus von Self*S* ist die Unterstützung von harter Echtzeitkommunikation. In Kapitel 2.1 werden die Echtzeiteigenschaften für ein allgemeines Echtzeitsystem beschrieben. An dieser Stelle werden die Bedeutungen einiger Echtzeiteigenschaften für das Self*S*-Protokoll konkretisiert:

- **Zykluszeit:** Ein Knoten i erzeugt immer nach einer konstanten Zeitperiode T_{Cyc_i} ein neues Echtzeitpaket. Diese Zeitperiode wird als Zykluszeit des Knotens i bezeichnet und kann für jeden Knoten unterschiedlich sein. Die konstante Zeitperiode $T_{\mathrm{Cyk}_{i,j}}$ ist die Zykluszeit für eine Echtzeitkommunikationsbeziehung von Knoten i zu einem Zielknoten j.

- **Jitter:** Der Jitter ist die Differenz zwischen der tatsächlichen Zykluszeit $T^k_{i,j}$ für die Ankunft eines Paketes p vom Knoten i gesendet an den Knoten j in der k-ten Runde und der erwarteten Zykluszeit $T_{\mathrm{Cyk}_{i,j}}$. Aus der Gleichung für den periodischen Jitter (2.1) folgt hier ein Jitter von

$$J^k_{i,j} = T^k_{i,j} - T_{\mathrm{Cyk}_{i,j}}. \tag{4.2}$$

- **Quadratischer Jitterfehler :** Der *quadratische Jitterfehler* J_{Err} des k-ten Paketes von Knoten i an den Knoten j wird definiert als

$$J^k_{\mathrm{Err}_{i,j}} = {J^k_{i,k}}^2. \tag{4.3}$$

Der quadratische Jitterfehler wurde eingeführt, um den Einfluss des Jitters auf das Echtzeitsystem zu bewerten. Ein kleiner Jitter ist in den meisten Fällen nicht zu verhindern und die meisten Echtzeitsysteme können mit einem geringen Jitter fehlerfrei arbeiten. Bei einem großen Jitter ist der Einfluss auf das Echtzeitsystem wesentlich größer und das System kann nicht mehr fehlerfrei arbeiten. Aus diesem Grund hat ein größerer Jitter einen wesentlich größeren Einfluss auf den quadratischen Jitterfehler. Jitter und somit auch quadratischer Jitterfehler treten auf, wenn ein Paket auf dem Weg zu seinem Ziel nicht unverzüglich weitergeleitet werden kann. Dieser Fehler kann z. B. im Sender auftreten, wenn ein Paket aufgrund eines höher priorisierten Paketes im Eingangspuffer nicht sofort gesendet werden kann. Zusätzlicher Jitter kann an jedem Knoten entstehen, der zwischen dem Sender und Empfänger liegt, wenn das Paket auf die Übertragung von höher priorisierten Paketen warten muss. Es ist die

Aufgabe des Synchronisationsprotokolls, die Summe der quadratischen Jitterfehler so klein wie möglich zu halten.

Basierend auf der Zykluszeit für jeden Knoten i und der Anzahl der Knoten n ist es möglich, eine Netzwerkauslastung des virtuellen Rings zu definieren, die von den Echtzeitpaketen verursacht wird:

- **Netzwerkauslastung:** Die *Netzwerkauslastung* λ wird definiert als die durchschnittliche Anzahl von Echtzeitpaketen, die in jedem Zeitschritt gesendet werden:

$$\lambda = \sum_{j=1}^{n} 1/T_{\mathrm{Cyc}_i} \tag{4.4}$$

Es ist offensichtlich, dass für $\lambda > 1$ einige Zykluszeiten nicht mehr eingehalten werden können und das Protokoll somit instabil wird. In Kapitel 4.2.3 wird gezeigt, dass schon bei geringer Netzwerklast der maximale Jitter sehr hoch werden kann.

4.2 Selbstsynchronisation

Ein gleichzeitiger Start der Kommunikation erfordert normalerweise die Synchronisation der lokalen Uhren aller n Knoten und die Festlegung eines gemeinsamen Startpunktes, an dem die Übertragung beginnen soll. Die explizite Uhrensynchronisation erfordert spezielle Synchronisationsprotokolle, die immer einen Teil der Bandbreite benötigen. Für die Uhrensynchronisation wird eine Referenzuhr benötigt. Als Referenzuhr wird entweder ein Knoten im Netzwerk bestimmt oder ein spezieller Knoten mit einer sehr genauen lokalen Uhr verwendet. In beiden Fälle kann ein Ausfall der Referenzuhr eine Synchronisation der Knoten verhindern. Zwar gibt es Verfahren, die bei einer Störung eine neue Referenzuhr bestimmen. Aber auch diese Verfahren benötigen Bandbreite und können die Echtzeitkommunikation stören. Das in diesem Kapitel vorgestellte Protokoll benötigt keine explizite Uhrensynchronisation, um einen gleichzeitigen Start der Kommunikation zu gewährleisten. Die Knoten synchronisieren sich beim Self*S*-Protokoll selbst und bleiben in dem synchronisierten Zustand, solange sich die Anzahl der Knoten im Ring nicht verändert. Der Prozess der Selbstsynchronisation wird verteilt auf jedem der n Knoten ausgeführt. Eine Referenzuhr, die ausfallen kann, ist nicht erforderlich. Im Folgenden wird der Prozess der Selbstsynchronisation im Detail beschrieben.

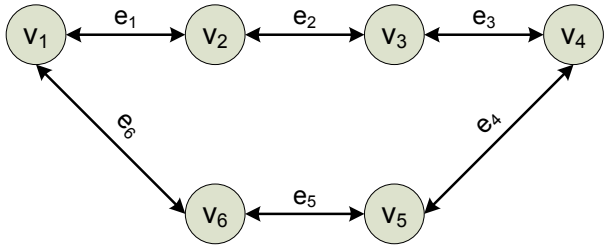


Abbildung 4.1: Ringtopologie eines Netzwerkes

4.2.1 Prozess der Selbstsynchronisation

Der Prozess der Selbstsynchronisation ist ein wichtiger Bestandteil des hier vorgestellten Echtzeitprotokolls. Im synchronisierten Zustand starten alle n Knoten gleichzeitig mit der Übertragung ihres Paketes. Die Echtzeitkommunikation beim Self$S$-Protokoll ist ausschließlich zeitgesteuert. Eine ereignisgesteuerte Kommunikation wird zunächst nicht betrachtet, lässt sich aber nahtlos in das Protokoll mit einbinden (Kapitel 4.4). Self$S$ basiert, wie in Abbildung 4.1 dargestellt, auf einem Netzwerk, in dem die n Knoten in einem (virtuellen) Ring angeordnet sind. Der Prozess der Selbstsynchronisation wird in diesem Abschnitt für den Fall untersucht, dass jeder der n Knoten alle n Zeitschritte ein eigenes Echtzeitpaket senden muss. Damit wird für jeden Knoten i eine Zykluszeit von $T_{\mathrm{Cyk_i}} = T_{\mathrm{RTT}}$ gewählt. Im Folgenden wird der Prozess der Selbstsynchronisation an einem Beispiel erläutert.

Beispiel: Am Anfang beginnt jeder Knoten mit der Übertragung seiner Pakete zu beliebigen Zeitpunkten. Abbildung 4.2 zeigt ein Beispiel für einen solchen asynchronen Paketstart für ein Netzwerk mit vier Knoten. Die Abbildung 4.2 illustriert den Paketfluss über die vier Kanten e_1 bis e_4 des Netzwerkes. Die Pakete sind mit ihrer Paket-ID markiert. Die Kommunikation ist in der ersten Runde noch asynchron und die Paketumlaufzeit der Pakete 1, 2 und 4 ist in dieser Runde größer als die minimale Paketumlaufzeit aus Gleichung (4.1). Grund für die Verzögerung bei der Übertragung der drei Pakete sind zusätzlich Wartezeiten in den Warteschlangen von Knoten 2 und 3. Die Pakete müssen in diesen Knoten erst auf die Übertragung des vorangegangenen Paketes warten. Das Warten der Pakete in dem Eingangspuffer eines Switches führt zu einer Aneinanderreihung der Pakete. In Abbildung 4.2 ist die Aneinanderreihung der Pakete für die Kante e_3 zu beobachten. Der Switch von Knoten 3 sendet die wartenden Pakete direkt nacheinander zum nächsten Switch. Hat der Switch jeweils ein Paket der vier Knoten übertragen, dann sind alle Knoten synchronisiert und die Übertragung der Pakete startet anschließend an allen Knoten immer zum selben Zeitpunkt.

Für den Prozess der Selbstsynchronisation ist kein globales Wissen notwendig und der Prozess wird völlig verteilt ausgeführt. Lediglich der Knoten muss wissen, ob er

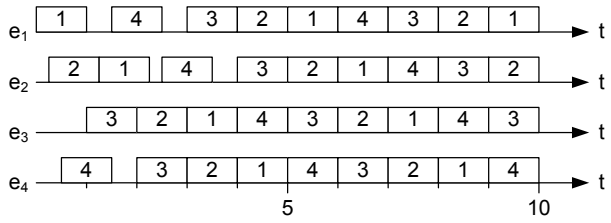


Abbildung 4.2: Prozess der Selbstsynchronisation

synchronisiert ist und ob er den Echtzeitanwendungen des Knotens eine Übertragung mit festen Zykluszeiten zur Verfügung stellen kann. Eine solche Echtzeitanwendung ist z. B. ein Regler, der Sensor- und Aktordaten mit einem anderen Knoten austauscht. Den Status der Synchronisation erkennt ein Knoten mithilfe von Algorithmus 1, der vom jedem Knoten ausgeführt wird.

Den Synchronisationsstatus eines Knotens detektiert der Algorithmus über eine Lücke in der Übertragung der Pakete. Befinden sich innerhalb einer Runde keine Lücken im Paketstrom, ist der Knoten synchronisiert. Der Algorithmus startet mit der Initialisierung der Variablen *detect_gap* und *synchronized*. Die beiden Variablen *detect_gap* und *synchronized* repräsentieren nur den Status der Selbstsynchronisation, haben aber keinen Einfluss auf den Synchronisationsprozess. Der Synchronisationsstatus wird nur für höhere Protokollebenen oder für die Echtzeitapplikationen benötigt. Die Variable *detect_gap* wird zunächst mit *true* initialisiert. Diese Variable hat immer dann den Wert *true*, wenn der Algorithmus eine Lücke zwischen zwei Paketen erkannt hat. Eine Lücke kann detektiert werden, indem der Algorithmus in jeder Schleifenrunde den Status des Eingangspuffers (Variable *inqueue*) überprüft. Ein leerer Eingangspuffer führt zu einer Lücke im Paketfluss. Die Statusvariable *synchronized* wird mit *false* initialisiert und sie erhält den Wert *true*, wenn der Synchronisationsprozess abgeschlossen ist. Als Nächstes ruft der Algorithmus, der vom Prozessor des Knotens i ausgeführt wird, die Funktion *send_packet(generate(RT), i)* auf und der Knoten i sendet das erste Paket mit der Paket-ID von i. Der Parameter RT der aufgerufenen Funktion setzt das Typfeld und markiert das Paket als ein Echtzeitpaket.

Nach der Initialisierung begibt sich der Algorithmus in eine unendliche *While*-Schleife. In dieser Schleife wird als Erstes der Status des Eingangspuffers von i überprüft. Befinden sich Pakete im Puffer und hat das nächste Paket in dieser Warteschlange die Paket-ID vom Knoten i, dann wird *detect_gap* auf *false* gesetzt und der Algorithmus sendet ein neues Echtzeitpaket mit der Paket-ID von i durch den Aufruf der Funktion *send_packet(generate(RT), i)*. Pakete mit einer anderen Paket-ID werden über die Funktion *forward_packet()* vom Knoten i einfach weitergeleitet. Empfängt der Knoten i nach einer Runde wieder sein eigenes Paket und ist *detect_gap* immer noch *false*, dann wird der Status *synchronized* auf *true* gesetzt und der Prozess der Selbstsynchro-

Algorithmus 1 Self$S$ protocol(node i)

```
1: detect_gap ← true
2: synchronized ← false
3: send_packet(generate(RT), i)
4: while true do
5:    if inqueue == empty then
6:       detect_gap ← true
7:    else if packet_ID == i then
8:       send_packet(generate(RT), i)
9:       if detect_gap then
10:         synchronized ← false
11:         detect_gap ← false
12:      else
13:         synchronized ← true
14:      end if
15:   else
16:      forward_packet()
17:   end if
18: end while
```

nisation ist für den Knoten i abgeschlossen. Der Algorithmus wird aber vom Knoten i weiter ausgeführt, um den Status der Synchronisation weiter zu beobachten. Änderungen, wie z. B. die Anzahl von Knoten im Netzwerk, können so jederzeit detektiert werden und die Selbstsynchronisation der Knoten wird erneut durchgeführt.

Lemma 4.2.1. *Die obere Schranke für die maximale Synchronisationszeit liegt bei* $T_{\text{Syn}} = (3 \cdot n - 2) \cdot T_{\text{Trn}}$.

Beweis. Es wird angenommen, dass der Synchronisationsprozess startet, wenn der Knoten i, der als Letzter im Ring sein erstes Paket sendet, mit der Ausführung des Algorithmus 1 beginnt. Innerhalb der Übertragungszeit T_{Trn} des ersten Paketes von Knoten i wird mindestens ein Paket vollständig von i empfangen. Jedes weitere Paket erreicht den Eingangspuffer von Knoten i während der Übertragung des vorangegangenen Paketes und kann direkt im Anschluss übertragen werden. Somit können alle Pakete der n Knoten von dem Knoten i direkt hintereinander ohne Lücke zwischen den Paketen in n Zeitschritten übertragen werden. Nachdem der Knoten i das letzte der n Pakete übertragen hat, benötigt dieses im schlechtesten Fall $(n-2)$ Zeitschritte, bis dieses Paket ihren Quellknoten erreicht. Der schlechteste Fall tritt dann ein, wenn das letzte Paket die Paket-ID des Knotens besitzt, der am weitesten von dem Knoten i entfernt ist. Nachdem ein Knoten sein eigenes Paket detektiert hat, dauert es weitere n Zeitschritte, bevor der Algorithmus den Status *synchronized* auf *true* setzt. Die Summe

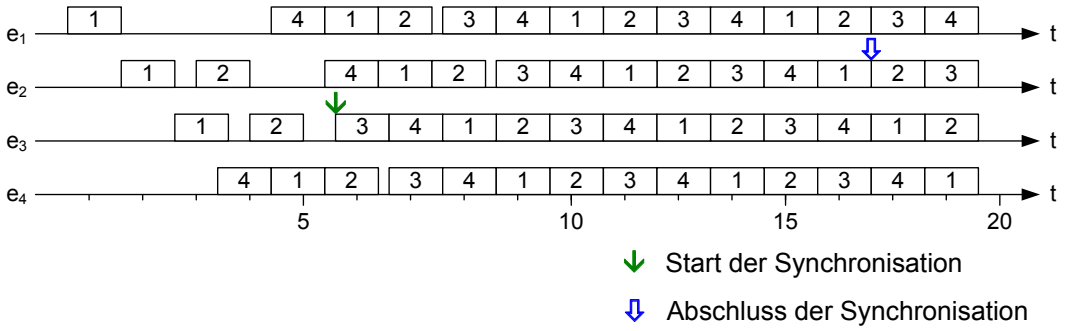


Abbildung 4.3: Beispiel eines Paketflusses mit maximaler Synchronisationszeit

über die beschriebenen Zeitschritte resultiert in der maximalen Synchronisationszeit, wie sie in Lemma 4.2.1 definiert ist. □

In dem folgenden Beispiel wird für ein Netzwerk mit vier Knoten ein Fall beschrieben, in dem die Synchronisation erst nach der maximalen Synchronisationszeit erfolgt.

Beispiel: Die Abbildung 4.3 zeigt den Paketfluss über die 4 Kanten eines Netzwerkes mit 4 Knoten. Der Beginn der Übertragung erfolgt asynchron und es entsteht, wie im vorangegangenen Beispiel geschildert, an dem Knoten 3 eine Aneinanderreihung der Pakete, die von den 4 Knoten gesendet wurden. Jedoch unterscheiden sich in diesem Beispiel die zeitliche Reihenfolge und der Abstand der Pakete zu Beginn der Übertragung. Dadurch ändert sich die Reihenfolge, in der die Pakete von Knoten 3 übertragen werden, sodass das Paket mit der ID von Knoten 2 als Letztes von Knoten 3 übertragen wird. Der Knoten 2 ist der Knoten, der vom Knoten 3 am weitesten entfernt ist, wodurch er als letzter Knoten synchronisiert wird. Der Prozess der Synchronisation startet mit dem Beginn der ersten Übertragung von Paket 3 durch den Knoten 3, weil dieser Knoten als Letztes mit der Übertragung seines ersten Paketes beginnt. Der Knoten 3 benötigt vier Zeitschritte, um die vier Pakete der Knoten zu senden. Nach weiteren zwei Zeitschritten wird das Paket mit der ID 2 von Knoten 2 empfangen. Es vergehen weitere 4 Zeitschritte, bis der Knoten 2 das Paket mit der ID 2 erneut empfängt und registriert, dass in dieser Zeit alle Pakete ohne Lücke übertragen wurden. Damit ist nach insgesamt 10 Zeitschritten der Prozess der Selbstsynchronisation für alle Knoten abgeschlossen. Die 10 Zeitschritte entsprechen der maximalen Synchronisationszeit nach Lemma 4.2.1 für ein Netzwerk mit $n = 4$ Knoten.

Lemma 4.2.2. *Bei einem synchronisierten Netzwerk sind der Jitter J und der quadratische Jitterfehler J_{Err} gleich null.*

Beweis. Im synchronisierten Zustand übertragen die n Knoten den Paketstrom ohne Lücken. Die Paketumlaufzeit T_{RRT} ist in diesem Fall für jedes Paket gleich und

kann mit der Formel (4.1) bestimmt werden. In Kapitel 4.1 wurde definiert, dass die minimale Paketumlaufzeit T_{RRT} aus Gleichung (4.1) für eine feste Anzahl von Knoten konstant ist. Da alle n Knoten ein neues Echtzeitpaket mit der eigenen Paket-ID senden, sobald sie das vorangegangene Paket mit der gleichen Paket-ID empfangen haben, ist die Zykluszeit T_{Cyc_i} gleich der Paketumlaufzeit T_{RRT}. Bei konstanten T_{Cyc_i} und bei lückenloser Paketübertragung ist die tatsächliche Zykluszeit $T^k_{i,j}$ des k-ten Paketes gleich der erwarteten Zykluszeit $T_{\mathrm{Cyk}_{i,j}}$ und der Jitter J und der quadratische Jitterfehler J_{Err} sind nach Gleichung (4.2) und (4.3) gleich null. □

4.2.2 Harmonische Zykluszeiten

In dem letzten Abschnitt wurde gezeigt, dass das Self*S*-Protokoll eine jitterfreie harte Echtzeitkommunikation unterstützt und Zykluszeiten von $T_{\mathrm{Cyc}_i} = T_{\mathrm{RTT}}$ garantiert. In einem verteilten Sensor-Aktor-System müssen nicht alle Informationen gleich häufig zwischen den Knoten ausgetauscht werden. Im einfachsten Fall kann natürlich für jeden Knoten die kleinste mögliche Zykluszeit gewählt werden. Einige RT-Daten werden in diesem Fall häufiger übertragen als eigentlich notwendig. Die dadurch verschwendete Bandbreite wird bei harmonischen Zykluszeiten für die NRT-Kommunikation verwendet. Bei einer harmonischen Zykluszeit entspricht die Zykluszeit von jedem Knoten einem beliebigen Vielfachen der Paketumlaufzeit und $T_{\mathrm{Cyc}_i} = T_{\mathrm{RTT}} \cdot K_i$. Das $K_i \in \mathbb{N}^+$ kann für jeden Knoten i unterschiedlich sein. Auch bei der digitalen Regelungstechnik sind unterschiedliche Abtastraten in einem System häufig harmonisch miteinander verbunden [Liu00].

Die Erweiterung des Protokolls um harmonische Zykluszeiten ermöglicht zum einen eine größere Flexibilität der Echtzeitkommunikation für jeden Knoten und zum anderen die gleichzeitige Übertragung von NRT-Daten. Eine Parametrierung des Netzwerkes durch den Benutzer beschränkt sich auf die Wahl der Zykluszeit für jeden Knoten als ein Vielfaches der Paketumlaufzeit. Die Paketumlaufzeit kann einfach mithilfe von Gleichung (4.1) bestimmt werden. Des Weiteren können die Anwendungen eines jeden Knotens die Zykluszeit zur Laufzeit des Systems ändern, ohne den Echtzeitdatenverkehr der anderen Knoten zu beeinflussen oder die Synchronisation der n Knoten zu stören.

NRT-Daten sollen so schnell wie möglich bzw. mit *Best Effort* von der Quelle zum Ziel übertragen werden. Dazu muss das Modell eines Knoten (siehe Kapitel 4.1) um zusätzliche Warteschlangen für die NRT-Pakete erweitert werden. Die Nutzdaten, die Quell- und die Zieladresse jedes empfangenen NRT-Paketes werden im NRT-Puffer des Knotens abgelegt. Weitergeleitet werden die NRT-Pakete nur in ungenutzten Zeitschlitzen, um den RT-Datenverkehr nicht zu stören. Jedes Mal, wenn ein Knoten i kein RT-Paket senden muss, verwendet er die Paket-ID des zuletzt empfangenen Pa-

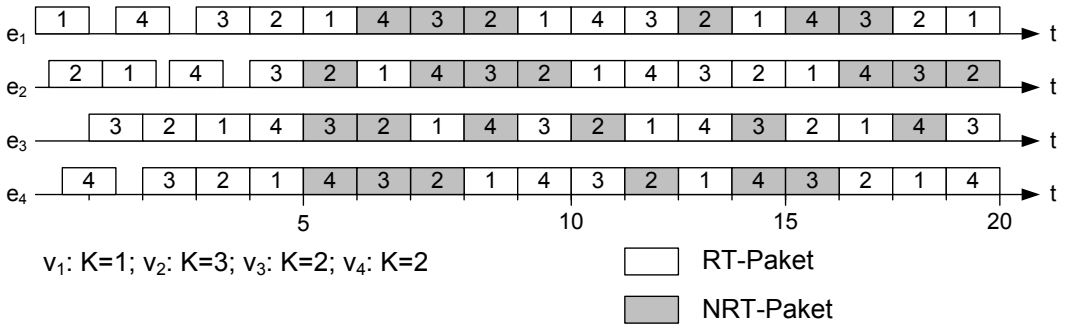


Abbildung 4.4: Selbstsynchronisation bei variablen Zykluszeiten

ketes, um das an erster Stelle in der NRT-Warteschlange stehende Paket zu senden. Hat der Knoten weder ein RT- noch ein NRT-Paket zu senden, sendet der Knoten ein leeres Paket. Das leere Paket besteht aus der Quelladresse des aktuellen Senders, einer Broadcast-Zieladresse und der Paket-ID des Knotens, bei dem das Paket seine Runde gestartet hat. Die Nutzdaten bestehen bei einem leeren Paket nur aus Nullen.

In Abbildung 4.4 wird ein Beispiel für die Selbstsynchronisation mit variablen Zykluszeiten und NRT-Datenverkehr gezeigt. In dem abgebildeten Beispiel beginnen die Knoten ebenfalls zu beliebigen Zeitpunkten mit der Übertragung ihres Paketes. Allerdings wurde das K_i in diesem Beispiel für die vier Knoten unterschiedlich gewählt. Die entstehenden freien Zeitschlitze werden mit NRT-Paketen aufgefüllt. Nachdem der Knoten 3 jeweils ein Paket der vier Knoten übertragen hat, sind die Knoten synchronisiert.

Algorithmus 2 ist eine Erweiterung von Algorithmus 1 und beschreibt den Prozess der Selbstsynchronisation bei harmonischen Zykluszeiten und die Übertragung von Echtzeitpaketen, NRT-Pakten und leeren Paketen. Die folgende Erläuterung des Algorithmus beschränkt sich auf die Erweiterungen im Vergleich zu Algorithmus 1, die eine RT- und NRT-Kommunikation und Zykluszeiten von $T_{\mathrm{Cyc}_i} = T_{\mathrm{RTT}} \cdot K_i$ ermöglichen.

In dem erweiterten Algorithmus wird eine neue Zählervariable k_i eingeführt und mit 1 initialisiert. Die Zählervariable k_i wird alle T_{RTT} Schritte inkrementiert und aus dem Ergebnis wird der Modulo K_i gebildet und k_i zugewiesen. Die Paketumlaufzeit T_{RTT} bestimmt der Algorithmus implizit über die Erkennung der eigenen Pakete. Die Erweiterung des Algorithmus beginnt in Zeile 9, nachdem Knoten i das erste Paket des Eingangspuffers als sein eigenes Paket identifiziert hat. Besitzt die Zählervariable k_i den Wert null, wird vom Algorithmus die Funktion *send_packet(generate(RT), i)* aufgerufen, worauf der Knoten i ein neues Echtzeitpaket mit der Paket-ID von i sendet. Für k_i größer oder gleich eins überträgt der Knoten mit dem Aufruf der Funktion *send_packet(pop(NRT_queue), i)* das erste Paket aus der NRT-Warteschlange oder

Algorithmus 2 Self$S$ protocol(node i, K_i)

```
1:  detect_gap ← true
2:  synchronized ← false
3:  send_packet(generate(RT), i)
4:  k_i ← 1
5:  while true do
6:    if inqueue == empty then
7:      detect_gap ← true
8:    else if packet_ID == i then
9:      if k_i == 0 then
10:       send_packet(generate(RT), i)
11:     else
12:       send_packet(pop(NRT_queue), i)
13:     end if
14:     k_i ← (k_i + 1) mod K_i
15:     if detect_gap then
16:       synchronized ← false
17:       detect_gap ← false
18:     else
19:       synchronized ← true
20:     end if
21:   else
22:     if packet_type == RT then
23:       forward_packet()
24:     else
25:       push(pop(inqueue), NRTqueue)
26:       send_packet(pop(NRT_queue))
27:     end if
28:   end if
29: end while
```

ein leeres Paket, falls keine Pakete in der NRT-Warteschlange verfügbar sind. Sowohl das NRT-Paket als auch das leere Paket erhalten die Paket-ID des Knotens i.

Die auf Zeile 21 folgenden Zeilen behandeln die Übertragung von Paketen mit einer Paket-ID, die sich von der ID des Knotens i unterscheidet. Besitzt ein solches Paket den Pakettypen *RT*, wird dieses Paket vom Knoten über den Funktionsaufruf *forward_packet()* direkt weitergeleitet. Besitzt das Paket einen anderen Pakettypen, wird das Paket über den Aufruf der Funktion *push(pop(inqueue), NRTqueue)* aus dem Eingangspuffer in die NRT-Warteschlange geschrieben. Erkennt die Funktion *push(pop(inqueue), NRTqueue)* ein leeres Paket, wird dieses nur aus dem Eingangspuffer entfernt. Die Paket-ID dieses Paketes wird nicht in den NRT-Puffer abgelegt, sondern zur Übertragung des nächsten Paketes aus der NRT-Warteschlange verwendet.

Über den Funktionsaufruf *send_packet(pop(NRT_queue))* wird das nächste Paket aus der NRT-Warteschlange entfernt und übertragen. Bei einer leeren NRT-Warteschlange sendet i ein leeres Paket. Auch dieses Paket trägt die Paket-ID des zuvor empfangenen Paketes.

Lemma 4.2.3. *Die obere Schranke der maximalen Synchronisationszeit ist* $T_{\mathrm{Syn}} = (3 \cdot n - 2) \cdot T_{\mathrm{Trn}}$ *und der Jitter* J *und der quadratische Jitterfehler* J_{Err} *sind gleich null.*

Beweis. Die Beweise zu Lemma 4.2.1 und Lemma 4.2.2 sind auch in diesem Fall weiterhin gültig. Auch hier gilt die Annahme, dass der Prozess der Selbstsynchronisation beginnt, wenn der letzte Knoten mit der Ausführung des Algorithmus beginnt. Des Weiteren sendet jeder Knoten im virtuellen Ring, der Algorithmus 2 ausführt, in jeder Runde ein Paket mit der Paket-ID des eigenen Knotens. Im Gegensatz zu Algorithmus 1 markiert ein Knoten i nur jede K_i-te Runde das eigene Paket als Echtzeitpaket. In den anderen Runden ist der Typ des Paketes entweder ein NRT-Paket oder ein Echtzeitpaket. Das bedeutet, dass sich nur der Inhalt des Paketes gegenüber Algorithmus 1 verändert hat. Die Häufigkeit, mit der die n Knoten Pakete senden, bleibt gleich. Daraus folgt, dass die Erweiterungen keinen Einfluss auf die maximale Synchronisationszeit, den Jitter und den quadratischen Jitterfehler haben. Die Beweise für Lemma 4.2.1 und Lemma 4.2.2 behalten daher ihre Gültigkeit für den Algorithmus 2. □

4.2.3 Beliebige Zykluszeiten

Im letzten Abschnitt wurde eine Protokollerweiterung eingeführt, die harmonische Zykluszeiten erlaubt, wobei die Zykluszeit jedes Knotens i die Bedingung $T_{\mathrm{Cyc}_i} = T_{\mathrm{RTT}} \cdot K_i$ für ein beliebiges $K_i \in \mathbb{N}^+$ erfüllen muss. In diesem Abschnitt werden beliebige Zykluszeiten T_{Cyc_i} für jeden Knoten i betrachtet. Der Effekt dieser Erweiterung auf den Ansatz der Selbstsynchronisation wird untersucht und eine untere Schranke für die Anzahl der Kollisionen und für den quadratischen Jitterfehler wird bestimmt.

Bei beliebigen Zyklen versucht ein Knoten i ein neues Echtzeitpaket nach einer Periode von T_{Cyc_i} zu übertragen. Die Zykluszeiten zwischen den n Knoten können sich unterscheiden. Der Start der Kommunikation erfolgt ebenfalls asynchron und resultiert in einer Pufferverzögerung der Pakete und im besten Fall sogar in einer Aneinanderreihung der Pakete, wie sie in Kapitel 4.2.1 beschrieben wird. Diese Aneinanderreihung der Pakete führt jedoch nicht in jedem Fall zu einem Synchronisationseffekt, weil die n Knoten ihr nächstes Paket basierend auf der Periode der Zykluszeit übertragen und die Aneinanderreihung durcheinandergebracht wird. Daher ist eine Selbstsynchronisation nicht immer möglich. In Abbildung 4.5 ist der Paketfluss für ein Netzwerk mit vier Knoten, die beliebige Zykluszeiten haben, dargestellt. Auch in diesem Beispiel

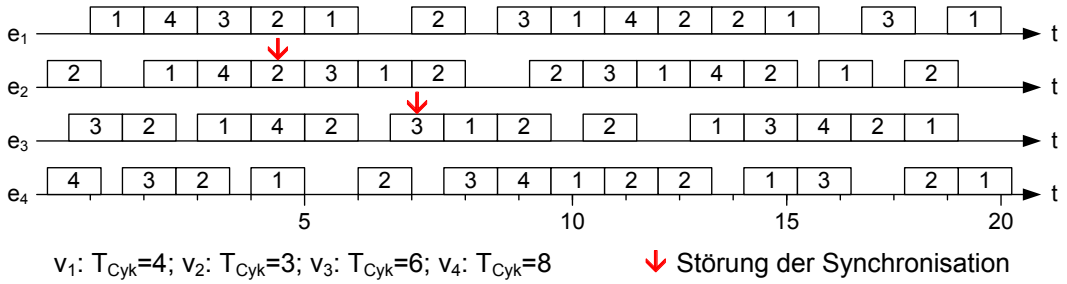


Abbildung 4.5: Selbstsynchronisation bei beliebigen Zykluszeiten

beginnen die Knoten mit der asynchronen Übertragung ihrer Pakete. An der Kante e_1 ist eine Aneinanderreihung der Pakete aller vier Knoten, die in Knoten 1 entstanden ist, zu sehen. Allerdings führt diese Aneinanderreihung nicht zur Synchronisation der Knoten, weil schon im nächsten Knoten, dem Knoten 2, das Paket mit der ID 3 auf das neue Paket 2 warten muss. Durch die zusätzliche Wartezeit entsteht ein Jitter für das Paket 3. Diese Störung der Synchronisation ist in der Abbildung 4.5 durch einen Pfeil oberhalb der Kante e_2 gekennzeichnet. Eine weitere Störung der Synchronisation ist oberhalb der Kante e_3 markiert. An dieser Stelle entsteht eine neue Lücke im Paketstrom, der durch das Senden eines neuen Paketes mit der ID 3 nach Ablauf der Zykluszeit von Knoten 3 entsteht. Die Aneinanderreihung wird dadurch aufgebrochen und eine Selbstsynchronisation ist in dem dargestellten Beispiel nicht möglich.

