Universität Siegen

Fachbereich Elektrotechnik und Informatik
Angewandte Informatik

Diplomarbeit

Entwurf und Entwicklung eines Simulationsmodells für ein neuartiges Echtzeit-Ethernet System im Industrieeinsatz

Design and development of a simulation model for a novel industrial Real-Time-Ethernet system

Henning
Westerholt

Matr. Nr.: 606556

Erstprüfer:
Prof. Dr. Roland Wismüller

Zweitprüfer:
Dipl. Inf. Frank Dopatka M. Sc.

Zusammenfassung

Im Rahmen dieser Arbeit wird ein Modell einer neuartigen Echtzeit-Ethernet Lösung für das Simulationssystem OMNeT++ entwickelt. Dafür müssen im Vorfeld verschiedene Echtzeit-Ethernetansätze in ihrer Funktionsweise untersucht und Grundlagen über das zu betrachtene Umfeld – Ethernet in der Automatisierungstechnik – recherchiert werden. Die einzelnen Echtzeitlösungen werden detailliert miteinander verglichen, weiterhin finden sie bei der Entwicklung des Modells Berücksichtigung. Zusätzlich zur Konzeption des Simulationsmodells ist der Entwurf und die Implementierung von verschiedenen Hilfswerkzeugen wie ein Netzwerkgenerator und ein Konverter notwendig. Anschließend lässt sich das Modell entwickeln und auf einwandfreie Funktionsweise überprüfen. Das Verhalten des Modells wird durch die Simulation von drei unterschiedlichen Netzwerk-Szenarien ausführlich untersucht. Anhand der ermittelten Ergebnisse ist die einwandfreie Funktion zu zeigen, des Weiteren lässt sich ein erster Vergleich mit den im Vorfeld betrachteten Systemen vornehmen.

Das entwickelte System bietet eine Leistung, die auch höchsten Echtzeitanforderungen genügt. Der gemischte Betrieb mit zusätzlichen nicht Echtzeit-Teilnehmern ist möglich, hier besteht aber noch Optimierungspotential. Durch die optimierte Berechnung der zeitlichen Ablaufpläne lässt sich die Leistung gegenüber gewöhnlichen, auf Zeitmultiplex-basierenden Echtzeit-Ethernet Verfahren erheblich erhöhen. Diese Arbeit bietet somit eine Ausgangsbasis für eine weitergehende Realisierung dieses neuartigen Systems.

Tag der Abgabe: 21.12.2006

Kontakt zum Autor über E-Mail: hw@skalatan.de

Die aktuelle Fassung der Arbeit und der erstellten Software kann von http://www.skalatan.de/rte/ heruntergeladen werden.

Informationen zur Versionskontrolle: $Revision: 805 $

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Der Einsatz von Echtzeit-Ethernet Ansätzen in der Automatisierungstechnik ist *das* Thema der letzten Jahre. Die Zeitschrift Computerwoche spricht in diesem Zusammenhang sogar von einer „Eroberung der Produktion“, laut einer dort zitierten Studie der ARC Advisory Group wird der Markt für Industrial Ethernet in den nächsten Jahren jährlich um die Hälfte wachsen [15]. Mittlerweile konkurrieren eine Vielzahl von Herstellern um diesen Markt, so führt eine Übersichtsseite der Universität Reutlingen [24] zu diesem Thema zum Zeitpunkt der Erstellung der Arbeit 19 verschiedene Verfahren auf.

Ziel des Einsatzs dieser Lösungen ist es, eine durchgängige Kommunikation – beginnend von Sensoren und Aktoren bis in die Büros – zu ermöglichen. Dadurch erhofft man sich Effizienzsteigerungen innerhalb der Produktion aufgrund von erhöhter Flexibilität, einfachere und schnellere Konfiguration bzw. Management und Prozessoptimierungen. Auf lange Sicht werden also Netze auf Ethernet-Basis die traditionellen Feldbusse ergänzen und sogar teilweise verdrängen. Deshalb ist es sinnvoll, sich mit den Funktionsprinzipien der unterschiedlichen Ansätze zu befassen, diese führt die erste Hälfte dieser Arbeit auf.

Da Ethernet in der verbreiteten Form nicht an die speziellen Anforderungen der Automatisierungstechnik angepasst ist, sind zahlreiche Anpassungen notwendig. Auch bei bestehenden Systemen existiert natürlich immer Optimierungspotential. Bei komplexen Systemen, wie sie Netze der Kommunikationstechnik darstellen, zeigt man die korrekte Funktionsweise gerne anhand von Modellen. Auch zur Entwicklung eines neuen Systems ist die Unterstützung durch ein Simulationssystem hilfreich. Die notwendigen Schritte von der Idee bis zum vollständigen Simulationsmodell bilden die zweite Hälfte dieser Arbeit.

1.2 Aufgabenstellung

Im Rahmen der Arbeit sollen zunächst gängige Echtzeit-Ethernet Ansätze untersucht werden. Dafür sind die technischen Daten und die Funktionsweise der Protokolle auf den Transport orientierten OSI-Schichten eins bis vier detailliert zu betrachten. Anschließend erfolgt ein Vergleich der einzelnen Verfahren anhand eines aufzustellenden Kriterienkataloges. Besonders von Interesse sind hierbei die (harte) Echtzeitfähigkeit und die Kompatibilität zu Standard-Ethernet.

Des weiteren sind gebräuchliche Netzwerksimulatoren auf ihre Eignung für den Einsatz im Echtzeit-Bereich zu analysieren. Nach Auswahl eines Simulators ist unter Verwendung von gängigen objektorientierten Analyse- und Entwurfstechniken ein Simulationsmodell für ein neu entwickeltes Echtzeit-Ethernet-Protokoll zu erstellen. Die Spezifikation dieses Protokolls ist aus der Arbeit einer Projektgruppe [39] abzuleiten.

Mit Hilfe des entwickelten Modells ist eine Simulation mehrerer Szenarien durchzuführen. Es ist zu evaluieren, inwieweit dabei eine Übernahme von Arbeitsergebnissen der Projektgruppe sinnvoll ist. Die Ergebnisse der einzelnen Simulationsläufe sind anschließend zu bewerten, insbesondere im Vergleich zu den bereits untersuchten, etablierten Echtzeit-Ethernet-Verfahren.

Im Rahmen dieser Diplomarbeit erfolgt kein umfassender Überblick über alle auf dem Markt befindlichen, oder in der Entwicklung stehenden Echtzeit-Ethernet-Systeme. Um eine Simulation sinnvoll vornehmen zu können, muss man eine Abstraktion und Vereinfachung des komplexen Systems „Ethernet“ durchführen. Es erfolgt eine Begrenzung der Simulation auf einige ausgewählte Szenarien. Da zu Beginn dieses Projektes die Arbeit der Projektgruppe noch nicht abgeschlossen ist, also wichtige Spezifikationen noch nicht endültig festgeschrieben sind, sind bei dem Grad der Interoperabilität Einschränkungen zu erwarten.

1.3 Rahmenbedingungen

Für die Simulation ist ein gängiges und frei verfügbares Simulationssystem für den Netzwerkbereich zu benutzen. Dabei wird keine komplette Eigenentwicklung, wie beispielsweise von der Projektgruppe, durchgeführt, sondern es sind bestehende Elemente des genutzten Systems anzupassen und gegebenfalls zu erweitern. Das Simulationssystem stellt also einen Rahmen bereit, in dass das neu entwickelte Echtzeit-Ethernet-Verfahren eingebettet wird.

Die System- und Protokolldefinition des zu erstellenden Modells wird von der Projektgruppe und dem Betreuer vorgegeben. An dieser Spezifikation sind keine größeren Modifikationen oder der Entwurf von neuen Komponenten vorzunehmen. Für die Simulation einzelner Szenarien, den Test des Simulators und das Herstellen von Interoperabilität zwischen der Arbeit der Projektgruppe und dem angepassten Simulationssystem sind jeweils eigene, voneinander abgegrenzte Module zu entwerfen. Aufgrund der begrenzten Zeit, die für die Ausführung der Arbeit zur Verfügung steht, erfolgt bei der Entwicklung gegebenenfalls eine Beschränkung auf wichtige Teilbereiche des Echtzeit-Ethernet Verfahren.

1.4 Gliederung

Nach Darstellung der Motivation und Definition der Aufgabenstellung wird noch noch kurz auf den weiteren Aufbau der Arbeit eingegangen. Im nachfolgenden Kapitel erfolgt eine Einführung in notwendiges Basiswissen zum Thema Automatisierungstechnik, Echtzeit-Netzwerke und Ethernet. Weiterhin werden gängige Echtzeit-Ethernet Ansätzen vorgestellt und anhand eines aufgestellten Kritierienkataloges bewertet (Kapitel 3). Es folgt die Einführung von Grundlagen zur Simulation und die Untersuchung des neu entwickelten Echtzeit-Ethernet-Systems mit Hilfe eines Simulationsframeworks in Kapitel 4 und 5. Den Schluß bildet eine Zusammenfassung der Ergebnisse und das Fazit, in Kapitel 7. Im Anhang erfolgt unter anderem eine ausführliche Darstellung der Bedienung und Erweiterung des erstellten praktischen Teils.

2 Grundlagen

In diesem Kapitel werden für das Verständnis der Arbeit wichtige Grundlagen eingeführt. Zunächst wird eine allgemeine Definition des Begriffs Echtzeit vorgenommen. Des Weiteren folgt eine Darstellung der allgemeinen Anforderungen an Echtzeit-Netzwerke und eine kurze Einführung in die Automatisierungstechnik. Daran schließt sich eine Übersicht über die Funktionsweise von Ethernet und eine Diskussion der Gründe für einen Einsatz dieser Technologie als Ersatz für Feldbusse in der Industrie an. Dieser Abschnitt endet mit einer Darstellung verschiedener Ansätze, um Echtzeitfähigkeit mit Ethernet zu erreichen.

2.1 Definition von Echtzeit

Allgemein unterteilt man Echtzeitsysteme anhand der auftretenen Folgen bei Nichteinhaltung der zeitlichen Anforderungen in zwei Klassen [50]:

- Bei harten Echtzeitsystemen muss die Reaktion auf eine Eingangsanregung immer innerhalb einer bestimmten Zeit erfolgen.
- Weiche Echtzeitsysteme haben eine Reaktionszeit, deren Mittelwert über eine definierte Zeitspanne nicht über einen spezifizierten Maximalwert liegt.

Die erste Kategorie ist erheblich schwerer innerhalb von Systemen zu realisieren. Ein Beispiel für ein System mit harten Echtzeitanforderungen ist eine elektronische Einspritzanlage für einen Automotor. Bei Überschreiten der Reaktionszeit entstehen sofort Störungen, der Motor bleibt stehen oder wird sogar beschädigt. Die IP-Telefonie[1] ist eine Anwendung mit weichen Echtzeit-Bedingungen. Falls die Latenz zu groß wird, oder zu viele Daten verloren gehen, treten Verständigungsprobleme und schlimmenfalls ein Verbindungsabbruch auf. Nach Verbesserung der Verbindung kann das Telefonat aber problemlos fortgesetzt werden.

Da der Begriff „Echtzeit“ nur die Reaktion auf ein Ereignis innerhalb einer bestimmten Zeit festlegt, werden diese zwei Kategorien nun nochmals differenziert. Die IAONA[2] unterscheidet vier Klassen ausgehend von dem maximalen Jitter der Zeitsynchronisation [44], wie in Tabelle 1 dargestellt.

Echtzeitklasse	Beschreibung	max. Jitter Synchronisation
1	Eigenschaften von Standardprodukten	< 1 ms
2	Optimierte Produkte nach heutigen Standards	100 μs - 3 ms
3	Produkte mit neuer Funktionalität in Software	10 μs - 400 μs
4	Neue Funktionalität in Hard- und Software	0,5 μs - 15 μs

Tabelle 1: Echtzeitklassen nach IAONA

Nachteil dieser Klassifizierung ist, dass sie aus Entwicklersicht formuliert ist, also keinerlei Aussagen über die Eignung der einzelnen Klassen für bestimmte Anwendungen in der Automatisierungstechnik trifft.

[1] auch Voice over IP genannt
[2] Industrial Automation Open Network Alliance

Die Norm IEC-61784-2 [19] schlägt drei Echtzeitklassen vor, sie bezieht sich dabei auf die maximale Reaktionszeit und auch explizit auf die Eignung für bestimmte Automatisierungfunktionen [13, 14]. Da dieser Ansatz für die Einteilung von Echtzeit-Lösungen erheblich besser geeignet ist, wird er innerhalb dieser Arbeit verwendet. Tabelle 2 gibt einen Überblick über die Klassifizierung nach IEC-61784-2.

Echtzeitklasse	Einsatz	maximale Reaktionszeit
1	Menschliche Überwachungsfunktionen	ca. 100 ms
2	„Normale“ Automatisierungsfunktionen	< 10 ms
3	Motion-Control Anwendungen	< 1 ms, Jitter < 1 μs

Tabelle 2: Echtzeitklassen nach IEC-61784-2

Die Klasse der menschlichen Überwachungsfunktionen lässt sich ohne Probleme mit heutigen Ethernet-Netzen erreichen. Zu diesem Bereich gehören weiterhin andere einfache Einsatzgebiete wie Füllstandsüberwachungen oder Temperaturregelungen. Für die Unterstützung von allgemeinen Automatisierungsfunktionen die sich in IEC-61784-2 Klasse 2 einordnen lassen, wie sie beispielsweise für Werkzeugmaschinen erforderlich sind, muss man entweder erhebliche Resourcen aufwenden, oder Vereinfachungen bzw. Optimierungen bei den eingesetzen Netzwerkstacks vornehmen. Klasse 3 Anwendungen erfordern die Veränderung des Medienzugriffs und Anpassung der Netzwerktopologie, ausserdem ist eine minimale Bitrate von 100 MBit/s notwendig [13].

Wie schon aus der Definition der einzelnen Klassen ersichtlich, wird also bei Echtzeit-Kommunikation nicht nur eine deterministische Übertragungsdauer (Latenz) gefordert, sondern gleichermaßen die Einhaltung von bestimmten Grenzwerten für deren Schwankungen, den Jitter. Besondere Anforderungen an den Durchsatz bestehen bei Echtzeit-Netzwerken im allgemeinen nicht, da die versendeten Datenmengen in der Regel gering sind. Wichtig für die Echtzeitfähigkeiten ist aber ein deterministischer Medienzugang für die Teilnehmer, damit in Konkurrenzsituationen[3] keine Probleme auftreten. Auf die Eigenschaften von Netzen in der Automatisierungstechnik wird in Kapitel 2.3.2 aber noch ausführlicher eingegangen.

2.2 Anforderungen an Echtzeit-Netzwerke

In erster Linie müssen Echtzeit-Netzwerke natürlich die von der Lösung geforderten und in Kapitel 2.1 eingeführten zeitlichen Anforderungen erfüllen. Weiterhin ergeben sich aus den Voraussetzungen innerhalb der Einsatzumgebung bestimmte Anforderungen, beispielsweise ist Kompatibilität zu anderen Feldbussen erforderlich. Für eine ausführliche Darstellung möglicher Kriterien siehe auch Kapitel 3.3. Eine weitere Diskussion der speziellen Bedingungen für Netze innerhalb der Automatisierungstechnik findet sich im nachfolgenden Kapitel 2.3.

Für einen Einsatz in Industrieanlagen, insbesondere innerhalb von sicherheitskritischen Systemen wie Chemiewerke oder Kernreaktoren sind andere Anforderungen an die Zuverlässlichkeit und Verfügbarkeit eines Netzwerkes zu stellen, als innerhalb einer normalen Büroumgebung. Eventuell aufgetretene Fehler dürfen keine kritischen Auswirkungen haben und sie müssen sich sicher und schnell diagnostizieren lassen. Ebenso muss die verwendete Hardware für die Umgebung geeignet sein, also höhere Temperaturen ertragen, gegebenenfalls beständig gegen Einwirkungen von Wasser, Öl oder Säure und die Einstrahlung von elektromagnetischen Störungen sein. Des Weiteren bestehen erhöhte Anforderungen an die

[3] Beispielsweise die gleichzeitige Übertragung von mehreren Nachrichten.

Explosionssicherheit, also Schutz vor Funkenflug oder ähnlichem. Das Feld der „sicherheitsrelevanten Eigenschaften“ wird innerhalb dieser Arbeit nicht behandelt. Eine Einführung in den Themenbereich der sicherheitskritischen Systeme und deren Anforderungen bietet zum Beispiel Storey [50]. Bereiche der elektromagnetischen Verträglichkeit, insbesondere bei der Installation, werden unter anderem in [3] behandelt.

2.3 Automatisierungstechnik

Zunächst wird das Modell der Automatisierungspyramide, dass eine Fertigung symbolisiert, betrachtet. Weiterhin erfolgt eine Einführung in den Netzwerkaufbau und die Kommunikationsformen innerhalb der Automatisierungstechnik.

2.3.1 Automatisierungspyramide

Die Automatisierungspyramide stellt das klassische Sinnbild einer Systemarchitektur in verfahrenstechnischen Unternehmen dar. Jede Ebene besitzt eigene, sie auszeichnende Eigenschaften, ein Informationsaustausch findet hier immer nur mit der benachbarten Ebene statt [7]. An der Spitze der Pyramide steht die Betriebs-(leitungs)ebene. Hier werden die Entscheidungen, die sich auf das gesamte Unternehmen beziehen, getroffen. An sie ist die Leitebene angrenzt, hier betrachten wir einzelne Fertigungen. In der Führungsleitebene vergrößert sich die Auflösung der Betrachtung auf einzelne Prozesse. Die letzten beiden Schichten sind die Steuerungs- und Feldebene, wo Steuerungen konkrete Sensoren und Aktoren kontrollieren. Zusätzlich zu der allgemeinen Darstellung der Archichtektur lassen sich weiter Rückschlüsse auf die Anzahl der Komponenten und die zu übertragende Datenmenge ziehen. Unten in der Pyramide ist die Anzahl der Komponenten groß, aber die Datenmenge klein. In der Spitze, der Unternehmensleitebene, konzentrieren sich die Daten auf einige wenige PCs, die eine Vielzahl von Daten beeinhalten. Die beschriebene Aufteilung wird in Abbildung 1 nochmals verdeutlicht.

Diese klassische Aufteilung wurde zunächst durch die große Verbreitung von Feldbussystemen und nun weiter durch die Einführung von Echtzeit-Ethernet-Lösungen aufgeweicht. Scheitlin spricht in diesem Zuammenhang sogar von einem „zusammenbrechen der Automatisierungspyramide“ [42]. Es fallen einzelne Ebenen zusammen und man geht heute von einer Struktur mit zwei Ebenen aus. Innerhalb dieses neuen Modells benutzt die IT-Ebene eine Netztopologie und ist über Koppelelemente mit einer schlanken Feldebene verbunden [7].

2.3.2 Netzwerkaufbau und Kommunikationsstruktur

Feldbussystemen im Industrieeinsatz benutzen hauptsächlich Bustopologien, die einen geringen Verkabelungsaufwand bei hoher Teilnehmerzahl benötigen [54]. Weiterhin gebräuchlich ist die Verwendung einer Linienstruktur, bei der einzelne Stichleitungen von einer Hauptleitung abzweigen, aber Stern- oder Ring-Vernetzungen sind ebenso möglich. Die meisten Verfahren benutzen eine deterministische Medienzugangskontrolle. Dabei unterscheidet man zentral gesteuerte Zugriffsmethoden mit Master/Slave Kommunikationsstruktur, wie CAN, Modbus oder auch PROFIBUS[4] und dezentrale Verfahren, bei denen das

[4] Genauer benutzt PROFIBUS ein hybrides Verfahren. Die Slaves senden nur auf Anforderung, die Master bilden untereinander einen logischen Token-Bus.

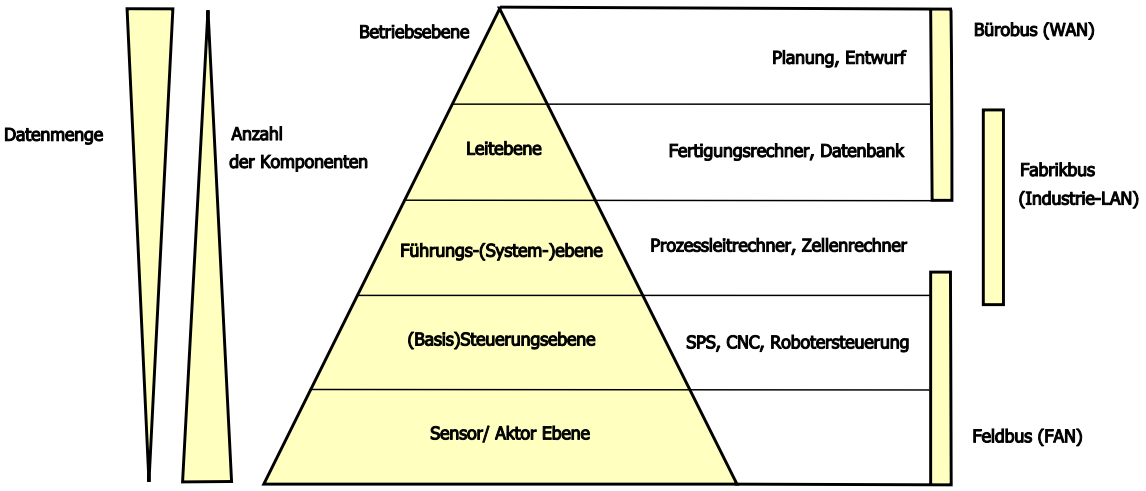

Abbildung 1: Automatisierungspyramide

Senderecht über ein Token geregelt wird. Feldbusse implementieren oft nur die OSI-Schichten 1, 2 und 7, sämtliche Zugriffe auf Objekte werden also direkt über das Anwendungsprotokoll abgewickelt. Dadurch lässt sich in erheblichen Ausmaß Protokoll-Overhead einsparen. Üblich sind dabei Datenraten von mehreren Kilobit/s bis mehreren Megabit/s. Die Variante PROFIBUS-DP unterstützt beispielsweise eine maximale Datenrate von 12 MBit/s, PROFIBUS-DA eine Übertragungsrate von 31,25 Kbit/s. Bei der Kommunikation überwiegen kleine Nachrichten aus nur wenigen Bytes, die häufig auftreten und schnell veralten. Man unterscheidet dabei synchrone, also zyklisch auftretene Nachrichten die die Prozessdaten beeinhalten, und azyklische Nachrichten für Alarme, Diagnose und Projektierung. Ein effektiver Fehlerschutz sorgt für eine unbeschädigte Übertragung der Daten. Eine kurze Einführung anhand von PROFIBUS und CAN bietet [54]; die Publikation von Siemens [46] bietet weitergehende Informationen zu PROFIBUS.

2.4 Ethernet

Ethernet als Technologie für lokale Netze hat sich gegenüber allen anderen Konkurrenten durchgesetzt. Zunächst wird die Entstehungsgeschichte kurz erläutert und auf zentrale Designprinzipien eingegangen. Das verwendete Rahmenformat, wichtige Leistungsdaten und die Medienzugriffssteuerung werden detaillierter betrachtet, da sie für das Verständnis der nachfolgenden Kapitel wichtig sind. Weiterhin erfolgt eine Diskussion der Gründe für den enormen Erfolg dieses Netzwerkstandards. Abschließend wird untersucht, warum Ethernet in der verbreiteten Form nicht für anspruchsvolle Echtzeitanforderungen geeignet ist. In der nachfolgenden Einführung erfolgt nur eine Darstellung von Ethernet über Kupferleitungen, die Varianten mit Glasfaser bleiben unberücksichtigt.

2.4.1 Geschichte

Ethernet wurde 1976 von Metcalfe und Boogs beim Xerox PARC entwickelt [30]. Bei dem Entwurf orientierten sich die Entwickler an dem von der Universität von Hawai entwickelten ALOHAnet. Eine Hauptidee dieses Netzwerks war die Retransmission bei durch Kollisionen verursachten Paketfehlern.

Sie gingen weiterhin davon aus, dass der vorherrschende Verkehr innerhalb eines LANs stoßweise auftritt, so dass die Verwendung eines TDMA-Verfahrens[5] uneffektiv wäre. Aufgrund der positiven Erfahrungen mit dem auf Hawai verwendeten Protokoll entschieden sie sich, ebenfalls auf eine zentrale Steuerung des Medienzugriffs zu verzichten. Für einen genaueren Überblick über die Entstehungsgeschichte von Ethernet und ALOHAnet sei auf [51, 30] verwiesen.

2.4.2 Design

In seiner Originalform wird ein gemeinsam genutztes, passives Datenübertragungsmedium, der sogenannte „Ether“ verwendet. Das Netz ist dabei als Bustopologie aufgebaut. Ethernet verhält sich probabilistisch, Datenübertragungen von einem Teilnehmer kommen also nur mit einer hohen Wahrscheinlichkeit beim Empfänger an. Es muss deshalb auf höheren Protokollschichten sichergestellt werden, dass trotz des verwendeten „best-effort“ Ansatzes keine Daten verloren gehen. Das erste Ethernet bot eine Bandbreite von 3 Mbit/s und arbeitete im Halbduplex-Betrieb.

Solange nur eine oder wenige Stationen Daten über das Netz übertragen, ist die Wahrscheinlichkeit für Kollisionen gering. Sobald allerdings die Zahl der sendewilligen Teilnehmer steigt, vergrößert sich gleichermaßen die Zahl der Kollisionen. Dieses Problem begrenzte den maximalen Durchsatz bei ALOHAnet auf ca. 18% [51]. Um die Effizienz zu verbessern und den Medienzugriff gerechter zu gestalten, muss jeder Teilnehmer eines Ethernet-Netzwerkes einen speziellen Algorithmus verwenden. Diesen Verfahren nennt man CSMA/CD, es wird in Kapitel 2.4.4 beschrieben. Dadurch lässt sich die Leistung erheblich steigern, Tanenbaum stellt in [51] die Berechnung der Effizienz anschaulich dar.

Die Ursprungsform des Netzes nutzte, wie schon angesprochen, ein gemeinsames Medium an dass alle Teilnehmer angeschlossen wurden. Da die Verwendung eines Busses, den man durch alle Teilnehmer führen musste, zu störanfällig war, benutzte man später Hubs zum Aufbau eines Netzes. Diese passiven Koppelelemente änderten aber nichts an der grundsätzlichen Funktionsweise des Netzes. Die Probleme aufgrund des gemeinsames genutzen Medium blieben bestehen, ab einer bestimmten Anzahl von sendewilligen Stationen sinkt die Kanaleffizienz erheblich.

2.4.3 Heutige Installationen

Mittlerweile ist der größte Teil der bestehenden Installationen auf die Geschwindigkeit von 100 Mbit/s ausgelegt, neu installierte Netzwerkschnittstellen unterstützen meist schon 1 Gbit/s. Da die Effizienz bei höherer Auslastung des gemeinsam genutzten Datenübertragungsmediums durch die zahlreichen Kollisionen und Retransmissionen begrenzt wird, schließt man heute alle Teilnehmer an Switches an. Dadurch erreicht man eine Entkopplung der Teilnehmer und es ist ein kollisionsfreies Ethernet möglich. Die Verwendung des Vollduplex-Modus ermöglicht dann auch dass gleichzeitige Senden und Empfangen von Daten, es lässt sich im besten Fall eine Verdopplung der Bandbreite erreichen.

[5] Abkürzung für Time Division Multiple Access, auf Deutsch Zeitmultiplexverfahren.

2.4.4 Carrier Sense Multiple Access / Collision Detection

CSMA/CD, auf Deutsch etwa „Mehrfachzugang mit Trägerprüfung und Kollisionserkennung“ ist ein Protokoll, um den Medienzugriff mehrerer Stationen zu kontrollieren. Zunächst überwacht jede Station, bevor sie sendet, den gemeinsamen Bus auf Übertragungen anderer Teilnehmer. Wenn eine Übertragung stattfindet, wartet die Station, bis diese beendet ist. Dann wartet sie eine bestimmte Zeit, das sogenannte „Interframe Spacing“[6] ab und sendet erst dann. Dadurch kann die Effizienz schon erheblich verbessert werden. Nun können aber immer noch Kollisionen auftreten, wenn zwei oder mehr Netzwerkteilnehmer auf ein Ende einer Datenübermittlung warten und dann gleichzeitig zu Senden beginnen. Deshalb hört jede Station auch während der Übertragung das Medium ab und generiert bei einer Kollision ein sogenanntes Jam-Signal, um anderen Teilnehmern den Fehler zu signalisieren. Kollisionen werden dadurch erkannt, dass eine Station mehr elektrische Leistung aus dem Netz erhält, als sie abgibt [51]. Dann warten die sendewilligen Teilnehmer eine zufällige Zeitspanne, die ein Vielfaches der sogenannten Slot-Zeit entspricht. Tritt beim nächsten Versuch wieder ein Fehler auf, verdoppeln sie die Wartezeit. Dadurch werden die Übertragungsversuche zeitlich entkoppelt, bis irgendwann die erste Station erfolgreich den Sendevorgang abschließen kann. Diesen Verfahren nennt man auch binären exponentiellen Backoff Algorithmus. Abbildung 2 zeigt den Ablauf des CSMA/CD-Verfahren als Zustandsdiagramm nochmals im Detail. Aus Gründen der Übersichtlichkeit wird auf die Darstellung der Berechnung der Backoff Zeit verzichtet.