Im folgenden Lemma wird gezeigt, dass für eine Netzwerkauslastung von $\lambda < 1$ die Anzahl der Kollisionen, die den Jitter verursachen, sogar bei einem für eine Ringtopologie optimalen Protokoll sehr groß werden kann. Eine Kollision entsteht immer dann, wenn mehr als ein Paket in einem Zeitschlitz übertragen werden soll. Mit Kollision ist nicht die Kollision zweier Pakete auf einem gemeinsamen Übertragungsmedium gemeint. Kollisionen im zeitlichen Ablauf einer Übertragung entstehen im Ring, wenn ein Knoten ein gerade empfangenes Paket weiterleiten und zum selben Zeitpunkt der Knoten ein neues Paket senden soll. Die Analyse der Kollisionen und dem daraus resultierenden Jitter erfolgt für ein optimales Protokoll und steht damit stellvertretend für das selbstsynchronisierende Echtzeitprotokoll. Die durch die Analyse für ein optimales Protokoll ermittelte untere Schranke kann vom Self$S$-Protokoll nicht unterboten werden.

Für den Beweis des folgenden Lemmas ist es ausreichend, die Analyse über die Anzahl der Kollisionen für einen Knoten zu zeigen, weil die Verzögerungen, die durch eine Kollision entstehen, innerhalb der Topologie nicht wieder aufgeholt werden können. In diesem Fall können alle Kollisionen in einem einzelnen Knoten beobachtet werden.

Lemma 4.2.4. *Ein Netzwerk mit n Knoten und einer Netzwerkauslastung von $\lambda < 1$ kann im schlechtesten Fall für einen beliebigen Algorithmus zu einem durchschnittli-*

chen quadratischen Jitterfehler von mindestens $O(n)$ über eine Periode von n Zeitschritten führen.

Beweis. Es wird davon ausgegangen, dass das Netzwerk beim Start bereits synchronisiert ist und alle Pakete ohne Verzögerungen übertragen werden können. Des weiteren wird angenommen, dass die Anzahl der Knoten gerade ist. Die Hälfte der Knoten hat eine Zykluszeit von $T_{\text{Cyc}} = n$ und die anderen Knoten haben eine Zykluszeit von $T_{\text{Cyc}} = n + 1$. Die durchschnittliche Netzwerkauslastung ist

$$\lambda = \frac{n}{2} \cdot \left(\frac{1}{n} + \frac{1}{n+1} \right) \leq 1. \tag{4.5}$$

Die Reihenfolge der n Zeitschlitze im synchronisierten Zustand wird so angeordnet, dass alle Pakete mit einer Zykluszeit von $T_{\text{Cyc}} = n$ geraden Zeitschlitzen und alle Pakete mit einer Zykluszeit von $T_{\text{Cyc}} = n + 1$ ungeraden Zeitschlitzen zugeordnet werden. Diese Neuanordnung kann mit der Hilfe einer virtuellen Nummerierung der Zeitschlitze erfolgen, ohne die tatsächliche Reihenfolge zu verändern. Bedingt durch die ungerade Zykluszeit $T_{\text{Cyc}} = n + 1$ werden diese Pakete jeden zweiten Zyklus einem geraden Zeitschlitz zugeordnet, was zu einer Kollision mit der ersten Hälfte der Pakete führt. Der quadratische Jitterfehler ist daher in jedem zweiten Zyklus mindestens 1. Daraus folgt ein durchschnittlicher quadratischer Jitterfehler von mindestens

$$J_{\text{Err}} = \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{n}{n+1} \cdot n \geq \frac{1}{8} \cdot n. \tag{4.6}$$

Der erste Faktor 1/2 in Formel (4.6) steht für die Pakete mit ungerader Zykluszeit. Der zweite Faktor 1/2 steht für das Auftreten eines quadratischen Jitterfehlers bei diesen Paketen von mindestens 1 in jeder zweiten Runde und der Faktor $n/(n + 1)$ steht für die Anzahl der ungeraden Pakete, die durchschnittlich in n Zeitschritten gesendet werden. □

Wie in Lemma 4.2.4 gezeigt wurde, kann die Anzahl der Kollisionen bei beliebigen Zykluszeiten sehr groß werden. Der quadratische Jitterfehler, der durch diese Kollisionen verursacht wird, ist dagegen eher klein. Das folgende Lemma wird zeigen, dass der quadratische Jitterfehler sehr viel größer wird, wenn der Jitter für den schlechtesten Fall über n Zeitschritte untersucht wird.

Lemma 4.2.5. *Der quadratische Jitterfehler in einem Netzwerk mit n Knoten und einer durch Echtzeitpakete verursachten Netzwerkauslastung von $\lambda < 1$ ist für eine Zeitperiode von n Zeitschritten im schlechtesten Fall mindestens $(n-1)^2$.*

Beweis. Der Beweis wird mit einer vollständigen Induktion über die Anzahl der Knoten n geführt. Als Induktionsstart wird $n = 2$ gewählt. In diesem Beweis wird gezeigt, dass bei einer ungünstigen Kombination der Zykluszeiten der einzelnen Knoten ein optimaler Algorithmus es nicht verhindern kann, dass die Pakete von allen Knoten in einem Zeitschlitz kollidieren. Die Zykluszeit von allen Knoten wird größer als $n_{\max} + 1$ gewählt, wobei $n_{\max}$ ein beliebiger Wert ist. Für den Induktionsstart soll die Zykluszeit der zwei Knoten $T_{\mathrm{Cyc}_1} = n_{\max} + 2$ und $T_{\mathrm{Cyc}_2} = n_{\max} + 1$ sein. Ohne Beschränkung der Allgemeinheit wird für den Induktionsstart angenommen, dass sich das System im synchronisierten Zustand befindet, Zeitschlitz 0 von Knoten 1 verwendet wird und Knoten 2 einen beliebigen Zeitschlitz von $0 < x_2 \leq n_{\max}$ benutzt, wobei x_2 vom optimalen Algorithmus bestimmt wird. Eine Kollision von zwei Paketen tritt auf, wenn Folgendes gilt:

$$k_1 \cdot T_{\mathrm{Cyc}_1} = x_2 + k_2 \cdot T_{\mathrm{Cyc}_2}$$

Die für die Knoten 1 und 2 gewählten Zykluszeiten führen zu einer Kollision für $k_1 = k_2 = x_2$. Aus der Kollision resultiert ein quadratischer Jitterfehler $J_{\mathrm{Err}} = 1$, was $(2-1)^2$ für eine Periode $n = 2$ entspricht.

Der Induktionsschritt von n Knoten nach $n+1$ Knoten geht davon aus, dass die Pakete aller n Knoten zu Beginn im Zeitschlitz 0 kollidieren. Die gemeinsame Periode der n Knoten wird definiert als $T_{\mathrm{all}} = \prod_{i=1}^{n} T_{\mathrm{Cyk}_i}$. Nach Ablauf der gemeinsamen Periode T_{all} wiederholen sich die Kollisionen der n Pakete im Zeitschlitz 0. Das Echtzeitpaket des $n+1$-ten Knotens wird vom optimalen Algorithmus in einem beliebigen Zeitschlitz von $0 < x_{n+1} \leq T_{\mathrm{all}}$ platziert. Bei einer Zykluszeit von $T_{\mathrm{Cyk}_{n+1}} = T_{\mathrm{all}} - 1$ dauert es T_{all} mal die gemeinsame Periode, bis das Paket des $n+1$-ten Knoten in Zeitschlitz 0 gesendet wird, wenn das Paket erstmals im Zeitschlitz $x_{n+1} = T_{\mathrm{all}}$ platziert wurde. Eine Zykluszeit von $T_{\mathrm{Cyk}_{n+1}} = T_{\mathrm{all}} - 1$ führt also zu einer Kollision von $n+1$ Paketen nach spätestens T_{all}^2 Zeitschritten. Diese Kollisionen führen zu einem quadratischen Jitterfehler von wenigstens $J_{\mathrm{Err}} = \sum_{i=1}^{n} i^2 \geq n^2$. □

Die geführten Beweise von Lemma 4.2.4 und Lemma 4.2.5 haben gezeigt, dass ein optimaler Algorithmus es nicht verhindern kann, dass die Anzahl der Kollisionen und der quadratische Jitterfehler bei beliebigen Zykluszeiten sehr groß werden können. Dieses Verhalten kann also auch nicht bei dem selbstsynchronisierenden Echtzeitprotokoll verhindert werden. Der Jitter, wie er in Lemma 4.2.5 für den schlechtesten Fall gezeigt wurde, ist viel größer als es die IAONA-Echtzeitklassen (Kapitel 2.5.1) zulassen würden. Daher ist es notwendig, die Zykluszeit, wie in Kapitel 4.2.2 vorgeschlagen, auf ein Vielfaches der Paketumlaufzeit zu beschränken.

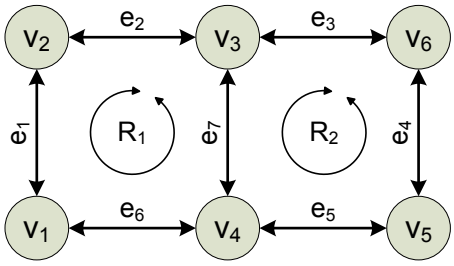


Abbildung 4.6: Netzwerk mit zwei Ringen

4.3 Weitere Topologien

Die bisherigen Betrachtungen des Self*S*-Protokolls bezogen sich auf einen einfachen Ring mit n Knoten. In diesem Abschnitt wird untersucht, ob sich das in Kapitel 4.2.2 vorgestellte Echtzeitprotokoll auch in anderen Netzwerktopologien einsetzen lässt. In dem ersten Schritt werden mehrere Ringe in eine Netzwerktopologie gelegt und so mehrere Kommunikationsgruppen gebildet. Die Knoten in einer Kommunikationsgruppe stehen in einer engen Verbindung zueinander. Sie gehören z. B. alle zu einem Teilsystem und tauschen hauptsächlich Daten untereinander aus. Bei der Bildung von Kommunikationsgruppen kommt es dazu, dass sich mehrere Ringe eine Verbindung teilen müssen. Diese geteilten Verbindungen werden in Kapitel 4.3.1 analysiert. Im zweiten Schritt in Kapitel 4.3.2 wird das Self*S*-Protokoll auf eine beliebige Netzwerktopologie angewendet. Dazu wird, wie es in vielen LANs üblich ist, ein Baum in das Netzwerk gelegt, um einen eindeutigen Pfad zwischen den Knoten zu erhalten. Ein virtueller Ring wird in den Baum eingebettet, um das Self*S*-Protokoll anzuwenden.

4.3.1 Geteilte Verbindungen

Die bisherigen Betrachtungen des Self*S*-Protokolls bezogen sich immer auf einen einzigen Ring im gesamten Netzwerk. In diesem Abschnitt wird eine Topologie aus mehreren Ringen betrachtet, die in ein Netzwerk eingebettet werden. Bei der Einbettung von mehreren Ringen in ein Netzwerk müssen sich einige Ringe eine oder mehrere Verbindungen mit anderen Ringen teilen. Diese geteilten Verbindungen verändern das Übertragungsverhalten im Ring und werden in diesem Abschnitt untersucht. Für die Analyse von mehreren Ringen wird angenommen, dass die Pakete eines Ringes nur in diesem bestimmten Ring übertragen werden. Ein Knoten i sendet Pakete mit der Paket-ID von i nur in die Ringe, zu denen der Knoten gehört. Die Untersuchungen bauen auf den vorangegangenen Protokollbeschreibungen auf und damit entspricht die Zykluszeit $T_{\mathrm{Cyc}_i} = T_{\mathrm{RTT}} \cdot K_i$ für ein beliebiges $K_i \in \mathbb{N}^+$.

Ein Beispiel für ein Netzwerk mit sechs Knoten und zwei Ringen, die sich eine Verbindung teilen, zeigt die Abbildung 4.6. Die beiden Ringe R_1 und R_2 bestehen in diesem Beispiel aus den vier Knoten $R_1 = \{v_1, \cdots, v_4\}$ und $R_2 = \{v_3, \cdots, v_6\}$. Die Knoten v_3 und v_4 gehören zu beiden Ringen und senden im Gegensatz zu den anderen Knoten zwei Pakete mit der Paket-ID des Knotens, wobei ein Paket im Ring R_1 und das andere Paket im Ring R_2 kreist.

Angenommen wird, dass die Bandbreite b für jede Verbindung im Netzwerk gleich groß ist, dann steht jedem der s Ringe, der diese Verbindung verwendet, eine Bandbreite von $b_w = b/s$ zur Verfügung. Trotz der Einschränkung der Bandbreite können für die Kommunikation die Algorithmen 1 oder 2 angewendet werden. Allerdings führt die Einschränkung der Bandbreite zu einer zusätzlichen Verzögerung der Pakete und somit zu einer Erhöhung der Paketumlaufzeit.

In der folgenden Untersuchung wird ein Netzwerk mit mehreren Ringen betrachtet, die sich eine Verbindung teilen. Dazu werden zwei Fälle unterschieden. Im ersten Fall speichert der Knoten i, der die Pakete über die geteilte Verbindung überträgt, die ankommenden Pakete in s unterschiedliche Eingangspuffer und leitet die Pakete aus den s Eingangspuffern nach der *Round-Robin*-Methode weiter. Die Eingangspuffer arbeiten nach dem FIFO-Prinzip (First In – First Out), d. h., die Pakete verlassen die Puffer in der Reihenfolge, in der sie in dem Puffer gespeichert wurden. Im zweiten Fall verwendet der Knoten i nur einen Ausgangspuffer, in dem alle ankommenden Pakete der s Ringe gespeichert werden.

Im folgenden Lemma wird die Paketumlaufzeit für den ersten Fall bestimmt und eine jitterfreie Übertragung nachgewiesen.

Lemma 4.3.1. *Ein Ring w mit n_w Knoten in einem Netzwerk mit s Ringen, die sich eine Verbindung teilen, hat eine Paketumlaufzeit von $n_w \cdot s$, wenn der Knoten i, der die Pakete aus den s Ringen über die geteilte Verbindung überträgt, einen Eingangspuffer für jeden Ring hat und die Pakete aus dem Eingangspuffer nach dem* Round-Robin-*Verfahren weiterleitet. Die Paketumlaufzeit ist konstant und das Self*S*-Protokoll bleibt jitterfrei.*

Beweis. Ohne Beschränkung der Allgemeinheit wird angenommen, dass alle n_w Pakete des Rings w sich in dem Eingangspuffer für den Ring w des Knotens i befinden. In jedem s-ten Zeitschlitz wird ein Paket aus diesem Eingangspuffer gesendet und das Paket benötigt weitere n_w Zeitschritte, bis es wieder in den Eingangspuffer abgelegt wird. In n_w Zeitschritten werden vom Knoten i weitere $\lfloor n_w/s \rfloor$ Pakete aus dem Eingangspuffer für den Ring w gesendet. Also befinden sich nach n_w Zeitschritten noch $n_w - \lfloor n_w/s \rfloor$ Pakete in diesem Eingangspuffer. Des Weiteren sind zu diesem Zeitpunkt $(n_w/s - \lfloor n_w/s \rfloor) \cdot s$ Zeitschritte verstrichen, seitdem der *Round-Robin*-Arbiter von Knoten i das letzte Paket aus dem Eingangspuffer für den Ring w gesendet hat. Ein Paket,

das den Knoten i erreicht und in dem Eingangspuffer gespeichert wurde, wartet also für $(n_w - \lfloor n_w/s \rfloor) \cdot s - (n_w/s - \lfloor n_w/s \rfloor) \cdot s$ Zeitschritte, bis es erneut von Knoten i übertragen wird. Nach n_w Zeitschritten erreicht dieses Paket wieder den Knoten i. Die Paketumlaufzeit aller Pakete im Ring w ist daher

$$\begin{aligned} T_{\mathrm{RTT}_w} &= n_w + \left(n_w - \left\lfloor \frac{n_w}{s} \right\rfloor\right) \cdot s - \left(\frac{n_w}{s} - \left\lfloor \frac{n_w}{s} \right\rfloor\right) \cdot s \\ &= n_w \cdot s. \end{aligned} \tag{4.7}$$

Aufgrund der fixen Datenrate von s Zeitschritten und der fixen Übertragungszeit von n_w Zeitschritten handelt es sich auch bei der Paketumlaufzeit um eine fixe Größe, die für kein Paket im Ring w variiert. Bei der Verwendung von Algorithmus 1 oder 2 entspricht die Zykluszeit eines Knoten T_{Cyc_i} der Paketumlaufzeit oder dem Vielfachen der Paketumlaufzeit, womit es sich auch bei der Zykluszeit um einen fixen Wert handelt. Die tatsächliche Zykluszeit in der k-ten Runde $T_{i,j}^k$ ist bei konstanter Zykluszeit und bei einer konstanten Übertragungszeit gleich der erwarteten Zykluszeit $T_{\mathrm{Cyk}_{i,j}}$ und der Jitter J und der quadratische Jitterfehler J_{Err} sind nach Gleichung (4.2) und (4.3) gleich null. □

Im nächsten Schritt wird eine geteilte Verbindung für den Fall betrachtet, dass alle eingehenden Pakete der s Ringe in einen Ausgangspuffer gespeichert werden. Im Gegensatz zu Lemma 4.3.1 wird das folgende Lemma zeigen, dass das Protokoll in diesem Fall nicht jitterfrei bleibt.

Lemma 4.3.2. *Die Algorithmen 1 bzw. 2 sind nicht jitterfrei, wenn das Netzwerk eine geteilte Verbindung hat, bei der die eingehenden Pakete aus den s Ringen in einen Ausgangspuffer gespeichert werden.*

Beweis. Der Beweis erfolgt über den Widerspruch der Annahme, die Algorithmen sind jitterfrei für ein Netzwerk mit einer geteilten Verbindung, bei der die ankommenden Pakete in einen Ausgangspuffer gespeichert werden. Die Anzahl der Knoten für Ring w sei $n_w = n_{w-1} + 1$ für $w = \{1, \ldots, s-1\}$, wobei n_0 gegeben ist. Alle Pakete befinden sich im Ausgangspuffer des Knotens i. Die Reihenfolge der Pakete startet mit einem Paket aus Ring $s-1$ und in absteigender Reihenfolge enthält der Puffer jeweils 1 Paket aus einem Ring bis hinunter zu einem Paket aus Ring 0. Die weiteren Pakete im Puffer sind ebenfalls in dieser Reihenfolge angeordnet. Werden die ersten Pakete in dieser Reihenfolge von dem Knoten i gesendet, dann erreichen die Pakete gleichzeitig den Knoten i, nachdem sie ihren Ring vollständig durchlaufen haben. Die Pakete müssen dann nacheinander in den Ausgangspuffer des Knotens i geschrieben werden, wobei es zu einer Vertauschung der Paketreihenfolge kommen kann. Durch die Vertauschung der Paketreihenfolge erhöht sich die Wartezeit des Paketes, bis es von Knoten i übertragen

wird und damit auch die tatsächliche Ankunftszeit des Paketes. Im schlechtesten Fall führt die Vertauschung der Paketreihenfolge zu einem Jitter von mindestens $s-1$. Dieser schlechteste Fall tritt z. B. dann auf, wenn die Pakete mit dem *Round-Robin*-Verfahren nacheinander in den Ausgangspuffer geschrieben werden und als Erstes das Paket aus Ring $s-2$ in den Puffer geschrieben wird. □

Der Beweis von Lemma 4.3.2 hat gezeigt, dass der Jitter durch eine Vertauschung der Paketreihenfolge entsteht, wenn die Pakete aus den s Ringen in den Ausgangspuffer abgelegt werden. Neben dem im Beweis angeführten Fall lassen sich noch weitere Fälle finden, bei denen es zu einer Vertauschung der Paketreihenfolge kommt. Die Vertauschung der Paketreihenfolge lässt sich verhindern, wenn die Pakete in ihrer Sendereihenfolge nummeriert werden und sie dann in dieser Reihenfolge wieder in den Ausgangspuffer geschrieben werden. Im folgenden Lemma werden für diesen Fall die Paketumlaufzeit und die Jitterfreiheit untersucht.

Lemma 4.3.3. *Die Paketumlaufzeit für einen Ring w mit n_w Knoten in einem Netzwerk mit s Ringen, die sich eine Verbindung teilen, ist gleich der Summe der Knoten der s Ringe $n_\mathrm{p} = \sum_{w=1}^{s-1} n_w$, wenn der Knoten i die Pakete in der gleichen Reihenfolge, wie er die Pakete gesendet hat, in den einen Ausgangspuffer ablegt. Die Paketumlaufzeit ist konstant und damit ist das Self*S*-Protokoll in diesem Fall jitterfrei.*

Beweis. Ohne Beschränkung der Allgemeinheit wird angenommen, dass sich alle $n_\mathrm{p} = \sum_{w=1}^{s-1} n_w$ Pakete aus den s Ringen im Ausgangspuffer von Knoten i befinden. Das erste von Knoten i gesendete Paket benötigt n_w Zeitschritte, bis es wieder in den Ausgangspuffer von Knoten i gespeichert wird. Nach n_w Zeitschritten befinden sich noch $n_\mathrm{p} - n_w$ Pakete im Ausgangspuffer von Knoten i. Das Paket muss also $n - n_w$ Zeitschritte warten, bis es erneut von Knoten i gesendet wird. Die Paketumlaufzeit aller Pakete in einem beliebigen Ring w ist dann

$$T_{\mathrm{RTT}_w} = n_w + (n_\mathrm{p} - n_w) = n_\mathrm{p}. \tag{4.8}$$

Angesichts der festen Pufferverzögerungszeit von $n - n_w$ und der festen Übertragungszeit des Paketes über den Ring n_w handelt es sich auch bei der Paketumlaufzeit um einen konstanten Wert. Wie auch schon im Beweis zu Lemma 4.3.2 kann aus einer konstanten Paketumlaufzeit die Jitterfreiheit der Übertragung gefolgert werden. □

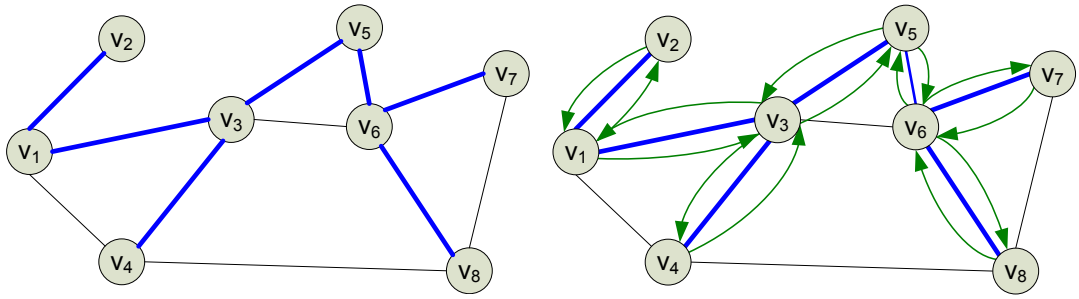
Die im Lemma 4.3.3 nachgewiesene Paketumlaufzeit von n_p Zeitschritten für Pakete eines Ringes in einem Netzwerk mit s Ringen, die sich eine Verbindung teilen, ist größer als die Paketumlaufzeit eines Netzwerkes mit nur einem Ring und der gleichen Anzahl von Knoten. In einem Netzwerk mit mehreren Ringen und einer geteilten Verbindung

werden mehr Pakete gesendet als in einem Netzwerk mit nur einem Ring. Grund dafür ist das Verhalten der beiden Knoten, zwischen denen die geteilte Verbindung besteht. Wie bereits erwähnt, senden diese beiden Knoten Pakete mit der eigenen Paket-ID in alle Ringe. Daher ist in einem Netzwerk mit n Knoten und s Ringen, die sich eine Verbindung teilen, die Gesamtanzahl der Pakete im Netzwerk $n_\mathrm{p} = n + 2 \cdot s$. Ein Netzwerk mit n Knoten und einer geteilten Verbindung überträgt also $2 \cdot s$ Pakete mehr als ein Netzwerk mit n Knoten, die alle nur in einem Ring angeordnet sind. Dieser Sachverhalt ist unabhängig davon, ob die Pakete aus den s Ringen in einen Ausgangspuffer oder in s Eingangspuffer geschrieben werden, bevor sie über die geteilte Verbindung weitergeleitet werden. Für den Fall, dass die Anzahl der Knoten in jedem Ring gleich groß sind und somit für einen Ring w die Anzahl der Knoten $n_w = n_\mathrm{p}/s$, folgt auch aus Gleichung (4.7) eine Paketumlaufzeit von $T_\mathrm{RTT} = n_\mathrm{p}$. Im Durchschnitt ist die Paketumlaufzeit auch für den in Lemma 4.3.1 beschriebenen Fall für ein Paket in einem der s Ringe größer als die Paketumlaufzeit eines Paketes in einem einzelnen Ring, der die n Knoten verbindet. Nur wenn die Bandbreite der geteilten Verbindung erhöht wird, sind kleinere Paketumlaufzeiten in jedem der s Ringe möglich.

4.3.2 In Bäume eingebettete Ringe

In den vorangegangenen Abschnitten wurden die Algorithmen beschrieben, die ein hartes Echtzeitverhalten innerhalb eines Ringnetzwerkes ermöglichen. Um solche Netzwerke aufzubauen, müssen die notwendigen Weiterleitungsmechanismen in die Knoten integriert werden. Dies beinhaltet zum einen den Umgang von Kreisen in einem Netzwerk und zum anderen den Umgang mit redundanten Pfaden, die durch die bidirektionale Verbindung verursacht werden. Standard LAN-Switches verwenden normalerweise keine redundanten Pfade und Kreise. Im Gegenteil, sie verwenden *Spanning Tree Protokolle*, um redundante Pfade und Kreise zu eliminieren und einen eindeutigen Pfad für die Kommunikation zu erhalten. Zur Erstellung eines Spannbaumes (siehe Abbildung 4.7(a)) in einem Netzwerk verwenden LAN-Switches die in der IEEE-Norm 802.1D-2004 [IEE04] beschriebenen Protokolle STP (Spanning Tree Protocol) oder RSTP (Rapid-Spanning Tree Protocol). Daher ist es nicht möglich, die in diesem Kapitel vorgestellten Algorithmen direkt anzuwenden, wenn das Netzwerk aus Standardkomponenten aufgebaut ist. Dazu muss der Ring als virtueller Ring in den Spannbaum eingebettet werden.

Einen Ring in einen Baum einzubetten entspricht einer Eulertour durch den Baum [BO03]. Eine Eulertour durch einen Graphen $G = (V,E)$ ist ein Kreis, der jede Kante nur einmal besucht. Ein gerichteter Graph besitzt eine Eulertour nur für den Fall, dass für alle Knoten aus V der Eingangsgrad von v gleich dem Ausgangsgrad von v ist [CLR00]. Ein Baum mit bidirektionalen Verbindungen zwischen den Knoten erfüllt



(a) Spannbaum in einem Netzwerk

(b) Einbetten eines Ringes als eine Euler-Tour durch einen Baum

Abbildung 4.7: Einbettung eines Ringes in eine allgemeine Netzwerktopologie

diese Bedingung, weil eine bidirektionale Verbindung zwei in unterschiedlichen Richtungen gerichteten Kanten entspricht. Infolgedessen durchläuft die Eulertour, wie in Abbildung 4.7(b) dargestellt, jede Verbindung des Baumes genau zweimal. Die Paketumlaufzeit T_{RTT} eines einzelnen Paketes erhöht sich daher von n Zeitschritten auf $2 \cdot (n-1)$ Zeitschritten. Weil die Verbindungen des Spannbaumes in beide Richtungen für den virtuellen Ring genutzt werden, kann der Echtzeitdatenverkehr den virtuellen Ring nur in einer Richtung durchlaufen. Dadurch verringert sich auch die vorhandene Bandbreite im Vergleich zur Ringtopologie um die Hälfte. Die Anzahl der Pakete, die dem virtuellen Ring pro Runde gesendet werden können, entspricht mit $2 \cdot (n-1)$ der Anzahl der Kanten im virtuellen Ring. Daher existieren Knoten in einem virtuellen Ring, die pro Runde mehr als ein Paket in eine Richtung senden können. Die Anzahl der Pakete, die ein Knoten i in einem virtuellen Ring pro Runde senden kann, ist abhängig von dem Ausgangsgrad von i.

Eine bessere Paketumlaufzeit in einem virtuellen Ring kann erzielt werden, wenn ein Hamiltonkreis in dem Netzwerk gefunden wird. Ein Hamiltonkreis eines Graphen $G = (V,E)$ ist ein einfacher Kreis, der alle Knoten aus V nur einmal enthält [Sed01]. Wird also ein Hamiltonkreis in einem Netzwerk mit n Knoten gefunden, entsteht ein virtueller Ring mit einer Paketumlaufzeit von n. Besitzen die Knoten untereinander eine bidirektionale Verbindung, können im Ring in beiden Richtungen Daten wieder ausgetauscht werden. Jedoch lässt sich nicht in jeder beliebigen Vernetzung von Knoten ein Hamiltonkreis finden und die Suche nach einem Hamiltonkreis ist NP-vollständig [CLR00]. Eine Eulertour lässt sich im Gegensatz zum Hamiltonkreis in linearer Zeit finden [Sed01]. Wie bereits erwähnt, kann in einem Spannbaum mit bidirektionalen Verbindungen eine Eulertour gefunden werden und ein Spannbaum kann bei einer beliebigen Vernetzung von Knoten in jedem Fall aufgebaut werden. Für beliebige Netzwerktopologien ist die Eulertour über einen Spannbaum zum Aufbau eines virtuellen Rings zu favorisieren. Ebenfalls spricht für die Eulertour die bereits erwähnte Tatsa-

che, dass Standard LAN-Switches bereits Protokolle einsetzen, um einen Spannbaum im Netzwerk aufzubauen.

4.4 Ereignisgesteuerte Nachrichten

Das in diesem Kapitel beschriebene Self*S*-Protokoll wurde speziell für eine periodische zeitgesteuerte Echtzeitkommunikation entworfen. In diesem Abschnitt werden zwei Erweiterungen vorgestellt, mit denen ereignisgesteuerte Nachrichten in das Self*S* integriert werden. Die eine Erweiterung ermöglicht eine ereignisgesteuerte Echtzeitkommunikation, ohne dabei den zeitgesteuerten Echtzeitdatenverkehr zu stören. Die zweite Erweiterung benutzt niederpriorisierte Zeitschlitze, um die Latenz der ereignisgesteuerten Nachrichten zu verringern. Beide Erweiterungen bauen auf das Self*S*-Protokoll mit variablen Zykluszeiten auf, welches in Kapitel 4.2.2 beschrieben wird.

Bei der ersten Erweiterung kann von einem Knoten i eine ereignisgesteuerte Nachricht gesendet werden, wenn das zuletzt empfangene Paket die Paket-ID von i besitzt und in diesem Zeitschlitz kein Echtzeitpaket gesendet werden soll. Soll in diesem Zeitschlitz auch kein ereignisgesteuertes Paket gesendet werden, dann wird, wie in Algorithmus 2 beschrieben, ein NRT-Paket oder ein leeres Paket von i gesendet. Im Gegensatz zu einem NRT-Paket markiert Knoten i ein ereignisgesteuertes Paket als Echtzeitpaket, um Verzögerungen in den NRT-Warteschlangen der anderen Knoten zu vermeiden. Die Übertragungszeit eines ereignisgesteuerten Paketes von Knoten i zu einem bestimmten Zielknoten ist somit immer konstant und genauso schnell wie die Übertragungszeit eines periodischen Echtzeitpaketes.

Allerdings ist die Zeit vom Eintreffen des Ereignisses bis zur Übertragung des Paketes zum Zielknoten im schlechtesten Fall deutlich höher. Die obere Schranke für diese Zeit liegt im schlechtesten Fall bei $(3 \cdot n - 2) \cdot T_{\mathrm{Trt}}$. Im schlechtesten Fall entsteht das Ereignis direkt nachdem der Knoten i ein Paket mit der eigenen Paket-ID gesendet hat und i in der nächsten Runde ein periodisches Echtzeitpaket sendet. Das ereignisgesteuerte Paket muss also $n - 1$ Zeitschlitze auf den Umlauf des ersten Paketes und weitere n Zeitschlitze auf den Umlauf des periodischen Echtzeitpaketes warten, bevor der Knoten i das ereignisgesteuerte Paket senden kann. Des Weiteren ist der Zielknoten im schlechtesten Fall der Knoten, der am weitesten von i entfernt ist. Das ereignisgesteuerte Paket benötigt dann noch $n - 1$ Zeitschlitze, um den Zielknoten zu erreichen.

Mit der zweiten Erweiterung lässt sich die hohe Latenz der ereignisgesteuerten Nachricht reduzieren, indem die ereignisgesteuerte Nachricht des Knotens i eine höhere Priorität als die periodische Echtzeitnachricht erhält. Durch die höhere Priorität kann die ereignisgesteuerte Nachricht auch in dem Zeitschlitz der periodischen Echtzeitnach-

richt gesendet werden, und die Latenz kann um n Zeitschlitze reduziert werden. Im schlechtesten Fall beträgt die Latenz bei der zweiten Erweiterung noch $2 \cdot (n-1) \cdot T_{\mathrm{Trt}}$. Diese Verbesserung der Latenz kann allerdings zu einem Verlust eines periodischen Echtzeitpaketes führen. Jedoch ist es bei Echtzeitsystemen nicht unüblich, dass eine wichtige ereignisgesteuerte Nachricht, wie z. B. ein Notfallstopp des Systems, eine höhere Priorität erhält. Auch der Verlust eines anderen Paketes kann vom System toleriert werden, wenn das System trotzdem ohne Schaden weiter arbeitet. Eine Priorisierung von zyklischen oder von ereignisgesteuerten Nachrichten kann durch den Benutzer im Vorfeld festgelegt werden oder durch eine EDF-Ablaufplanung erfolgen. Die EDF-Ablaufplanung wählt die Nachricht mit der nächsten Zeitschranke für die Übertragung aus.

4.5 Zusammenfassung

Das Self*S*-Protokoll ermöglicht eine Variation der Zykluszeit jedes einzelnen Knotens während des Betriebs, ohne die Zykluszeiten anderer Knoten zu beeinflussen. Das Protokoll ist trotz seiner Einfachheit in der Lage, harte Echtzeitanforderungen zu erfüllen. Dazu wird bei Self*S* weder ein Master noch eine explizite Uhrensynchronisation benötigt. Der simultane Start der Paketübertragung erfolgt beim Self*S*-Protokoll durch eine Selbstsynchronisation. Obwohl Self*S* zeitgesteuert ist, können auch ereignisgesteuerte Nachrichten integriert werden.

Das Self*S*-Protokoll und dessen Prozess der Selbstsynchronisation wurden mithilfe eines theoretischen Modells für eine Ringtopologie analysiert. Für den Prozess der Selbstsynchronisation konnte mithilfe des Modells eine obere Schranke für die maximale Synchronisationszeit nachgewiesen werden. Ebenfalls konnte mithilfe des Modells die Jitterfreiheit des Protokolls bei unterschiedlichen Zykluszeiten für jeden einzelnen Knoten unter der Voraussetzung gezeigt werden, dass alle Zykluszeiten ein beliebiges Vielfaches der Paketumlaufzeit sind. Die Einstellung der Zykluszeit als ein Vielfaches der Paketumlaufzeit ist ideal für digitale Regelungen, weil verschiedene Abtastraten bei einer digitalen Regelung häufig auch als ein Vielfaches einer anderen Abtastrate gewählt werden. Bei größeren Zykluszeiten geben die Knoten die von ihnen nicht benötigte Bandbreite frei. Diese wird bei Self*S* für die Übertragung von NRT-Paketen verwendet. Dabei werden Echtzeitanforderungen des Echtzeitdatenverkehrs vom NRT-Datenverkehr nicht beeinflusst.

Völlig frei gewählte Zykluszeiten würden die Selbstsynchronisation stören und zu einem hohen Jitter führen. Nicht nur das Self*S*-Protokoll würde bei beliebigen Zykluszeiten einen hohen Jitter verursachen. Es konnte gezeigt werden, dass selbst ein für die Ringtopologie optimales Protokoll einen hohen Jitter bei beliebigen Zykluszeiten nicht verhindern kann.

Neben einem einfachen Ring kann das Self*S*-Protokoll auch in mehreren Ringen eingesetzt werden, die sich eine Verbindung teilen. Eine Jitterfreiheit kann das Protokoll in diesem Fall unter der Voraussetzung garantieren, dass der Knoten, an dem die Pakete aus mehreren Ringen ankommen, die Pakete aus jedem Ring in einen separaten Eingangspuffer ablegt. Werden alle ankommenden Pakete in einen gemeinsamen Ausgangspuffer geschrieben, kann ein Jitter entstehen.

Auch in einer beliebigen Topologie lässt sich das Self*S*-Protokoll anwenden, indem ein virtueller Ring in die Topologie eingebettet wird. Dazu muss zunächst ein Spannbaum in der beliebigen Topologie gebildet werden. Zum Erstellen des Spannbaums kann auf Verfahren zurückgegriffen werden, die in Standard-LANs verwendet werden. Mithilfe einer Euler-Tour kann in jeden Baum ein Ring gelegt werden. Durch dieses Verfahren können die im LAN-Bereich verwendeten Verfahren für die Spannbaumbildung bei Self*S* wieder verwendet werden. Einziger Nachteil ist bei der Euler-Tour eine beinahe Verdopplung der Paketumlaufzeit. Allerdings sind andere Algorithmen zur Suche eines Ringes wie z. B. der Hamiltonkreis NP-vollständig.

Die Jitterfreiheit des Self*S*-Protokolls, die in diesem Kapitel nachgewiesen wurde, bezieht sich nur auf das Protokoll. Es wurde untersucht, ob das Protokoll oder die Selbstsynchronisation einen Jitter verursacht. Der Jitter, der z. B. in einem Switch entsteht, wurde an dieser Stelle nicht mit berücksichtigt. Der Einfluss dieser switchspezifischen Eigenschaften auf das Self*S*-Protokoll wird im nächsten Kapitel untersucht.

Kapitel 5

Evaluierung einer Self*S*-Portierung auf Ethernet

Im Bereich der Echtzeitkommunikation bildet Ethernet den Standard, auf dem die meisten modernen Echtzeitprotokolle aufbauen (vgl. Kapitel 2.4 und 2.5). In diesem Kapitel wird mithilfe einer Netzwerksimulation untersucht, ob sich der in Kapitel 4 vorgestellte Ansatz eines selbstsynchronisierenden Echtzeitprotokolls auf Ethernet portieren lässt. Mit der computergestützten Netzwerksimulation können Netzwerkmodelle aufgebaut und durch die Simulation analysiert werden. Durch einen möglichst detailgetreuen Aufbau des Modells, der anschließenden Simulation und der Auswertung der Ergebnisse können die Korrektheit und die Leistungsfähigkeit eines Netzwerkes, einer Netzwerkkomponente oder eines Netzwerkprotokolls bereits evaluiert werden, bevor das System tatsächlich implementiert wird. Im Gegensatz zum mathematischen Modell können komplexere Modelle mit einem höheren Detaillierungsgrad leichter untersucht werden. Des Weiteren erhält der Entwickler bei der Netzwerksimulation bereits einen Einblick in das praktische Verhalten des Systems. Gegenüber einer prototypischen Implementierung des Netzwerkes kann bei der Simulation das Verhalten der Netzwerkkomponenten und des Protokolls schrittweise beobachtet werden und ist dadurch auch besser zu verstehen.

Die Modellierung, die Simulation und die Auswertung der Ergebnisse erfolgen mithilfe eines Netzwerksimulators. Ein verbreiteter frei verfügbarer Netzwerksimulator ist der *Network Simulator v.2 (ns-2)* [BEF+00a], der von der *University of Southern California* bereitgestellt wird. Der Netzwerksimulator ns-2 ist ein objektorientierter diskreter Ereignissimulator. In der diskreten Ereignissimulation werden nur die Statusveränderungen des Systems betrachtet. Typische Statusveränderungen in der Netzwerksimulation sind z. B. der Beginn und das Ende der Übertragung eines Paketes. Ein bekannter kommerzieller ereignisbasierter Netzwerksimulator ist der OPNET Modeller [OPN08], der ursprünglich am MIT entwickelt wurde. OPNET wird insbesondere in der Industrie zur Netzwerkmodellierung eingesetzt.

In dieser Arbeit wird zur Netzwerksimulation der frei verfügbare Simulator OMNeT++ (Object Modular Network Test bed in C++) [Var01] verwendet. OMNeT++ ist ein objektorientierter modulbasierter diskreter Ereignissimulator. Die für OMNet++ zur

Verfügung gestellte INET-Bibliothek beinhaltet bereits ein Basismodell für Ethernet, welches die Modellierung von Self*S* erleichtert. Der modulare Aufbau der Modelle ermöglicht es, die hierarchische Struktur des tatsächlichen Systems nachzubilden. OMNeT++ ist ebenfalls eine weit verbreitete Plattform zur Simulation von Netzwerkprotokollen in Forschung und Lehre und wird auch zum Teil in der Industrie eingesetzt. Unter anderem wurden z. B. mit OMNeT++ die Echtzeiteigenschaften des zeitgesteuerten PROFINET IRT untersucht [JE04].

Vor der Netzwerksimulation werden in Kapitel 5.1 zunächst die für die Portierung erforderlichen Anpassungen des Protokolls beschrieben. Aufbauend auf diesen Anpassungen wird in Kapitel 5.2 ein Simulationsmodell eines Ethernet-basierten Self*S*-Netzwerkes vorgestellt, mit dessen Hilfe die Selbstsynchronisation in Kapitel 5.3 und die Echtzeiteigenschaften des Protokolls in Kapitel 5.4 evaluiert werden. Abhängig sind die Echtzeiteigenschaften des Self*S*-Protokolls von der Switchimplementierung. In Kapitel 5.5 werden die Echtzeiteigenschaften des Self*S*-Protokolls für den in Kapitel 3.2 vorgestellten Hardware- und Software-Switch miteinander verglichen. Mithilfe eines experimentellen Aufbaus eines Self*S*-Netzwerkes werden in Kapitel 5.6 die Echtzeiteigenschaften und die Selbstsynchronisation in einem realen Netzwerk untersucht und den Echtzeiteigenschaften aus der Netzwerksimulation gegenübergestellt. Die Leistungsfähigkeit eines Ethernet-basierten Self*S*-Protokolls wird in Kapitel 5.7 mit anderen Ethernet-basierten Echtzeitprotokollen verglichen. Schließlich werden in Kapitel 5.8 noch einige Erweiterungen des Ethernet-basierten Self*S*-Protokolls vorgestellt. Die Erweiterung befasst sich mit der Variation der Basiszykluszeit und der Fehlertoleranz.

5.1 Portierung auf Ethernet

In Kapitel 2.5 sind mehrere Echtzeitprotokolle beschrieben worden, die als Basis den Ethernet-Standard verwenden. Auch das Self*S*-Protokoll lässt sich auf Ethernet portieren. Dazu ist es notwendig, das Self*S*-Protokoll, wie in Kapitel 4 beschrieben, an die Ethernet-spezifischen Anforderungen, wie z. B. der definierten IFG, anzupassen. Ebenfalls müssen die von Self*S* benötigten Protokollinformationen, die sich nicht in einem Ethernet-Paket befinden, in das Paketformat eingebettet werden.

5.1.1 Paketformat

Pakete des Self*S*-Protokolls benötigten, wie bereits in Kapitel 4.1 erläutert, eine Zieladresse, eine Quelladresse, eine Paket-ID und ein Typfeld. Für die Ziel- und die Quelladresse können die Adressfelder für Ziel und Quelle des Ethernet-Paketes verwendet werden. Nur für die Paket-ID und das Typfeld wird ein insgesamt 2 Byte Self*S*-Paketkopf in den Nutzdatenteil des Ethernet-Paketes eingebettet (Abbildung 5.1).

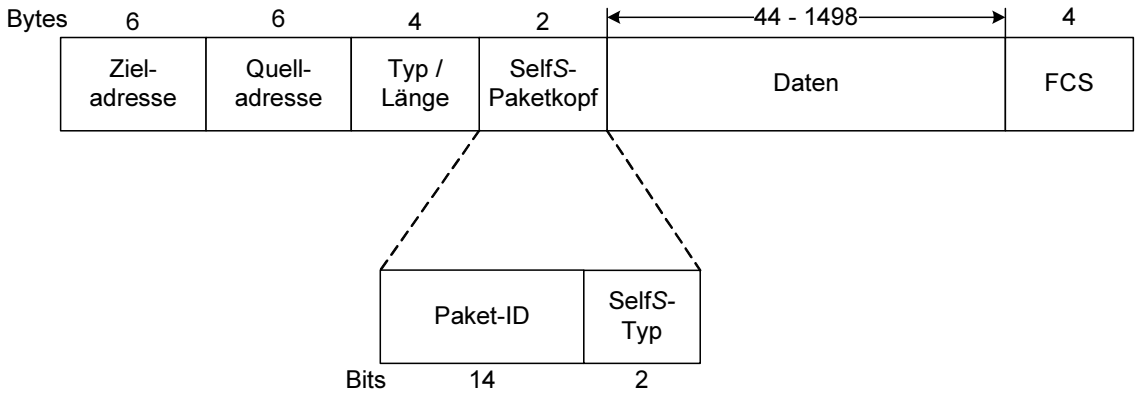


Abbildung 5.1: Self*S*-Ethernet-Paketformat

Das Typfeld des Self*S*-Paketkopfes wird als Self*S*-Typ gekennzeichnet, um es vom Ethernet-Typfeld zu unterscheiden.

Die Bedeutung der Felder eines Self*S*-Ethernet-Rahmens wird wie folgt zusammengefasst:

- **Zieladresse** ist die Adresse des Knotens, für den die Nutzdaten des Paketes bestimmt sind.
- **Quelladresse** ist die Adresse des Knotens, der die Nutzdaten generiert hat.
- **Typ/Länge** beschreibt den Typ oder die Länge der gültigen Nutzdaten des Ethernet-Rahmens. Im Falle eines RT-Paketes sollte immer die Länge der gültigen Nutzdaten des Ethernet-Paketes, d.h. einschließlich des Self*S*-Paketkopfes, angegeben werden.
- **Paket-ID** identifiziert den Knoten, der das Paket jeder neuen Runde generiert und erneut sendet. Dieses Feld hat eine Breite von 14 Bit, womit sich 16.384 Knoten unterscheiden lassen.
- **Self*S*-Typ** kennzeichnet ein Paket als RT-, NRT- oder als ein leeres Paket.

Besteht das Netzwerk aus mehreren Ringen, die sich eine Verbindung teilen, wird der Self*S*-Paketkopf in ein VLAN-Ethernet-Paket (Abbildung 2.10) eingefügt. In diesem Fall wird jedem Ring eine VLAN-Kennung zugeordnet. Alle Pakete, die zu einem Ring gehören, tragen die VLAN-Kennung des Ringes und können so von einem Switch in den richtigen Ring weitergeleitet werden. Das im VLAN-Standard IEEE 802.1Q spezifizierte Verfahren muss dazu nicht angepasst werden.

5.1.2 Erweiterung des Switches

Bei einem Software-Switch, wie z. B. dem in Kapitel 3.2.1 beschriebenen Switch, muss nur die Programmierung des Switches verändert werden, damit der Switch die Self*S*-Pakete korrekt weiterleitet. Eine Softwarelösung hat aber im Gegensatz zu einer Hardwarelösung höhere Verarbeitungszeiten und einen höheren Jitter. Für harte Echtzeitanforderungen empfiehlt sich der Einsatz eines Hardware-Switches. Die Abbildung 5.2 zeigt den für das Self*S*-Protokoll notwendigen erweiterten Aufbau einer Teilswitch-Komponente des in Kapitel 3.2.2 beschriebenen Hardware-Switches.

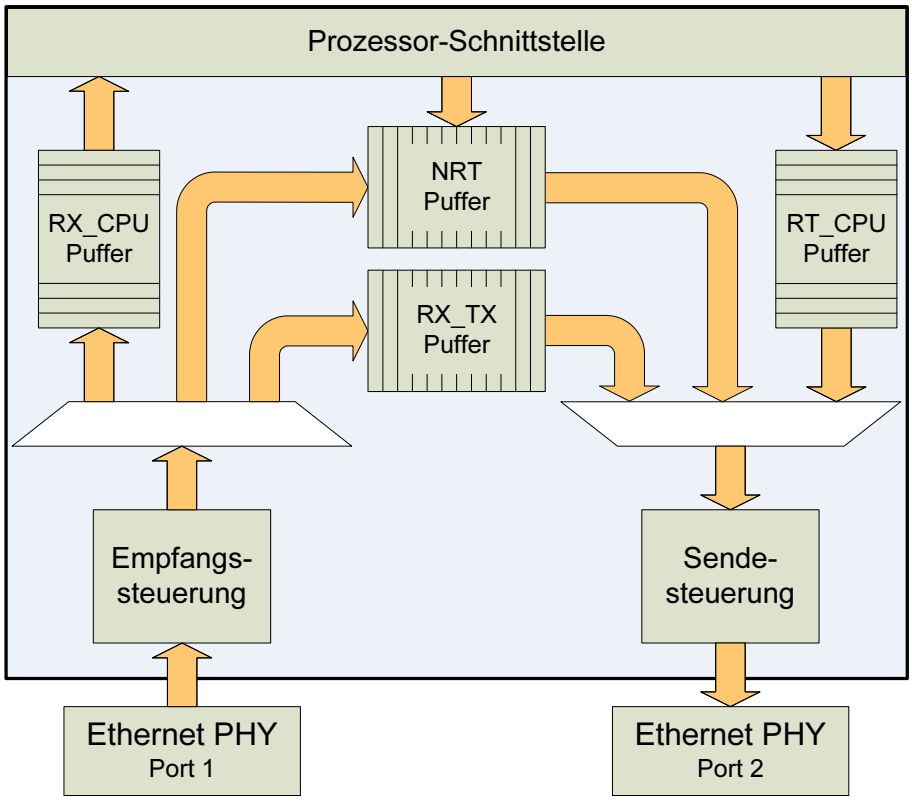


Abbildung 5.2: Blockdiagramm eines Self*S*-Teilswitches

Die wesentliche Veränderung im Aufbau des erweiterten Teilswitches ist der zusätzliche NRT-Puffer, in den alle ankommenden NRT-Pakete abgelegt werden. Alle anderen Puffer sind im Aufbau des Switches erhalten geblieben. Aber auch die Funktion der Puffer hat sich zum Teil verändert. Der ehemalige TX_CPU-Puffer (Abbildung 3.6), jetzt umbenannt in RT_CPU-Puffer, enthält nur noch die vom Prozessor zu versendenden Echtzeitdaten. Alle NRT-Daten vom Prozessor werden genau wie die ankommenden NRT-Pakete in den NRT-Puffer geschrieben. Nur die Funktion des RX_CPU-Puffers ist gleich geblieben. In diesen Puffer werden die Pakete abgelegt, die vom Prozessor weiter verarbeitet werden. Alle Puffer arbeiten nach dem FIFO-Prinzip (First In First

Out), d.h., das Paket, das als Erstes in den Speicher geschrieben wurde, wird auch als Erstes wieder gelesen.

Nicht nur der Aufbau des Switches, sondern vor allem die Paketverarbeitung des Switches muss angepasst werden. Generell muss der Switch weiterhin zwischen Paketen unterscheiden, die weitergeleitet werden müssen und die an den Prozessor gerichtet sind. Zusätzlich muss der Switch die drei Self*S*-Pakettypen RT, NRT und leeres Paket unterscheiden.

Pakete weiterleiten: Vom Switch weitergeleitet werden die RT- und die NRT-Pakete, die nicht die Zieladresse des Prozessors tragen. RT-Pakete werden vom Switch direkt in den RX_TX-Puffer geschrieben. Sobald das Paket vollständig in den Puffer geschrieben ist, wird es über den Ausgangsport versandt. Im Gegensatz dazu werden die Ziel- und die Quelladresse, das Typ/Längen-Feld sowie die Nutzdaten eines NRT-Paketes in den NRT-Puffer geschrieben. Die Paket-ID wird direkt an die Sendersteuerung weitergegeben, die die Paket-ID in das erste Paket aus dem NRT-Puffer einfügt und dann das Paket überträgt. Erreicht ein leeres Paket den Switch, wird ebenfalls die Paket-ID direkt an die Sendesteuerung übergeben. Die übrigen Daten werden verworfen. Ist der NRT-Puffer leer, überträgt die Sendesteuerung ein neues leeres Paket mit der erhaltenen Paket-ID. Befinden sich noch Pakete im NRT-Puffer, wird in das erste Paket aus dem Sendepuffer die Paket-ID eingefügt und das Paket wird übertragen. Eine Ausnahme sind Pakete, die die Paket-ID des Knotens haben. Enthält dieses Paket NRT-Daten, werden diese ebenfalls in den NRT-Puffer geschrieben. In Abhängigkeit der eingestellten Zykluszeit wird ein neues RT-Paket aus dem CPU_RT-Puffer übertragen (vgl. Kapitel 4.2.2). Soll in der Runde kein RT-Paket gesendet werden, wird ein NRT- bzw. ein leeres Paket übertragen.

Pakete empfangen und senden: Alle Pakete, die im Zieladressfeld den Prozessor des Knotens adressieren, werden in dem RX_CPU-Puffer gespeichert und vom Prozessor weiter verarbeitet. Ebenfalls wird die Paket-ID des empfangenen Paketes an die Sendersteuerung übergeben. Für den Fall, dass die Paket-ID nicht mit der ID des Knotens übereinstimmt, wird auch bei einem empfangenen NRT-Paket ein NRT-Paket bzw. ein leeres Paket mit der Paket-ID des empfangenen Paketes übertragen. Wird im selben Fall ein RT-Paket empfangen, dann kann nur an den Knoten, von dem die Paket-ID stammt, ein RT-Paket zurückgesendet werden. Dieses RT-Antwort-Paket trägt natürlich auch dieselbe Paket-ID wie das empfangene Paket. Auf diese Weise kann ein Regler innerhalb einer Runde die Soll-Werte zu einem Gerät senden und die Ist-Werte dieses Gerätes empfangen. Soll kein RT-Antwort-Paket übertragen werden, dann wird ein NRT-Paket bzw. ein leeres Paket gesendet. Pakete mit der Paket-ID des eigenen Knotens werden, wie oben beschrieben, verarbeitet.

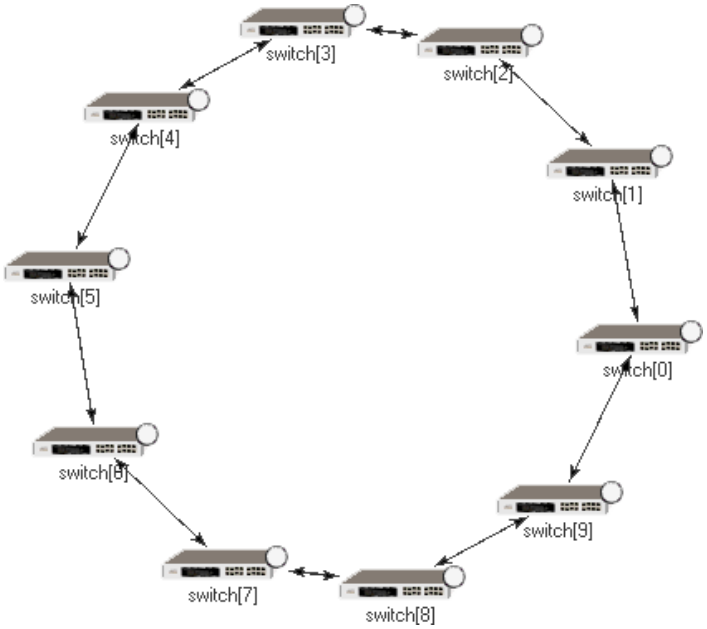


Abbildung 5.3: OMNeT++-Darstellung eines Netzwerkes mit 10 Knoten

5.1.3 Synchronisationsstatus

In Kapitel 4.2.1 wird beschrieben, dass ein Knoten die Synchronisation erkennt, wenn er alle Pakete innerhalb einer Runde ohne eine Lücke überträgt. Bei Ethernet existiert kein lückenfreier Transport von Paketen. Die IFG und auch der Jitter der Ethernet-Ports sorgen dafür, dass zwischen jedem übertragenen Paket eine Lücke eingefügt wird. In diesem Fall müssen nur die Lücken erkannt werden, die länger als die Summe aus IFG und dem Jitter der Ethernet-Ports sind. Wenn innerhalb der Paketumlaufzeit des eigenen Paketes keine Lücke detektiert wird, die größer als IFG und Portjitter ist, kann der Synchronisationsstatus auf *Synchronisiert* gesetzt werden. Die Erkennung des Synchronisationsstatus entspricht generell den in Kapitel 4.2 beschriebenen Algorithmen 1 und 2. Lediglich die Detektierung der Paketlücke muss, wie oben beschrieben, angepasst werden.

5.2 Simulationsmodell

Der allgemeine Ansatz des selbstsynchronisierenden Protokolls wurde in Kapitel 4 mithilfe eines mathematischen Modells beschrieben und analysiert. Ethernet-spezifische Protokolleigenschaften, wie z. B. die definierte Paketlücke IFG und gerätespezifische Eigenschaften, wie z. B. die Verarbeitungszeit des Switches und der durch die physikalischen Ethernet-Ports verursachte Jitter wurden dabei nicht betrachtet. Diese Ei-

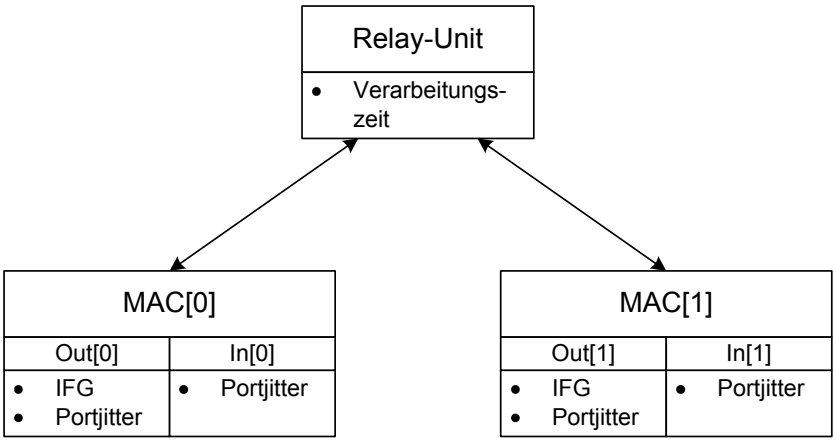


Abbildung 5.4: Modell der Switch-Architektur

genschaften werden an dieser Stelle für die Evaluierung der Selbstsynchronisation, der Zykluszeit und des Jitters des Ethernet-basierten Self*S*-Protokolls mit berücksichtigt.

Im mathematischen Modell wird ein Netzwerk aus n Knoten durch einen Graphen $G = (V,E)$ mit V der Menge der n Knoten und E der Menge der m Kanten repräsentiert. Ein Netzwerk im Simulationsmodell wird aus n Switches und m Verbindungen zwischen den Switches aufgebaut. Die Verbindungen und Switches bilden, wie in Abbildung 5.3 dargestellt, zusammen einen Ring, in dem jeder Switch ein Paket zu seinem direkten Nachbarn senden kann. Die Pakete haben das in Abbildung 5.1 dargestellte Format und, falls nicht anders angegeben, eine minimale Länge von $l = 72$ Bytes.

5.2.1 Modell der Switch-Architektur

Das Simulationsmodell der Switch-Architektur wird in Abbildung 5.4 dargestellt. Das Modell wurde von der prototypischen Umsetzung des Hardware-Switches aus Kapitel 3.2.2 abgeleitet und besteht aus einer *Relay-Unit* und zwei MACs. In der Relay-Unit erfolgt die Verarbeitung und die Weiterleitung der Pakete nach dem Store-and-Forward-Prinzip mit einer festen Verarbeitungszeit T_{Swi} von 280 ns. Beim Self*S*-Protokoll kann der in Kapitel 3.3.2 beschriebene Jitter der Verarbeitungszeit des Hardware-Switches von 40 ns nicht auftreten. In diesem Fall werden die Sendepuffer des Prozessors und die Sendepuffer zur Weiterleitung der Pakete nicht sequenziell taktweise abgefragt, sondern der Status des empfangenen Self*S*-Paketes bestimmt, aus welchem Puffer ein Paket übertragen wird.

Sowohl die Sendeports (TX) als auch die Empfangsports (RX) der beiden MACs verursachen einen Portjitter J_{Port} von jeweils 40 ns. In dem realen System haben die MACs und die physikalischen Ethernet-Schnittstellen einen voneinander unabhängigen Takt von 25 MHz bei einer Datenrate von 100 MBit/s. Die Taktflanken von den MACs und

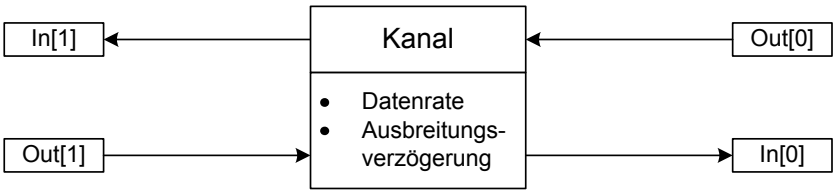


Abbildung 5.5: Modell des Verbindungskanals

den physikalischen Schnittstellen sind in diesem Fall um maximal 40 ns verschoben, wodurch beim Senden und Empfangen der Portjitter entsteht. Messungen an der prototypischen Umsetzung des Hardware-Switches mit dem Oszilloskop bestätigen den Portjitter von 40 ns. Der Portjitter wird im Modell in den Sende- und Empfangsports der MACs durch eine Zufallszahl z im Intervall $0 \leq z \leq 40 \cdot 10^{-9}$ bestimmt. In den Berechnungen wird für den Portjitter J_{Port} immer sein Maximalwert von 40 ns verwendet. Die Sendeports fügen zusätzlich zwischen den Paketen, die direkt hintereinander übertragen werden, die im Ethernet-Standard definierte Paketlücke IFG von 0,96 μs ein. Nach Ablauf der Zeit einer IFG wird noch der Portjitter abgewartet, bevor das Switch-Modell mit der Übertragung des Paketes beginnt.

Der Synchronisationsstatus wird im Modell nach dem in Kapitel 5.1.3 beschriebenen Anpassungen des Algorithmus 1 ermittelt. Dazu detektiert der Switch im Paketstrom die Lücken, die größer sind als die erlaubte Paketlücke. Als erlaubte Paketlücke wird in der Simulation ein Wert von 1,04 μs gewählt. Dies entspricht der Paketlücke, die vom Switch durch die IFG und den Portjitter verursacht wird. Nur wenn der Switch innerhalb der Paketumlaufzeit des eigenen Paketes keine Lücke erkennt, wird der Status des Switches auf *Synchronisiert* gesetzt.

Der Startzeitpunkt des ersten Paketes, das der Switch sendet, wird durch eine Zufallszahl z bestimmt. Für den Startzeitpunkt liegt z bei den meisten Simulationen in einem Intervall zwischen 0 und 2 ms. Eine Intervallobergrenze von 2 ms ist ausreichend, um alle möglichen Startreihenfolgen in einem Netzwerk von über 200 Knoten zu simulieren. Wesentlich größere Intervallobergrenzen führen nur zu einer unnötigen Erhöhung der Simulationszeit.