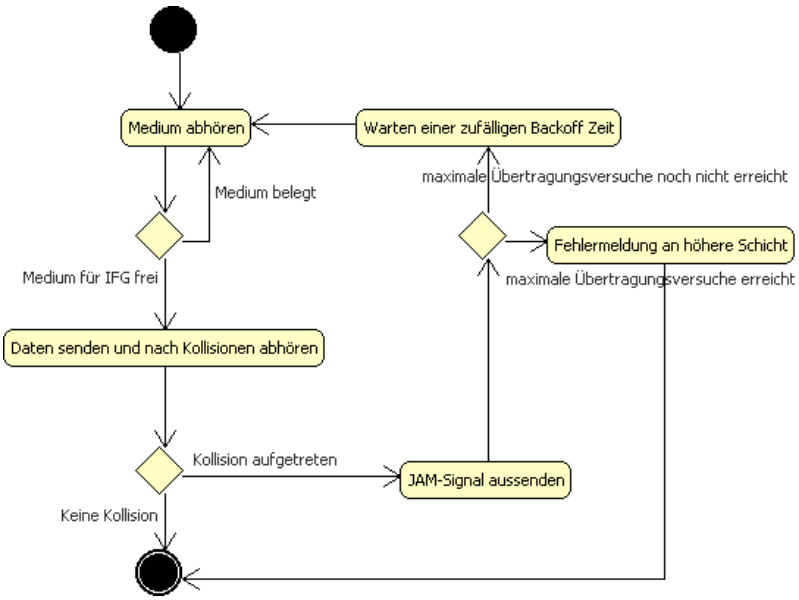

Abbildung 2: CSMA/CD Algorithmus

Um Kollisionen zuverlässig zu erkennen, ergeben sich einige Anforderungen an das Netz. Da bei Sendevorgängen das Jam-Signal immer vor dem Ende einer Übertragung beim Sender ankommen muss, legt man eine minimale Rahmengröße fest. Diese wird aus der vollen Round-Trip Signallaufzeit abgeleitet und ergibt sich damit aus der maximalen Größe des Netzes und den Verzögerungen eventuell vorhandener zusätzlicher Netzwerkhardware. Da bei höheren Geschwindigkeiten die Dauer eines Bits immer geringer wird, muss man in diesem Fall entweder größere Rahmen einsetzen oder eine Verkleinerung

[6] Abgekürzt „IFS“ oder auch „IFG - Interframe Gap“

des Netzes vornehmen. Da bei der Nutzung von CSMA/CD auch bei Verwendung der minimalen Wartezeit ebenfalls eine Erkennung von Kollisionen möglich sein muss, entspricht die Slot-Zeit ebenfalls der vollen Round-Trip Signallaufzeit. Durch diese Abhängigkeit der Netzwerkgröße von dem CSMA/CD Algorithmus treten bei höheren Geschwindigkeiten einige Probleme auf, siehe auch den Abschnitt „Zu große Rahmen“ auf Seite auf Seite 12.

Da die meisten Echtzeit-Ethernet-Ansätze eine komplette Verhinderung von Kollisionen anstreben, beispielsweise durch die Verwendung von Vollduplex-Verbindungen mit Hilfe von Switches, genügt diese Darstellung für den folgenden Teil der Arbeit. Für eine exakte Spezifikation des CSMA/CD-Verfahrens und des binären exponentiellen Backoff Algorithmus sei auf den IEEE 802.3 Standard verwiesen [25]. Auch bei Vollduplex-Verbindungen treten immer noch Probleme auf, die einen Einsatz von gewöhnlichen Ethernet für Echtzeitanwendungen erschweren, diese werden in Kapitel 2.4.8 dargestellt.

2.4.5 Wichtige Leistungsdaten

Um eine zeitliche Perspektive, die für die Betrachtung der Echtzeitfähigkeit notwendig ist, zu erhalten, sind in Tabelle 3 einige wichtige Leistungsdaten von 100 Mbit/s Ethernet aufgeführt. Diese Leistungsklasse stellt den Großteil der Installationen derzeit dar, und wird ebenfalls von den meisten Echtzeit-Ethernet-Ansätzen verwendet oder zumindest empfohlen.

Bezeichnung	Wert	Anmerkung
Bitdauer	10 ns	Übertragungsdauer eines Bits auf der Leitung
IFG	0.96 μs	Zeitdauer ab der das Medium als frei angenommen wird
Slot-Zeit	512 Bitzeiten	Dauer eines Slots für den CSMA/CD-Algorithmus
Jam-Länge	32 Bit	Größe des Jam-Signals zur Kennzeichnung einer Störung
Min. Rahmengröße	512 Bit/ 64 Bytes	Minimale Größe eines Frames (ohne Präambel und SFD)
Max. Rahmengröße	1518 Bytes	Maximale Größe eines Frames (ohne VLAN Tag)

Tabelle 3: Leistungsdaten 100 MBit/s Ethernet

2.4.6 Ethernet Rahmenformat

Der Standard IEEE 802.3 [25] definiert zwei Rahmenformate, das sogenannte „Basic MAC frame format“ sowie das „Tagged MAC frame format“. Diese Formate unterscheiden sich nur durch ein zusätzliches VLAN-Tag. Abbildung 3 zeigt den allgemeinen Aufbau des Rahmens in den beiden Varianten [53].

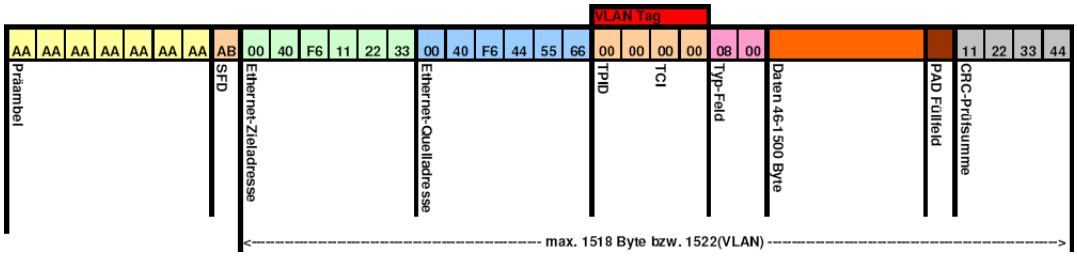

Abbildung 3: Rahmenformat Ethernet nach IEEE 802.3 [53]

Header des Ethernet-Rahmens Beide Rahmenformate beginnen mit einer sieben Byte langen Präambel, die zur zeitlichen Synchronisation des Empfängers dient und einen SFD-Byte, dem „Start-Frame-Delimiter“, der den Start des Rahmens ankündigt. Nun folgen die Ziel- und Quelladresse von jeweils sechs Byte Länge. Als Ziel können individuelle Adressen oder Multi- bzw. Broadcastadressen angegeben werden. Die Quelladresse wertet die Medienzugriffsschicht nicht aus. Das zwei Byte lange Typ- bzw. Längenfeld wird je nach Inhalt unterschiedlich ausgewertet. Ist sein Wert kleiner oder gleich der maximalen Rahmenlänge, wird es als Längenfeld ausgewertet. Ist sein Wert größer als $0x0600_{hex}$, gibt es den Typ des Rahmens an.

Besonderheiten beim Tagged-Rahmen Bei Tagged-Rahmen wird das Typfeld auf $0x8100_{hex}$ gesetzt, und noch zwei Byte Tag-Informationen hinzugefügt. Es folgt ein zusätzliches Typ-/Längenfeld, ebenfalls von zwei Bytes, dass die Länge oder gegebenenfalls den ursprünglichen Typ des markierten Rahmens angibt. Die Tag-Informationen enthalten drei Bit Informationen über die Benutzerpriorität, ein Bit für Sonderzwecke, und eine zwölf Bit lange VLAN-Id. Das genaue Format dieser Erweiterung ist in dem IEEE Standard 802.1Q [26] definiert. Die Auswertung der Benutzerpriorität erfolgt als Binärzahl und kann damit acht Prioritätsabstufungen wiedergeben. Weitere Informationen über die Nutzung und Interpretation dieses Feldes kann man dem IEEE Standard 802.1D [27] entnehmen. Der CFI, „Canonical Format Indicator“ legt die Bit-Anordnung der Adressfelder fest, er ermöglicht also die Verwendung von little-endian oder big-endian Adressen. Im Id-Feld, auch VID - „VLAN Identifier“ genannt, wird festgelegt zu welchem VLAN der Rahmen gehört. Er wird gleichermaßen als einfache binäre Nummer ausgewertet und der Nutzer kann ihn frei[7] verwenden.

Benutzerdaten, Padding und Fehlerschutz Der restliche Aufbau ist nun bei beiden Formaten wieder identisch. Es folgen die Benutzerdaten und ein Padding-Feld, das wenn notwendig den Rahmen auf die minimale Größe auffüllt. Die 32 Bit CRC-Prüfsumme im FCS, der „Frame-Check-Sequence“ ermöglicht die Erkennung von Übertragungsfehlern. Bei speziellen Übertragungsmodi[8] wird noch ein Erweiterungsfeld angehangen, um die Daten auf eine Slot-Zeit zu verlängern. Für eine exakte Spezifikation sei auf die schon genannten Standards verwiesen.

2.4.7 Gründe für den Erfolg

Ethernet wurde vor ca. 30 Jahren entwickelt und hat alle anderen Konkurrenten verdrängt. Es ist heute die mit Abstand wichtigste Technologie für den Aufbau von lokalen Netzen. Diese lange Erfolgsgeschichte ist gerade in der schnellen Umbrüchen unterworfenen IT-Welt etwas besonderes. Warum hat sich Ethernet durchgesetzt?

Hauptgrund für den Erfolg der Ethernet-Technologie ist wohl, dass sie einfach und flexibel ist. Durch die Beschränkung auf einen „best-effort“ Ansatz bei der Datenübertragung konnten Netzwerkschnittstellen

[7] Bis auf einige Werte für Sonderzwecke.
[8] Geschwindigkeit größer als 100 Mbit/s im Halbduplex-Betrieb.

sehr günstig entwickelt und produziert werden. Die Netzwerke sind einfach aufzubauen, Rechner lassen sich problemlos an einen Switch anschließen und funktionieren dann sofort mit fast allen Anwendungen, ohne aufwändige Konfiguration. Weiterhin arbeitet es sehr gut mit TCP/IP zusammen, es konnte also direkt von dem Internet-Boom profitieren. IP passt als verbindungsloses Protokoll genau zu dem ebenfalls verbindungslosen Ethernet. Und Ethernet konnte sich in vielen wichtigen Punkten anpassen und weiterentwickeln, so wird mittlerweile mit 10 Gbit/s eine um mehrere Größenordnungen höhere Geschwindigkeit bereitgestellt. Durch die Verwendung von Switches anstelle von Hubs konnte die Effektivität der Netzwerke erheblich gesteigert werden. Ein entschiedener Ansatz bei der Erhaltung der Kompatibilität zu Vorgängergenerationen machte es Konkurrenten sehr schwer, in den Markt einzudringen.

2.4.8 Echtzeitanforderungen mit Ethernet

Ethernet ist in der verbreiteten und durch die IEEE standardisierten Form nicht für Echtzeitanwendungen geeignet. Für diesen Anwendungsfall reicht der Ansatz die Daten „so gut wie möglich“ zu übertragen nicht aus, es sind höhere Anforderungen zu stellen. Im Folgenden werden ausgewählte Problemfelder des Ethernet-Standards ausführlich dargestellt.

Nichtdeterministisches Verhalten Durch die Verwendung des CSMA/CD-Algorithmus und eines gemeinsamen Übertragungsmediums treten bei Kollisionen unvermeidlich Verzögerungen auf. Diese sind nicht deterministisch und hängen von der Auslastung des Netzes ab. Bei zu großer Auslastung können unter Umständen nur noch sehr wenige Daten korrekt übertragen werden, die Verzögerung wird sehr groß.

Durch die Nutzung von Switches anstelle von Hubs kann man die Problematik des CSMA/CD Verfahrens entschärfen, bei Vollduplex-Verbindungen können keine Kollisionen mehr auftreten. Man erhält dadurch aber zusätzliche Latenzen, da Rahmen oder zumindestens Teile davon in den Switches zur Auswertung zwischengespeichert werden müssen. Hubs haben eine Latenz von ca. 0,5 μs, da sie Rahmen nicht zwischenspeichern. Switches dagegen, selbst wenn sie als „Cut-through-Switch“[9] arbeiten, benötigen etwa 10 μs für die Weiterleitung eines Rahmens [12], laut Bormann sogar eine Latenz von ca. 40 μs [3]. Aus diesem Grund verzichten einige Echzeit-Ethernet-Verfahren auf die Verwendung von Switches oder modifizieren sie, um eine höhere Leistung zu erreichen.

Switches können noch weitere negative Effekte innerhalb eines Netzwerks verursachen. Bei Überlast einzelner Schnittstellen müssen einzelne Rahmen gepuffert werden. Es entstehen nichtdeterministische Verzögerungen, der daraus resultierende Jitter beeinträchtigt Echtzeitanwendungen in ihrer Leistung. Bei zu starker Überlastung senden gewöhnliche Switche spezielle Pause-Rahmen[10], um ihre Kommunikationspartner zum temporären Einstellen der Datenübertragung zu veranlassen. Dadurch wird die Verbindung für eine vom Switch festgelegte Zeitdauer unterbrochen. Die maximale Länge der Pause beträgt ca. 0,33 s und ist damit zu lang für Echtzeitanforderungen. Bei Deaktivierung der Flusskontrolle können Daten verloren gehen bzw. werden bei Überlast sogar verworfen, auch durch diese Maßnahme lässt sich also kein determinisches Netzwerkverhalten erreichen.

[9] Hierbei erfolgt keine vollständige Zwischenspeicherung des Rahmens, sondern er wird direkt nach Auswertung der Adresse weitergeleitet.

[10] Hier wird nur der verbreitete Vollduplex-Betrieb beschrieben, bei Halbduplex verwendet man mit „back-pressure“ ein anderes Flusskontrollverfahren.

Zu große Rahmen Weiterhin sind die maximalen zulässigen Rahmen zu groß. Die Übertragungsdauer eines Rahmens beeinflusst direkt die mögliche Zykluszeit der Anlage. Bei der Verwendung von 100 MBit/s Ethernet dauert die Übertragung eines Rahmens der maximalen Größe 122 μs.

$$\begin{aligned} t_{max} &= t_{bit} \cdot n_{max\,rahmen} \\ &= 10\,\text{ns} \cdot (1518 + 8) \cdot 8 \\ t_{max} &\approx 122\,\mu\text{s} \end{aligned}$$

Dazu kommen Verzögerungen durch den IFG, er beträgt bei 100 MBit/s Netzwerken 0,96 μs. Ein System, welches die Kompatibilität zu gewöhnlichen Ethernet und deren großen Rahmen erhalten will, hat dadurch ab einer geringen Netzwerkgröße schon Schwierigkeiten, die in Tabelle 2 aufgeführte Klasse 3 „Motion-Control“ zu erreichen. Durch den Einsatz eines Netzes mit einer höheren Geschwindigkeit wie bei 1 Gbit/s Ethernet verringern sich diese Probleme zwar, ein solcher Systemwechsel bringt allerdings neue Komplikationen mit sich. Um bei dieser Geschwindigkeit unter der Verwendung von Halbduplex-Verbindungen eine akzeptable maximale Netzwerkgröße zu erreichen, wird die minimal erlaubte Rahmenlänge angehoben [25]. Dadurch sinkt die Effizienz bei der Nutzung kleiner Frames. Weiterhin können mehrere Rahmen zusammengefasst werden, dies kann andere Übertragungen unverhersehbar verzögern[11]. Als zusätzliche Maßnahme darf die Medienzugangsschicht eine bestimmte Anzahl zusätzliche „Füllrahmen“ übertragen. Weitere Informationen zu dem Verhalten bei dieser Geschwindigkeit finden sich in dem genannten Standard [25].

Rahmenaufbau Die minimale mögliche Zykluszeit ergibt sich analog aus der minimalen Rahmengröße von 64 Bytes, sie beträgt in diesem Fall[12] ein Vielfaches von 5,76 μs. Allerdings ist die Effizienz bei Verwendung der kleinstmöglichen Rahmen nicht gut, da der Anteil des Headers zu groß wird. Bei 72 Bytes realer Rahmengröße ergibt sich eine Effizienz von unter 64%.

$$\begin{aligned} E &= \frac{n_{daten}}{n_{min\,rahmen}} \\ &= \frac{64 - ((2 \cdot 6) + 2 + 4)}{64 + 8} \\ E &\approx 0,64 \end{aligned}$$

Hier veringert der IFG ebenfalls die Effizienz noch weiter. Weiterhin hängt bei Ethernet die Kanaleffizienz direkt von der Rahmengröße ab. Nach [51] ist die Kanaleffizienz von Ethernet

$$K = \frac{1}{1 + \frac{2BLe}{cF}}$$

wobei B die Bandbreite, L die Kabellänge, e die Anzahl der Konkurrenzzeitschlitze[13], c die Ausbreitungsgeschwindigkeit und F die Rahmenlänge darstellt. Man erkennt anhand dieser Gleichung, dass mit der Verwendung von größeren Rahmen die Kanaleffizienz steigt.

[11] Diese beiden Maßnahmen werden auch „carrier extension“ und „frame bursting“ genannt.

[12] Inklusive Präambel und SFD.

[13] Damit wird die Anzahl der Zeitschlitze pro Konkurrenzintervall bezeichnet, weitere Informationen dazu finden sich in der angegebenen Quelle [51].

Inadäquate Netzwerktopologie Eine weitere Einschränkung ergibt sich aus der heute verwendeten Netzwerk-Topologie. Bei den Feldbussen hat man aus Kostengründen eine Sternverkabelung vermieden. Mit der Ethernet-Technologie benötigt man wieder eine sternförmige Verbindung der Teilnehmer. Da dies für die Automatisierungstechnik nicht akzeptabel ist, kann man eine Busstruktur durch Einsatz eines Switches in jeden Endgerät aufbauen. Dadurch erhält man aber zusätzliche Komplexität in jedem Teilnehmer und unerwünschte zusätzliche Latenzen, siehe auch den Abschnitt „Nicht deterministisches Verhalten“ auf auf Seite 11.

2.5 Gründe für den Einsatz von Echtzeit-Ethernet

Warum möchte man nun Ethernet trotz der zahlreichen zu überwindenen Schwierigkeiten als Feldbus-Ersatz in der Industrie einsetzen? Zunächst hofft man, von den in Kapitel 2.4.7 genannten Vorteilen wie günstige Netzwerkadapter, Flexibilität und einfache Konfiguration des Netzwerkes auch innerhalb der Industrie zu profitieren. Die weite Verbreitung der Internetprotokolle TCP/IP erfordert ebenfalls teilweise innerhalb der Automatisierungstechnik eine Verbindung zum Internet. Als Beispiel für eine solche Anforderung ließe sich die Visualisierung des Produktionsprozesses bestellter Waren für den Kunden im Internet aufführen. Man hofft, mit der Verwendung von Echtzeit-Ethernet eine Durchgängkeit der Unternehmensprozesse von der Verwaltung bis hin zur Produktion zu erhalten. Dabei spricht man auch von vertikaler Integration der Unternehmensebenen. Davon verspricht man sich eine Effizienzsteigerung der Produktion, einen besseren Informationsfluß von der Fertigung bis in die Unternehmensleitung und ganz allgemein eine höhere Wettbewerbsfähigkeit des Unternehmens.

Sicherlich findet überdies im Moment ein Marketing-Hype zu dem Thema statt. Einige Hersteller versuchen, mit halbfertigen Lösungen auf den RT-Ethernet-Zug[14] aufzuspringen. Potentielle Anwender realisieren oft nicht, dass für den Einsatz in der Industrie eben nicht die gewöhnlichen Ethernet-Netzwerkschnittstellen oder Switche „für einige Euro aus dem Technologiemarkt“ ausreichen, sondern höhere Anforderungen bestehen (siehe auch Kapitel 2.2). So kostet beispielsweise ein 4-Port Ethernetswitch, der für den Industrieeinsatz geeignet ist, mehrere hundert Euro [2].

Echtzeit-Ethernet Systeme müssen teilweise erheblichen Aufwand betreiben, um die eigentlich nicht echtzeitfähige Grundlage an die Anforderungen anzupassen. Dies treibt die Kosten nach oben und vergrößert die Komplexität der Protokolle und Hardware. Komplexe Systeme lassen sich schwerer verstehen, analysieren und warten, sind unsicherer und weniger zuverlässig [50, 43]. Schneiers Aussage, „Komplexität ist der schlimmste Feind der Sicherheit“ [43] ist sicherlich auch im Bereich der Ethernet-Echtzeitnetzwerke gültig. Für bestimmte abgegrenzte Anwendungsfälle, bei denen keine Vorteile durch die Integration von Prozessen zu erwarten sind oder hohe Sicherheitsanforderungen bestehen, ist die Verwendung von einfachen Feldbussystemen sinnvoller und wohl auch ökonomischer.

Durch die Verbindung der Fertigungsnetze mit dem Internet, oder auch nur mit dem firmeninternen Intranet treten neue Gefahren auf. Der Übergang der Netze muss gut gesichert werden, sonst können Viren oder ähnliche Schädlinge, die sonst „nur“ in Büronetzwerken für Probleme sorgen, sich auf die Produktion auswirken. Schon im Jahr 2003 drang ein Internet-Wurm über das öffentliche Netz in die Steuerung eines amerikanischen Kernkraftwerkes ein und deaktivierte für mehrere Stunden wichtige Sicherheitssysteme [37]. Diese neue Risiken müssen angemessen berücksichtigt werden, sie vergrößern unvermeidlich

[14] Im Folgenden werden die Abkürzung RT und RTE für „Real-time“ bzw. „Real-Time Ethernet“ verwendet.

die Komplexität der Gesamtlösung. Im Rahmen dieser Arbeit erfolgt aber keine weitergehende Betrachtung dieser Problemfelder.

2.6 Ansätze für Echtzeit-Ethernet

Zur Vorbereitung auf den folgenden Teil der Arbeit erfolgt nun einen Überblick über die verschiedenen Ansätze, um mit Ethernet nach dem Standard IEEE 802.3 Echtzeitfähigkeit zu erreichen. Zunächst lassen sich die Verfahren in zwei große Klassen einteilen [44]. Des Weiteren können die einzelnen Systeme anhand der Herangehensweise der Hersteller nochmals differenziert werden.

2.6.1 Sicherstellen einer rechtzeitigen und deterministischen Datenübertragung

Zum einen kann durch verschiedene Maßnahmen eine rechtzeitige und deterministische Übertragung der Daten zu den Teilnehmern sichergestellt werden. Das von Ethernet bei Halbduplex-Verbindungen verwendete CSMA/CD stellt ein Problem vor allen für die deterministische Kommunikation dar. Daher befasst sich eine Vielzahl von Lösungen mit der Umgehung dieses Verfahrens. Der einfachste Weg zur Erreichung dieses Ziels ist die durchgängige Verwendung von Vollduplex-Verbindungen und dass garantieren einer geringen Netzwerklast. Durch die geringe Auslastung des Netzes und gleichzeitiger Verwendung von 100 MBit/s Verbindungen lassen sich die durch Pufferung in Switches auftretenen Latenzen vermeiden. Durch Modifikation der Netzwerkgeräte lässt sich mit Hilfe eines Zeitmultiplexverfahrens sicherstellen, dass immer nur ein Teilnehmer zeitgleich senden darf. Durch Master-Slave Architekturen, bei denen man die Teilnehmer nacheinander abfragt, ist auch ein zeitlich differenzierter Zugriff auf das gemeinsame Netzwerkmedium zu erreichen. Durch zusätzliche Maßnahmen wie die Priorisierung von wichtigen Kommandos und die Verringerung der Latenz innerhalb des Netzwerkes lässt sich allgemein die Echtzeitfähigkeiten der Systeme vergrößern.

2.6.2 Exakte Synchronisation der Teilnehmer

Zum anderen ist es für ein System, dass sich sychron verhalten soll, ausreichend wenn alle Uhren innerhalb der Teilnehmer exakt gleichlaufen. Nun lassen sich Kommandos mit genügend Vorlauf übertragen, Zeitstempel sorgen dann für eine gleichzeitige Ausführung innerhalb des Netzwerkes. Die einzelnen Geräte können hier selbst entscheiden, wann sie ihre Befehle auswerten, sie benötigen also eine größere Eigenintelligenz als in der vorherigen Klassifizierung. Die verschiedenen Echtzeit-Ethernet-Ansätze verwenden einen proprietären Ansatz oder nutzen das offene und standardisierte Protokoll nach IEEE 1588 [17] zur Synchronisation der einzelnen Teilnehmer. In Abbildung 4 werden die vorgestellten Ansätze nochmals zusammengefasst dargestellt.

2.6.3 Herangehensweise der Hersteller

Eine zusätzliche Klassifizierung bietet die Herangehensweise der Hersteller an das Problem „Echtzeit-Ethernet“. Der einfachste Ansatz ist sicherlich, die applikationsspezifischen Daten des Feldbus in einen gewöhnlichen Ethernet-Rahmen zu kapseln. Allerdings verliert man dadurch die Echtzeiteigenschaften

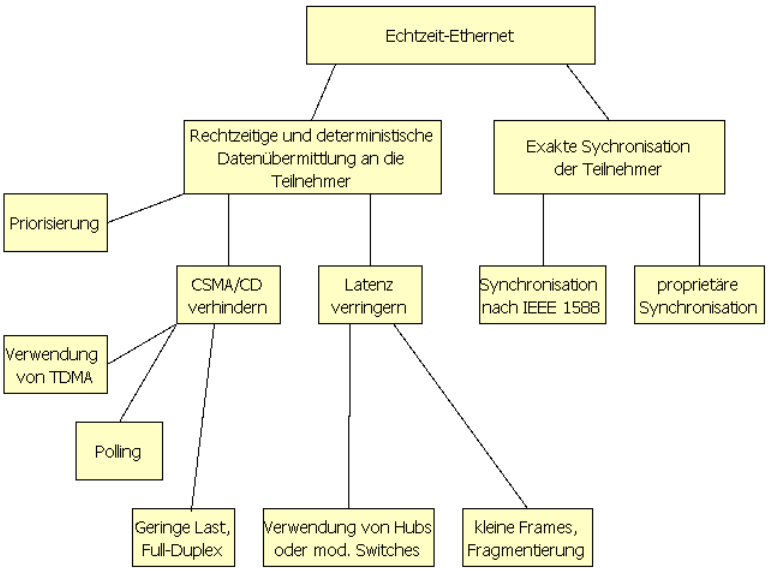

Abbildung 4: Ansätze für Echtzeit-Ethernet

des ursprünglichen Protokolls, zusätzlich steigt der Overhead beträchtlich an. Zwar kann die Applikationsschicht des ursprünglichen Systems fast unverändert übernommen werden, aber die entstehende Gesamtlösung bleibt natürlich weiter proprietär [45]. Der Gegensatz zu diesem Vorgehen besteht in der Entwicklung eines komplett neuen System inklusive der Applikationsschicht. Dabei besteht allerdings das Problem, dass eine Brücke zu bestehenden Installationen geschaffen werden muss. Natürlich sind auch Mischlösungen zwischen diesen beiden Vorgehensweisen möglich.

3 Untersuchung gängiger Echtzeit-Ethernet Ansätze

Nachdem die grundsätzlichen Ansätze für die Realisierung von RTE-Lösungen[15] dargestellt wurden, werden nun konkrete Systeme betrachtet. Zunächst erfolgt eine Diskussion der Kriterien, die zu der Auswahl der einzelnen Ansätze geführt haben. Daran schließt sich eine detaillierte Darstellung ihrer Funktionsweise an. Zum Schluss dieses Kapitels wird ein Bewertungskatalog formuliert und auf die vorgestellten Verfahren angewendet.