5.2.2 Modell des Verbindungskanals

Die Vollduplexverbindung wird, wie in Abbildung 5.5 dargestellt, durch zwei unidirektionale Verbindungen repräsentiert, die beide mit dem Übertragungskanal verbunden sind. Der Kanal erlaubt Datenraten von 10 MBit/s, 100 MBit/s und 1 GBit/s. Für die Simulation wird eine Datenrate von 100 MBit/s gewählt. Die Ausbreitungsverzögerung T_{Prd} des Kanals beträgt, falls nicht anders angegeben, 556 ns und entspricht damit der

Modellparameter	Erläuterung	Wert
numOfNodes	Anzahl der Knoten im Netzwerk	n
paketLength	Länge l eines Ethernet-Paketes	72 Byte
processingTime	Verarbeitungszeit des Switches T_{Swi}	280 ns
IFG	Inter Frame Gap	0,96 μs
portJitter	Der Jitter der Ethernet-Port J_{Port}	Zufallszahl z für $0 \leq z \leq 40$ ns
startTime	Zeitpunkt der Übertragung des ersten eigenen Paketes	Zufallszahl z für $0 \leq z \leq 2$ ms
allowedGapTime	Erlaubte Paketlücke zur Detektierung des Synchronisationsstatus	1,04 μs
delay	Ausbreitungsverzögerung über ein Ethernet-Kabel T_{Prd}	556 ns
datarate	Datenrate r	100 Mbit/s

Tabelle 5.1: Parameter des Simulationsmodells

Verzögerung über ein Ethernet-Twistet-Pair-Kabel mit einer Länge von 100 Metern. Eine größere Kabellänge ist im Ethernet-Standard IEEE 802.3 [IEE05] nicht erlaubt.

5.2.3 Zusammenfassung der Modellparameter

Die Tabelle 5.1 listet noch einmal alle Modellparameter auf. Die folgenden Simulationen beruhen auf diesen Parametern, solange keine anderen Angaben gemacht werden. Ethernet-Pakete werden in der Simulation immer mit einer minimalen Länge l von 72 Byte übertragen, weil diese für die meisten Echtzeitanwendungen ausreichend ist. Der Einfluss der Paketlänge auf einen Store-and-Forward-Switch wurde bereits in Kapitel 3.3.2 diskutiert.

5.3 Selbstsynchronisation

Mithilfe des zuvor beschriebenen Simulationsmodells wird zunächst der Einfluss der Ethernet- und gerätespezifischen Eigenschaften, wie IFG, Ausbreitungsverzögerung, Verarbeitungszeit und Jitter, auf die Selbstsynchronisation untersucht. Dazu wird der Startzeitpunkt, in dem ein Knoten sein erstes Paket sendet, für jeden Knoten zufällig bestimmt. Ein Netzwerk gilt als synchronisiert, nachdem der letzte Knoten innerhalb der Paketumlaufzeit seines eigenen Paketes keine Lücke erkannt hat, die größer als 1,04 μs ist.

Die Simulationsergebnisse für die Selbstsynchronisation in einem Self*S*-Netzwerk mit vier Switches wird in der Abbildung 5.6 skizziert. Dargestellt wird in dieser Abbildung der Paketstrom über die vier Verbindungen zwischen den vier Switches. Die Nummern

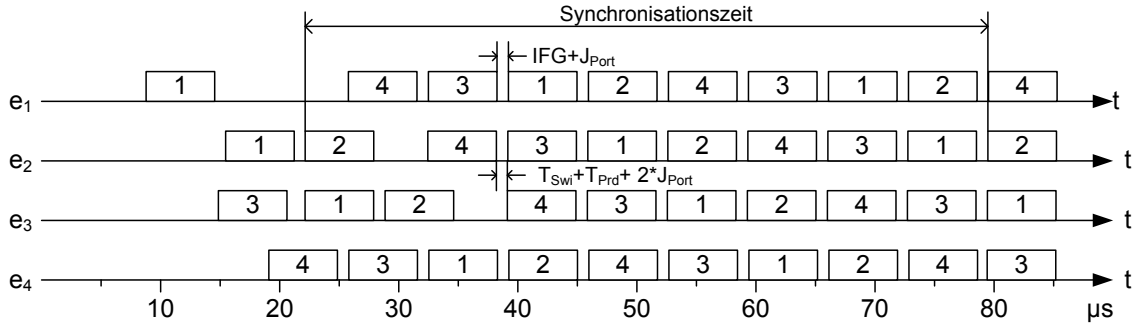


Abbildung 5.6: Selbstsynchronisation im Ethernet-Modell ($T_{\mathrm{Prd}} = 556\,\mathrm{ns}$)

in den Paketen stehen für die Paket-ID des Paketes. Das Simulationsbeispiel mit einem Netzwerk von vier Knoten ist an die in Kapitel 4.2.1 dargestellten Beispiele der Selbstsynchronisation der Abbildungen 4.2 und 4.3 angelehnt.

In Abbildung 5.6 ist, wie auch in den Beispielen in Kapitel 4.2.1, zunächst der asynchrone Start der Teilnehmer zu erkennen, bei dem der Teilnehmer 1 nach 8 μs das erste Paket sendet und der Teilnehmer 2 als Letzter nach 22 μs sein erstes Paket überträgt. Das Intervall für die zufällig ermittelten Startzeitpunkte wurde in diesem Simulationsbeispiel auf 20 μs reduziert, um die Selbstsynchronisation von Beginn an skizzieren zu können. Der zufällig generierte Startzeitpunkt für den Teilnehmer 2 liegt tatsächlich bei 19 μs und damit noch innerhalb des Intervalls. Allerdings beginnt die Übertragung dieses Paketes erst nach 22 μs, weil der Teilnehmer 2 die Übertragung von dem Paket mit der ID 1 erst abschließen muss. Auch in dem Simulationsbeispiel erfolgt eine Aneinanderreihung der Pakete. Im Gegensatz zu den theoretischen Beispielen bleibt im Paketstrom die durch die IFG und den Portjitter verursachte Lücke erhalten. Die Aneinanderreihung führt auch bei Ethernet zu einer Synchronisation des Netzwerks, die in diesem Beispiel als Letztes von Switch 2 erkannt wird. Die Synchronisationszeit beträgt in diesem Beispiel 59 μs.

Ein weiterer Unterschied zu den theoretischen Beispielen ist der Abstand zwischen dem Abschluss der Übertragung am Vorgängerknoten und dem Beginn der erneuten Übertragung, der durch die Ausbreitungsverzögerung T_{Prd}, die Verarbeitungszeit T_{Swi} und den doppelten Portjitter J_{Port} entsteht. In dem Beispiel aus Abbildung 5.6 beträgt dieser Abstand 916 ns und ist damit nicht viel geringer als die Paketlücken im Paketstrom. Daher entsteht in dieser Abbildung der Eindruck, alle Knoten würden gleichzeitig mit der Übertragung eines Paketes beginnen. Wird dieser Abstand, z. B. durch eine kleinere Ausbreitungsverzögerung, verringert, bildet sich eine Verschiebung der Paketströme heraus, die in der Abbildung 5.7 verdeutlicht wird. Für das in Abbildung 5.7 dargestellte Simulationsbeispiel wurde die Ausbreitungsverzögerung auf 50 ns reduziert. Die restlichen Parameter einschließlich der Startzeiten sind mit dem Beispiel aus Abbildung 5.6 identisch.

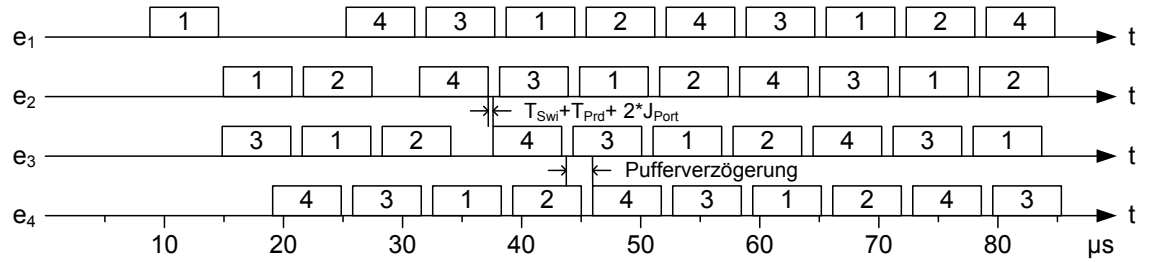


Abbildung 5.7: Selbstsynchronisation im Ethernet-Modell ($T_{\mathrm{Prd}} = 50\,\mathrm{ns}$)

Durch die Verschiebung der Paketströme beginnen die Switches nicht gleichzeitig mit der Übertragung eines Paketes, wodurch während des Umlaufs eines Paketes durch den Ring Pufferwartezeiten entstehen. In dem Simulationsbeispiel aus Abbildung 5.7 entstehen bei dem Switch 4 Pufferverzögerungen, weil jedes Paket auf die Beendigung der Übertragung seines Vorgängers warten muss. Pufferverzögerungen können den Determinismus und die Echtzeitfähigkeit des Protokolls negativ beeinflussen, wenn sie eine zufällige Größe haben und daher nicht vorhersagbar sind. Allerdings ist dies bei der Selbstsynchronisation über Ethernet nicht der Fall. Die Pufferverzögerungen T_{Buf}, die während des Umlaufs eines Pakets entstehen, werden bestimmt durch

$$T_{\mathrm{Buf}} = n \cdot ((T_{\mathrm{Trn}} + T_{\mathrm{IFG}}) - (T_{\mathrm{Trn}} + T_{\mathrm{Prd}} + T_{\mathrm{Swi}} + 2 \cdot J_{\mathrm{Port}}))\,. \tag{5.1}$$

Verursacht wird die Pufferverzögerung dadurch, dass ein Paket den Ring schneller durchläuft als ein Switch die n Pakete des Ringes übertragen kann. In dem Simulationsbeispiel aus Abbildung 5.7 benötigt der vierte Switch 26,88 μs, um vier Pakete zu übertragen. Das zuerst gesendete Paket erreicht den Ausgangspuffer des vierten Switches bei einem maximalen Jitter in jedem Ethernet-Port bereits nach 24,68 μs. Somit liegt die Pufferverzögerung bei 2,2 μs. Die Verschiebung der Paketströme hat aufgrund der deterministischen Pufferverzögerungen keinen Einfluss auf das Echtzeitverhalten des Protokolls. Auch bei Ethernet findet eine Selbstsynchronisation statt.

5.3.1 Synchronisationszeit

In Lemma 4.2.1 wurde gezeigt, dass für die Synchronisationszeit eine theoretische obere Schranke von $T_{\mathrm{syn}} = (3 \cdot n - 2)$ Zeitschritten existiert. Ein Zeitschritt bei einem Ethernet-basierten Self$S$-Protokoll ist gleich der Übertragungszeit von dem Sendepuffer des einen Switches in den Sendepuffer des nächsten Switches und wird bestimmt durch:

$$T_{\mathrm{Trt}} = (T_{\mathrm{Trn}} + T_{\mathrm{Prd}} + T_{\mathrm{Swi}} + 2 \cdot J_{\mathrm{Port}}) + T_{\mathrm{Buf}}/n \tag{5.2}$$

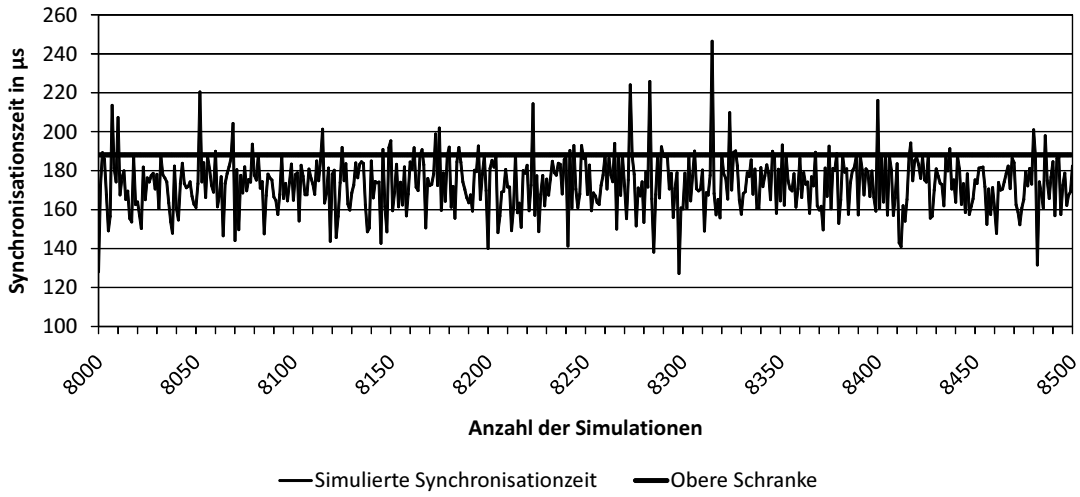


Abbildung 5.8: Synchronisationszeiten für ein Netzwerk mit 10 Knoten

Mithilfe der Simulationen wird die Synchronisationszeit ermittelt und überprüft, ob die obere Schranke überschritten wird. Dazu wurde 10.000 mal die Synchronisationszeit in Netzwerken mit einer Anzahl von 10 bis 100 Knoten mit zufällig generierten Startzeiten ermittelt. Das Intervall für die Startzeiten wurde in diesem Fall auf 20 ms erhöht. Die Abbildung 5.8 zeigt einen Auszug von 500 Werten aus den Ergebnissen der Simulation eines Netzwerks mit 10 Knoten. Ebenfalls eingezeichnet ist die theoretische obere Schranke der Synchronisationszeit für ein Netzwerk mit 10 Knoten. In der Darstellung ist zu erkennen, dass diese obere Schranke in einigen Fällen von den simulierten Synchronisationszeiten überschritten wird.

Die Abbildung 5.9 zeigt die mittleren und maximalen Simulationszeiten für Netzwerke mit 10 bis 100 Knoten. Die mittleren simulierten Synchronisationszeiten liegen unterhalb der oberen Schranke. Aber die maximalen simulierten Synchronisationszeiten liegen bei allen simulierten Netzwerken über der theoretischen oberen Schranke. Für ein Netzwerk mit 10 Knoten liegt die maximale Synchronisationszeit z. B. bei 259 μs und die theoretische obere Schranke bei 188 μs. Der Grund für diese Überschreitung ist der Portjitter, der sich in den ersten Runden über mehrere Knoten aufaddieren kann. Wird dasselbe Netzwerk ohne den Portjitter simuliert, liegt die maximale simulierte Synchronisationszeit mit einem Wert von 187,7 μs noch unterhalb der theoretischen oberen Schranke.

In den ersten Runden addiert sich der Portjitter in jedem Knoten bei der Übertragung eines Paketes auf. Dadurch entstehen Paketlücken, die größer sind als die eingestellte erlaubte Paketlücke von 1,04 μs. Obwohl die Selbstsynchronisation schon abgeschlossen ist, führt eine größere Paketlücke dazu, dass der Algorithmus erst in der nächsten

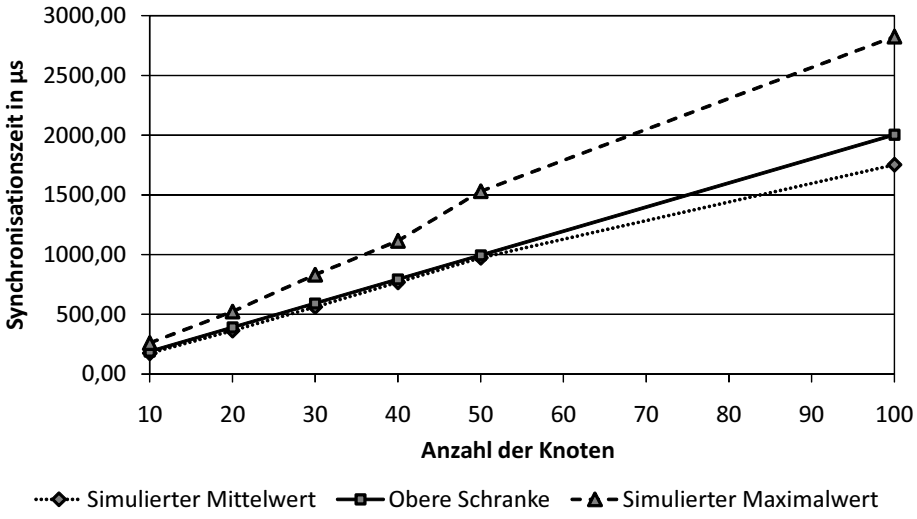


Abbildung 5.9: Synchronisationszeiten für Netzwerke von 10 bis 100 Knoten

Runde ein Netzwerk als synchronisiert erkennt. In der Simulation erkennt der Algorithmus die Netzwerke daher erst nach ca. vier Runden und nicht schon nach drei Runden als synchronisiert. Nach ca. vier Runden ist das Netzwerk spätestens synchronisiert, weil die größeren Paketlücken aufgrund der Kompensierung des Portjitters durch die IFG nur in den ersten Runden entstehen können. Erläutert wird die Kompensierung des Portjitters im Detail in Kapitel 5.4.3.

5.4 Echtzeiteigenschaften

In der theoretischen Beschreibung und Analyse des selbstsynchronisierenden Protokolls konnte in Lemma 4.2.2 nachgewiesen werden, dass das Self$S$-Protokoll eine gleich bleibende feste Zykluszeit und damit auch keinen Protokolljitter besitzt. Der Einfluss von Ethernet auf die Zykluszeit, den Jitter und die Latenz wird mithilfe der Simulation untersucht. Die Zykluszeit ist abhängig von der Paketumlaufzeit und wurde mit $T_{\mathrm{Cyc}_i} = T_{\mathrm{RTT}} \cdot K_i$ mit $K_i \in \mathbb{N}^+$ definiert. Der Faktor K_i hat keinen Einfluss auf die Paketumlaufzeit und den Jitter, weil sich über diesem Faktor nur der Pakettyp verändert. Daher erfolgt die Simulation mit $K_i = 1$, womit die Zykluszeit jedes Knotens der Paketumlaufzeit entspricht.

Die Paketumlaufzeit wird in der Simulation für jeden Switch immer dann ermittelt, wenn ein Switch ein Paket mit der Paket-ID dieses Switches vollständig empfangen hat. Es wurde der Empfangszeitpunkt für die Bestimmung der Paketumlaufzeit herangezogen, um den periodischen Empfang von Paketen und damit die Echtzeitfähigkeit

Anzahl der Knoten	berechnete mittlere RTT in μs	simulierte mittlere RTT in μs	Standardabweichung der sim. RTT in ns
10	67,20	67,20	23,02
20	134,40	134,40	23,02
30	201,60	201,60	23,28
40	268,80	268,80	22,84
50	336,00	336,00	23,14
100	672,00	672,00	22,88
1000	6720,00	6720,00	23,01

Tabelle 5.2: Mittlere Paketumlaufzeit (RTT)

nachzuweisen. Errechnet wird die Paketumlaufzeit in jeder Runde in einem Switch i mit

$$T_{\mathrm{RTT}_i} = t_{k_i} - t_{(k-1)_i}, \tag{5.3}$$

wobei t_{k_i} die Ankunftszeit des eigenen Paketes in der k-ten Runde ist. Auch bei diesen Simulationen beginnen die Knoten mit der Übertragung ihres ersten Paketes nach den zufällig generierten Startzeitpunkten. Die Paketumlaufzeit wird für einen Knoten aber erst ermittelt, nachdem der Knoten synchronisiert ist. Anderenfalls würden die Paketumlaufzeiten, die während der Synchronisation entstehen, das Gesamtergebnis verfälschen.

5.4.1 Mittlere Paketumlaufzeit

Die Mittelwerte der Paketumlaufzeit für Netzwerke mit einer Anzahl von 10 bis 1000 Knoten sind in der Tabelle 5.2 aufgelistet. Sie wurden aus einem Satz von 10.000 Simulationsergebnissen mit einer Standardabweichung von 23 ns gebildet. Größere Netzwerke mit mehr als 1000 Knoten wurden aufgrund der langen Simulationsdauer nicht betrachtet. Die Simulationsdauer eines Netzwerkes mit 1000 Knoten über 10.000 Runden beträgt bereits mehrere Tage. Die durchschnittliche Paketumlaufzeit entspricht, bis auf eine minimale Abweichung von 0,00002 ‰, einer berechneten Paketumlaufzeit von

$$T_{\mathrm{RTT}} = n \cdot (T_{\mathrm{Trn}} + T_{\mathrm{IFG}}). \tag{5.4}$$

Keinen Einfluss auf die durchschnittliche Paketumlaufzeit haben die Verarbeitungszeiten und die Ausbreitungsverzögerungen, solange $T_{\mathrm{IFG}} > T_{\mathrm{Swi}} + T_{\mathrm{Prd}} + 2 \cdot J_{\mathrm{Port}}$ ist. Wenn

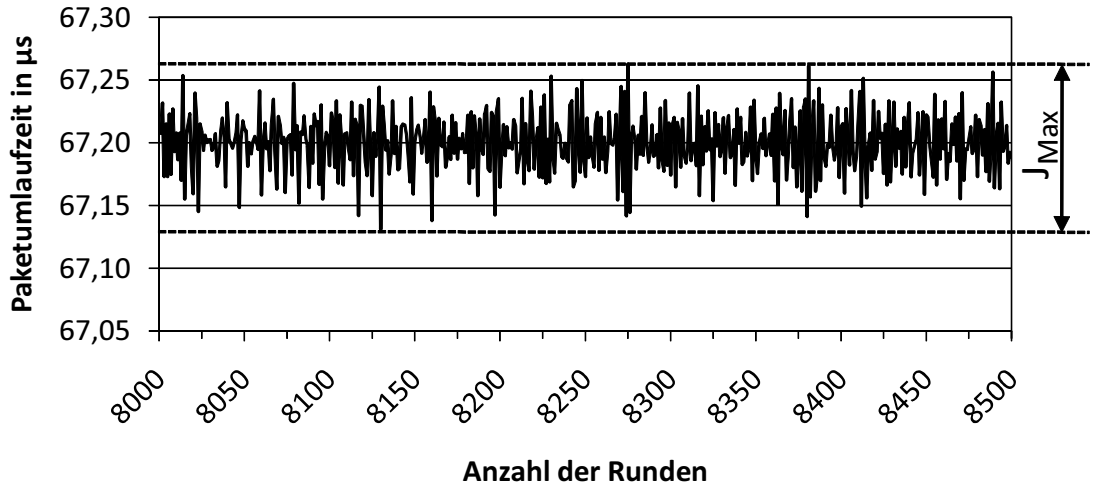


Abbildung 5.10: Simulierte Paketumlaufzeit im Netz mit 10 Knoten

diese Bedingung zutrifft, und das ist im Simulationsmodell der Fall, dann durchläuft ein Paket p schneller den gesamten Ring, als ein Switch i die n Pakete des Ringes versenden kann. Das Paket p kann erst wieder von dem Switch i gesendet werden, wenn dieser die Übertragung der anderen Pakete aus dem Ring abgeschlossen hat. Aus diesem Grund ist die Paketumlaufzeit in der Simulation im Mittel auch gleich $T_{\mathrm{RTT}} = n \cdot (T_{\mathrm{Trn}} + T_{\mathrm{IFG}})$.

5.4.2 Jitter der Paketumlaufzeit

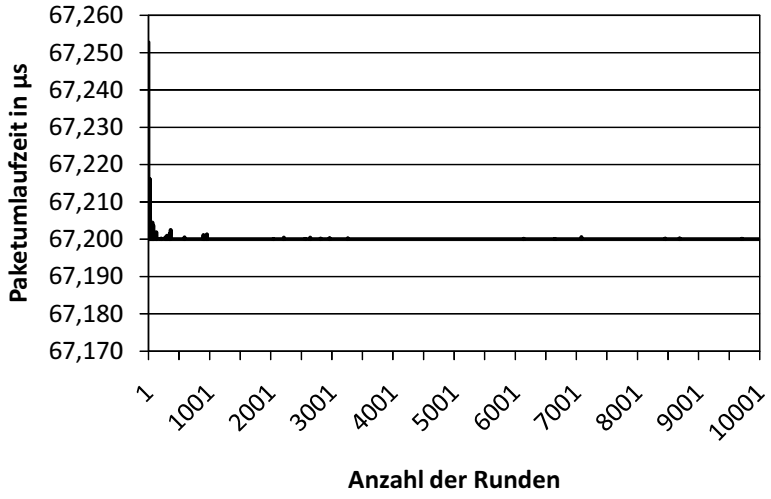
An dieser Stelle werden nicht nur die Mittelwerte betrachtet, sondern die Abbildung 5.10 zeigt eine Stichprobe von 500 Werten aus den simulierten Paketumlaufzeiten eines Knotens aus einem Ring mit insgesamt 10 Knoten. Auch für diese Konfiguration beträgt der Mittelwert der Paketumlaufzeit 67,2 μs. Der Jitter der Paketumlaufzeit ist durch die Ausschläge des Graphen um den Mittelwert herum zu erkennen. Sein Maximum J_{Max} ergibt sich aus der Differenz der größten Paketumlaufzeit und der geringsten Paketumlaufzeit. In der Abbildung 5.10 wird der maximale Jitter durch die zwei Geraden gekennzeichnet, innerhalb derer sich sämtliche Werte für die Paketumlaufzeit befinden. Für die abgebildete Stichprobe liegt der maximale Jitter bei einem Wert von 130 ns.

Im gesamten Simulationsintervall von 10.000 Runden wurden für ein Netzwerk mit 10 Knoten nach Gleichung (2.5) ein durchschnittlicher Jitter von 18 ns zusammen mit einem maximalen Jitter von 144 ns und einer Standardabweichung von 23 ns ermittelt. Der durchschnittliche quadratische Jitterfehler beträgt in diesem Fall 533 ns^2.

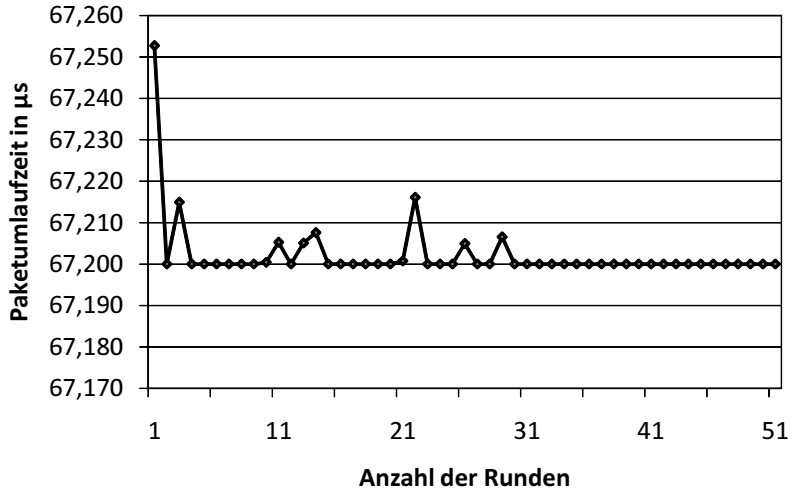
Anzahl der Knoten	mittlerer Jitter in ns	maximaler Jitter in ns	mittlerer quad. Jitterfehler in ns^2
10	18,6	141	530
20	18,5	147	530
30	18,9	144	542
40	18,4	144	522
50	18,7	142	536
100	18,5	145	524
1000	18,6	141	530
10000	22,1	117	812

Tabelle 5.3: Simulierter Jitter der Paketumlaufzeit (Verarbeitungszeit 280 ns)

Weitere Simulationen für eine unterschiedliche Anzahl an Knoten haben gezeigt, dass der durchschnittliche Jitter, der maximale Jitter und der durchschnittliche quadratische Jitterfehler bei einer steigenden Anzahl von Knoten, wie in Tabelle 5.3 zu erkennen, ungefähr gleich ist, solange $T_{\mathrm{IFG}} > T_{\mathrm{Swi}} + T_{\mathrm{Prd}} + 2 \cdot J_{\mathrm{Port}}$ ist. Normalerweise wäre zu erwarten, dass der maximale Jitter mit der Anzahl der Knoten ansteigt. Denn auf ein Paket, das den Ring durchläuft, übt jeder Knoten den 2-fachen Portjitter aus. Daraus folgt, dass sich die Portjitter addieren und theoretisch ein maximaler Jitter von $J_{\mathrm{Max}} = 2 \cdot n \cdot J_{\mathrm{Port}}$ erreicht wird. Allerdings wird im Fall $T_{\mathrm{IFG}} > T_{\mathrm{Swi}} + T_{\mathrm{Prd}} + 2 \cdot J_{\mathrm{Port}}$ der Portjitter jedes Knotens nicht addiert, sondern der Jitter wird von der IFG kompensiert. Daraus folgt, dass der Jitter der Ankunftszeit t_{k_i} eines Paketes, das den Ring durchlaufen hat, nur von den beiden Portjitter bei der letzten Übertragung beeinflusst wird. Sein Maximum entspricht theoretisch dem zweifachen maximalen Portjitter. Die Paketumlaufzeit wird nach Gleichung (5.3) aus der aktuellen und der vorangegangenen Ankunftszeit eines Paketes berechnet. Daher entspricht der maximale Jitter der Paketumlaufzeit in der Simulation dem Vierfachen des maximalen Portjitters. In der simulierten Konfiguration beträgt der maximale Jitter der Paketumlaufzeit theoretisch 160 ns. In der Simulation über 10.000 Runden liegt für eine unterschiedliche Anzahl an Knoten im Netzwerk der maximale Jitter aber nur bei ca. 144 ns. Der Grund für die Abweichung vom simulierten und theoretischen Maximum ist die geringe Wahrscheinlichkeit, dass innerhalb von 10.000 Runden beide in der Simulation zufällig generierten Portjitter zur gleichen Zeit null sind und in der nächsten Runde beide Portjitter zufällig ihren maximal Wert erreichen. Bei einer höheren Anzahl von simulierten Runden nähert sich der simulierte maximale Jitter der Paketumlaufzeit langsam dem theoretischen Wert an und beträgt z. B. in einem für zwei Millionen Runden simulierten Netzwerk von 10 Knoten bereits 157 ns. Die Standardabweichung liegt, wie bei der Simulation über 10.000 Runden, bei 23 ns.



(a) 10.000 Runde



(b) Die ersten 50 Runden

Abbildung 5.11: Paketumlaufzeit RTT_{Sen}

5.4.3 Kompensierung des Jitters durch die IFG

Der simulative Nachweis der Kompensierung des Jitters durch die IFG wird dadurch erbracht, dass die Paketumlaufzeit immer dann bestimmt wird, wenn der Switch mit der Übertragung an die physikalische Schnittstelle beginnt. Dies geschieht, direkt nachdem die IFG abgelaufen ist und kurz bevor der Ausgangsportjitter entsteht. Um Verwechselungen zu vermeiden, wird diese Paketumlaufzeit durch RTT_{Sen} symbolisiert und die Paketumlaufzeit, die bei der Ankunft des Paketes ermittelt wird, durch RTT_{Emp} symbolisiert. Wenn die IFG den Jitter absorbiert, ist RTT_{Sen} jitterfrei. Die Abbildung 5.11 zeigt die Paketumlaufzeit RTT_{Sen} für ein Netzwerk mit 10 Knoten über ein Si-

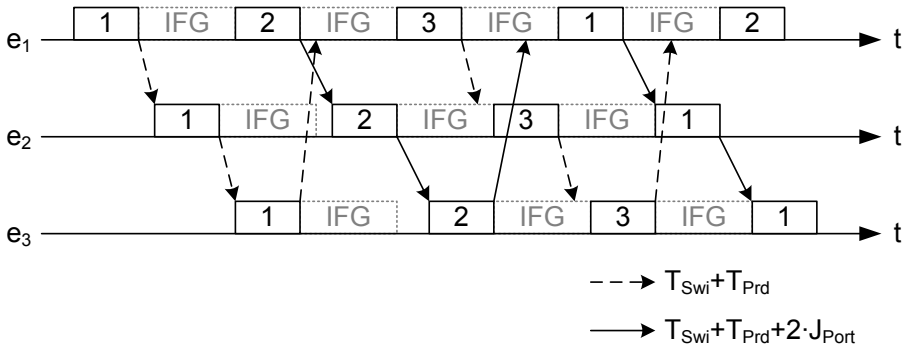


Abbildung 5.12: Kompensierung des Jitters durch die IFG

mulationsintervall von 10.000 Runden. Zu erkennen ist ein Einschwingvorgang, in dem der Jitter dieser Paketumlaufzeit in der ersten Runde nach der Synchronisation noch 53 ns beträgt. Allerdings fällt dieser Jitter sehr schnell ab und in der Abbildung 5.11(b) ist er ab der 30-ten Runde schließlich gleich null. Zwar tritt in der Abbildung 5.11(a) auch noch nach über 8000 Runden sporadisch ein Jitter auf, der aber mit 0,1 ns so gering ist, dass er keinerlei Auswirkung auf die Paketumlaufzeit RTT_{Emp} hat, die bei der Ankunft eines Paketes ermittelt wird.