3.1 Auswahlkriterien der Systeme

Zur Klassifizierung der einzelnen Ansätze für Echtzeit-Ethernet bietet sich eine Einteilung anhand des OSI-Schichtenmodells an. Felser teilt in [13] die Verfahren danach in verschiedene Klassen ein. Die einfachste Form von RTE-Systemen sind mit ihren Protokollschichten nur oberhalb von TCP/IP angesiedelt. Sie nutzen einen herkömmlichen Netzwerkstack bis einschließlich OSI-Schicht 4. Der nächste Klasse tauscht die Vermittlungs- und Netzwerkschicht durch ein Echtzeitprotokoll aus, diese Verfahren bewegen sich also oberhalb von Ethernet. Die Modifikation der Ethernet-Schichten stellt schließlich den einschneidensten Ansatz zur Erlangung von Echtzeitfähigkeit dar. Hier werden alle Schichten verändert, bei manchen Verfahren sogar die Protokolle der Bitübertragungsschicht[16]. Die vorgestellte Einteilung ist in Abbildung 5 nochmals zusammengefasst dargestellt.

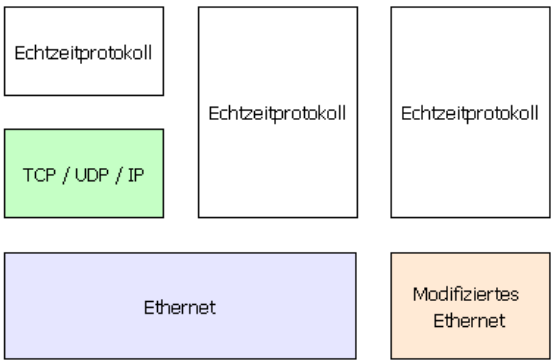

Abbildung 5: Einteilung Echtzeit-Ethernet Verfahren

Bei der Auswahl der einzelnen Systeme wird darauf geachtet, dass diese Klassen abgedeckt sind. Weiterhin erfolgt eine Berücksichtigung der in Abschnitt 2.6 vorgestellten Ansätze für RT-Ethernet[17]. Die nachfolgend betrachteten Verfahren implementieren einen oder mehrere der dort genannten Herangehensweisen.

3.2 Vorstellung der einzelnen Verfahren

Bei der genaueren Darstellung der einzelnen Verfahren ist gleichermaßen eine Anlehnung an das OSI-Schichtenmodell sinnvoll. Hierbei wird sich größenteils auf die Transport orientierten unteren vier Schichten beschränkt und nur kurz auf die jeweils verwendeten Applikationsmodelle eingegangen. Der Fokus

[15] Im Folgenden wird die Abkürzung RTE für „Real-time Ethernet“ verwendet.
[16] Wie bei EtherCAT, das einen sogenannten E-Bus als alternative Schnittstelle zu Ethernet anbietet.
[17] Im weiteren Verlauf der Arbeit wird die Abkürzung RT für den Begriff „Real-time“ verwendet.

liegt also auf der Untersuchung der konkreten technischen Funktionsweise der implementierten unteren Protokolle. Eine Untersuchung der Mechanismen zur Projektierung, Inbetriebnahme und Wartung findet nicht statt. Weiterhin erfolgt eine Betrachtung der möglichen Netzwerktopologien und der allgemeinen Leistungsmerkmale, da diese Parameter von entscheidener Bedeutung für den Einsatz im Feld sind. Zum Schluss stehen dann Eigenschaften im Vordergrund, die die Investitionssicherheit und Zusammenarbeit mit anderen Systemen beeinflussen. Hier sind insbesondere die Anzahl bereits installierter Systeme, die Kompatibilität zu vorhandenen Installationen bzw. normalen Ethernet und eventuelle Standardisierungsbemühungen zu nennen.

3.2.1 EtherNet/IP

Das „EtherNet/Industrial Protocol“ stellt eine Lösung auf der Basis von TCP/IP dar. Es ist ein vergleichsweise einfaches Protokoll, das von der Firma Rockwell konzeptioniert wurde. EtherNet/IP nutzt erweiterte Ethernet-Frames wie im Kapitel 2.4.6 beschrieben, die es über normale Switche versendet. Dabei erfolgt eine Priorisierung von Echtzeitnachrichten gegenüber den normalen Daten durch die Auswertung eines VLAN-Tags im Header der Frames nach dem Standard IEEE-802.1Q [26]. Durch die Verwendung von 100 Mbit/s Verbindungen im Vollduplex-Betrieb können keine Kollisionen innerhalb des Datenübertragungsmediums auftreten. Eventuelle Verzögerungen oder Datenverluste aufgrund der Überlastung eines Switch-Ports lassen sich mittels der Vergabe von Prioritäten für Echtzeitnachrichten verringern. Durch den verwendeten Ansatz können ohne weiteres gewöhnliche Teilnehmer in das Netzwerk eingebunden werden, um beispielsweise Recherchen im Internet durchzuführen.

Bei Nutzung der Erweiterung CIPsync, eine Implementation des IEEE-1588 Protokolls zur Uhrensynchronisation, lässt sich eine zeitliche Genauigkeit von 0,5 μs erreichen. Dadurch lassen sich geplante Ereignisse unabhängig von eventuellen durch das Netzwerk verursachten Verzögerungen ausführen. EtherNet/IP ist damit also eine Lösung, die für hohe Echtzeitanforderungen geeignet ist, aber aufgrund der Beschränkung auf gewöhnliches Ethernet mit Nachrichten-Priorisierung keine deterministische Kommunikation ermöglicht [13]. Als mögliche Leistung wird in [4] ein Jitter von unter 200 ns bei einer Zykluszeit von 1 ms und 100 belieferten Achsen genannt. Diese Leistungswerte erscheinen allerdings ein wenig optimistisch[18], der eingangs genannte Jitter von etwa einer halben Mikrosekunde ist wohl realisischer.

Aktuell stellt EtherNet/IP keine Anforderungen an die konkrete Implementierung oder Leistungsfähigkeit von Netzwerkteilnehmern, es verfolgt einen „best-effort“ Ansatz. Es sind aber Performance-Metriken geplant, mit deren Hilfe kann dann eine bestimmte Geschwindigkeit von Endgeräten angefordert werden [33]. Da Standard-Ethernet genutzt wird, ist nur eine Stern- und Baumtopologie möglich, es besteht aber die Möglichkeit, in jedem Endgerät einen kleinen Switch vorzusehen, und darüber eine Linienstruktur zu verwirklichen. Die RTE-Lösung stellt ein Erzeuger/Verbraucher Modell für die Kommunikation einzelner Geräte zur Verfügung, dabei wird auch der Informationsaustausch von einem Teilnehmer zu mehreren mittels IP-Multicast unterstützt. Die Einbindung von gewöhnlichen Ethernetgeräten ist problemlos möglich, durch die bevorzugte Behandlung von Echtzeitnachrichten kann die Leistungsfähigkeit des Netzes für diese Teilnehmer bei eventueller Überlastung leiden.

[18] Insbesondere unter dem Gesichtspunkt, dass die Autoren des zitierten Artikels bei der Herstellerfirma beschäftigt sind.

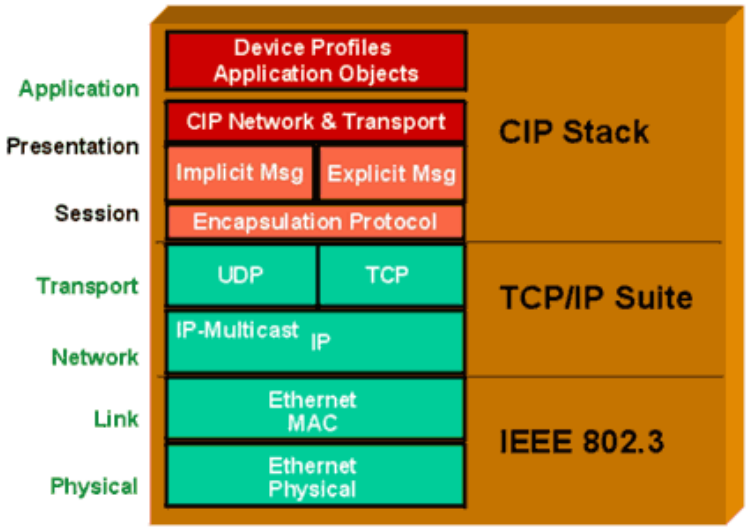

Abbildung 6: Protokollaufbau EtherNet/IP [16]

Innerhalb der Applikationsschicht wird CIP, das „Control and Information Protocol“ verwendet; dieses Protokoll nutzen ebenfalls die Feldbussysteme ControlNet und DeviceNet. Es erfolgt eine Kapselung der CIP-Daten in TCP/IP bzw. UDP Paketen, je nach dem ob eine zuverlässige oder schnelle Übertragung gefordert ist [44]. Die Abbildung 6 zeigt den Protokollaufbau des EtherNet/IP-Gesamtsystems [16]. Bestehende Feldbussysteme auf CIP-Basis lassen sich durch die Beibehaltung der Anwendungsschicht relativ leicht in das Netzwerk eingliedern.

Die Spezifikation von EtherNet/IP erfolgte erstmals 1998 von ControlNet International. Seit dem Jahr 2000 erfolgt eine zusätzliche Unterstützung durch die ODVA (Open DeviceNet Vendor Association). Das Protokoll ist standardisiert in der Norm IEC 61784-1 [18] als Communication Profile 2/2 [13]. EtherNet/IP ist durch sein Alter und den verwendeten vergleichsweise einfachen Ansatz weit verbreitet, Chaffee [4] gibt eine Anzahl von einer Million gelieferter Netzwerkgeräte für den Mai 2006 an. Die Erweiterung CIPsync, auch CIP Motion genannt, ist erst ab 2007 verfügbar. Zusätzliche Angaben über den genauen Aufbau lassen sich der Spezifikation entnehmen. Sie ist nach Angabe der persönlichen Daten auf der Webseite der ODVA [31] herunterzuladen.

3.2.2 PROFINET

Im Folgenden erfolgt nur eine Betrachtung von PROFINET IO, PROFINET CBA wird nicht behandelt. PROFINET Input Output ist eine Automatisierungslösung für den Feldbereich und arbeitet mit modifizierten Ethernet, PROFINET Component Based Automation ermöglicht es, verschiedene verteilte Anlagen miteinander zu vernetzen und ist deshalb innerhalb dieser Arbeit nicht von Interesse. Die Kommunikation wird im untersuchten Ansatz in drei Stufen mit jeweils unterschiedlichen Leistungsanforderungen eingeteilt. Normaler Verkehr ohne Echtzeitanforderungen nutzt TCP/UDP, darunter fallen auch Aufgaben wie Parametrierung und Konfiguration. Die nächste Stufe stellt die Echtzeitkommunikation im Modus RT (Real Time) dar, noch höhere Anforderungen lassen sich mit der IRT (Isochronous Real Time) Betriebsart erfüllen. Abbildung 7 stellt den Aufbau der Protokollstruktur von PROFINET schematisch dar [47].

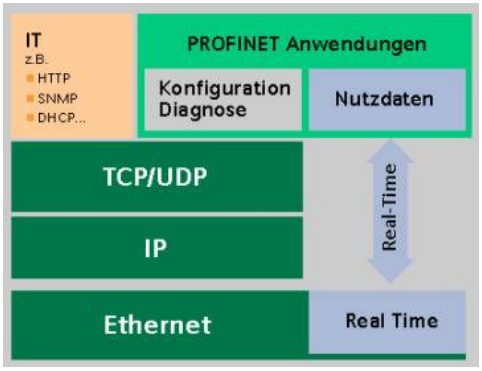

Abbildung 7: Protokollstruktur PROFINET [47]

Im RT-Modus (auch asynchroner Modus genannt) kann man Standard-Ethernetkontroller in den Geräten nutzen, es handelt sich also um eine reine Softwarelösung. Zum garantieren der Leistung dürfen innerhalb des Netzwerkes ausschließlich Switche im Vollduplex-Betrieb mit einer Geschwindigkeit von 100 Mbit/s verwendet werden, zusätzlich ist ein Sicherheitsintervall am Ende des Zyklus notwendig [36]. Echtzeitdaten erhalten wie in EtherNet/IP nach IEEE-802.1Q eine erhöhte Priorität zugewiesen, bei Verwendung von PROFINET Switchen ist aber auch die Priorisierung nur anhand des Rahmentyps möglich. Durch diese Maßnahmen lassen sich Zykluszeiten von fünf bis zehn Mikrosekunden realisieren. Für den IRT-Betrieb – er wird auch als synchroner Betrieb bezeichnet – sind spezielle zwei- oder vier-Port-Switche notwendig, hier benötigen die Netzwerkteilnehmer ebenfalls eine spezielle Netzwerkschnittstelle mit integrierten Switch. Dadurch lassen sich zusätzlich zu einer gewöhnlichen Sternstruktur auch Linien- und Ringtopologien verwirklichen. Alle PROFINET-Switche synchronisieren sich untereinander mit Hilfe eines modifizierten IEEE-1588 Verfahren [13], nach Popp [36] handelt es sich dabei um das „Precision Transparent Clock Protocol“ nach IEC-61158. Damit sind Zykluszeiten von weniger als 1 ms möglich, der zu erreichende Jitter hängt direkt von der Anzahl hintereinander gereihter Switche ab. Bei 25 Switchen ergibt sich ein Jitter von 1 μs [45]. Laut Schaffee und Hirschinger ist damit eine Aktualisierung von 150 Achsen möglich, die minimale Zykluszeit ist wie bei EtherNet Powerlink 200 ms [4]. Popp gibt in [36] allerdings eine minimale Zykluszeit von 156,25 μs für den IRT-Betrieb an.

Die RT-Betriebsart unterstützt den Datenaustausch nach einem Erzeuger/Verbraucher-Modell, wobei auch Multicast möglich ist. Der Prozessdatenaustausch erfolgt zyklisch, asynchroner Datenaustausch und die Nutzung von Alarmen sind ebenfalls möglich. Normale Daten der Prozessautomatisierung und nicht Echtzeit-Ethernet-Frames werden frei innerhalb eines Zyklus gesendet, gewöhnliche Ethernet Teilnehmer lassen sich also ohne weiteres benutzen. Zugriff auf das Internet ist aber nur über ein Gateway realisierbar. Ähnlich wie bei EtherNet/IP sind hier bei Überlast gleichermaßen Beeinträchtigungen des nicht Echtzeitverkehrs und im Extremfall sogar des Echtzeitbetriebs zu erwarten [36]. Bei Verwendung des IRT-Modus erfolgt eine Aufteilung des Zyklus in zwei Phasen. Innerhalb des IRT-Intervalls ist nur die Verarbeitung von IRT-Daten erlaubt[19], die Switche leiten dann ankommende Daten nach einen festen, bei der Projektierung vorgegebenen Zeitplan weiter. Durch die Zeitsychronisation und genaue Kenntnis der einzelnen Verzögerungen innerhalb der Switche lassen sich die einzelnen Zeitschlitze sehr exakt festlegen. PROFINET IO im IRT-Modus stellt also ein Zeitmultiplexverfahren dar. Dadurch, dass keinerlei Adressinformationen ausgewertet werden müssen, können die Switche laut Felser [12] innerhalb von nur

[19] Eventuell ankommende nicht Echtzeit-Daten werden gepuffert und später gesendet.

3 μs ihre Daten weiterleiten. Im zweiten, dem offenen Intervall arbeitet dieses Verfahren dann genau wie in der RT-Betriebsart. In der Abbildung 8 ist der Zyklus von PROFINET IO IRT abgebildet [48]. Minimale Zeiten der einzelnen Zeitschlitze sind laut Popp für den IRT-Kanal 31,25 μs, und 125 μs für den Slot, der die gewöhnliche Kommunikation beeinhaltet. Damit ist die Verwendung von gewöhnlichen Ethernetrahmen der maximalen Größe im nicht Echtzeitkanal möglich. Die Nutzung von Segmentierungs- und Reassemblierungstechniken, um kleinere Zeitschlitze zu erreichen, wird nicht unterstützt.

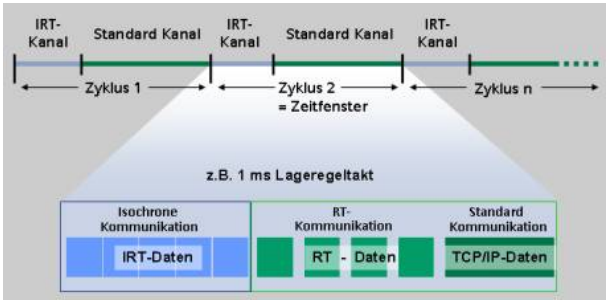

Abbildung 8: PROFINET IO IRT Zyklus [48]

Die Spezifikation dieses RTE-Ansatz ist seit Frühjahr 2001 veröffentlicht, die herstellerübergreifende Organisation PROFIBUS International [38] leistet Unterstützung. Eine Vielzahl von Herstellern hat sich ihr angeschlossen, als zwei große Firmen sind Siemens und Bosch-Rexroth zu nennen. PROFINET stellt ein umfassendes Konzept zur Integration von bestehenden Automatisierungslösungen zur Verfügung. Über Proxies lassen sich andere Feldbusse wie PROFIBUS in das Netzwerk integrieren. PROFINET-IO ist mittlerweile recht weit verbreitet, der schnellere IRT-Ansatz ist wohl laut [4] erst ca. ein halbes Jahr verfügbar. Durch die große Popularität von PROFIBUS, insbesondere im europäischen Raum, lässt sich aber auch eine gute Akzeptanz dieser Erweiterung erwarten. PROFINET ist in der IEC 61158 und IEC 61784 standardisiert. Für weitere Informationen sei auf das Buch von Manfred Popp [36] verwiesen.

3.2.3 Ethernet Powerlink

Ethernet Powerlink nutzt Ethernet-Frames mit einem eigenen Protokoll-Typ, es stellt also eine Lösung oberhalb des normalen Ethernets dar. Es bietet zwei Betriebsarten an, den „Open Mode“ und „Protected Mode“. Hier wird aber nur der ausgereifteren Protected Mode betrachtet, da er die bessere Leistung liefert und der Open Mode noch nicht verfügbar ist [55]. Dieser Ansatz nutzt ein Polling-Buszugriffsverfahren, wie von vielen Feldbussen eingesetzt. Dabei verwaltet ein Master jeweils einzelne Ethernet-Segmente, auf die nacheinander die Slaves zugreifen können. Alle Netzwerkteilnehmer nutzen dabei eine Standard-Netzwerkschnittstelle, aus Leistungsgründen sollen allerdings Hubs anstelle von Switches eingesetzt werden [45].

Ein Master teilt den Beginn eines Zyklus allen Teilnehmern über eine Broadcastnachricht mit. Diese Nachricht gilt auch als gemeinsame Zeitbasis, wobei die einzelnen Master zur ihrer Sychronisation das IEEE-1588 Protokoll verwenden. Dann fragt er nacheinander alle Endgeräte ab, die ihrerseits mittels Multicast-Nachrichten ihre Informationen an die Empfänger übermitteln. Diesen Zeitspanne nennt man auch isochrone Phase, sie ist für zeitkritische Daten vorgesehen. Innerhalb dieses Zeitraums kann ein Gerät zusätzlichen Sendebedarf anmelden, dieses darf dann nach Ende der Phase in der asychronen

Phase erneut kommunizieren. Innerhalb dieses letzten Slots kann es dann zusätzlich normale Ethernet-Frames und damit TCP/IP oder UDP für die Übermittlung von unkritischen Daten nutzen. Am Ende des Zyklus sorgt eine kurze Idle-Phase für einen definierten Abschluss [13], der gesamte Ablauf ist in Abbildung 9 noch einmal zusammengefasst [13].

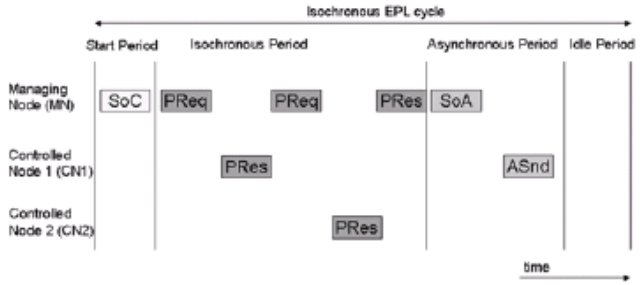

Abbildung 9: Ethernet Powerlink Zyklus [13]

Durch den Einsatz von Geräten mit integrierten Hub ist eine Linienstruktur realisierbar, andernfalls ist nur eine Stern- bzw. Baumanordnung möglich. Ethernet Powerlink ist im Protected Mode für hohe Echtzeitanforderungen geeignet, [45] nennt Jitterwerte von unter 400 ns und eine Zykluszeit von 1 ms für die Ansteuerung von 30 Achsen. Chaffee und Hirschinger nennen in [4] wiederum recht optimistische Zahlen, die dort aufgeführte Mindestzykluszeit von 200 μs erscheint angesichts des Zyklusaufbau aber realistisch. Durch die Sychronisation der einzelnen Master untereinander ist außerdem Echtzeitverkehr über Ethernet-Segmentgrenzen hinweg möglich. Die Verwendung von Standard Ethernet-Geräten ist innerhalb des Protected Modes nicht im Netzwerk möglich. Um auf Informationen von externen Netzen wie dem Internet zuzugreifen, muss auch hier ein Gateway vorgesehen werden. Es handelt sich also um ein geschlossenes Netz, im Gegensatz zu PROFINET oder auch dem REAL-Ansatz.

Ethernet Powerlink wurde von Bernecker und Rainer im Jahr 2001 vorgestellt, es wird unterstützt von der Ethernet Powerlink Standardization Group. Das verwendete Applikationsmodell ist CANopen, es lassen sich dadurch normale CANopen Geräte einfach innerhalb des Netzwerks nutzen. Laut [4] sind bis zum Mai 2006 100.000 Geräte ausgeliefert worden. Die offene Anwender- und Anbietergruppe Ethernet Powerlink Standardization Group arbeitet an der Spezifikation und Weiterentwicklung des Standards, er ist auch für Nichtmitglieder gegen eine Schutzgebühr erhältlich. Die auf der Webseite der EPSG [10] verfügbare Broschüre [11] liefert weiterführende Informationen.

3.2.4 EtherCAT

EtherCAT stellt einen recht radikalen Bruch mit den normalen Arbeitsprinzipien und dem Standard von Ethernet dar. Ein Master, für gewöhnlich das Kontrollsystem, erzeugt einen Ethernet-Rahmen, der nacheinander durch alle Slaves übertragen wird. Dabei koppeln die einzelnen Slaves die an sie addresierten Daten aus und können auch neue Informationen in den Rahmen einfügen. Dieser Vorgang kann durch die Verwendung von spezieller Hardware mit normaler Ethernet-Geschwindigkeit stattfinden, pro Slave tritt nur eine Verzögerung von ca. 60 ns auf. Dabei enthält der verwendete ASIC[20] zusätzlich einen integrierten Switch mit zwei Ports. Das letzte Gerät in der Kette sendet die Daten dann wieder an den

[20] Application-Specific Integrated Circuit

Master zurück, so dass sich aufgrund der Nutzung von Vollduplex-Verbindungen eine physikalische Ringtopologie ergibt. Die Abbildung 10 illustriert das Arbeitsprinzip und stellt dar, wie die einzelnen Netzwerkteilnehmer ihren Datenbereich des Ethernet-Rahmens nutzen [21].

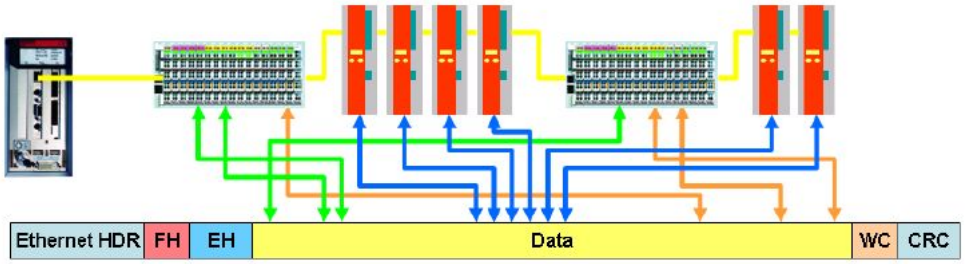

Abbildung 10: EtherCAT Arbeitsprinzip [21]

Das gesamte Netzwerk stellt sich dabei als ein einziges virtuelles[21] Ethernet-Device dar, es lässt sich als Stern, Baum oder Linie strukturieren [44]. Um eine Weiterleitung von Daten auch über Netzwerkgrenzen hinweg zu ermöglichen, unterstützt EtherCAT, wie auch andere Verfahren, die Kapselung der Daten in UPD-Paketen. Der Austausch von Nachrichten erfolgt über einen Mailbox-Mechanismus, gepufferter Prozessdatenaustausch wird über einen separaten Dienst realisiert.

Die Leistung dieses Echtzeit-Ethernet-Ansatzes ist sehr gut, Zykluszeiten von 30 μs lassen sich in einem Netz mit reinem Echtzeitverkehr erreichen. Bei Verwendung von Rahmen der maximalen Größe lässt sich immer noch eine Zykluszeit von 300 μs erreichen, dabei ist der Austausch von fast 12.000 digitalen Ein- und Ausgangswerten möglich [9]. Durch die Kapselung und Fragmentierung von nicht RT-Verkehr lässt sich auch im gemischten Betrieb eine Zykluszeit von 100 μs erreichen[22] [9]. Aufgrund der genutzen Ringstruktur kann der Master die Verzögerungen zu den jeweiligen Slaves exakt berechnen, zwischen verschiedenen Segmenten kann man dann beispielsweise ein Protokoll nach IEEE-1588 zur Synchronisation nutzen. Als mögliche Leistungsfähigkeit wird in [4] ein Jitter von unter 1 μs bei einer Zykluszeit von 100 μs und 100 belieferten Achsen genannt. Der Einsatz von gewöhnlichen Ethernetgeräten ist durch Kapselung dieser Daten im Echzeitprotokoll mit Hilfe eines Switches problemlos möglich, natürlich ist auch hier ein Gateway für den Übergang ins Internet erforderlich. EtherCAT verwendet wie auch Ethernet Powerlink ein abgeschlossenes Netz, dass allerdings sehr performant arbeitet.

Als Applikationsmodell nutzt dieser RTE-Ansatz CANopen, die Nutzung von SERCOS ist aber ebenso möglich. Es existieren offene Protokoll-Implementierungen für den Master, durch die Verwendung von spezieller Hardware ist die Realisierung eines Slaves nicht so einfach durchführbar. EtherCAT wurde 2003 von Beckhoff vorgestellt, wird von der EtherCAT Technology Group unterstützt und ist 2005 in den IEC Standard IEC 61784-2 [19] aufgenommen worden. EtherCAT ist ein ziemlich neuer Ansatz, [4] berichtet von 3.500 gelieferten Geräten bis zum Mai 2006, die ersten Slave-ASICs sind seit ungefähr einem Jahr verfügbar. Zusätzliche Informationen liefert beispielsweise die Broschüre [9] der EtherCAT Technology Group, weitere Dokumente lassen sich auch von der EtherCAT Webseite [8] beziehen.

[21] Virtuell, da es ja in Wirklichkeit aus vielen einzelnen EtherCAT Geräten besteht.
[22] Natürlich nicht mit Rahmen der maximalen Größe.

3.2.5 REAL

Der Name „REAL“ ist der Arbeitstitel des von Dopatka in [5, 6] vorgeschlagenen neuen Echtzeit-Ethernet Verfahren, das in dieser Arbeit simuliert werden soll. Auch dieses System basiert auf einem Zeitmultiplex-Ansatz. Bei den vorgestellten Verfahren Ethernet Powerlink und EtherCAT ist aufgrund der Funktionsweise zu jedem Zeitpunkt nur eine Kommunikation möglich. PROFINET erlaubt demgegenüber schon das gleichzeitige Versenden von Nachrichten, sofern sie unabhängig voneinander sind.

Der REAL-Ansatz versucht nun, optimale zeitliche Ablaufpläne für die Kommunikation zu finden, die eine möglichst große Parallelität erlauben. Dazu werden zunächst sogenannte „Kommunikationslinien“ generiert, die die Verkehrsbeziehungen der einzelnen Knoten beschreiben. Aus diesen lässt sich dann ein Konflikt-Graph berechnen, für den dann nach einem speziellen Algorithmus eine möglichst gute Lösung[23] gefunden wird. Verkehr mit nicht Echtzeit-Anforderungen kann in freien Zeitschlitzen befördert werden. Aufgrund der optimierten Berechnung der Schedules und der besseren Ausnutzung von leeren Zeitschlitzen ist eine bessere Leistung zu erwarten.

Für genauere Informationen zu diesem Ansatz sei auf die eingangs genannten Quellen und die Dokumentation der Projektgruppe [39] zu diesem Thema verwiesen. Genauere Informationen zur konkreten Implementierung des REAL-Systems innerhalb des Simulationsmodells finden sich in Kapitel 5.1.4.