Um den in Abbildung 5.11 gezeigten Einschwingvorgang des Jitters und damit auch die Kompensierung des Portjitters durch die IFG zu verdeutlichen, zeigt die Abbildung 5.12 ein Beispiel für den Einschwingvorgang in einem Netzwerk mit drei Knoten. Dargestellt sind die Paketflüsse einschließlich der IFG für die drei Switches. Der Switch 1 sendet alle $T_{\mathrm{Trn}} + T_{\mathrm{IFG}}$ ein Paket. Es wird angenommen, dass das erste Paket den Switch 2 bei einem Portjitter von null nur verzögert durch die T_{Prd} erreicht. Dann beginnt die Übertragung in Switch 2, nachdem T_{Swi} verstrichen ist. Das zweite Paket wird von dem Switch erst nach einer zusätzlichen Verzögerung durch den zweifachen Portjitter empfangen. Nachdem auch T_{Swi} verstrichen ist, beginnt die Übertragung des zweiten Paketes $T_{\mathrm{Trn}} + T_{\mathrm{IFG}} + 2 \cdot J_{\mathrm{Port}}$, nachdem der Switch 2 das erste Paket übertragen hat. Ein Jitter entsteht in Abbildung 5.11 nur, wenn ein Paket erst zur Übertragung bereitsteht, nachdem die IFG abgelaufen ist. Daher ist der Jitter in der Abbildung 5.11 auch immer positiv. Als Folge der verzögerten Übertragung verschiebt sich auch das Ende der IFG und das Paket 3 muss erst warten, bis T_{IFG} abgelaufen ist.

In Abbildung 5.12 ist ebenfalls zu erkennen, dass die Verschiebung der IFG nach der Übertragung des zweiten Paketes in Switch 2 bei gleichem Jitter auch die IFG von Switch 3 verschiebt. Eine ständige Verschiebung der IFG im Kreis würde dazu führen, dass sich der Start der Übertragung auf der Zeitachse ständig weiter verschiebt und damit auch der nachfolgende Portjitter. Aber in Folge der Verschiebung der IFG in Switch 3 folgt keine Verschiebung der IFG in Switch 1, weil das Paket 2 den Switch 1 erreicht hat, bevor dieser das Paket 1 übertragen konnte. Nicht nur in dem abgebildeten

Beispiel durchläuft ein Paket p schneller den Ring, als ein Switch alle n Pakete des Ringes übertragen kann. Dies ist auch in der Simulation gegeben, wie bereits in Kapitel 5.3.1 erwähnt.

Entspricht nun J_{Port} dem maximalen Portjitter, wird das Ende der IFG so weit verschoben, dass kein Paket nach Ablauf der IFG den Switch mehr erreicht. Das System ist dann völlig eingeschwungen und jeder Portjitter wird von der IFG kompensiert. Dass auch noch nach über 8000 Runden ein Jitter in Abbildung 5.11 auftritt, liegt daran, dass es sehr unwahrscheinlich ist, dass beide Portjitter zufällig zum gleichen Zeitpunkt ihren maximalen Wert erreichen und somit auch die Summe der beiden Portjitter maximal wird.

Die Kompensierung des Portjitters der vorangegangenen Übertragung durch die IFG führt dazu, dass der Jitter der Paketumlaufzeit RTT_{Emp} auch nur vom Jitter des Sendeports vom vorangegangen Knoten und vom Jitter des eigenen Empfangsports beeinflusst wird. Daraus folgt, dass der maximale Jitter einen Wert von 160 ns nicht übersteigt, solange $T_{\text{IFG}} \geq T_{\text{Swi}} + T_{\text{Prd}} + 2 \cdot J_{\text{Port}}$. Ein maximaler Jitter von 160 ns ist im Vergleich mit anderen Ethernet-basierten Echtzeitprotokollen sehr gering. PROFINET IRT erreicht bereits bei einem Netzwerk mit 25 Knoten, die in einer Linie angeordnet sind, einen maximalen Jitter von 1 μs [Pop05].

5.4.4 Jitter bei großen Verarbeitungszeiten

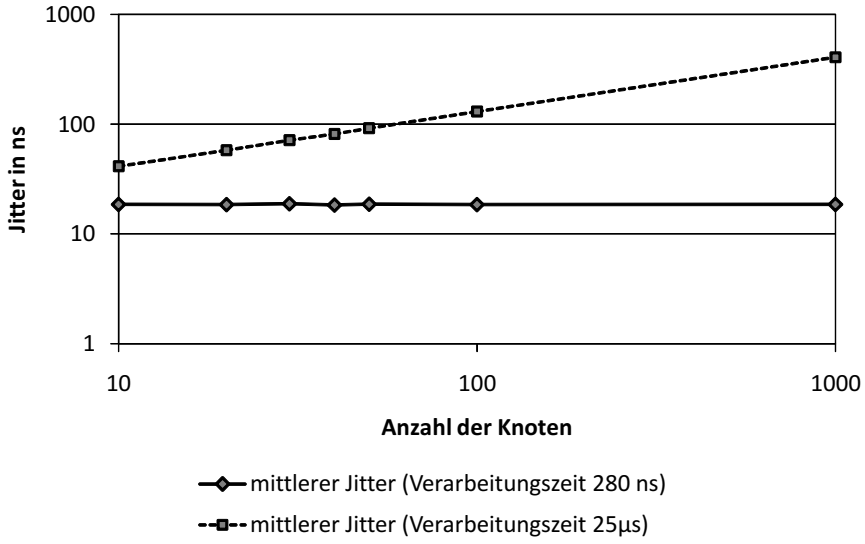


Abbildung 5.13: Mittlerer Jitter bei unterschiedlichen Verarbeitungszeiten

Für den Fall, dass $T_{\text{IFG}} < T_{\text{Swi}} + T_{\text{Prd}} + 2 \cdot J_{\text{Port}}$ ist, kann die IFG den Portjitter zwischen beliebigen Switches nicht mehr absorbieren und im schlechtesten Fall erhöht

sich der Jitter über den gesamten Ring. Insbesondere wenn ein Software-Switch anstelle eines Hardware-Switches verwendet wird, ist bereits die Verarbeitungszeit des Switches größer als die Dauer einer IFG. In der Simulation, deren Ergebnis die Abbildung 5.13 illustriert, wurde T_{Swi} auf 25 μs erhöht. Die 25 μs entsprechen der durchschnittlichen Verarbeitungszeit des in Kapitel 3.2.1 beschriebenen Software-Switches. Der Jitter der Verzögerungszeit des Software-Switches wird nicht an dieser Stelle, sondern in Kapitel 5.5 mit berücksichtigt. In diesem Abschnitt soll nur der Einfluss des Portjitters auf den Jitter der Paketumlaufzeit für den Fall untersucht werden, dass der Portjitter nicht durch die IFG kompensiert wird.

In der Abbildung 5.13 wird der durchschnittliche Jitter bei einer Verarbeitungszeit von 25 μs und 280 ns dargestellt. Der durchschnittliche Jitter der Paketumlaufzeit RTT_{Emp} ist für $T_{\text{IFG}} < T_{\text{Swi}} + T_{\text{Prd}} + 2 \cdot J_{\text{Port}}$ nicht nur deutlich höher, sondern er steigt auch mit der Anzahl der Knoten. Auch der maximale Jitter und der quadratische Jitterfehler sind ohne die Kompensierung durch die IFG wesentlich höher, wie die Tabelle 5.4 zeigt.

Anzahl der Knoten	**mittlerer Jitter in ns**	**maximaler Jitter in ns**	**mittlerer quad. Jitterfehler in ns^2**
10	41,3	376	2692
20	57,8	540	5269
30	71,4	641	7993
40	81,5	691	10344
50	91,9	964	13313
100	130,4	1340	26577
1000	408,6	3893	265349

Tabelle 5.4: Simulierter Jitter der Paketumlaufzeit (Verarbeitungszeit 25 μs)

5.4.5 Latenzzeit

Die Latenzzeit ist, wie die Abbildung 5.14 verdeutlicht, vor allem abhängig von der Anzahl der Knoten, die sich zwischen der Quelle und dem Ziel eines Paketes befinden. Die Abbildung 5.14 zeigt den linearen Verlauf der durchschnittlichen Latenzzeit für eine Verarbeitungszeit von 280 ns und von 25 μs. Ermittelt wurden die Latenzzeiten aus den Sendebeginnzeiten und den Ankunftszeiten eines simulierten Netzwerkes mit 100 Knoten über ein Intervall von 10.000 Runden. Betrachtet wurden die Latenzzeiten von einem Quellknoten zu mehreren Zielknoten. Die *Anzahl der Knoten* bezeichnet in der Abbildung 5.14 die Anzahl der Switches, die das Paket auf seinem Weg von der Quelle bis zum Ziel übertragen.

Bei einer Verarbeitungszeit von 280 ns haben die simulierten Latenzzeiten eine Standardabweichung von 16 ns, einen durchschnittlichen Jitter von 13 ns, einen maximalen

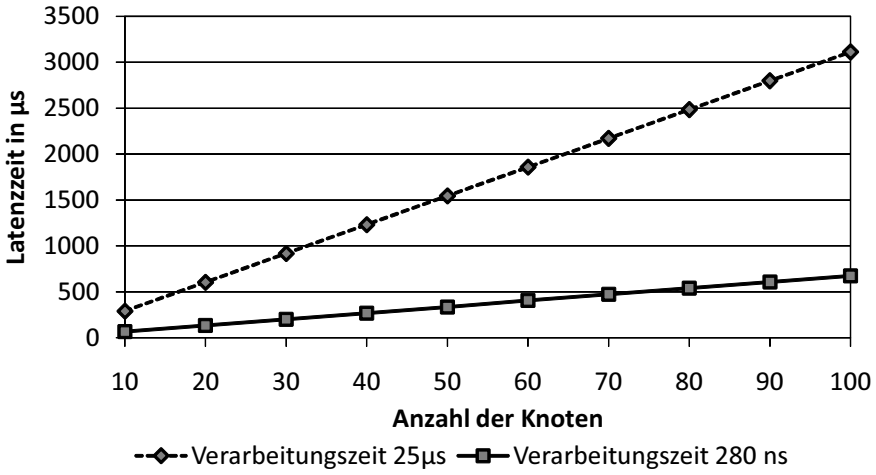


Abbildung 5.14: Simulierte durchschnittliche Latenzzeiten

Jitter von 80 ns und einen durchschnittlichen Jitterfehler von 267 ns^2, unabhängig von der Anzahl der Knoten. Der simulierte maximale Jitter entspricht dem theoretischen Maximum der Portjitter zwischen zwei Knoten. Die zuvor beschriebene Kompensierung des Portjitters durch die IFG für den Fall $T_{\mathrm{IFG}} \geq T_{\mathrm{Swi}} + T_{\mathrm{Prd}} + 2 \cdot J_{\mathrm{Port}}$ beschränkt ebenfalls den Jitter der Latenz. Verursacht wird der Jitter der Latenzzeit daher nur durch den Portjitter, der bei der Übertragung vom letzten Switch bis zum Ziel entsteht. Aus diesem Grund liegt das theoretische Maximum des Jitters der Latenzzeit in dieser Konfiguration bei 80 ns.

Generell steigt die Latenzzeit im Falle eines Switches mit einer Verarbeitungszeit von 280 ns im Durchschnitt für jeden weiteren Knoten, den das Paket durchlaufen muss, um jeweils 6,676 μs an. Dieser Wert entspricht der Summe der Verzögerungszeiten, die bei der Übertragung über die Leitung und durch den Switch entstehen, und wird berechnet mit $T_{\mathrm{Swi}} + T_{\mathrm{Prd}} + 2 \cdot J_{\mathrm{Port}}$. Die Tabelle 5.5 listet die Latenzzeiten und den Anstieg der Latenzzeiten als Differenz aus $T_{\mathrm{Lat}_n} - T_{\mathrm{Lat}_{n-1}}$ für 10 Knoten auf. Die simulierten Ergebnisse aus der Tabelle stammen von einer Simulation eines Netzwerkes mit 10 Knoten mit einem Simulationsintervall von 100.000 Runden. Die Ergebnisse aus dieser Simulation sind zum einen genauer als die Ergebnisse der Abbildung 5.14. Zum anderen verdeutlicht dieses Simulationsbeispiel die Abweichungen vom Anstieg der Latenzzeit von Knoten zu Knoten, weil für alle Knoten des Netzwerkes die Latenzzeiten aufgelistet sind.

In der Tabelle 5.5 weicht nur der Anstieg der Latenzzeit von Knoten 1 und Knoten 8 von dem Wert 6,676 μs ab. Die durchschnittliche Latenz von Knoten 1 repräsentiert die Latenz zwischen zwei benachbarten Knoten. Sie ist um T_{Swi} und um weitere 40 ns geringer als die durchschnittliche Latenz, die entsteht, wenn das Paket vom Switch weitergeleitet wird. Die Latenz wird ab dem Sendebeginn eines Paketes ermittelt. Aus

Anzahl der Knoten	simulierte mittlere Latenz in μs	$T_{\mathrm{Lat}_n} - T_{\mathrm{Lat}_{n-1}}$ in μs	berechnete mittlere Latenz in μs
1	6,36	6,3560	6,40
2	13,03	6,6758	13,12
3	19,71	6,6759	19,84
4	26,38	6,6759	26,56
5	33,06	6,6759	33,28
6	39,74	6,6758	40,00
7	46,41	6,6759	46,72
8	53,53	7,1172	53,44
9	60,20	6,6759	60,16
10	66,88	6,6759	66,88

Tabelle 5.5: Latenzzeit für eine Verarbeitungszeit von 280 ns

diesem Grund hat die Verarbeitungszeit bei der ersten Übertragung keinen Einfluss auf die Latenzzeit. Des Weiteren erhöhen die beiden Portjitter bei der ersten Übertragung die Latenz bei gleich verteiltem Portjitter im Durchschnitt nur um 40 ns. Wird aber das Paket weitergeleitet, sorgt die Kompensierung des Jitters durch die IFG im eingeschwungenen Zustand dafür, dass sich die Latenzzeit nicht um den durchschnittlichen zweifachen Portjitter, sondern immer um den maximalen zweifachen Portjitter erhöht. Aufgrund der in Kapitel 5.3 beschriebenen Pufferverzögerung ist die Erhöhung der Latenzzeit von Knoten 7 nach Knoten 8 mit einem Wert von 7,112 μs um T_{Buf} höher als bei den anderen Knoten. Auch bei der Simulation mit 100 Knoten erfährt ein Knoten eine um T_{Buf} erhöhte Latenz, die aber aufgrund der geringen Größe in Abbildung 5.14 nicht zu erkennen ist.

Rechnerisch kann die durchschnittliche Latenzzeit eines Paketes, die bei der Übertragung über u Knoten von der Quelle bis zum Ziel entsteht, für den Fall, dass $T_{\mathrm{IFG}} \geq T_{\mathrm{Swi}} + T_{\mathrm{Prd}} + 2 \cdot J_{\mathrm{Port}}$ ist, wie folgt beschrieben werden:

$$T_{\mathrm{Lat}} = u \cdot \left(T_{\mathrm{Trn}} + T_{\mathrm{Prd}} + J_{\mathrm{Port}} + \frac{T_{\mathrm{Buf}}}{n} \right) - (u-1) \cdot (T_{\mathrm{Swi}} + J_{\mathrm{Port}}) \tag{5.5}$$

Wird in die Gleichung (5.5) für T_{Buf} die Gleichung (5.1) eingesetzt, folgt daraus eine durchschnittliche Latenzzeit von

$$T_{\mathrm{Lat}} = u \cdot (T_{\mathrm{Trn}} + T_{\mathrm{IFG}}) - T_{\mathrm{Swi}} - J_{\mathrm{Port}}. \tag{5.6}$$

Die theoretischen Latenzzeiten, die auch in der Tabelle 5.5 aufgelistet sind, weichen von den simulierten Werten ab, weil T_{Buf} nur in einem Knoten des Netzwerkes entsteht. In welchem Knoten T_{Buf} entsteht, ist von dem zufälligen Start der Pakete und der

Anzahl der Knoten	maximaler Jitter in ns	mittlerer Jitter in ns	mittlerer quad. Jitterfehler in ns^2
10	453	41	2657
20	678	58	5331
30	808	71	7994
40	940	82	10659
50	1032	92	13309
60	1124	101	15982
70	1295	109	18638
80	1433	117	21318
90	1523	124	23986
100	1559	130	26679

Tabelle 5.6: Simulierter Jitter der Latenzzeit (Verarbeitungszeit 25 μs)

Synchronisation abhängig und kann nicht vorherbestimmt werden. Daher wird bei der Berechnung der Latenzzeiten T_{Buf} anteilig auf jeden Knoten verteilt.

Letztendlich verdeutlicht die Gleichung (5.6), dass die Latenzzeit eines Paketes, das den Ring vollständig durchläuft, durch die Paketumlaufzeit des Paketes begrenzt wird. Weil die Latenzzeit vom Sendebeginn bis zur Ankunft des Paketes und nicht zwischen der Ankunftszeit und der vorherigen Ankunftszeit ermittelt wird, ist die Latenzzeit um $T_{\mathrm{Swi}} + J_{\mathrm{Port}}$ geringer als die Paketumlaufzeit.

Wie in der Abbildung 5.14 verdeutlicht wird, sind die Latenzzeiten für eine Verarbeitungszeit von 25 μs deutlich größer als die Latenzzeiten bei einer Verarbeitungszeit von 280 ns. In diesem Fall ist $T_{\mathrm{IFG}} < T_{\mathrm{Swi}} + T_{\mathrm{Prd}} + 2 \cdot J_{\mathrm{Port}}$ und der Portjitter kann nicht durch die IFG kompensiert werden. Wie auch bei der Paketumlaufzeit steigt der maximale Jitter, der durchschnittliche Jitter und der durchschnittliche quadratische Jitterfehler in Abhängigkeit von der Anzahl der Knoten, die ein Paket von der Quelle bis zum Ziel übertragen. Die Tabelle 5.6 listet den in einem Netzwerk mit 100 Knoten simulierten Jitter der Latenzzeit für $T_{\mathrm{Swi}} = 25\,\mu$s auf.

In der Tabelle 5.7 sind die durchschnittlichen Latenzen für jeden 10-ten Knoten eines simulierten Netzwerkes von 100 Knoten aufgelistet. Jeder zusätzliche Knoten, über den das Paket übertragen werden muss, erhöht die Latenzzeit für diese Konfiguration um 31,356 μs und jeder 10-te Knoten somit um 313,56 μs. Nur aufgrund der Bestimmung der Latenzzeit vom Sendebeginn an ist die Erhöhung der Latenz nach den ersten 10 Knoten 25μs geringer. Daher werden die durchschnittlichen Latenzzeiten für den Fall, dass $T_{\mathrm{IFG}} < T_{\mathrm{Swi}} + T_{\mathrm{Prd}} + 2 \cdot J_{\mathrm{Port}}$ ist, rechnerisch nach der folgenden Gleichung bestimmt:

$$T_{\mathrm{Lat}} = u \cdot (T_{\mathrm{Trn}} + T_{\mathrm{Swi}} + T_{\mathrm{Prd}} + J_{\mathrm{Port}}) - T_{\mathrm{Swi}} \tag{5.7}$$

Anzahl der Knoten	**Simulierte mittlere Latenz in μs**	$T_{\mathrm{Lat}_n} - T_{\mathrm{Lat}_{n-1}}$ **in μs**	**Berechnete mittlere Latenz in μs**
10	289	288,56	289
20	602	313,56	602
30	916	313,56	916
40	1229	313,56	1229
50	1543	313,56	1543
60	1856	313,56	1856
70	2170	313,56	2170
80	2483	313,56	2483
90	2797	313,56	2797
100	3111	313,56	3111

Tabelle 5.7: Latenzzeit für eine Verarbeitungszeit von 25 μs

Die berechneten Latenzzeiten sind auch in der Tabelle 5.7 aufgelistet und weichen im Gegensatz zum Fall $T_{\mathrm{IFG}} \geq T_{\mathrm{Swi}} + T_{\mathrm{Prd}} + 2 \cdot J_{\mathrm{Port}}$ nicht von den simulierten Werten ab.

5.5 Vergleich von Hardware- und Software-Switch

In diesem Abschnitt werden die Echtzeiteigenschaften des Hardware-Switches aus Kapitel 3.2.2 und des Software-Switches aus Kapitel 3.2.1 miteinander für den Fall verglichen, dass beide Switches das Self*S*-Protokoll anwenden. Die Echtzeiteigenschaften des Self*S*-Protokolls wurden für den Hardware-Switch bereits in den vorangegangenen Abschnitten untersucht und werden an dieser Stelle den Echtzeiteigenschaften des Software-Switches gegenübergestellt.

Im Gegensatz zum Hardware-Switch hat der Software-Switch keine konstante Verarbeitungszeit. Daher ist es für die Untersuchung der Echtzeiteigenschaften des Software-Switches notwendig, das Simulationsmodell aus Kapitel 5.2 um einen Jitter der Verarbeitungszeit zu erweitern. Die Messergebnisse aus Kapitel 3.3.2 haben für die Verarbeitungszeit einen maximalen Jitter von 4 μs ergeben. In der Simulation wird der Jitter durch eine Zufallszahl im Bereich von $\pm 2\,\mu$s generiert und zu der Verarbeitungszeit von 25 μs hinzuaddiert.

In der Tabelle 5.8 werden die Simulationsergebnisse für die durchschnittliche Paketumlaufzeit des Hardware-Switches und des Software-Switches gegenübergestellt. Die hohe Verarbeitungszeit des Software-Switches führt dazu, dass die Paketumlaufzeit bei gleicher Anzahl von Knoten gegenüber dem Hardware-Switch fünfmal so groß ist. Wird das Netzwerk nur aus Software-Switches aufgebaut, folgt daraus auch, dass die realisierbaren minimalen Zykluszeiten eines Knoten um das Fünffache größer sind. Auch die Latenzzeit, die, wie bereits in Kapitel 5.4.5 beschrieben, durch die Paketumlaufzeit

Anzahl der Knoten	mittlere Paketumlaufzeit in μs	
	Hardware-Switch	Software-Switch
10	67	315
20	134	631
30	202	946
40	269	1.261
50	336	1.577
100	672	3.154
1000	6.720	31.538

Tabelle 5.8: Simulierte Paketumlaufzeit von Software- und Hardware-Switch

begrenzt wird, ist beim Software-Switch ebenfalls fünfmal so groß. Die höheren Zykluszeiten und die höheren Latenzen führen dazu, dass bereits ab einer Anzahl von 30 Knoten in einem Netzwerk aus Software-Switches die höchste IAONA-Echtzeitklasse für den Bereich Latenz (Kapitel 2.5.1) nicht mehr erfüllt wird. Der Hardware-Switch erfüllt die höchste Echtzeitklasse noch bei einem Netzwerk von über 100 Knoten.

Während der Jitter der Paketumlaufzeit beim Hardware-Switch unabhängig von der Anzahl der Knoten einen maximalen Wert von 160 ns nicht überschreitet, liegt der Jitter der Paketumlaufzeit beim Software-Switch im Durchschnitt in einem Netzwerk mit nur 10 Knoten schon bei 3 μs. Sowohl der durchschnittliche als auch der maximale simulierte Jitter der Paketumlaufzeit steigt beim Software-Switch, wie in der Abbildung 5.15 dargestellt, mit der Anzahl der Knoten weiter an. Bereits bei einer Anzahl von 10 Knoten erreicht der maximale Jitter einen Wert von 27 μs. Mit diesem Wert kann nur noch die zweithöchste Echtzeitklasse im Bereich Jitter erfüllt werden. Bei einem Netzwerk mit 1000 Knoten beträgt der maximale Jitter nach 10.000 simulierten Runden schließlich 290μs und erreicht nur noch die Echtzeitklasse C. Beim Hardware-Switch kann im Bereich des Jitters für eine beliebige Anzahl von Knoten immer die höchste Echtzeitklasse F abgedeckt werden.

Theoretisch beträgt der maximale Jitter beim Software-Switch in einem Netzwerk mit 1000 Knoten sogar 4,08ms und wird berechnet mit

$$J_{\mathrm{Max}} = (J_{\mathrm{CPU}} + 2 \cdot J_{\mathrm{Port}}) \cdot n. \tag{5.8}$$

Den größten Einfluss auf den Jitter der Paketumlaufzeit beim Software-Switch hat der Jitter der Verarbeitungszeit J_{Swi}. Der Anteil des Portjitters am Jitter der Paketumlaufzeit ist wesentlich geringer. Bei einem Netzwerk mit 1000 Knoten beträgt der Anteil des Portjitters ca. 2 % bzw. 80 μs.

Das Self*S*-Protokoll, angewandt vom Hardware-Switch, erfüllt höchste Echtzeitanforderungen von Zykluszeiten unter einer Millisekunde und einem Jitter weit unter einer

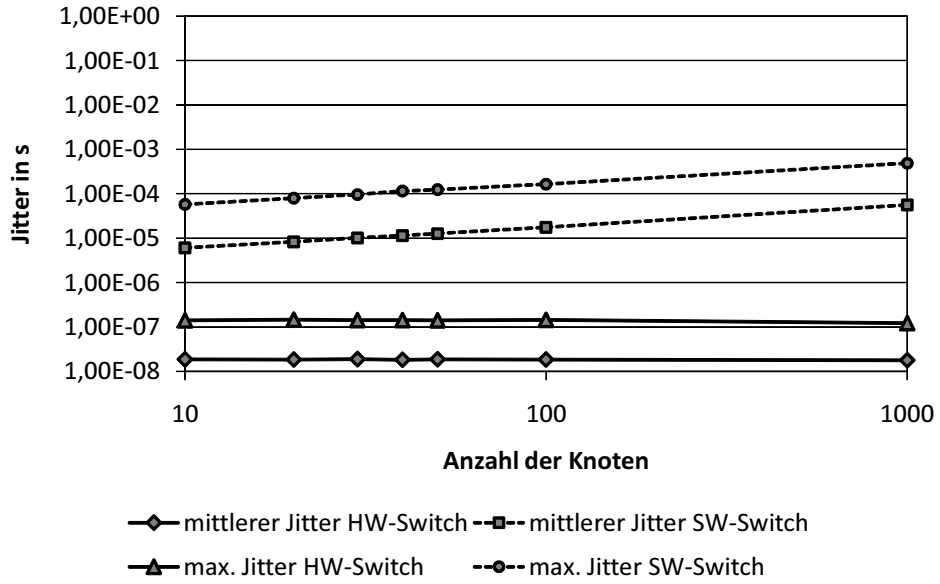


Abbildung 5.15: Simulierter Jitter des HW- und SW-Switches

Mikrosekunde, wie sie z. B. bei der Regelung von sich bewegenden Systemen (Motoren, Roboterarme, etc.) benötigt werden. Neben der höheren Zykluszeit und der höheren Latenz sorgt insbesondere der hohe Jitter beim Einsatz des Software-Switches dafür, dass die Echtzeiteigenschaften im Vergleich zum Hardware-Switch wesentlich schlechter sind. Nur geringe Echtzeitanforderungen können auch vom Software-Switch erfüllt werden.

5.5.1 Dynamische Rekonfiguration

Generell kann auch beim Self*S*-Protokoll die partielle dynamische Rekonfiguration vom Software- zum Hardware-Switch und umgekehrt benutzt werden. In diesem Fall müssen für die Entscheidung über eine Rekonfiguration die Anforderungen aller Teilnehmer im Ring betrachtet werden. Fordert nur ein Teilnehmer einen Jitter an, der kleiner als 1 μs ist, müssen alle Knoten im Ring als Hardware-Switch konfiguriert sein, damit diese Anforderung erfüllt wird. Nur ein Software-Switch im Ring ist ausreichend, um den Jitter der Paketumlaufzeit auf über 1 μs zu erhöhen. Des Weiteren erhöht jeder Software-Switch im Ring die minimal mögliche Zykluszeit für jeden Knoten.

Die dynamische Rekonfiguration der Switches muss in diesem Fall global gesteuert und kontrolliert werden. Sie benötigt einen Austausch der Anforderungen unter allen Knoten und die gleichzeitige Rekonfiguration von mehreren Switches. Jede Rekonfiguration führt auch zu einer erneuten Synchronisation der Knoten.

Das Self*S*-Protokoll wurde entwickelt, damit Zykluszeiten angepasst werden können, ohne dass die Echtzeiteigenschaften der anderen Knoten beeinflusst werden. Bei der Rekonfiguration kann eine Beeinflussung der anderen Teilnehmer nicht verhindert werden. Auch ein sehr geringer Jitter ist beim Self*S*-Protokoll nur zu erreichen, wenn im Ring ausschließlich Hardware-Switches verwendet werden.

5.6 Experimentelle Validierung von Self*S*

Die Echtzeiteigenschaften eines Ethernet-basierten Self*S*-Protokolls wurden in den vorangegangen Abschnitten mithilfe eines Simulationsmodells untersucht. In diesem Abschnitt wird die Gültigkeit der Simulationsergebnisse für ein reales System validiert. Dazu wurde ein Testsystem implementiert, das die Messung der Paketumlaufzeit und des Jitters der Paketumlaufzeit ermöglicht.

5.6.1 Aufbau des Testsystems

Das Testsystem besteht aus einem Netzwerk mit fünf Knoten, die ringförmig miteinander über ca. 1 m lange Kabel verbunden sind. Die Knoten basieren auf der in Kapitel 3.2 beschriebenen prototypischen Umsetzung des Software-Switches und des Hardware-Switches, die für die experimentelle Validierung des Self*S*-Protokolls angepasst wurden. Implementiert werden Hardware- und Software-Switch in diesem Fall nur auf einem Virtex-II Pro, der direkt an ein DB-Ethernet angeschlossen ist. Als Plattform für den Virtex-II Pro und das DB-Ethernet wird, wie beim rekonfigurierbaren Switch, das RAPTOR2000-Prototypingsystem verwendet. Im Gegensatz zu dem rekonfigurierbaren Switch wird ein Virtex-II hier nicht benötigt, weil eine dynamische partielle Rekonfiguration für diese experimentelle Validierung nicht erforderlich ist. Um Experimente mit beiden Switchvarianten durchzuführen, können die fünf Knoten zwischen Hardware-Switch und Software-Switch umgeschaltet werden.

An einen der fünf Knoten ist ein *Tektronix DPO 70404* digitales Oszilloskop angeschlossen. Vom Oszilloskop wird ein Ausgangssignal des FPGAs aufgenommen, welches vom Switch auf logisch 1 gesetzt wird, sobald der Switch sein eigenes Paket detektiert hat. Dieses Signal bleibt während der Empfangsdauer des eigenen Paketes auf 1. Gemessen wird die Paketumlaufzeit zwischen zwei aufeinanderfolgenden steigenden Taktflanken.

Im Testsystem unterscheidet sich der Ablauf der Kommunikation vom Ablauf in der Simulation nur beim Start der Kommunikation. In der Simulation beginnen die Knoten nach einer zufälligen Wartezeit mit der Generierung und Übertragung des ersten eigenen Paketes. Im Testsystem kann die Kommunikation erst dann starten, nachdem

alle Knoten konfiguriert und damit betriebsbereit sind. Nach der Konfiguration wird von einem der Knoten ein Initialisierungspaket versendet, das an jeden Knoten weitergeleitet wird. Nachdem ein Knoten das Initialisierungspaket erhalten hat, erstellt er sein eigenes Paket und überträgt dieses. Wie in der Simulation werden auch im Testsystem nur RT-Pakete von den Knoten versendet. Damit entspricht die Zykluszeit der Paketumlaufzeit.

5.6.2 Ergebnisse Hardware-Switch

Mit dem Oszilloskop wurden 10.000 Messwerte der Paketumlaufzeit in dem aufgebauten Netzwerk, bestehend aus fünf Hardware-Switches, aufgenommen. Aus diesen Messwerten resultiert eine mittlere Paketumlaufzeit von 33,6 μs, ein maximaler periodischer Jitter von 80 ns und eine Standardabweichung von 4 ns. Das Ergebnis einer zweiten Messung des Jitters der Paketumlaufzeit mit dem Oszilloskop ist in Abbildung 5.16 dargestellt. Das Oszilloskop wird dazu so eingestellt, dass mehrere steigende Flanken angezeigt werden. Dazu wird bei dem Oszilloskop die Anzeigeoption *unendliches Nachleuchten (engl.: infinite persistence)* gewählt. Bei dieser Messung wird der maximale periodische Jitter über den größten Abstand zwischen zwei steigenden Flanken bestimmt. In der Abbildung 5.16 sind die beiden Messpunkte t_1 und t_2 durch zwei vertikale Linien markiert. Der Abstand zwischen t_1 und t_2, also das Δt, beträgt 80 ns und bestätigt damit das Ergebnis der ersten Messung.

Des Weiteren ist in der Abbildung 5.16 zu erkennen, dass nur drei unterschiedliche Werte für die Paketumlaufzeiten entstehen. Der Abstand zwischen zwei nebeneinanderliegenden steigenden Taktflanken beträgt exakt 40 ns. Dieser Abstand entspricht genau der Taktperiode des Hardware-Switches. Ein anderer Abstand zwischen den Taktflanken ist nicht möglich, weil der Switch auch nur alle 40 ns ein eigenes Paket detektieren kann.

Im Testsystem wird die Übertragung eines Paketes zusätzlich durch das DB-Ethernet verzögert. Die physikalische Sende- und Empfangseinheit LTX974 [Int01] und das zur Synchronisierung mit der MAC auf dem DB-Ethernet eingesetzte CPLD erhöhen beim Senden und Empfangen die Übertragung um 580 ns. Um die gemessenen Werte mit der Simulation zu vergleichen, wird die Verzögerung durch das DB-Ethernet bei der Simulation eines Netzwerkes mit 5 Knoten mit berücksichtigt. Die Leitungslänge in dieser Simulation beträgt 1 m. Ebenfalls wurden die Paketumlaufzeit und der maximale Jitter für ein Netzwerk mit fünf Knoten theoretisch bestimmt. Zur Berechnung der Paketumlaufzeit wird die Gleichung (5.4) verwendet. Der theoretische Jitter ergibt sich aus den Erläuterungen in Kapitel 5.4.2. Die Ergebnisse der Messung, der Simulation und der Berechnung sind in Tabelle 5.9 aufgelistet.

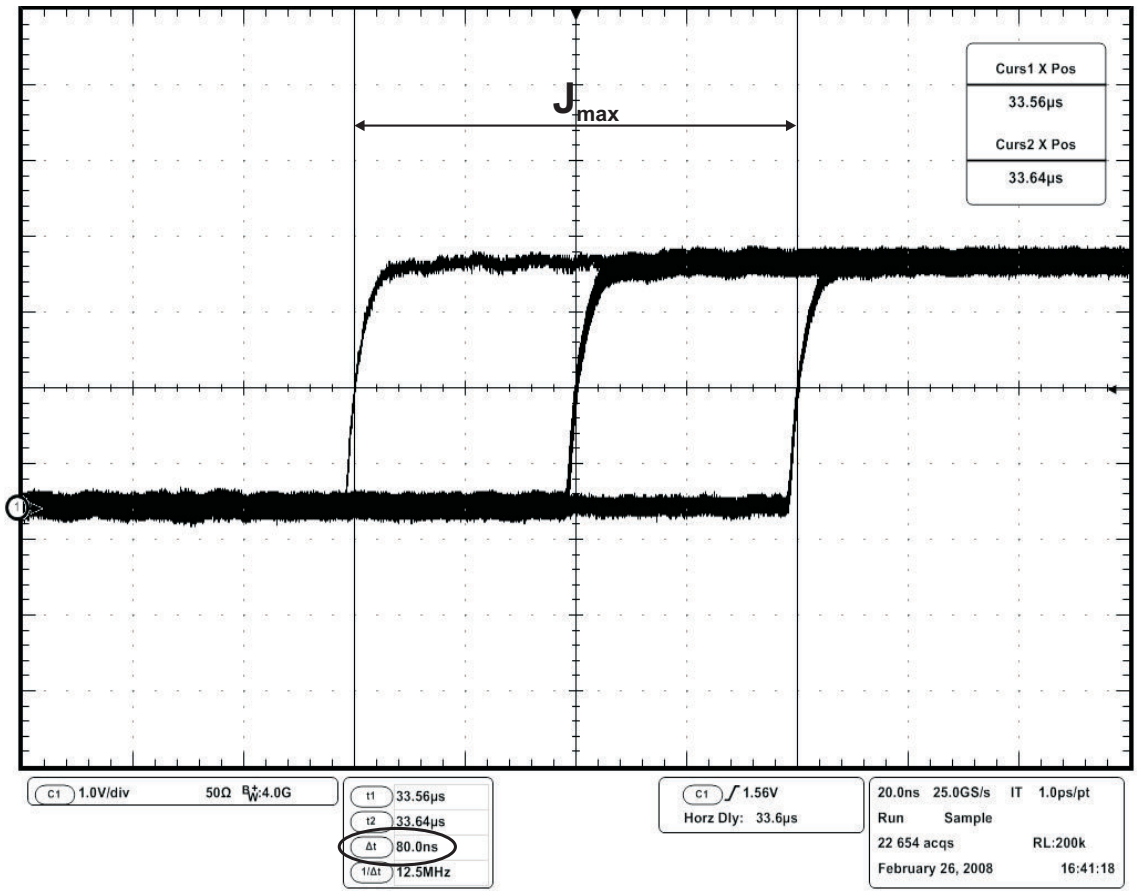


Abbildung 5.16: Oszilloskopaufnahme vom Jitter des Hardware-Switches

	Messung	**Simulation**	**Berechnung**
mittlere Paketumlaufzeit in μs	33,6	33,6	33,6
Jitter in ns	80	140,7	160

Tabelle 5.9: Gegenüberstellung für den Hardware-Switch

Bei der Messung, der Simulation und der Berechnung ist die durchschnittliche Paketumlaufzeit in etwa gleich. Werden alle Nachkommastellen berücksichtigt, ist die gemessene durchschnittliche Paketumlaufzeit um 3 ns größer als der simulierte bzw. der berechnete Wert. An der Dicke der steigenden Taktflanken in Abbildung 5.16 ist zu erkennen, dass der maximale Wert der Paketumlaufzeit häufiger vorkommt als der minimale Wert. Da bei der Simulation die maximalen und minimalen Werte gleichmäßig verteilt sind, ist der Mittelwert der gemessenen Paketumlaufzeit etwas größer als der Mittelwert der simulierten Paketumlaufzeit.

Bei der Simulation eines Netzwerkes mit 5 Knoten über 10.000 Runden beträgt der Jitter 140,7 ns. Theoretisch beträgt das Maximum des simulierten Jitters 160 ns. Durch

die Kompensierung des Jitters durch die IFG beeinflusst nur der Jitter, der bei der Übertragung eines Paketes vom direkten Nachbarknoten zum Zielknoten entsteht, den Jitter der Paketumlaufzeit. Bei der Übertragung eines Paketes zwischen zwei Knoten entsteht ein Jitter von 80 ns. In der Simulation ist der theoretische Jitter doppelt so groß, weil, wie bereits in Kapitel 5.4.2 beschrieben wurde, die Paketumlaufzeit aus der aktuellen und der vorherigen Ankunftszeit des eigenen Paketes ermittelt wird. Der gemessene Jitter beträgt nur 80 ns und entspricht damit dem Jitter, der bei einer Übertragung eines Paketes zwischen zwei Knoten entsteht.

Der in der Messung ermittelte geringe Jitter und die nahezu identischen Werte für die mittlere Paketumlaufzeit bei Messung, Simulation und Berechnung belegen, dass das Self*S*-Protokoll, ausgeführt auf einem Hardware-Switch, deterministisch ist. Alle Zykluszeiten können im Voraus berechnet werden. Bedingt durch den geringen Jitter ist sichergestellt, dass harte Echtzeitbedingungen erfüllt werden.

5.6.3 Ergebnisse Software-Switch

Um die Paketumlaufzeit und den Jitter der Paketumlaufzeit beim Software-Switch zu messen, wurden die fünf Switches des Testsystems als Software-Switch konfiguriert. Auch bei dieser Messung wurden mehr als 10.000 Messwerte der Paketumlaufzeit vom Oszilloskop aufgenommen. Aus den Messwerten wurde eine mittlere Paketumlaufzeit von 142,45 μs ermittelt. Der maximale Jitter der Paketumlaufzeit beträgt 12,5 μs und die Standardabweichung 1,98 μs.

In der Abbildung 5.17 ist das Ergebnis einer zweiten Jittermessung dargestellt. Zur Bestimmung des maximalen periodischen Jitters wird bei dieser Messung, wie auch beim Hardware-Switch, die Anzeigeoption *unendliches Nachleuchten* des Oszilloskopes verwendet. Der maximale periodische Jitter, also das Δt zwischen t_1 und t_2, beträgt bei dieser Messung 12,44 μs. Diese Messung bestätigt ungefähr den Jitter von 12,5 μs der ersten Messung. Der Unterschied liegt im Rahmen der Messungenauigkeit, die beim Einstellen der beiden Messpunkte t_1 und t_2 entstehen kann.

In der Abbildung 5.17 wird auch deutlich, dass im Schwankungsbereich des Jitters nur fünf unterschiedliche Werte für die Paketumlaufzeit existieren. Der Abstand zwischen den steigenden Flanken dieser fünf Werte beträgt ca. 3,1 μs. Dieser Wert entspricht ungefähr dem Jitter der Latenzzeit des im Testsystem verwendeten Software-Switches. Der gemessene Jitter der Latenzzeit dieses Software-Switches beträgt 3,19 μs.

Wird ein Netzwerk, bestehend aus fünf Software-Switches, simuliert, ergibt dies eine durchschnittliche Paketumlaufzeit von 157,5 μs und einen maximalen Jitter von 1,99 μs. Diese Werte liegen einige Mikrosekunden über den gemessenen Werten. Grund hierfür ist, dass in der Simulation die gemessene Latenzzeit und der Jitter (vgl. Kapitel

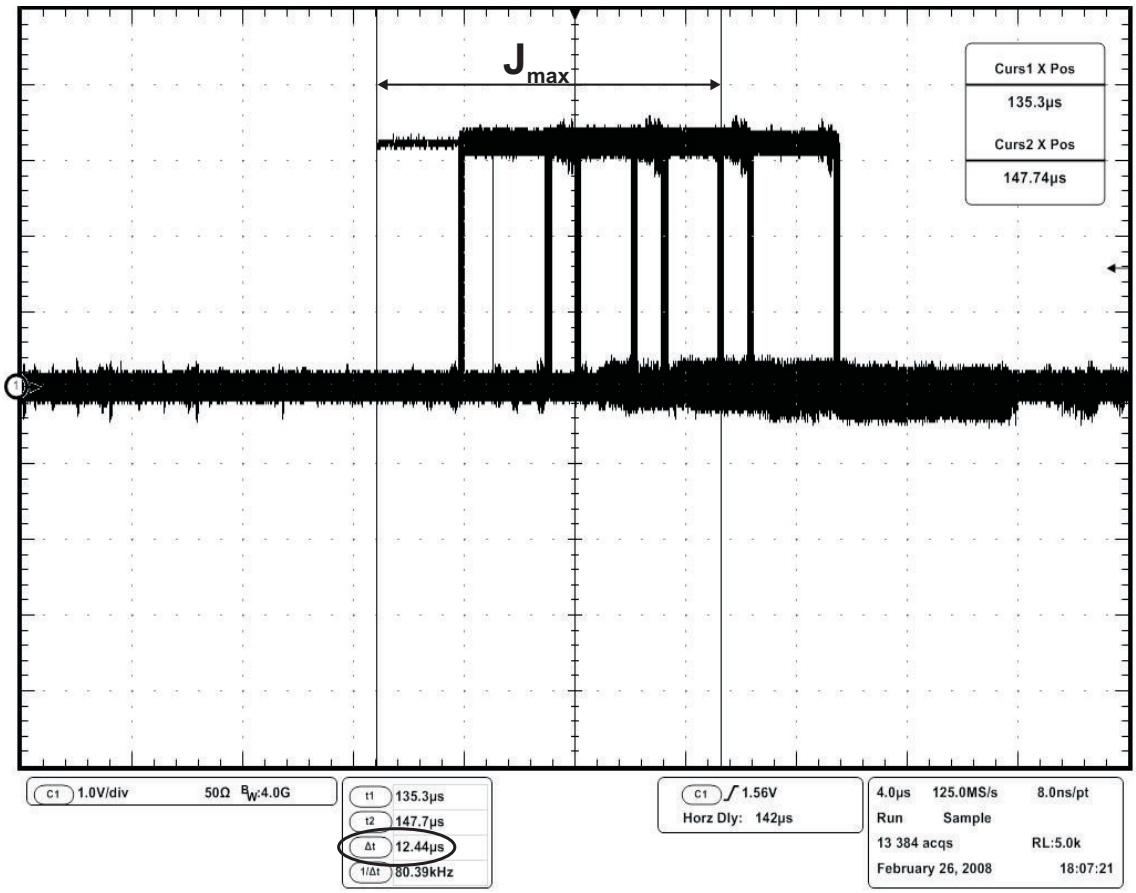


Abbildung 5.17: Oszilloskopaufnahme vom Jitter des Software-Switches

3.3.2) des ursprünglichen Software-Switches verwendet wurden. Wie bereits erwähnt, wird im Testsystem ein für dieses Experiment angepasster Software-Switch eingesetzt. Im Gegensatz zum ursprünglichen Switch werden beim angepassten Switch keine Interrupts benötigt, um ankommende Pakete zu verarbeiten. Der im Experiment eingesetzte Switch verwendet das sogenannte *Polling*, um die Ankunft neuer Pakete periodisch abzufragen. Neben dem Weiterleiten von Paketen überwacht der ursprüngliche Switch zusätzlich die Netzwerklast und die Pufferstände, um gegebenenfalls eine automatische Rekonfiguration einzuleiten. Damit diese Überwachung nicht die Reaktionszeit bei Ankunft eines neuen Paketes erhöht, wurden Interrupts benutzt. In diesem Experiment wird eine automatische Rekonfiguration nicht benötigt. Daher wurden die Überwachungsmechanismen und die Interruptverarbeitung beim angepassten Software-Switch entfernt.

Der ursprüngliche Software-Switch hat eine Verarbeitungszeit von 25 μs und einen Jitter von 4 μs. Die Messung des angepassten Software-Switches resultiert in einer Verarbeitungszeit von 22,15 μs und einem Jitter von 3,19 μs. Zusätzlich muss für einen Vergleich mit den gemessenen Werten in der Simulation die Verzögerungszeit auf dem

	Messung	**Simulation**	**Berechnung**
mittlere Paketumlaufzeit in μs	142,45	143,18	142,48
Jitter in μs	12,5	15,69	16,35

Tabelle 5.10: Gegenüberstellung für den Software-Switch

DB-Ethernet T_{Phy} mit berücksichtigt werden. In der Tabelle 5.10 sind die Ergebnisse der Messung, der Simulation und der Berechnung aufgelistet. Die berechnete Paketumlaufzeit T_{RTT} für den Software-Switch wird in diesem Fall mit

$$T_{\text{RTT}} = n \cdot (T_{\text{Trn}} + T_{\text{Swi}} + T_{\text{Phy}} + T_{\text{Prd}}) \tag{5.9}$$

ermittelt.

Für die Berechnung des Jitters der Paketumlaufzeit wird die Gleichung (5.8) verwendet.

In der Tabelle 5.10 sind der gemessene und der berechnete Wert für die mittlere Paketumlaufzeit bis auf wenige Nanosekunden identisch. Die simulierte Paketumlaufzeit wurde über 10.000 Runden ermittelt. Der Unterschied zwischen simulierter und gemessener durchschnittlicher Paketumlaufzeit liegt ebenfalls im Nanosekundenbereich, ist aber größer als beim berechneten Wert. Bei dem Jitter der Paketumlaufzeit liegt der simulierte Wert näher am gemessen Wert als der berechnete Wert. Allerdings sind sowohl der simulierte als auch der berechnete Jitter um über $3\,\mu$s größer. Berechnet wird der Jitter nach der Gleichung (5.8). Dabei wird der maximale Jitter jeder einzelnen Netzwerkkomponente addiert. In den durchgeführten Messungen ist der Fall nicht aufgetreten, dass alle fünf Switches in derselben Runde eines Paketes ihren maximalen Jitter erreichen. Daher ist der gemessene Jitter geringer als der berechnete Jitter.

Im Gegensatz zum Hardware-Switch weichen die gemessen, simulierten und berechneten Werte für die mittlere Paketumlaufzeit voneinander ab. Diese Abweichung ist zwar gering, aber die Zykluszeiten für das Self*S*-Protokoll können beim Software-Switch im Voraus nicht genauso exakt bestimmt werden, wie beim Hardware-Switch. Der hohe Jitter und die höhere Paketumlaufzeit sorgen dafür, dass ein Self*S*-Protokoll mit nur fünf Software-Switches nicht mehr die höchste Echtzeitklasse (vgl. Kapitel 2.5.1) erfüllt. Harte Echtzeiteigenschaften können insbesondere bei einer größeren Anzahl von Software-Switches nicht mehr erfüllt werden.

5.7 Vergleichende Leistungsbewertung

In den vorangegangenen Abschnitten wurden die Paketumlaufzeit, der Jitter und die Latenz des Self*S*-Protokolls eingehend analysiert. In diesem Abschnitt sollen die Echtzeiteigenschaften von der Ethernet-basierten Self*S*-Lösung mit den in Kapitel 2.5 vorgestellten Echtzeit-Protokollen PROFINET IRT und EtherCAT verglichen werden. In der Leistungsbewertung der standardisierten Echtzeitprotokolle in Kapitel 2.6 hat PROFINET von den Producer/Consumer- Protokollen und EtherCAT von den Master-Slave-Protokollen die höchste Leistungsfähigkeit erreicht. Daher wurden diese beiden Protokolle für den Vergleich mit Self*S* ausgewählt. Als Topologie wird für Self*S* der Ring und für PROFINET und EtherCAT eine Linie gewählt, wie sie in Abbildung 2.19 dargestellt ist. Die Kabellänge zwischen jedem Knoten beträgt für den Vergleich 10 m.

5.7.1 Zykluszeit

Ein Vorteil vom Self*S*-Protokoll ist die Möglichkeit, dass alle Knoten in einem Ring innerhalb eines Zyklus ein Paket übertragen und jeweils eine Antwort vom Zielknoten des Paketes erhalten. Wird bei der Ringtopologie ein Paket in beide Richtungen gesendet, kann jeder Knoten in einem Zyklus sogar zwei Pakete senden. Für ein einheitliches Kommunikationsszenario werden alle Pakete zum selben Zielknoten übertragen, der daraufhin ein Paket zum jeweiligen Quellknoten sendet. Gewählt wird als Zielknoten der erste Knoten der Linie und der erste Knoten des Ringes. Der Zielknoten entspricht hier der Steuerung im Kommunikationsszenario aus Kapitel 2.6. Beide Kommunikationsszenarien sind daher identisch. Die Zykluszeit von PROFINET kann nach der Gleichung (2.27) und die Zykluszeit von EtherCAT nach der Gleichung (2.29) bestimmt werden. Beim Self*S*-Protokoll entspricht die minimale Zykluszeit der Paketumlaufzeit und wird nach der Gleichung (5.4) bestimmt.

Die Ergebnisse des Vergleichs der minimalen Zykluszeit für eine variierende Nutzdatenmenge zeigt die Abbildung 5.18. Bei kleinen Nutzdatenmengen ist die Zykluszeit von EtherCAT etwas geringer als die vom Self*S*-Protokoll. Bei EtherCAT werden alle Nutzdaten bis zur maximalen Paketlänge in ein Ethernet-Paket eingebettet. Aus diesem Grund müssen bei EtherCAT weniger Daten übertragen werden und die Zykluszeit ist etwas geringer. Bei großen Nutzdatenmengen ist die Paketlänge und damit auch die Zykluszeit in etwa gleich groß. Obwohl PROFINET Cut-Throught-Switches verwendet, sind die Zykluszeiten von Self*S* und PROFINET in etwa gleich groß. Store-and-Forward-Switches haben zwar eine höhere Weiterleitungszeit eines Paketes als Cut-Through-Switches. Trotzdem ist die Zykluszeit von PROFINET nur bei großen Nutzdatenmengen etwas geringer als die Zykluszeit vom Self*S*-Protokoll. Die unter-

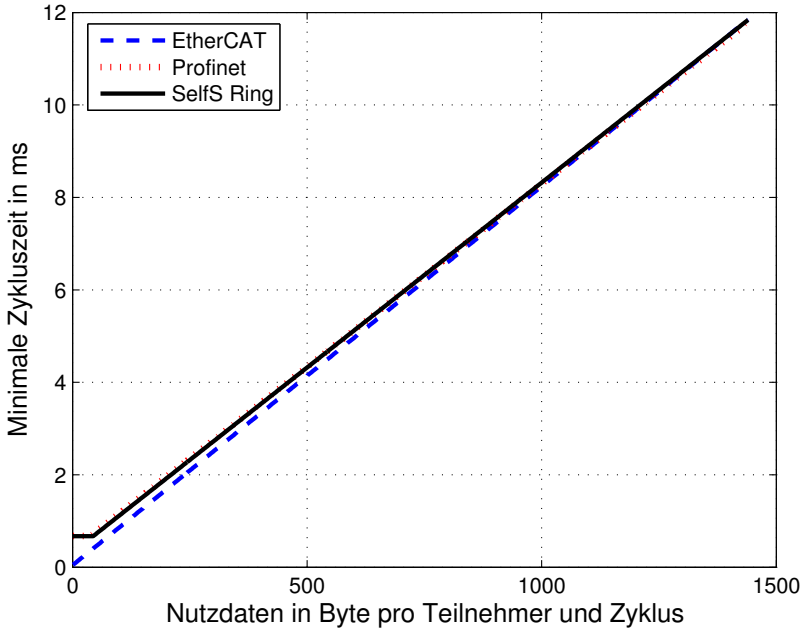


Abbildung 5.18: Berechnete minimale Zykluszeiten in einem Netzwerk mit 100 Knoten

schiedlichen Switch-Architekturen haben keine Auswirkung auf die Zykluszeit, weil auch bei PROFINET im optimalen Fall die Pakete von den Teilnehmern zur Steuerung gleichzeitig übertragen werden. In diesem Fall muss auch beim Cut-Through-Switch ein empfangenes Paket auf den Abschluss der Übertragung des vorangegangenen Paketes warten. Die Weiterleitungszeit von einem Cut-Through-Switch und von einem Store-and-Forward-Switch ist daher gleichgroß.

Aufgrund der Ringstruktur muss ein Paket bei Self*S* über eine Kante mehr als bei einer Linienstruktur übertragen werden. Daher ist die Zykluszeit von Self*S* bei großen Nutzdatenmengen größer als bei PROFINET. Bei den in der Automatisierung häufiger benötigten kleinen Nutzdatenmengen ist die Zykluszeit von Self*S* jedoch geringer als die von PROFINET. Bei PROFINET ist der Zyklus nämlich erst abgeschlossen, nachdem ein zusätzliches Paket zur Uhrensynchronisation versendet wurde. Bei großen Nutzdatenmengen hat die zusätzliche Übertragung über eine weitere Kante aber eine größere Auswirkung als die Übertragungsdauer des Synchronisationspaketes.

Bei etwa gleichgroßen Zykluszeiten hat Self*S*-Protokoll gegenüber PROFINET und EtherCAT einen weiteren Vorteil. Aufgrund der Ringstruktur können beim Self*S*-Protokoll die doppelte Datenmengen übertragen werden, wenn jeder Knoten jeweils ein Paket in beide Richtungen des Ringes überträgt.

Wird das Self*S*-Protokoll nicht in einer Ring-Topologie eingesetzt, sondern in einer beliebigen Topologie als virtueller Ring, sind bei gleicher Anzahl von Knoten die Zy-

kluszeitzeit von Self$S$ höher als bei PROFINET. In einem virtuellen Ring werden nicht n Pakete, sondern $2 \cdot (n-1)$ Pakete übertragen, um den Zyklus abzuschließen. Für die Echtzeitübertragung sind aber nur n Pakete notwendig. In diesem Fall können also $n-2$ NRT-Pakete zusätzlich versendet werden. Bei PROFINET werden die NRT-Pakete in der offenen Phase übertragen. Die Dauer der offenen Phase wird im Vorfeld festgelegt und verlängert den gesamten PROFINET-Zyklus. Es wird angenommen, dass in einem Netzwerk mit 100 Knoten im selben Kommunikationsszenario vom Self$S$-Protokoll mit virtuellem Ring und von PROFINET in einem Zyklus zusätzlich 98 NRT-Pakete gesendet werden. In diesem Fall beträgt die Zykluszeit bei minimaler Paketlänge von Self$S$ 16,6 ms und von PROFINET 16,9 ms.

5.7.2 Latenzzeit

Als Latenzzeit wird in diesem Vergleich die Zeit ermittelt, die bei der Übertragung eines Paketes vom ersten Knoten bis zum weit entferntesten Knoten entsteht. Für PROFINET und EtherCAT wird diese Zeit bestimmt durch:

$$T_{\mathrm{Lat}} = T_{\mathrm{Trn}} + (n-1) \cdot (T_{\mathrm{Prd}} + T_{\mathrm{Cut}}) \tag{5.10}$$

Bei einer PROFINET-Komponente ist T_{Cut} die Verarbeitungszeit eines Paketes durch den ERTEC400.

Die Verzögerungszeiten bei EtherCAT unterscheiden sich, wie in Kapitel 2.6.6 erläutert, auf Hin- und Rückweg. Bei der Bestimmung der Latenzzeit wird nur die Verzögerung in Hinrichtung berücksichtigt werden. Des Weiteren wird bei EtherCAT nur die erste Station physikalisch über Ethernet mit dem Master verbunden. Die anderen Stationen sind an den EBUS angeschlossen. Daher unterscheiden sich die Verarbeitungszeiten der ersten Station $T_{\mathrm{S1_{to}}}$ von der Verarbeitungszeit der übrigen Stationen $T_{\mathrm{S_{to}}}$. Die Latenz von EtherCAT wird mit

$$T_{\mathrm{Lat}} = T_{\mathrm{Trn}} + T_{\mathrm{S1_{to}}} + (n-1) \cdot T_{\mathrm{Prd}} + (n-1) \cdot T_{\mathrm{S_{to}}} \tag{5.11}$$

ermittelt. Bei Self$S$ wird die Latenz mithilfe der Gleichung (5.6) bestimmt. In diesem Vergleichsszenario ist die Anzahl der Knoten zwischen der Quelle und dem Ziel $u = n-1$.

Die Ergebnisse der berechneten Latenzzeiten für die minimale Nutzdatenmenge von 1 Byte und für die maximale Nutzdatenmenge von 1440 Byte sind in der Tabelle 5.11

	Nutzdaten in Byte	**Latenzzeit in ms**
EtherCAT	1	0,03
PROFINET	1	0,31
Self*S*	1	0,66
EtherCAT	1440	0,14
PROFINET	1440	0,42
Self*S*	1440	11,72

Tabelle 5.11: Berechnete Latenzzeiten in einem Netzwerk mit 100 Knoten

aufgelistet. Die 1440 Byte entsprechen der maximalen Nutzdatenmenge von PROFINET.

Die Latenzzeiten von EtherCAT und PROFINET sind insbesondere bei großen Paketlängen deutlich geringer als die Latenzzeit des Self*S*-Protokolls. Grund dafür sind die Weiterleitungsverzögerungen der Store-and-Forward-Switches, die bei einer Erhöhung der Paketlänge ebenfalls ansteigt. Die geringste Latenz hat EtherCAT, weil jeder Slave die Übertragung des Paketes nur geringfügig verzögert.

5.7.3 Jitter

In diesem Vergleich wird der maximal mögliche Jitter der Latenzzeit ermittelt. Bestimmt wird der Jitter von PROFINET und EtherCAT durch:

$$J_{\mathrm{Max}} = 2 \cdot u \cdot J_{\mathrm{Port}} \tag{5.12}$$

Bei PROFINET beträgt der Portjitter 40 ns. Bei EtherCAT werden die Daten entweder über eine Ethernet-Physik oder über den EBUS übertragen. Der Portjitter beträgt bei einer Anbindung über eine Ethernet-Physik 40 ns und bei einer Verbindung über den EBUS 10 ns. In der Tabelle 5.12 sind die Ergebnisse für den Jitter aufgelistet, der bei einer Übertragung eines Paketes über 10 und 100 Knoten entsteht.

Den geringsten maximalen Jitter von nur 80 ns hat das Self*S*-Protokoll. Aufgrund der in Kapitel 5.4.3 beschriebenen Kompensierung bleibt der Jitter beim Self*S*-Protokoll konstant, auch wenn die Anzahl der Knoten, über die ein Paket übertragen wird, ansteigt. Auch in einem virtuellen Ring übersteigt der Jitter beim Self*S*-Protokoll diesen Wert nicht. Im Gegensatz dazu steigt bei EtherCAT und PROFINET der Jitter deutlich an, wenn die Anzahl der Knoten erhöht wird. Der PROFINET-Switch hat im Vergleich zum EtherCAT-Slave einen höheren Jitter. Daher ist auch der Jitter der Latenz bei PROFINET größer als bei EtherCAT.

	Anzahl der Knoten	Jitter in μs
EtherCAT	10	0,13
PROFINET	10	0,36
Self*S*	10	0,08
EtherCAT	100	1,03
PROFINET	100	3,96
Self*S*	100	0,08

Tabelle 5.12: Gegenüberstellung des berechneten Jitters

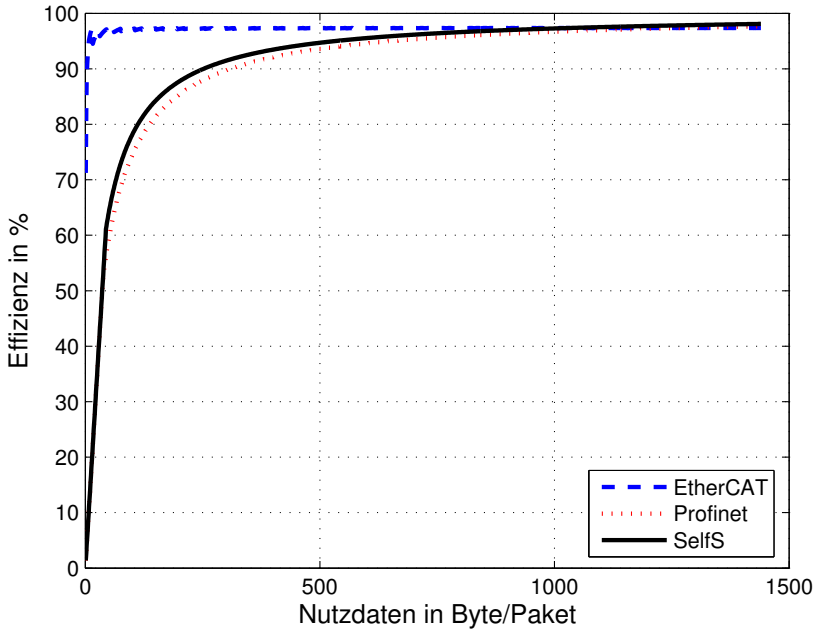


Abbildung 5.19: Vergleich der Protokolleffizienz

5.7.4 Protokolleffizienz

Die Protokolleffizienz für PROFINET wird bestimmt nach der Gleichung (2.28) und für EtherCAT nach der Gleichung (2.30). Beim Self*S*-Protokoll wird die Protokolleffizienz mithilfe der Gleichung (2.13) berechnet. Die Ergebnisse der Berechnung der Protokolleffizienz für eine variierende Nutzdatenmenge ist in der Abbildung 5.19 dargestellt.