3.2.6 RTnet

Der RTnet-Ansatz ist insbesondere aufgrund der Herstellerunabhängigkeit durch die Bereitstellung der vollständigen Implementation als Opensource interessant. Bei freier Software ist die Offenheit des Systems immer gewährleistet, da der gesammte Quellcode frei verfügbar ist. RTnet ist ein einfaches TDMA-Verfahren, das mit Standard Ethernet Hardware arbeitet. Der existierende, leistungfähige Linux-Netzwerkstack wird zur Bereitstellung von Echtzeitdiensten modifiziert, die Betriebssystembasis (RTAI Linux) ist dabei auch mit Echtzeiterweiterungen ausgestattet. Es wurde Wert auf einen modularen Aufbau gelegt, um das gesamte System leicht anpassen und erweitern zu können [23]. Die Struktur ist in Abbildung 11 dargestellt [22].

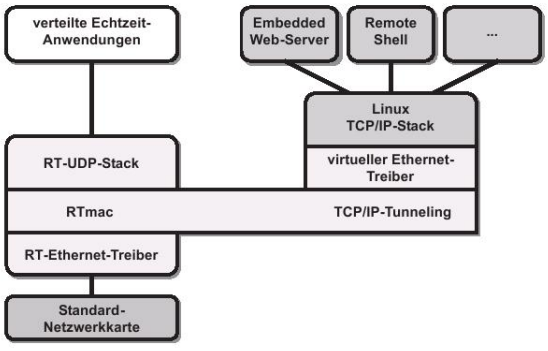

Abbildung 11: RTnet Struktur [22]

RTnet unterstützt die Verwendung von gewöhnlichen Ethernetprodukten und der Standardprotokolle IP, ICMP und UDP, der Einsatz von TCP ist nicht vorgesehen. Es nutzt einen Master/Slave-Ansatz, um

[23] Lösung im Sinne eines optimalen Ablaufplans für den Verkehr.

Zeitmultiplex zu ermöglichen. Ein Master gibt den zentralen Takt vor, damit lassen sich die Uhren aller Teilnehmer synchronisieren. Die Slaves senden dann innerhalb des zugeteilten Slots ihre Daten. Zur Verbindung der einzelnen Teilnehmer werden Hubs verwendet. Bei den Netzwerkteilnehmern generierter nicht Echtzeit-Verkehr kann nach einer Kapselung ebenfalls versendet werden. Genaue Leistungswerte lassen sich nicht nennen, da sie sehr stark von der verwendeten Hardware abhängen. Kiszka zeigt aber, dass sich eine Zykluszeit von 2,5 ms bei einem Jitter von unter 40 μs erreichen lässt [22].

Die Universität Hannover übernahm 2001 Grundlagen für das Projekt von Lineo und beschloss es selbst weiterzuentwickeln. RTnet bietet noch kein eigenes Applikationsmodell oder die Unterstützung eines verbreiteten Protokolls wie CANopen an. Aktuell ist die Programmierung nur mittels einem erweiterten POSIX API und einigen anderen Ansätzen möglich [23]. Weitere Informationen finden sich auf der Homepage des Projekts [41].

3.3 Kriterienkatalog für Bewertung der Systeme

Nach der Darstellung der einzelnen Systeme erfolgt jetzt die Formulierung eines Kriterienkataloges für die Bewertung der einzelnen Echtzeit-Ethernet-Verfahren. Dafür wird die Leistungsfähigkeit und die Kompatibilität der einzelnen RTE-Anwendungen untersucht.

3.3.1 Anforderungen an die Leistungsfähigkeit

Das wichtigeste Kritierium im Rahmen dieser Arbeit ist die Eignung für RT-Anwendungen, also der Grad der Echtzeitfähigkeit. Die Norm IEC-61784-2 [19] schlägt nach Felser einheitliche Leistungsklassen für Echtzeit-Ethernet-Lösungen vor [13]. Diese werden detailliert in Tabelle 4 dargestellt. Auch die Bewertung der vorgestellten RTE-Ansätze verwendet diese Klassifizierung.

Leistungsklasse	Anmerkung
Auslieferungszeit	Transport m/o Fehler von Quelle zu Ziel auf Anwendungsebene
Anzahl Endknoten	Maximale Anzahl der RTE-Geräten als Endknoten
Netzwerktopologie	Beliebige Kombination von Stern, Baum, Ring oder Linie
Anzahl Switche zwischen Endknoten	Switche bedingen immmer Latenz und Jitter
Durchsatz im RTE-Betrieb	Anzahl der Bytes auf Anwendungsebene pro Link und Sekunde
Bandbreite im nicht-RTE-Betrieb	Prozent der Bandbreite für nicht Echtzeitverkehr
Genauigkeit der Zeitsynchronisation	Maximale Abweichung zwischen zwei Knoten
Erholungszeit im Redundanzfall	Erholungszeit bei einem permanten Fehler

Tabelle 4: Leistungsklassen nach IEC-61784-2

Die direkten Leistungsparameter hängen bei den vorgestellten Netzwerken teilweise von deren Topologie, der Anzahl angeschlossener Teilnehmer und anderen Einflussfaktoren ab. Dieser Teil der Arbeit beruht zu einem großen Teil auf Daten der Hersteller. So konnten keine Tests durchgeführt werden, bei denen man vergleichbare Bedingungen schaffen könnte. Deshalb wird in diesen Fällen ein beispielhafter Wert für eine typische Konfiguration des Systems angegeben.

3.3.2 Einbindung in bestehende Systeme

Für einen sinnvollen Einsatz in der Industrie ist aber noch die Betrachtung anderer Gesichtspunkte wichtig. Da komplette Neuinstallationen selten auftreten, muss man neue Lösungen gut in bestehende Umgebungen eingliedern können. Die Kompatibilität zu anderen Systemen stellt daher ein wichtiges Kriterium dar. Hierbei werden sowohl die Möglichkeiten der Einbindung von herkömmlichen Feldbussystemen, als auch anderer RTE-Ansätze untersucht. Da ein explizites Designkriterium des REAL-Systems die Zusammenarbeit mit normalen Ethernetsystemen darstellt, erfolgte eine zusätzliche Bewertung der Kompatibilität in diesem Bereich.

Für die Evaluation mit dem Ziel eines konkreten Einsatzes stellt die Verbreitung eines Systems eine wichtige Eigenschaft dar. Von ihr hängt unter anderem der Umfang der Erfahrungen im Markt, die Verfügbarkeit von Fachpersonal und damit ebenso teilweise das Risiko eines Einsatzes ab. Für die Vermeidung einer zu starken Abhängigkeit von einem Lieferanten ist die Offenheit und der Grad der Standardisierung einzelner Ansätze ein wichtiges Merkmal. Diese kann sich beispielsweise positiv auf Parameter wie Preisgestaltung und auch Verfügbarkeit von erfahrenen Personal auswirken. Als letztes, die Zusammenarbeit mit anderen Feldbussen betreffende Merkmal wird das verwendete Applikationsprotokoll betrachtet.

Tabelle 5 fasst die erstellten Kriterien für die Einbindung in bestehende Systeme noch einmal zusammen. Im Gegensatz zu den genannten Leistungskriterien lassen sich einige dieser Anforderungen schlecht in absoluten Zahlenwerten formulieren. In diesem Fällen wird eine textuelle Beschreibung vorgenommen.

Kriterium	Anmerkung
Kompatibilität Feldbusse	zu anderen RTE-Systemen und Feldbussen
Kompatibilität Ethernet	zu normalen Ethernet-Netzwerkteilnehmern
Verbreitung	Anzahl der Installationen im Feld
Offenheit	des Standards und des Systems
Standardisierung	Standardisierungsprozesse, Normen
Applikationsmodell	Verwendete Applikationsprotokoll

Tabelle 5: Kriterien für die Einbindung in bestehende Echtzeit-Ethernet-Systeme

Weitere für die Evaluation von RTE-Systemen wichtige Parameter wie die Verfügbarkeit von Werkzeugen für Entwicklung, Projektierung, Fehlerdiagnose und Änderungen werden im Rahmen dieser Arbeit nicht betrachtet. Es erfolgt ebenfalls keine Bewertung von wirtschaftlichen Aspekten wie der Preis und die langfristige Verfügbarkeit der Komponenten.

3.4 Bewertung

Nachdem nun die gewählten Kriterien dargestellt wurden, lassen sich die einzelnen Verfahren genauer einordnen. Da eigene Tests im Rahmen dieser Arbeit nicht durchführbar waren, wurden Herstellerangaben bzw. andere Quellen für die Bewertung der einzelnen Eigenschaften verwendet. Teilweise ergeben sich dadurch wiedersprüchliche Angaben, allgemein sind die verfügbaren Daten zu der Leistung der einzelnen Ansätze recht lückenhaft. Die Hersteller beziehen sich in ihrer Literatur meist auf einen, optimalen Einsatzfall, der nicht ohne weiteres mit anderen Betriebsarten vergleichbar ist.

Aufgrund dieser Schwierigkeiten, brauchbare und vergleichbare Leistungsdaten zu bekommen, müssen die Bewertungskriterien modifiziert werden. So wurde der Parameter „Auslieferungszeit“ zugunsten eines allgemeinen Leistungsbegriffes ersetzt. Auch eine Angabe der Erholungszeit ist aufgrund der verwendeten Datenbasis nicht möglich, so ist beispielsweise für PROFINET die Redundanz laut [36] noch nicht spezifiziert. Auf die Angabe einer maximalen Netzwerkgröße wurde verzichtet, da in der Praxis die theoretisch mögliche Anzahl nie erreicht wird, da die Netzwerklatenzen zu groß werden. PROFINET geht hier von einer maximalen Anzahl von 20 Geräten für eine Latenz von 1 μs aus. Deshalb erfolgt hier nur die Angabe der Verzögerungen pro Knoten bzw. Switch.

Die Tabelle 6 stellt nun die einzelnen RTE-Verfahren im Vergleich dar, die Daten stammen aus den schon mehrfach zitierten Quellen [1, 44, 45, 13]. Für genauere Informationen sei auf die jeweiligen Detailbetrachtungen im vorherigen Kapitel verwiesen.

Verfahren/ Kriterium	Leistung - Zykluszeit	Topologie	Latenz Switch/Knoten Linie	Jitter
EtherNet/IP	100 Achen - 1 ms	Stern, Baum, Linie	10 μs - 40 μs	< 0,5 μs
ProfiNET IO IRT	150 Achsen - 1 ms	Baum, Stern, Linie, Ring	3 μs/40 ns	< 1 μs
Ethernet PowerLink	100 Achsen - 1 ms	Stern, Baum, Linie	0,5 μs	< 1 μs
EtherCAT	100 Achsen - 0,1 ms	Stern, Baum, Linie	/60 ns	< 1 μs
REAL	85 Achsen - 1 ms	Stern, Baum, Linie	3 μs/	-
RTnet	- 2,5 ms	Stern, Baum	10 μs - 40 μs	< 40 μs

Tabelle 6: Leistungsvergleich Echtzeit-Ethernet

Für die Leistungsbewertung des REAL-Ansatzes wurde Szenario „realer Betrieb“ in Kapitel 6.2.2 zu Grunde gelegt. Parameter wie die Latenz der Switche und der Jitter werden in Kapitel 6.1 und 6.1.1 genauer erläutert. Tabelle 7 stellt nun die für die Kompatibilität der einzelnen RTE-Verfahren wichtigen Eigenschaften vor.

Verfahren/ Kriterium	Feldbusse	Ethernet	Verbreitung	Offenheit	Standardisierung	Applikationsmodell
EtherNet/IP	ControlNet, DeviceNet	ja	> 1 Million	ja	ja	CIP, CIP Motion Drive
ProfiNET IO IRT	PROFIBUS, Interbus	ja, Geräte	?	Mitglieder	ja	PROFIsafety, PROFIdrive
Ethernet PowerLink	CANopen	nein	> 100.000	ja	in Vorbereitung	CANopen
EtherCAT	CANopen, Sercos	ja, Geräte	> 3500	Mitglieder	ja	CANopen, Sercos
REAL	-	ja, Geräte	-	-	-	-
RTnet	-	nein	?	ja, Opensource	nein	-

Tabelle 7: Kompatiblitätsvergleich Echtzeit-Ethernet

Da es sich bei der REAL-Lösung erst um einen Entwurf handelt, lassen sich hier natürlich nur lückenhafte Aussagen treffen. Bei PROFINET IO, EtherCAT und REAL[24] können gewöhnliche Ethernet-Geräte an die Switche angeschlossen werden. EtherNet/IP unterstützt gewöhnliche Geräte ohne spezielle Switche, und Ethernet PowerLink[25] bietet keinerlei Kompatibilität zu Standard-Ethernet. Für PROFINET IO und RTnet konnten keine genaueren Verbreitungszahlen gefunden werden.

[24] In der hier implementierten Variante mit speziellen Switchen.
[25] In der hier untersuchten Betriebsart „Protected Mode“.

3.5 Zusammenfassung

Die betrachteten Lösungen stellen jeweils sehr unterschiedliche Ansätze für das Problem „Echtzeit-Ethernet“ dar. Bei der Auswahl eines konkreten Verfahrens spielen die Anforderungen und die schon vorhandene Infrastruktur eine große Rolle. Deshalb wird auf die Benennung eines eindeutigen „Gewinners“ dieses Vergleichs verzichtet. Mittlerweile existieren eine Vielzahl verschiedener RTE-Lösungen, eine im Internet verfügbare Auflistung nennt alleine 19 [24]. Im Folgenden werden die einzelnen Ansätze nochmal kurz zusammengefasst.

PROFINET stellt gewisserweise die traditionelle Herangehensweise dar. Hier wird das Problem mit erheblichen Technologieeinsatz gelöst. Innerhalb des Zeitmultiplexes sind die Switches für die Einhaltung des Ablaufplanes zuständig, die einzelnen Slaves führen nur Kommandos aus. Darüber hinaus stellt es eine umfassende Umgebung für den gesamten Systemlebenslauf zur Verfügung. Dem gegenüber lagert EtherNet/IP mit CIPsync Entscheidungen in die Netzwerkteilnehmer aus. Auf eine exakte Synchronisation des Netzwerkverkehrs wird verzichtet, durch den Einsatz von Zeitstempeln können die Slaves bei Kenntnis der genauen Zeit selbst eigenständig handeln. Durch den Verzicht auf komplexe TDMA-Verfahren konnte man die Entwicklungszeit kurz halten, auch dadurch ist diese Lösung ist mit Abstand am weitesten in der Praxis verbreitet.