Das Self*S*-Protokoll hat aufgrund eines kleineren Paketkopfes und des Wegfalls von gesonderten Paketen zur Uhrensynchronisation eine höhere Protokolleffizienz als PROFINET. Solange die Nutzdaten nicht ausreichen, damit das Ethernet-Paket seine minimale Länge erreicht, sind die Protokolleffizienzen von Self*S* und PROFINET scheinbar gleich. Das Synchronisierungspaket, welches nach jedem Zyklus bei PROFINET gesen-

det wird, verringert die Protokolleffizienz von PROFINET gegenüber der Protokolleffizienz von Self*S* auch in diesem Fall. Insbesondere bei kleinen Nutzdatenmengen ist die Protokolleffizienz von EtherCAT am größten, weil bei EtherCAT die Nutzdaten aller Teilnehmer in ein Paket integriert werden. Als Erweiterung könnten auch bei Self*S* in einem Paket Nutzdaten für mehrere Knoten integriert werden. Als Zieladresse müsste eine *Broadcast*-Adresse verwendet werden, damit alle Knoten die empfangenen Daten an den Prozessor weitergeben. Eine zusätzliche Adressierung wird notwendig, wenn nur ein Teil der Nutzdaten für einen Knoten bestimmt ist. Die zusätzlichen Adressen führen aber wieder zu einer Verringerung der Protokolleffizienz.

5.8 Protokollerweiterungen

Im bisherigen Teil dieses Kapitels sind die Synchronisationseigenschaften und die Echtzeiteigenschaften des Self*S*-Protokolls analysiert worden. Bisher nicht diskutiert wurden mögliche Protokollerweiterungen, die die Wahl der Zykluszeit erleichtern und die sich mit der Fehlertoleranz der Kommunikation befassen.

5.8.1 Variation der Basiszykluszeit

Bei Self*S* kann die Zykluszeit jedes Knotens als ein beliebiges Vielfaches der Paketumlaufzeit gewählt werden. Allerdings kann es vorkommen, dass in einem System eine Basiszykluszeit gefordert wird, die z. B. zwischen der Paketumlaufzeit und der zweifachen Paketumlaufzeit liegt. Die Basiszykluszeit ist die geringste Zykluszeit eines Systems. Wie bereits in Kapitel 4.2.2 beschrieben, werden auch in der digitalen Regelung Zykluszeiten als ein Vielfaches einer Basiszykluszeit erzeugt. Um eine geforderte Basiszykluszeit zu erhalten, muss beim Self*S*-Protokoll die Paketumlaufzeit angepasst werden.

Außer in der Planungsphase ist die Anzahl der Knoten in einem Netzwerk eine konstante Größe. Als Parameter zur Anpassung der Paketumlaufzeit bleibt nach Gleichung (5.4) nur die Übertragungsverzögerung, die über die Paketlänge bestimmt wird und die Dauer einer IFG. Die Paketlänge zu erhöhen, um damit eine Basiszykluszeit einzustellen, hat den Vorteil, dass gleichzeitig die Nutzdatenmenge, die mit einem Paket übertragen wird, erhöht wird. Während die Paketlänge im Normalfall nur byteweise erhöht wird, kann die Dauer einer IFG nur um eine Bitzeit erhöht werden. Die Bitzeit ist die Zeit, die eine Netzwerkkomponente benötigt, um ein Bit bei einer bestimmten Datenrate zu übertragen. Jedoch kann die IFG nicht nur für einen Knoten erhöht werden, sondern die IFG muss bei allen Knoten gleich groß sein. Ein Knoten mit großer IFG erzeugt eine große Lücke zwischen jedem Paket. Durch die große Lücke zwischen

den Paketen ist eine kleinere IFG eines anderen Knotens bereits abgelaufen, bevor das nächste Paket diesen Knoten erreicht. Der Portjitter kann in diesem Fall nicht kompensiert werden. Die Paketlänge kann im Gegensatz zur IFG für jeden Knoten einzeln festgelegt werden. Allerdings sind unterschiedliche Paketlängen nur erforderlich, wenn die Basiszykluszeit mit einer einheitlichen Paketlänge nicht zu erreichen ist. Die Erhöhung der Paketlänge von nur einem Paket ermöglicht also eine genauere Einstellung der Paketumlaufzeit. In diesem Fall wird ein Knoten so konfiguriert, dass er sein eigenes Paket mit einer größeren Paketlänge generiert. Alle anderen Pakete, die der Knoten weiterleitet, werden nicht verändert. Die Paketumlaufzeit wird bei unterschiedlichen Paketlängen durch

$$T_{\mathrm{RTT}} = \sum_{i=1}^{n} T_{\mathrm{Trn}_i} + T_{\mathrm{IFG}} \tag{5.13}$$

bestimmt. T_{Trn_i} ist die Übertragungsverzögerung, die bei der Übertragung des Paketes entsteht, das von dem Knoten i erstellt wurde.

Die Basiszykluszeit bzw. Paketumlaufzeit kann z. B. vor der Inbetriebnahme des Netzes ermittelt und die Parameter für die IFG und die Paketlänge eingestellt werden. Ebenfalls möglich ist eine Veränderung der Paketumlaufzeit im Betrieb. Dazu ist es gegebenenfalls sogar ausreichend, dass nur ein Knoten den Parameter für die Paketlänge verändert. Allerdings resultiert eine Veränderung der Paketumlaufzeit im Betrieb in einen erneuten Einschwingvorgang und wird kurzzeitig den Jitter erhöhen. Über die Veränderung der Paketumlaufzeit kann eine beliebige Basiszykluszeit eingestellt werden, die größer ist als die Paketumlaufzeit bei minimaler Paketlänge und bei einer Standard-IFG.

5.8.2 Fehlertoleranz

Eine wichtige Eigenschaft eines Echtzeitkommunikations-Protokolls ist die Fehlertoleranz. In diesem Abschnitt werden einige Möglichkeiten diskutiert, um die Fehlertoleranz von Self$S$ zu erhöhen.

Abbruch der Verbindung: Wird eine physikalische Verbindung durch eine Störung oder einen Kabelbruch unterbrochen, kann kein Paket den Ring durchlaufen und es ist keine Kommunikation im Netzwerk möglich. Werden nur in einer Richtung Echtzeitdaten übertragen, kann die Kommunikation bei einem Kabelbruch weitergeführt werden, wenn der Ring an beiden Knoten, die zuvor über das defekte Kabel verbunden sind, wieder geschlossen wird. Dazu müssen beide Knoten, nachdem sie den Verbindungsabbruch erkannt haben, alle eingehenden Pakete an denselben Port weiterleiten,

über den das Paket empfangen wurde. Die Pakete werden also über die Sendeleitung an den Knoten zurückgeschickt, von dem das Paket zuvor übertragen wurde. Auf diese Weise entsteht ein neuer virtueller Ring. Dieses Verfahren wird auf ähnliche Weise auch bei EtherCAT verwendet [JB03b].

Zwar ist der Ring wieder geschlossen, aber die Paketumlaufzeit erhöht sich, weil die Pakete nicht mehr über n Kanten, sondern über $2 \cdot (n-1)$ Kanten übertragen werden. Wird die Basiszykluszeit schon von Beginn an so gewählt, dass sie im virtuellen Ring eingehalten wird, dann kann die Paketlänge oder die IFG verringert werden und jeder Knoten kann wie zuvor in derselben Zykluszeit sein Echtzeitpaket übertragen. Auch bei einem Ausfall eines Knotens kann dieses Verfahren verwendet werden, um den Ring wieder zu schließen und um die Kommunikation weiter zu ermöglichen.

Paketverlust: Der Verlust eines Paketes führt dazu, dass ein Knoten keine eigenen Echtzeitpakete mehr generieren kann. Des Weiteren erhöht der Paketverlust den gesamten Jitter, weil durch den Paketverlust eine Lücke entsteht, die größer als eine IFG ist. Daher kann nicht mehr bei jedem Paket der Jitter kompensiert werden. Ein Paketverlust entsteht, wenn ein oder mehrere Bits bei der Übertragung durch Störungen gekippt werden. Empfängt ein Switch ein nicht korrekt übertragenes Paket, erkennt der Switch den Fehler mithilfe des CRCs.

Normalerweise verwirft der Switch ein Paket, dessen CRC nicht korrekt ist. Damit beim Self*S*-Protokoll in diesem Fall keine große Lücke zwischen den Paketen entsteht, sendet jeder Switch, der ein fehlerhaftes Paket erkannt hat, anstelle des fehlerhaften Paketes ein leeres Paket. Würde der CRC ignoriert und das fehlerhafte Paket weitergeleitet, würde dies z. B. dazu führen, dass der Zielknoten falsche Daten verarbeitet oder ein falscher Knoten adressiert wird.

Durch das Einfügen eines leeren Paketes wird der Jitter durch eine Störung bei der Übertragung nicht beeinflusst. Dieses leere Paket kann jedoch nicht die Paket-ID des fehlerhaften Paketes erhalten, weil auch die Paket-ID ihren Wert verändert haben könnte. Aus diesem Grund werden alle 14 Bits der Paket-ID auf 1 gesetzt. Alle Knoten erkennen diese Paket-ID als Fehler-ID. Die ursprüngliche Paket-ID wird mithilfe der Paketreihenfolge vom Quellknoten der Paket-ID wieder hergestellt. Ein Knoten ersetzt die Fehler-ID durch seine eigene Paket-ID, nachdem der Knoten die Paket-ID des Paketes erkannt hat, das immer vor seinem eigenen Paket übertragen wurde. Durch dieses Verfahren kann auch bei einem Paketverlust jeder Knoten in jeder Runde ein eigenes Echtzeitpaket generieren.

5.9 Zusammenfassung

Das Self*S*-Protokoll basiert auf der Verwendung von Store-and-Forward-Switches. Dadurch wird eine Portierung von Self*S* auf Ethernet erleichtert. Für die Portierung auf Ethernet muss das Self*S*-Protokoll in einigen Bereichen angepasst werden. Einer der wichtigsten Schritte ist die Integration von Self*S* in das Ethernet-Paketformat und die Erweiterung des Paketformats um zusätzliche Felder, die vom Ethernet-Standard nicht vorgesehen sind. Des Weiteren muss auch die Erkennung des Synchronisationsstatus aufgrund der IFG angepasst werden.

Von zentraler Bedeutung für das Self*S*-Protokoll ist die verwendete Netzwerkkomponente. Daher wird detailliert erläutert, wie die Pakete bei einem Ethernet-basierten Self*S*-Protokoll von einem Ethernet-Switch weitergeleitet werden müssen. Für Self*S* notwendige Erweiterungen des Switches wurden am Beispiel des in dieser Arbeit entwickelten Hardware-Switches verdeutlicht.

Wichtigster Punkt der Portierung von Self*S* auf Ethernet ist die Analyse des Einflusses von Ethernet auf den Prozess der Selbstsynchronisation und die Echtzeiteigenschaften des Self*S*-Protokolls. Durchgeführt wurde diese Analyse mithilfe einer rechnergestützten Netzwerksimulation. Basis der Netzwerksimulation ist das Modell des Ethernet-Übertragungskanals und eines Ethernet-Switches. Das Modell des Ethernet-Switches berücksichtigt die IFG und den Portjitter und basiert auf den realen Parametern des in dieser Arbeit entwickelten Hardware- und Software-Switches.

Mithilfe der Netzwerksimulationen konnte gezeigt werden, dass das Protokoll trotz des Portjitters und der IFG seine selbstsynchronisierenden Eigenschaften behält und Zykluszeiten als ein Vielfaches der Paketumlaufzeit garantiert werden können. Im Falle des Hardware-Switches konnte sogar nachgewiesen werden, dass sich der Jitter der Paketumlaufzeit nicht mit der Anzahl der Knoten aufaddiert, sondern dass der Jitter unabhängig von der Anzahl der Knoten einen maximalen Wert von nur 160 ns erreicht. Beim Software-Switch ist der Jitter allerdings größer und ist abhängig von der Anzahl der Knoten. Die höhere Verarbeitungszeit und der Jitter der Verarbeitungszeit verhindern eine Kompensierung des Portjitters. Sie führen gegenüber dem Hardware-Switch nicht nur zu einem größeren Jitter, sondern auch zu einer höheren Paketumlaufzeit und zu einer höheren Latenzzeit. Bestätigt wurden die Ergebnisse aus der Simulation mittels eines experimentellen Netzwerkes, das aus den prototypischen Implementierungen von fünf Hardware- und Software-Switches aufgebaut wurde. Für dieses experimentelle Netzwerk wurde die in Kapitel 3.2 beschriebene Umsetzung des Hardware- und des Software-Switches angepasst.

Die Flexibilität und die Adaptivität des Self*S*-Protokoll führen nicht zu einer geringen Leitungsfähigkeit des Protokolls. Im Vergleich von Self*S*, PROFINET und EtherCAT konnte gezeigt werden, dass das Self*S*-Protokoll auch mit den leistungsfähigsten

Ethernet-basierten Echtzeitprotokollen konkurrieren kann. Im Bereich der minimalen Zykluszeiten sind alle drei Protokolle in etwa gleich. Insbesondere bei kleinen Nutzdatenmengen ist die Protokolleffizienz von EtherCAT am größten. Steigen die Nutzdatenmengen an, überschreitet die Protokolleffizienz von PROFINET und Self*S* die von EtherCAT. Nur die Latenzzeit ist bei Self*S* größer als bei den anderen beiden Protokollen. Jedoch kann der sehr geringe Jitter des Self*S*-Protokolls von keinem der anderen beiden Protokolle unterboten werden.

Darüber hinaus wurden einige Protokollerweiterungen vorgestellt, mit denen die Redundanz erhöht und die Flexibilität der einstellbaren Zykluszeiten vergrößert werden kann.

Kapitel 6

Zusammenfassung und Ausblick

Echtzeitkommunikationsnetze werden zur Vernetzung verteilter Sensoren, Aktoren und Steuerungen eingesetzt. Die Entwicklung von Echtzeitkommunikationsnetzen begann mit der Einführung der Feldbusse, die zwar Echtzeitanforderungen garantierten, aber nur über eine sehr geringe Datenrate verfügten. Höheren Anforderungen an die Kommunikation und einer gestiegenen Anzahl von Knoten sind die Feldbusse nicht mehr gewachsen. Ethernet-Netzwerke verfügen über eine hohe Bandbreite und sind seit vielen Jahren im Bürobereich etabliert. Die hohe Bandbreite und die hohe Verfügbarkeit von Ethernet-Komponenten führten zu der Entwicklung von Ethernet-basierten Echtzeitkommunikationsnetzen, die harte Echtzeitanforderungen erfüllen. Insbesondere für harte Echtzeitanforderungen sind komplexe Planungswerkzeuge erforderlich, die die Kommunikation im Vorfeld planen, optimieren und den Ablauf der Übertragungen festlegen.

In Zukunft wird nicht nur die Anzahl der Komponenten in einem verteilten System ansteigen, sondern verteilte Systeme werden in der Lage sein, sich dynamisch an neue Anforderungen aus der Umwelt oder des Benutzers eigenständig anzupassen. In solchen verteilten selbstoptimierenden Systemen werden sich auch die Anforderungen an das Echtzeitkommunikationsnetz bei jeder Anpassung des Systems dynamisch verändern. Eine Planung der Kommunikation im Vorfeld ist bei einem selbstoptimierenden System nicht mehr möglich. Daher werden neue adaptive Echtzeitkommunikationsnetze benötigt, die sich im Betrieb an sich ändernde Echtzeitanforderungen anpassen. In dieser Arbeit wurde die Adaption eines Echtzeitkommunikationsnetzes auf der Ebene der Netzwerkkomponente und auf der Protokollebene untersucht.

Adaption auf der Ebene der Netzwerkkomponente bedeutet die Anpassung des Ressourcenbedarfs der Netzwerkkomponente an die Kommunikationslast und an die Echtzeitanforderungen. Bei geringer Kommunikationslast können so nicht benötigte Ressourcen anderen Applikationen zur Verfügung gestellt werden. Die partielle dynamische Rekonfiguration eines FPGAs ermöglicht eine Anpassung eines Teils einer Hardwarearchitektur, während der Rest der Architektur ungestört weiterarbeitet. Weil Netzwerkkomponenten in Teilen immer eine Hardwareimplementierung erfordern, ist die dynamisch partielle Rekonfiguration eine ideale Plattform, um den Ressourcenbedarf einer Netzwerkkomponente zu verändern. Auf der Basis der partiellen dyna-

mischen Rekonfiguration ist ein rekonfigurierbarer Ethernet-Switch entstanden, der seinen Ressourcenbedarf auf dem FPGA an die Anforderungen anpasst. Bei geringen Echtzeitanforderungen und geringer Kommunikationslast werden die Pakete vom rekonfigurierbaren Switch mittels einer Softwarelösung weitergeleitet. Auf dem FPGA werden in diesem Fall nur zwei einfache Netzwerkschnittstellen benötigt, die nur wenig FPGA-Ressourcen einnehmen. Die restlichen Ressourcen können von anderen Applikationen verwendet werden. Steigt die Netzwerklast an, wird, bevor der Software-Switch die Kommunikationslast nicht mehr bewältigen kann und Pakete verloren gehen, automatisch ein integrierter Hardware-Switch durch eine partielle Rekonfiguration auf das FPGA geladen. Der Hardware-Switch benötigt mehr FPGA-Ressourcen, die anderen Applikationen nun nicht mehr zur Verfügung stehen. Aber gegenüber dem Software-Switch erreicht der Hardware-Switch eine höhere Datenrate und die Echtzeiteigenschaften, z.B. Latenz und Jitter, sind beim Hardware-Switch wesentlich besser. Allerdings kann ein Switch nicht einfach durch eine andere Variante ersetzt werden, weil sonst die Pakete, die gerade übertragen werden oder sich noch in den Puffern befinden, verloren gingen. Aus diesem Grund wurden spezielle Rekonfigurationsstrategien entwickelt, die einen Paketverlust bei einer Rekonfiguration des Switches verhindern. Die Echtzeiteigenschaften und der FPGA-Ressourcenbedarf beider Switchvarianten wurden anhand einer prototypischen Implementierung des rekonfigurierbaren Switches untersucht. Die Ergebnisse der Untersuchung belegen die besseren Echtzeiteigenschaften und den höheren Ressourcenbedarf des Hardware-Switches gegenüber dem Software-Switch. Bei großen FPGAs ist der prozentuale Unterschied der belegten Ressourcen zwischen Hardware- und Software-Switch gering. In diesem Fall rentiert sich die Rekonfiguration nicht und es ist ausreichend, nur den Hardware-Switch zu verwenden. Aber insbesondere bei kleinen FPGAs lohnt sich der Einsatz des rekonfigurierbaren Switches, um FPGA-Ressourcen anderen Applikationen zur Verfügung zu stellen, wenn die Kommunikationsanforderungen gering sind.

Für eine Adaption auf Protokollebene sind Verfahren notwendig, die ohne eine aufwendige Planungsphase harte Echtzeitanforderungen garantieren. Zusätzlich müssen diese Protokolle gewährleisten, dass die Adaption einer Echtzeitanforderung die anderen Echtzeitanforderungen nicht beeinflusst. Das Self$S$-Protokoll erfüllt diese Anforderungen an ein verteiltes adaptives Echtzeitprotokoll. Self$S$ wurde für virtuelle Ringtopologien entwickelt. Das Protokoll ist selbstsynchronisierend und benötigt daher keine explizite Uhrensynchronisation. Bei Self$S$ können die Zykluszeiten von jedem einzelnen Knoten als ein beliebiges Vielfaches der Paketumlaufzeit gewählt und im Betrieb verändert werden. Die Bandbreite, die die Echtzeitzeitkommunikation nicht benötigt, wird bei Self$S$ für die Übertragung von nicht echtzeitkritischen Daten bereitgestellt. Somit muss im Gegensatz zu den zeitgesteuerten Echtzeitprotokollen keine zusätzliche Phase für die Übertragung der nicht echtzeitkritischen Daten eingeplant werden. Die Bandbreite für die NRT-Kommunikation wird dynamisch zugeteilt. Steigt z. B. der

Bandbreitenbedarf für die Echtzeitkommunikation, verringert sich die Bandbreite, die der NRT-Kommunikation zur Verfügung steht.

Eine obere Schranke für den Prozess der Selbstsynchronisation konnte mithilfe eines mathematischen Modells für die Ringtopologie nachgewiesen werden. Des Weiteren wurde mittels des Modells die Jitterfreiheit von Self*S* auf Protokollebene belegt. Sowohl der Prozess der Selbstsynchronisation als auch die Jitterfreiheit des Protokolls wird weder von einer simultanen Übertragung von nicht echtzeitkritischen Daten noch von der Adaption der Zykluszeit von einem oder von mehreren Knoten beeinflusst. Diese Ergebnisse gelten nicht nur für eine einfache Ringtopologie, sondern auch für einen virtuellen Ring, der in beliebige Topologien eingebettet wird.

Der Ethernet-Standard bildet die Grundlage für viele leistungsfähige Echtzeitprotokolle. Zwar sind das Self*S*-Protokoll und das mathematische Modell unabhängig von der physikalischen Netzwerkinfrastruktur, sie basieren aber auf derselben Switch-Architektur wie der rekonfigurierbare Ethernet-Switch. Daher sind auch nur einige Anpassungen notwendig, um Self*S* auf Ethernet zu portieren. Mithilfe einer rechnergestützten Netzwerksimulation wurden der Prozess der Selbstsynchronisation und die Echtzeiteigenschaften eines Ethernet-basierten Self*S*-Protokolls untersucht. Das für die Netzwerksimulation erstellte Simulationsmodell basiert neben den Eigenschaften eines Ethernet-Übertragungskanals auf den Parametern der prototypischen Umsetzung des Hardware- und des Software-Switches. Das Simulationsmodell berücksichtigt im Gegensatz zum allgemeineren theoretischen Modell auch Netzwerk- und Komponentenspezifische Eigenschaften, wie z. B. den Portjitter und die IFG. Trotz des Portjitters und der IFG konnte anhand der Netzwerksimulation belegt werden, dass auch eine Ethernet-basiertes Self*S*-Protokoll selbstsynchronisierend ist. Auch die Echtzeiteigenschaften des Ethernet-basierten Self*S*-Protokolls werden vom Portjitter nicht negativ beeinflusst. Beim Hardware-Switch kann der Portjitter sogar durch die IFG kompensiert werden. Dadurch wird ein Aufaddieren des Portjitters in jedem Knoten verhindert. Die Paketumlaufzeit bei einem Self*S*-Protokoll mit Hardware-Switch hat daher nur einen Jitter von maximal 160 ns. Bei der Latenzzeit ist der Jitter mit 80 ns sogar noch geringer. Aufgrund der höheren Verarbeitungszeit sind beim Software-Switch die Paketumlaufzeit, die Latenzzeit und der Jitter wesentlich größer als beim Hardware-Switch. Die Gültigkeit der simulierten Ergebnisse für reale Systeme wurde mittels eines experimentellen Aufbaus eines Self*S*-Netzwerkes evaluiert. Aufgebaut wurde das Netzwerk zum einen aus den prototypischen Implementierungen des Hardware-Switches und zum anderen aus den prototypischen Implementierungen des Software-Switches. Für beide Switches bestätigen die experimentellen Ergebnisse die Ergebnisse aus der Netzwerksimulation.

Die Adaptivität und die hohe Flexibilität des Self*S*-Protokolls führen gegenüber anderen Ethernet-basierten Echtzeitprotokollen nicht zu einer geringeren Leistungsfä-

higkeit von Self*S*. Bei einem Vergleich von Self*S* mit dem Hardware-Switch sowie der Ethernet-basierten Echtzeitprotokolle PROFINET und EtherCAT, die beide die höchsten Anforderungen der IAONA-Echtzeitklassen erfüllen, erreichen die Echtzeiteigenschaften vom Self*S*-Protokoll vergleichbare Ergebnisse wie die anderen Protokolle. Nur die Latenzzeiten sind bei Self*S* höher. Aber für Echtzeitanwendungen, wie z. B. digitale Regelungen, sind etwas höhere Latenzen meist besser zu handhaben als ein großer Jitter. Der geringe Jitter des Self*S*-Protokolls, der dazu noch unabhängig von der Anzahl der Knoten ist, kann von keinem der beiden anderen Protokolle unterboten werden. Insbesondere bei einer großen Anzahl von Knoten ist der Jitter von PROFINET und EtherCAT deutlich größer.

Die in dieser Arbeit vorgestellten Verfahren zur Adaption auf der Ebene der Netzwerkkomponenten und der Protokollebene ermöglichen adaptive Echtzeitkommunikationsnetze, die Veränderungen von Echtzeitanforderungen im Betrieb zulassen und ihren eigenen Ressourcenbedarf an diese Anforderungen anpassen. Offen bleiben noch weitere Möglichkeiten, den Ressourcenbedarf einer Netzwerkkomponente an die Kommunikationsanforderungen anzupassen. Schon bei Ethernet mit einer Datenrate von 1 GBit/s spielt der Energieverbrauch einer Netzwerkkomponente eine wichtige Rolle. Neben der rekonfigurierbaren Logik eines FPGAs sind andere Architekturen vorstellbar, die ihren Energieverbrauch an die Kommunikationsanforderungen anpassen. Beispiel für eine solche Architektur wäre ein Multiprozessorsystem, das bei geringer Kommunikationslast nicht benötigte Prozessoren abschaltet.

Die adaptiven Echtzeitkommunikationsnetze bilden die Basis, auf der verteilte selbstoptimierende Systeme realisiert werden. Aber insbesondere der reduzierte Planungsaufwand beim adaptiven Echtzeitprotokoll bei gleichzeitiger Erfüllung von harten Echtzeitanforderungen macht das Self*S*-Protokoll zu einer interessanten Alternative im gesamten Bereich der Echtzeitkommunikationsnetze. Der Einsatz von Self*S* in vielen Bereichen der Echtzeitkommunikation wird auch durch die Erweiterbarkeit des Protokolls hinsichtlich der Fehlertoleranz unterstützt. Einige Erweiterungen, um z. B. den Ausfall einer Verbindung zu kompensieren, wurden bereits in Kapitel 5.8 dieser Arbeit diskutiert. Mit der Umsetzung dieser Verfahren und der Entwicklung von weiteren Verfahren können mithilfe des Self*S*-Protokoll sowohl echtzeitfähige als auch sichere adaptive Netzwerke aufgebaut werden.

Verzeichnis der verwendeten Abkürzungen und Formelzeichen

Abkürzungen

ASIC	Application Specific Integrated Circuit
ATM	Asynchronous Transfer Mode
BEB	Binary Exponential Backhoff
CAN	Cntroll Area Network
CIP	Control and Information Protocol
CBA	Component Based Automation
CN	Controlled Node
CoS	Class of Service
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
DIX	DEC-Intel-Xerox-Konsortium
EDF	Earliest Deadline First
EIB	European Installation Bus
EPSG	Ethernet Powerlink Standardization Groupe
FCS	Frame Ceck Sequenz
FDDI	Fiber Distributed Data Interface
FIFO	First In - First Out
FPGA	Field Programmable Gate Array
FTP	File Transfer Protocol
IAONA	Industrial Automation Open Networking Alliance
IEA	Industrial Ethernet Association
IEC	International Electrotechnical Commision
IEEE	Institute of Electrical and Electronics Engineers

IFG	Inter Frame Gap
IP	Internet Protocol
IPIF	Intellectual-Property-Interface
IRT	Isochornous Real-Time
ISO	International Standards Organisation
LAN	Local Area Network
LLC	Logical Link Control
LON	Local Operating Network
MAC	Media Access Controller
MBAP	Modbus Aplikation
MII	Media Independent Interface
MN	Managing Node
NRT	Non Real-Time
OPB	On-Chip Peripheral Bus
ODVA	Open Device Vendor Association
OMNeT++	Object Modular Network Test bed in C++
OSI	Open System Interconnection
PC	Personal Computer
PDU	Protocol Data Unit
PLB	Processor Local Bus
RM	Rate Monotonic
PNO	Profibus Nutzerorganisation
PSE	Packet Starvation Effect
PTCP	Precision Time Clock Protocol
PTP	Precision Time Protocol
QoS	Quality of Service
RT	Real-Time
RTT	Round Trip Time
SCNM	Slot Communication Network Management
SFD	Start of Frame Delimiter
SoC	Star-of-Cyclic-Nachricht
SPS	Speicher Programmierbare Steuerung
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TTE	Time Triggered Ethernet

TTP	Time Triggered Protocol
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network
WAN	Wide Area Network

Lateinische Buchstaben

$\boldsymbol{c}$	Lichtgeschwindigkeit
$\boldsymbol{b}$	Bandbreite
$\boldsymbol{d}$	Länge einer Leitung, über die Daten transportiert werden
$\boldsymbol{e}$	Element aus der Menge der Kanten $\boldsymbol{E}$
$\boldsymbol{g}$	Datenrate
$\boldsymbol{h}$	Länge eines Paketkopfes
$\boldsymbol{k_i}$	Zählervariable eines Knotens $\boldsymbol{i}$
$\boldsymbol{l}$	Länge eines Paketes einschließlich Paketkopf, bei Ethernet einschließlich Präambel und SFD
$\boldsymbol{l_{\mathrm{N}}}$	Länge der Nutzdaten eines Paketes
$\boldsymbol{m}$	Systemgröße; Anzahl der Kanten oder Verbindungen eines Netzwerkes
$\boldsymbol{n}$	Systemgröße; abstraktes Maß für die Anzahl der in einem System befindlichen Teilkomponenten. Beschreibt im Rahmen dieser Arbeit häufig die Anzahl der Knoten oder Switches
$\boldsymbol{n_{\mathrm{Last}}}$	Anzahl der Schleifendurchläufe im Leerlauf-Task im Intervall $\boldsymbol{T_I}$ unter Last
$\boldsymbol{n_{\mathrm{Leer}}}$	Anzahl der Schleifendurchläufe im Leerlauf-Task im Intervall $\boldsymbol{T_I}$ bei Leerlauf
$\boldsymbol{n_{\mathrm{IFG}}}$	Anzahl der Paketlücken (IFGs) im Intervall $\boldsymbol{T_I}$
$\boldsymbol{n_{\mathrm{p}}}$	Maß für die Anzahl der Pakete in einem Netz oder Anzahl der empfangenen bzw. gesendeten Pakete
$\boldsymbol{n_w}$	Anzahl der Knoten in einem Ring $\boldsymbol{w}$
$\boldsymbol{r}$	Datenrate innerhalb eines Netzwerkes oder über eine Verbindung
$\boldsymbol{s}$	Anzahl der Ringe in einem Netzwerk
$\boldsymbol{t_{\mathrm{err}}}$	Abweichung zwischen der lokalen und der zentralen Uhr
$\boldsymbol{t_{k_i}}$	Die Ankunftszeit des Paketes $\boldsymbol{i}$ in der $\boldsymbol{k}$-ten Runde
$\boldsymbol{t_{\mathrm{ld}}}$	Verzögerungszeit zwischen zwei Knoten
$\boldsymbol{u}$	Systemgröße; Anzahl der Knoten zwischen der Quelle und dem Ziel
$\boldsymbol{v}$	Element aus der Menge der Knoten $\boldsymbol{V}$

$\boldsymbol{z}$	Zufallszahl
$\boldsymbol{B}$	Die Größe eines Paketpuffers
$\boldsymbol{D}$	Menge der tatsächlich übertragenen Daten
$\boldsymbol{E}$	Menge der Kanten
$\boldsymbol{G}$	Ein Graph, bestehend aus der Menge der Knoten $\boldsymbol{V}$ und der Menge der Kanten $\boldsymbol{E}$
$\boldsymbol{J}$	Jitter. Der Jitter ist die Schwankung eines periodisch auftretenden Ereignisses.
$\overline{\boldsymbol{J}}$	Durchschnittlicher Jitter
$\boldsymbol{J}_{\mathbf{Akk}}$	Akkumulierter Jitter
$\boldsymbol{J}_{\mathbf{Cyc}}$	Cycle-to-Cycle-Jitter
$\boldsymbol{J}_{\mathbf{Err}}$	Quadratischer Jitterfehler
$\boldsymbol{J}_{i,j}^{k}$	Der Jitter des Paketes $\boldsymbol{p}$ vom Knoten $\boldsymbol{i}$ an den Knoten $\boldsymbol{j}$ nach der $\boldsymbol{k}$-ten Runde
$\boldsymbol{J}_{\mathbf{Err}_{i,j}}^{k}$	Der quadratische Jitterfehler nach dem $\boldsymbol{k}$-ten Paket von Knoten $\boldsymbol{i}$ an den Knoten $\boldsymbol{j}$
$\boldsymbol{J}_{\mathbf{Max}}$	Maximaler Jitter
$\boldsymbol{J}_{\mathbf{Per}}$	Periodischer Jitter
$\boldsymbol{J}_{\mathbf{Port}}$	Durch den Ethernet-Port verursachte Jitter beträgt bei einer Datenrate von 100 MBit/s maximal 40 ns
$\boldsymbol{J}_{\mathbf{Swi}}$	Jitter der Paketverarbeitungszeit
$\boldsymbol{K}$	Systemgröße; Anzahl der Zyklen bzw. der Runden
$\boldsymbol{K}_{i}$	Die Anzahl der Runden, nach denen ein Knoten $\boldsymbol{i}$ zyklisch ein Echtzeitpaket generiert
$\boldsymbol{L}_{\mathbf{CPU}}$	Prozessorlast gemessen im Intervall $\boldsymbol{T_I}$
$\boldsymbol{L}_{\mathbf{Netz}}$	Netzwerklast gemessen im Intervall $\boldsymbol{T_I}$
$\boldsymbol{N}$	Menge der Nutzdaten
$\boldsymbol{N}_{\mathbf{In}}$	Eingangsnutzdaten
$\boldsymbol{N}_{\mathbf{Out}}$	Ausgangsnutzdaten
$\boldsymbol{T}_{\mathbf{Cut}}$	Verzögerungszeit eines Cut-Through-Switches
$\boldsymbol{T}_{\mathbf{Cyc}}$	Zykluszeit
$\boldsymbol{T}_{\mathbf{Buf}}$	Die durch die Wartezeit in einem Paketpuffer verursachte Pufferverzögerung
$\boldsymbol{T}_{\mathbf{DV}}$	Zeitbereich, in dem RX_DV von der MII getrieben wird
$\boldsymbol{T}_{\mathbf{FPGA}}$	Zeit für die partielle Rekonfiguration eines Moduls auf dem FPGA
$\boldsymbol{T}_{\mathbf{Hub}}$	Verzögerungszeit eines Hubs

$\boldsymbol{T_{\mathrm{I}}}$	Vordefiniertes Messintervall
$\boldsymbol{T_{\mathrm{IFG}}}$	Zeitlicher minimaler Abstand zwischen zwei Paketen. Bei dem im Rahmen dieser Arbeit betrachteten Ethernet Protokoll beträgt dieser Abstand 9,6 $\boldsymbol{\mu}$s bei einer Datenrate von 10 MBit/s und 0,96 $\boldsymbol{\mu}$s bei einer Datenrate von 100 MBit/s
$\boldsymbol{T_{\mathrm{Lat}}}$	Latenzzeit
$\boldsymbol{T_{\mathrm{Phy}}}$	Verarbeitungszeit der physikalischen Empfangs- und Sendeeinheit
$\boldsymbol{T_{\mathrm{Prd}}}$	Ausbreitungsverzögerung durch das Übertragungsmedium
$\boldsymbol{T_{\mathrm{PRq-PRs}}}$	Reaktionszeit eines Ethernet Powerlink Teilnehmers
$\boldsymbol{T_{\mathrm{PRs-PRq}}}$	Reaktionszeit einer Ethernet Powerlink Steuerung
$\boldsymbol{T_{\mathrm{RdReq}}}$	Übertragungsverzögerung einer *Read-Request*-Nachricht
$\boldsymbol{T_{\mathrm{Rek}}}$	Gesamte Rekonfigurationszeit von einer Switchvariante zur anderen
$\boldsymbol{T_{\mathrm{RTT}}}$	Paketumlaufzeit nach dem Empfang des selben Paketes
$\boldsymbol{T_{\mathrm{RTT_{Emp}}}}$	Paketumlaufzeit nach dem Senden des selben Paketes
$\boldsymbol{T_{\mathrm{RTT_{Sen}}}}$	Paketumlaufzeit (engl.: Round-Trip Time)
$\boldsymbol{T_{\mathrm{Stack}}}$	Verarbeitungszeit eines Protokollstapels
$\boldsymbol{T_{\mathrm{Start}}}$	Dauer der Startphase bei Ethernet Powerlink
$\boldsymbol{T_{\mathrm{ST\&FW}}}$	Verzögerungszeit bzw. Latenzzeit eines Store-and-Forward-Switches
$\boldsymbol{T_{\mathrm{Swi}}}$	Paketverarbeitungszeit eines Switch
$\boldsymbol{T_{\mathrm{Syn}}}$	Zeit, die für den Selbstsynchronisationsprozess benötigt wird
$\boldsymbol{T_{\mathrm{Trn}}}$	Übertragungsverzögerung ist die Zeit, die ein Sender für die Übertragung eines Paketes benötigt
$\boldsymbol{T_{\mathrm{Trt}}}$	Übertragungszeit ist die Zeit, die ein Paket für die Übertragung vom Sender bis zum Empfänger über eine Verbindung benötigt
$\boldsymbol{T_{\mathrm{Um}}}$	Zeit, die der Umschaltvorgang von einer zur anderen Switchvariante in Anspruch nimmt
$\boldsymbol{T_{\mathrm{w}}}$	Dauer eines Zeitfensters
$\boldsymbol{T_{\mathrm{WrReq}}}$	Übertragungsverzögerung einer *Write-Request*-Nachricht
$\boldsymbol{T_{\mathrm{WrRes}}}$	Übertragungsverzögerung einer *Write-Response*-Nachricht
$\boldsymbol{S}$	Protokolleffizienz
$\boldsymbol{V}$	Menge der Knoten

Griechische Buchstaben

$\boldsymbol{\gamma}$	Ausbreitungskoeffizient eines physikalischen Übertragungsmediums
$\boldsymbol{\lambda}$	Netzwerkauslastung

$\boldsymbol{\nu}$	Ausbreitungsgeschwindigkeit in einem physikalischen Übertragungsmedium
$\boldsymbol{\Delta}$	Differenz

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[ARSG06] Anand, Himanshu ; Reardon, Casey ; Subramaniyan, Rajagopal ; George, Alan D.: Ethernet Adaptive Link Rate (ALR): Analysis of a MAC Handshake Protocol. In: *Proceedings of the 31st IEEE Conference on Local Computer Networks (LCN)*, 2006

[Bag98] Baginski, Alfredo: *INTERBUS : Grundlagen und Praxis.* 2. überarbeitete Auflage. Hüthig Verlag, 1998

[BEF+00a] Breslau, Lee ; Estrin, Deborah ; Fall, Kevin ; Floyd, Sally ; Heidemann, John ; Helmy, Ahmed ; Huang, Polly ; McCanne, Steve ; Varadhan, Kannan ; Xu, Ya ; Yu, Haobo: Advances in Network Simulation. In: *IEEE Computer* 33 (2000), Nr. 5, S. 59–67

[BH06] Bormann, Alexander ; Hilgenkamp, Ingo: *Industrielle Netze : Ethernet-Kommunikation für Automatisierungsanwendungen.* Heidelberg : Hüthig Verlag, 2006

[BO03] Baldi, Mario ; Ofek, Yoram: A comparison of ring and tree embedding for real-time group multicast. In: *IEEE/ACM Transactions on Networking* 11 (2003), Nr. 3

[But04] Buttazzo, Giorgio: *Hard Real-Time Computing Systems : Predictable Scheduling Algorithms and Applications.* Second Edition. New York, NY, USA : Springer, 2004

[BW01] Burns, Alan ; Wellings, Andy J.: *Real-Time Systems and Programming Languages.* 3rd. Addison Wesley, 2001

[CC97] Connelly, Kay H. ; Chien, Andrew A.: FM-QoS: real-time communication using self-synchronizing schedules. In: *Proceedings of the 1997 ACM/IEEE Conference on Supercomputing.* USA, 1997

[Cha96] Chandrasekharan, P.: *Robust Control of Linear Dynamical Systems.* Academic Press, 1996

[CLR00] Coremen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L.: *Introduction to Algorithms.* The MIT Press, 2000

[Dec01] Decotignie, Jean-Dominique: A perspective on Ethernet as a fieldbus. In: *Proceedings of the 4th International Conference on Fieldbus Systems and their Applications (FeT)*. Nancy, France, November 2001, S. 214–219

[Dec05] Decotignie, Jean-Dominique: Ethernet-based real-time and industrial communications. In: *Proceedings of the IEEE* 93 (2005), Nr. 6, S. 1102–1117

[DIN03] Norm DIN EN 50325-4 Juli 2003. *Industrielles Kommunikationssubsystem basierend auf ISO 11898 (CAN) – Teil 4: CANopen*. – Deutsche Fassung EN 50325-4 2002, Beuth-Verlag, Berlin

[Dit99] Ditze, Carsten: *Towards Operating System Synthesis*. Paderborn, Department of Computer Science, Paderborn University, PhD thesis, 1999

[DK00] Dietrich, Dietmar ; Kastner, Wolfgang ; Sauter, Thilo (Hrsg.): *EIB : Gebäudebussysteme*. Heidelberg : Hüthig Verlag, 2000

[DL98] Dietrich, Dietmar ; Loy, Dietmar ; Schweinzer, Hans-Jörg (Hrsg.): *LON-Technologie : Verteilte Systeme in der Anwendung*. Heidelberg : Hüthig Verlag, 1998

[EHS97] Ermedahl, Andreas ; Hansson, Hans ; Sjodin, Mikael: Response-time guarantees in ATM networks. In: *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS)*. San Francisco, CA, USA, December 1997, S. 274–284

[EPL03] Ethernet Powerlink Standardisation Group: *Ethernet Powerlink V2.0 : Communication Profile Specification*. Version 0.1.0. 2003

[Ert06] Siemes AG: *ERTEC 400 – Enhanced Real-Time Ethernet Controller – Datenblatt*. Version 1.1.1. 2006

[Eth01] ControlNet International and Open DeviceNet Vendor Association: *EtherNet/IP Specification*. Release 1.0. June 2001

[Eth04] EtherCAT Technology Group: *EtherCAT Communication Specification*. Version 1.0. 2004

[Eth05] EtherCAT Technology Group: *EtherCAT - Technische Einführung und Überblick*. Juli 2005

[Eth07] *IEEE P802.3ba 40Gb/s and 100Gb/s Ethernet Task Force* . Version: Dezember 2007. `http://www.ieee802.org/3/ba/index.html`, Abruf: 8. März 2008. – Website

[Eth08] Beckhoff Automation GmbH: *Hardware Datasheet – ET1200 – EtherCAT Slave Controller*. Versio 1.3. January 2008

[Ets02] Etschberger, Konrad: *Controller area network: Grundlagen, Protokolle, Bausteine, Anwendungen*. 3., aktualisierte Aufl. Hanser, 2002

[Fer05] Ferber, Michael: *Ein dynamisch rekonfigurierbarer Ethernet-Switch*, Schaltungstechnik, Heinz Nixdorf Institut, Universität Paderborn, Diplomarbeit, Januar 2005

[FGK+04] Frank, Ursula ; Giese, Holger ; Klein, Florian ; Oberschelp, Oliver ; Schmidt, Andreas ; Schulz, Bernd ; Vöcking, Henner ; Witting, Katrin: *Selbstoptimierende Systeme des Maschinenbaus – Definitionen und Konzepte*. Paderborn : Heinz Nixdorf Institut, Universität Paderborn, 2004

[FJ05] Fan, Xinx ; Jonsson, Magnus: Guaranteed Real-Time Services over Standard Switched Ethernet. In: *Proceedings of the 30th IEEE Conference on Local Computer Networks (LCN)*. Sydney, Australia, 2005, S. 490–492

[Fle04] FlexRay Consortium: *FlexRay Communications System Protocol Specification Version 2.0*. Juni 2004

[FN01] Friedman, David ; Nagle, David: Building Firewalls with Intelligent Network Interface Cards / Carnegie Mellon University, School of Computer Science. 2001. – Research Report

[FR05] Francisco, André L. ; Rammig, Franz J.: Fault-Tolerant Hard-Real-Time Communication of Dynamically Reconfigurable, Distributed Embedded Systems. In: *Proceedings of the 8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC)*. USA, 2005

[Fur03] Furrer, Frank J.: *Industrieautomation mit Ethernet-TCP/IP und Web-Technologie*. Heidelberg : Hüthig Verlag, 2003

[FZ00] Foh, Chuan H. ; Zukerman, Moshe: CSMA with reservations by interruptions (CSMA/RI): a novel approach to reduce collisions in CSMA/CD. In: *IEEE Journal on Selected Areas in Communications* 18 (2000), Nr. 9, S. 1572–1580

[GNVV04] Guo, Zhi ; Najjar, Walid ; Vahid, Frank ; Vissers, Kees: A quantitative analysis of the speedup factors of FPGAs over processors. In: *Proceedings of the 12th International Symposium on Field Programmable Gate Arrays*. Monterey, California, USA : ACM Press, 2004, S. 162–170

[Gro82] Grow, Robert M.: A timed token protocol for local area networks. In: *Proceedings of the Electro'82, Token Access Protocols*, 1982 (Paper 17/3)

[HJ03] Hanssen, Fredy ; Jansen, Pierre G.: Real-time Communication Protocols: An Overview / Centre for Telematics and Information Technology, University of Twente. 2003 (TR-CTIT-03-49). – Research Report

[HKKP07a] Hagemeyer, Jens ; Kettelhoit, Boris ; Koester, Markus ; Porrmann, Mario: A Design Methodology for Communication Infrastructures on Partially Reconfigurable FPGAs. In: *Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL)*, IEEE, 2007

[HKKP07b] Hagemeyer, Jens ; Kettelhoit, Boris ; Koester, Markus ; Porrmann, Mario: Design of Homogeneous Communication Infrastructures for Partially Reconfigurable FPGAs. In: *Proceedings of the Int. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, CSREA Press, 2007

[HKT03] Haubelt, Christian ; Koch, Dirk ; Teich, Jürgen: ReCoNet: Modeling and Implementation of Fault Tolerant Distributed Reconfigurable Hardware. In: *Proceedings of 16th Symposium on Integrated Circuits and Systems Design (SBCCI2003)*. São Paulo, Brazil, September 2003, S. 343–348

[HMJ06] Hanssen, Fredy ; Mader, Angelika ; Jansen, Pierre G.: Verifying the Distributed Real-Time Network Protocol RTnet Using Uppaal. In: *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation (MASCOTS)*. Monterey, CA,USA, September 2006, S. 239–246

[Hoa04] Hoang, Hoai: Real-time communication for industrial embedded systems using switched Ethernet. In: *Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Santa Fe, NM, USA, April 2004

[HSB06] Hübner, Michael ; Schuck, Christian ; Becker, Jürgen: Elementary Block Based 2-Dimensional Dynamic and Partial Reconfiguration for Virtex-II FPGAs. In: *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2005) - Reconfigurable Architectures Workshop (RAW)*, IEEE Computer Society, 2006

[IDT07] Integrated Device Technology (IDT): *The role of jitter in timing signals*. February 2007. – White Paper

[IEC06] Norm IEC 61158 December 2006. *Digital data communication for measurement and control – Fieldbus for use in industrial control systems.* – IEC, Geneva, Switzerland

[IEC07] Norm IEC 61784-2 December 2007. *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3.* – ISO/IEC, Geneva, Switzerland

[IEE98] Norm IEEE 802.5 (ISO/IEC 8802-5) May 1998. *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 5: Token Ring Access Method and Physical Layer Specification.* – IEEE, New York, NY, USA

[IEE03] Norm IEEE 802.1Q May 2003. *IEEE Standard for Local and metropolitan area networks: Virtual Bridged Local Area Networks.* – IEEE, New York, NY, USA

[IEE04] Norm IEEE 802.1D June 2004. *IEEE Standard for Local and metropolitan area networks : Media access control (MAC) Bridges.* – IEEE, New York, NY, USA

[IEE05] Norm IEEE 802.3 December 2005. *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications.* – IEEE, New York, NY, USA

[Int01] Intel Corporation: *LXT974/LXT975 Fast Ethernet 10/100 Quad Transceivers.* Revision 1.4. Santa Clara, CA, USA, January 2001

[ISO94] Norm ISO/IEC 7498-1 November 1994. *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model.* – ISO/IEC, Geneva, Switzerland

[Jas02] Jasperneite, Jürgen: *Leistungsbewertung eines lokalen Netzwerkes mit Class-of-Service Unterstützung für die prozessnahe Echtzeitkommunikation*, Otto-von-Guericke Universität Magdeburg, Dissertation, 2002

[JB03a] Janssen, Dirk ; Büttner, Holger: EtherCAT - Der Ethernet-Feldbus, Adressierung und Aufbau der Protokolle – 2. Teil. In: *Elektronik* 25 (2003), S. 62–67

[JB03b] Janssen, Dirk ; Büttner, Holger: EtherCAT - Der Ethernet-Feldbus, Funktion und Eigenschaften – 1. Teil. In: *Elektronik* 23 (2003), S. 64–72

[JE04] Jasperneite, Jürgen ; Elsayed, Ehab: Investigations on a Distributed Time-triggered Ethernet Realtime Protocol Used by Profinet. In: *Proceedings of the 3rd International Workshop on Real-Time Networks (RTN)*, 2004

[JG07] Jasperneite, Jürgen ; Gamper, Sergej: Echtzeit-Betrieb im Ethernet - Industrial Ethernet Switches auf dem Prüfstand - Teil2. In: *Elektronik* 16 (2007), Aug, S. 60–65

[JN01] Jasperneite, Jürgen ; Neumann, Peter: Switched Ethernet for factory communication. In: *Proceedings of the 8th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Antibes-Juan les Pins, France, October 2001, S. 205–212

[JSW04] Jasperneite, Jürgen ; Shehab, Khaled ; Weber, Karl: Enhancements to the time synchronization standard IEEE-1588 for a system of cascaded bridgest. In: *Proceedings of the 5th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2004, S. 239–244

[KAGS05] Kopetz, Hermann ; Ademaj, Astrit ; Grillinger, Petr ; Steinhammer, Klaus: The Time-Triggered Ethernet (TTE) Design. In: *Proceedings of the 8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC)*. Seattle, WA, USA, May 2005, S. 22–33

[KG94] Kopetz, Hermann ; Grünsteidl, Günter: TTP - A Protocol for Fault-Tolerant Real-Time Systems. In: *IEEE Computer* 27 (1994), Nr. 1, S. 14–23

[KKK+05] Kalte, Heiko ; Kettelhoit, Boris ; Koester, Markus ; Porrmann, Mario ; Rückert, Ulrich: A System Approach for Partially Reconfigurable Architectures. In: *International Journal of Embedded Systems (IJES)* 1 (2005), Nr. 3/4, S. 274–290

[Kop97] Kopetz, Hermann: *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic, 1997

[Kop01] Kopetz, Hermann: A Comparison of TTP/C and FlexRay / Technische Universität Wien, Institut für Technische Informatik. Vienna, Austria, 2001 (10/2001). – Research Report

[KP06] Kalte, Heiko ; Porrmann, Mario: REPLICA2Pro: Task relocation by bitstream manipulation in Virtex-II/Pro FPGAs. In: *Proceedings of the 3rd conference on Computing Frontiers, (CF '06)*. New York, NY, USA : ACM Press, 2006, S. 403–412

[KPR02] Kalte, Heiko ; Porrmann, Mario ; Rückert, Ulrich: A Prototyping Platform for Dynamically Reconfigurable System on Chip Designs. In: *Proceedings of the IEEE Workshop Heterogeneous reconfigurable Systems on Chip (SoC)*. Hamburg, Germany, 2002

[KPR05] Koester, Markus ; Porrmann, Mario ; Rückert, Ulrich: Placement-Oriented Modeling of Partially Reconfigurable Architectures. In: *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS 2005) - Reconfigurable Architectures Workshop (RAW)*, IEEE Computer Society, 2005

[KS97] Krishna, C. M. ; Shin, Kang G.: *Real time systems*. McGraw-Hill, 1997

[KSD+06] Koch, Dirk ; Streichert, Thilo ; Dittrich, Steffen ; Strengert, Christian ; Haubelt, Christian ; Teich, Jürgen: An Operating System Infrastructure for Fault-Tolerant Reconfigurable Networks. In: *Proceedings of the 19th International Conference on Architecture of Computing Systems (ARCS 2006)*. Frankfurt, Germany : Springer, März 2006, S. 202–216

[LH04] Löser, Jork ; Härtig, Hermann: Low-Latency Hard Real-Time Communication over Switched Ethernet. In: *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS)*. Catania, Sicily, Italy, June 2004, S. 13–22

[Liu00] Liu, Jane W. S.: *Real-Time Systems*. Prentice Hall, 2000

[LJR02] Lankes, Stefan ; Jabs, Andreas ; Reke, Michael: A time-triggered Ethernet protocol for Real-Time CORBA. In: *Proceedings of the 5th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC)*, 2002, S. 215–222

[LL05] Lüder, Arndt (Hrsg.) ; Lorentz, Kai (Hrsg.): *IAONA Handbook: Industrial Ethernet*. Industrial Automation Open Networking Alliance e.V., 2005

[LLR93] Le Lann, Gérard ; Rivierre, Nicolas: Real-time communications over broadcast networks: the CSMA-DCR and the DOD-CSMA-CD protocols

/ Institut National de Recherche en Informatique et en Automatique (INRIA). 1993 (1863). – Research Report

[LMR+03] Lockwood, John W. ; Moscola, James ; Reddick, David ; Kulig, Matthew ; Brooks, Tim: Application of Hardware Accelerated Extensible Network Nodes for Internet Worm and Virus Protection. In: *In Proceedings of the International Working Conference on Active Networks (IWAN)*, 2003

[MB76] Metcalfe, Robert M. ; Boggs, David R.: Ethernet: Distributed Packet Switching for Local Computer Networks. In: *Communication of the ACM* 19 (1976), Juli, Nr. 7, S. 395–404

[MHG03] Martínez, José M. ; Harbour, Michael G. ; Gutiérrez, Javier J.: RT-EP: Real-Time Ethernet Protocol for Analyzable Distributed Applications on a Minimum Real-Time POSIX Kernel. In: *Proceedings of the 2nd International Workshop on Real-Time LANs in the Internet Age (RTLIA)*. Porto, Portugal, July 2003

[MK85] Molle, Mart L. ; Kleinrock, Leonard: Virtual Time CSMA: Why Two Clocks Are Better than One. In: *IEEE Transactions on Communications* 33 (1985), September, Nr. 9, S. 919–933

[MKZ96] Malcolm, Nicholas ; Kamat, Sanjay ; Zhao, Wei: Real-time communication in FDDI networks. In: *Real-Time Systems* 10 (1996), Nr. 1, S. 75–107

[Mod04a] Modbus-IDA: *Modbus Application Protocol Specification*. V1.1a. June 2004

[Mod04b] Modbus-IDA: *Modbus messaging on TCP/IP implementation guide*. V1.0a. June 2004

[MZ95] Malcolm, Nicholas ; Zhao, Wei: Hard real-time communication in multiple-access networks. In: *Real-Time Systems* 8 (1995), Nr. 1, S. 35–77

[OPB04] Xilinx Inc: *On-Chip Peripheral Bus v2.0 with OPB Arbiter (v1.10b)*. January 2004

[OPN08] OPNET Technologies, Inc.: *Verifying Statistical Validity of Discrete Event Simulations*. Bethesda, MD, USA, 2008. – White Paper

[PA02] Pedreiras, Paulo ; Almeida, Luís: Flexibility, Timeliness and Efficiency over Ethernet. In: *Proceedings of the 1st International Workshop on Real-Time LANs in the Internet Age (RTLIA).* Vienna, Austria, June 2002, S. 41–44

[PGAB05] Pedreiras, Paulo ; Gai, Paolo ; Almeida, Luis ; Buttazzo, Giorgio: FTT-Ethernet: A flexible real-time communication protocol that supports dynamic QoS management on Ethernet-based systems. In: *IEEE Transactions on Industrial Informatics* 1 (2005), Nr. 3, S. 162–172

[PL04] Patrin, John ; Li, Mike: Characterizing Jitter Histograms for Clock and DataCom Applications. In: *Proceedings of the IEC DesignCon 2004.* Santa Clara, California, USA, February 2004

[PLB04] Xilinx Inc: *Processor Local Bus (PLB) v3.4.* January 2004

[Pop00] Popp, Manfred: *Profibus-DP-DPV1: Grundlagen, Tipps und Tricks für Anwender.* 2., überarb. Aufl. Heidelberg : Hüthig Verlag, 2000

[Pop05] Popp, Manfred: *Das PROFINET IO-Buch: Grundlagen und Tipps für Anwender.* Heidelberg : Hüthig Verlag, 2005

[PRO06] PROFINET Nutzerorganisation e.V.: *Profinet Technologie und Anwendung, Systembeschreibung.* April 2006

[RY94] Ramakrishnan, K. K. ; Yang, Henry: The Ethernet capture effect: Analysis and solution. In: *Proceedings of the 19th IEEE Conference on Local Computer Networks (LCN)*, 1994, S. 228–240

[RZKJ01] Raha, Amitava ; Zhao, Wei .. ; Kamat, Sanjay ; Jia, Weijia: Admission control for hard real-time connections in ATM LANs. In: *IEE Proceedings - Communications* 148 (2001), August, Nr. 4, S. 217–228

[Sch05] Schlenger, Marc: *Dynamische Rekonfiguration mit Hilfe eingebetteter Prozessoren*, Schaltungstechnik, Heinz Nixdorf Institut, Universität Paderborn, Studienarbeit, März 2005

[Sch08] Schwager, Jürgen: *Informationsportal für Echtzeit-Ethernet in der Industrieautomation.* Version: März 2008. `http://www-pdv.fh-reutlingen.de/rte/`, Abruf: 8. März 2008

[Sed01] Sedgewick, Robert: *Algorithms in C, Part 5: Graph Algorithms, 3rd Edition.* Addison Wesley, 2001

[Sei00] Seifert, Rich: *The Switch Book : The Complete Guide to LAN Switching Technology*. New York, NY, USA : John Wiley & Sons, Inc., 2000

[SKS02] Song, Yequiong ; Koubaa, Anis ; Simonot, François: Switche Ethernet For Real-Time Industrial Communication : Modelling And Message Buffering Delay Evaluation. In: *Proceedings of the 4th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2002, S. 27–35

[Son01] Song, Yequiong: Time constrained communication over switched Ethernet. In: *Proceedings of the 4th International Conference on Fieldbus Systems and their Applications (FeT)*. Nancy, France, November 2001, S. 138–143

[Ste06] Steinhammer, Klaus: *Design of an FPGA-Based Time-Triggered Ethernet System*, Technische Universität Wien, Institut für Technische Informatik, Dissertation, 2006

[SZ03] Scheitlin, Hans ; Zuber, Roger: Kommunizieren Sie pünktlich / Institut für Embedded Systems (InES)/Zürcher Hochschule Winterthur (ZHW). 2003. – Forschungsbericht

[Tan03] Tanenbaum, Andrew S.: *Computernetzwerke*. Prentice Hall, 2003

[Tra04] Trader, Michael T.: How to calculate CPU utilization. In: *Embedded Systems Design* (2004), July

[USL01] Underwood, Keith D. ; Sass, Ron R. ; Ligeon, Walter B.: A Reconfigurable Extension to the Network Interface of Beowulf Clusters. In: *Proceedings of the IEEE Conference on Cluster Computing (Cluster 2001)*, 2001

[Var01] Varga, András: The OMNeT++ Discrete Event Simulation System. In: *Proceedings of the 16th European Simulation Multiconference (ESM)*, 2001

[VC94] Venkatramani, Chitra ; Chiueh, Tzi-cker: Supporting real-time traffic on Ethernet. In: *Proceedings of the 15th IEEE Real-Time Systems Symposium (RTSS)*. San Juan, Puerto Rico, Dezember 1994, S. 282–286

[Ven97] Venkatramani, Chitra: *The Design, Implementation and Evaluation of RETHER: A Real-Time Ethernet Protocol*, State University of New York, PhD thesis, 1997

[Vit01] Vitturi, Stefano: On the use of Ethernet at low level of factory communication systems. In: *Computer Standards & Interfaces* 23 (2001), Nr. 4, S. 267–277

[Wil07] Willms, Eduard: *Leistungsanalyse von Ethernet-basierten Echtzeitprotokollen*, Schaltungstechnik, Heinz Nixdorf Institut, Universität Paderborn, Studienarbeit, Januar 2007

[Wis02] SiliCore, Opencore.org: *Wishbone System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores*. Rev. B3. September 2002

[WSF94] Whetten, Brian ; Steinberg, Stephen ; Ferrari, Domenico: The packet starvation effect in CSMA/CD LAN's and a solution. In: *Proceedings of the 19th IEEE Conference on Local Computer Networks (LCN)*, 1994, S. 206–217

[WXL+00] Wang, Zhi-Ping ; Xiong, Guang-Ze ; Luo, Jin ; Lai, Ming-Zhi ; Zhou, Wanlei: A hard real-time communication control protocol based on the Ethernet. In: *Proceedings 7th Australasian Conference on Parallel and Real-Time Systems (PART)*. Sydney, Australia, November 2000, S. 161–170

[YB99] Yodaiken, Victor ; Barabanov, Michael: RTLinux Version Two. In: *Proceedings of the 1st Real-Time Linux Workshop*, 1999

[ZR87] Zhao, Wei ; Ramamritham, Krithi: Virtual time CSMA protocols for hard real-time communication. In: *IEEE Transactions on Software Engineering* 13 (1987), Nr. 8, S. 938–952

Eigene Veröffentlichungen

[GBP08] Griese, Björn ; Brinkmann, André ; Porrmann, Mario: SelfS – A Real-Time Protocol for Virtual Ring Topologies. In: *Proceedings of the 16th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*. Miami, Florida, USA, April 2008

[GKP06] Griese, Björn ; Kettelhoit, Boris ; Porrmann, Mario: Evaluation of on-chip interfaces for dynamically reconfigurable coprocessors. In: *Proceedings of the 5th International Symposium on Parallel Computing in Electrical Engineering (PARELEC)*. Bialystok, Poland, September 2006, S. 214–219

[GOP05] Griese, Björn ; Oberthür, Simon ; Porrmann, Mario: Component case study of a self-optimizing RCOS/RTOS system: A reconfigurable network service. In: *Proceedings of the International Embedded Systems Symposium (IESS)*. Manaos, Brazil, August 2005, S. 267–277

[GP06] Griese, Björn ; Porrmann, Mario: A Reconfigurable Ethernet Switch for Self-Optimizing Communication Systems. In: *Proceedings of the IFIP Conference on Biologically Inspired Cooperative Computing (BICC)*. Santiago de Chile, Chile, August 2006, S. 115–125

[GVPR04] Griese, Björn ; Vonnahme, Erik ; Porrmann, Mario ; Rückert, Ulrich: Hardware Support for Dynamic Reconfiguration in Reconfigurable SoC Architectures. In: *Proceedings of the 14th International Conference on Field Programmable Logic and its Applications (FPL)*. Antwerp, Belgium, 30 August - 1 September 2004, S. 842–846

[OBG05] Oberthür, Simon ; Böke, Carsten ; Griese, Björn: Dynamic Online Reconfiguration for Customizable and Self-Optimizing Operating Systems. In: *Proceedings of the 5th ACM International Conference on Embedded Software (EMSOFT)*. Jersey City, NJ, USA, September 2005, S. 335–338

[VGPR04a] Vonnahme, Erik ; Griese, Björn ; Porrmann, Mario ; Rückert, Ulrich: Dynamic reconfiguration of real-time network interfaces. In: *Proceedings of the 4th International Conference on Parallel Computing in Electrical Engineering (PARELEC)*. Dresden, Germany, September 2004, S. 376–379

[VGPR04b] Vonnahme, Erik ; Griese, Björn ; Porrmann, Mario ; Rückert, Ulrich: Dynamische Rekonfiguration echtzeitfähiger Netzwerkschnittstellen. In: *VDE Kongress 2004 - ITG Fachtagung Ambient Intelligence* Bd. 1. Berlin, Germany : VDE Verlag, October 2004, S. 99–104