Ethernet Powerlink nutzt ein traditionelles Pollingverfahren, wie es bei Feldbussen weit verbreitet ist. Es ist komplexer als EtherNet/IP, aber benötigt nicht soviel Aufwand wie PROFINET. Dies spiegeln auch die Verbreitungszahlen wieder. EtherCAT ist der schnellste von den hier betrachteten Ansätzen. Durch eine geschickt ausgelegte Systemarchitektur kann eine sehr gute Leistung und Netzauslastung erreicht werden, man muss allerdings bei der Kompatibilität zu normalen Ethernet Abstriche machen.

Auf eine Bewertung von RTnet wird hier verzichtet, da zu geringe Informationen über den Einsatz in der Praxis vorhanden sind. Die Simulationsergebnisse zur Bewertung des REAL-Verfahrens werden in Kapitel 6.3 ausführlich diskutiert.

4 Diskrete Ereignissimulatoren

Bevor auf die auf die Auswahl eines Simulationssystems eingegangen wird, werden noch einige grundlegende Begriffe eingeführt. Zunächst ist die Fragestellung zu betrachten, welche Vorteile, aber auch Einschränkungen durch eine Modellbildung zu erwarten sind. Dann erfolgt eine Darstellung der Grundlagen der diskreten Ereignissimulation und des allgemeinen Vorgehens bei der Durchführung einer Simulation. Weiterhin werden die Anforderungen an ein Simulationssystem für den Einsatz innerhalb dieser Arbeit und die Begründung der Entscheidung für das Simulationssystem OMNeT++ dargelegt. Zum Schluss dieses Kapitels wird noch kurz die Funktionsweise und Bedienoberfläche dieses Simulators vorgestellt.

4.1 Modellbildung

Um Erkenntnisse über komplexe Systeme zu erlangen, ist es oft sinnvoll, ein Modell zu erstellen, dass die wesentlichen Eigenschaften eines Systems abbildet und die Untersuchung dann daran vorzunehmen. Die Beschränkung auf einige problemrelevante Eigenschaften vermindert die Komplexität und macht die Untersuchung häufig erst realisierbar [34]. Aufgrund der Ergebnisse der Simulation lassen sich dann in gewissen Grenzen Rückschlüsse auf das Verhalten des Originals ziehen. Im Rahmen der Arbeit ist die Untersuchung am realen System nicht möglich, da keine Prototypen vorhanden sind. Es handelt sich erst um einen Entwurf für ein Protokoll. Man spricht in diesem Fall auch von einem „Gestaltungsmodell“. Die Simulation soll in hier Erkenntnisse für den weiteren Entwurfsprozess bringen, also als Entscheidungshilfe dienen.

Allerdings muss man aufgrund der während der Modellbildung vorgenommenen Abstraktion und Idealisierung eine größere Ungenauigkeit in Kauf nehmen. Wenn zu viele Details der Lösung weggelassen werden, können unter Umständen größere Fehler entstehen. Nicht nur aus diesem Grund müssen die Ergebnisse einer Simulation immer interpretiert und auf ihre Stichhaltigkeit und Korrektheit überprüft werden. Weitere Informationen zu Möglichkeiten und Grenzen der Modellbildung und Simulation findet sich beispielsweise in [34]. Welche Eigenschaften des Originals wesentlich für die Simulation sind, hängt von der Fragestellung und Zielsetzung des Modells ab, deren detaillierte Vorstellung in Kapitel 5.1 erfolgt. Einschränkungen des Simulators, vorgenommene Abstraktionen und Idealisierungen werden in Abschnitt 5.3.1 diskutiert. In Kapitel 4.3 wird das Vorgehensmodell zur Erstellung eines Modells ausführlich eingeführt.

4.2 Simulationsmodelle

Simulationsmodelle lassen sich auf unterschiedliche Art und Weise charakterisieren. Neben der Klassifizierung nach der Untersuchungsmethode oder anhand des Abbildungsmedium erscheint eine Einteilung nach Art der Zustandsübergänge am sinnvollsten.

Für das Anwendungsfeld der Netzwerksimulation sind zeitdiskrete Simulationen verbreitet, da sie für diesen Einsatzzweck gut geeignet sind [34, 52]. Bei der diskreten Simulation werden, im Gegensatz zur kontinuierlichen Simulation, nur endlich viele Zustandsänderungen pro Zeitintervall betrachtet. Es lässt sich also durch Verwendung einer zeitdiskreten Simulationsumgebung eine erheblich bessere Leistung

erzielen, da allgemein eine geringere Anzahl von Ereignissen betrachtet werden muss. Innerhalb des Feldes der diskreten Simulation haben sich verschiedene Sichtweisen herausgebildet, wovon aber nur der ereignisorientierte Ansatz für diese Arbeit interessant ist. Diskrete Ereignis-Simulatoren sind zunächst Systeme, bei denen Zustandsveränderungen nur an diskret auf der Zeitachse verteilten Punkten auftreten können. Diese Zustandsveränderungen, also Ereignisse, benötigen dabei für sich genommen keine Zeit[26]. Weiter wird angenommen, dass nichts interessantes zwischen zwei aufeinanderfolgenden Ereignissen passiert [52], bzw. dass der exakte Ablauf dieser Tätigkeiten vernachlässigbar ist. Die interne Uhr des Modells wird also nach Verarbeitung eines Ereignisses jeweils auf den nächsten Ereigniszeitpunkt vorgerückt. Für eine ausführlichere Darstellung der unterschiedlichen Ansätze sei nochmals auf Page [34] verwiesen.

Ereignisorientierte Simulationsmodelle bestehen im allgemeinen aus einem Zustandsmodell, das die statischen Komponenten beeinhaltet und einer Ereignisliste, die die nächsten abzuarbeitenden Ereignisse enthält. Durch die Spezifikation von Ereignisroutinen bestimmt man das Verhalten des Modells. Diese können unter anderen Attribute oder Aufbau des Zustandsmodells ändern und die Ereignisliste modifizieren. Die Verwendung dieses Modellansatzes ermöglicht eine klare Trennung zwischen statischer Struktur und dynamischen Verhalten eines zu simulierenden Systems [34]. Verschiedene weitere Hilfsfunktionen sorgen für zusätzliche Ablaufsteuerung, so startet eine Initialisierungsfunktion die Simulation, ihre Ergebnisse werden zum Schluss von einer Auswertungsfunktion erfasst. Im Kapitel 4.5 erfolgt eine Darstellung der gerade vorgestellten Komponenten nochmals anhand des OMNeT++ Simulationssystems.

Durch Angabe der internen Zeit und des aktuellen Zustandes der aufgezählten Komponenten lässt sich ein ereignisorientierten Simulationsmodells eindeutig beschreiben.

4.3 Vorgehen bei der Erstellung des Modells

Wie schon diskutiert, erfolgt im Rahmen der Diplomarbeit die Erstellung eines Gestaltungsmodells. Bei Entwicklungsarbeiten, speziell der Softwareentwicklung ist es sinnvoll, sich an einem Vorgehensmodells zu orientieren. Dadurch reduziert sich die Komplexität der Aufgabe, da sie in handlichere Teilbereiche zerlegt wird, und man erhält einen Leitfaden, den man zum Erreichen des Ergebnisses einfach folgen kann.

Vorgehensmodelle zum Simulationsentwurf beziehen sich in der Regel auf die Modellierung anhand eines Originalssystems, die meisten Überlegungen lassen sich aber auch sinngemäß auf die Simulation eines erdachten Systems übertragen. Page schlägt einen detaillierten Modellbildungszyklus vor [34], der in Abbildung 12 vereinfacht dargestellt wird.

Vorgehensmodelle sollten, um sie sinnvoll einzusetzen zu können, an die jeweiligen Bedürfnisse angepasst werden[27]. Deshalb erfolgt in den weiteren Teil dieses Abschnittes nun eine Zuordnung der einzelnen Modellphasen an die konkreten Erfordernisse der Aufgabenstellung, gegebenenfalls erfolgt ein Verweis auf die entsprechenden Kapitel.

Aus dem Protokollentwurf bzw. der Spezifikation der Simulation der Projektgruppe lässt sich die **Problemdefinition** ableiten. Sie legt den durch die Simulation zu untersuchenden Aufgabenbereich und die

[26] Mit „Zeit“ ist hier die Modellzeit gemeint. Selbstverständlich erfordert die Verarbeitung einzelner Ereignisse innerhalb des Prozessors real eine bestimmte Zeit.

[27] Dieses Prozess nennt man auch „Tailoring“ und ist insbesondere bei umfangreichen Vorgehensmodellen notwendig.

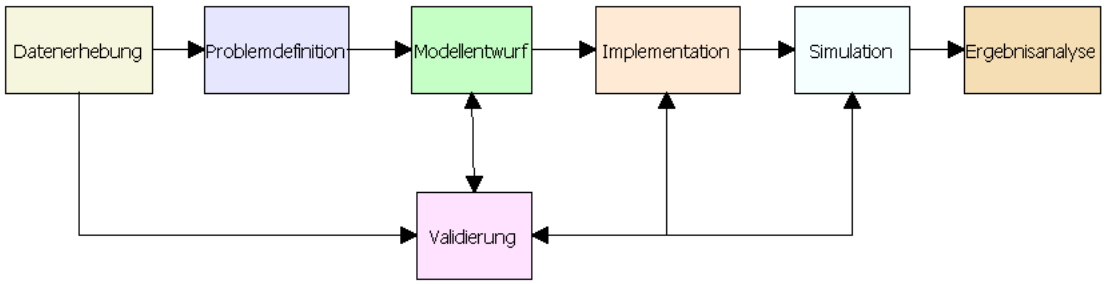

Abbildung 12: Modellbildungszyklus nach Page

zu erreichenden Ziele fest. Dieser Teil der Arbeit befindet sich in Kapitel 5.1. Nun wird für den **Modellentwurf** zunächst ein erstes konzeptionelles Modell erstellt. Innerhalb dieser Phase erfolgt durch Abstraktion, Idealisierung und Modularisierung der Entwurf eines informalen, beschreibenes Abbild der zu erstellenden Simulation. Die Darstellung von wichtigen Details erfolgt zusätzlich in einer formalen Spezifikation. Abschnitt 5.2.5 stellt diese Schritte ausführlich dar, hier wird ebenso eine Aufwandschätzung der Realisierung vorgenommen. Die notwendige **Datenerhebung** im Grundlagenteil der Arbeit (Kapitel 2) bietet die Basis für den Modellentwurf und dessen **Validierung**. Die **Modellimplementierung** setzt dann den Entwurf in eine ausführbare Simulation um, siehe auch Kapitel 5.2.6. Hier muss wieder gleichermaßen ein Abgleich mit den Erwartungen stattfinden, um sicherzustellen, dass sich die Simulation korrekt verhält. Mit dem funktionsfähigen und validierten Modell kann man nun Experimente durchführen. Die **Simulation** von ausgewählten Szenarien wird im Abschnitt 6.2 dargestellt. Die Aufbereitung und Bewertung der Resultate im Rahmen der **Ergebnisanalyse** bildet das Fazit der Arbeit.

4.4 Auswahl eines Simulationssystems

Nachdem nun ausführlich auf die theoretischen Hintergründe der diskreten Ereignissimulation eingegangen wurde, erfolgt nun eine Darlegung der Gründe, die zur Wahl von OMNeT++ führen.

4.4.1 Anforderungen an ein Simulationssystems

Page definiert die Aufgaben eines Simulationssystems folgendermaßen:

> „Ein Simulationssystem ist ein Softwaresystem, das die Bearbeitung der drei Aufgabenbereiche *Modellbildung*, *Durchführung von Simulationsexperimenten* und *Ergebnisanalyse* im Rahmen einer Simulationsstudie unterstützt.“ [34]

Unter Zuhilfenahme dieser Einteilung lassen sich die Anforderungen an ein System nach diesen drei Bereichen nun genauer spezifizieren. Hierbei erfolgte eine Priorisierung anhand der Zwecke dieser Arbeit. Eine weitere Diskussion von möglichen Anforderungen findet sich in der oben genannten Quelle.

Zunächst ist eine einfache Eingabe und Veränderung von Modellen sehr wichtig, um nach kurzer Einarbeitungszeit sofort Simulationen erstellen zu können. Weiterhin müssen eventuelle weitere Eingabeparameter sich leicht, ohne Quellcodeänderung, modifizieren lassen. Dies erleichtert und beschleunigt die Modellierung erheblich, da sich Veränderungen schnell und unkompliziert vornehmen lassen. Letztlich

soll, nach Durchführung der Simulation, eine ansprechende Präsentation der Ergebnisse durchgeführt werden können. Aufgrund des begrenzten Zeitrahmens einer Diplomarbeit ist die Entwicklung von eigenen Visualisierungswerkzeugen nicht sinnvoll realisierbar; es muss also auf vorhandene Werkzeuge zurückgegriffen werden.

Zusätzlich zu den genannten Kritierien lassen sich weitere Anforderungen formulieren, die sich auf die Gesamtheit eines Simulationssystems beziehen. Erleichternd für eine Arbeit im akademischen Umfeld ist die Verwendung von freier Software, also Software, die im Quellcode vorliegt und für die keine teuren Lizenzen notwendig sind. Ferner ist es für die Anerkennung und eventuelle Nachvollziehbarkeit von Ergebnissen gut, wenn ein Simulator gängig und verbreitet ist. Dies hat zusätzlich den Vorteil, dass schon umfangreiche Erfahrungen vorliegen, dies verringert das Risiko gegenüber dem Einsatz eines Simulationssystems, das noch nicht im Netzwerkbereich eingesetzt wurde. Attribute wie leichte Erweiterbarkeit, Portabilität und eine aktive Entwicklergemeinschaft ermöglichen eine schnelle Einarbeitung und versprechen eine gute Ausgangsbasis für eigene Veränderungen. Des Weiteren wurde Wert auf eine gute Unterstützung der Internet-Netzwerkprotokolle, speziell von Ethernet, gelegt. Letztlich spielen auch subjektive Gründe wie die Gestaltung der graphischen Benutzeroberfläche und die verwendete Programmiersprache natürlich eine Rolle.

4.4.2 Gründe für OMNeT++

Nach Recherchen anhand der genannten Kriterien wurden zunächst zwei Simulationssysteme ausgewählt: ns-2 und OMNeT++. Nach Durchführung einer Testinstallation von beiden Simulatoren folgte eine Evaluation mit kleineren erstellten Netzwerken.

Das Simulationsframework ns-2 ist möglicherweise der bekannteste und am meisten verbreitete Netzwerksimulator [28]. Es unterstützt eine Vielzahl von Netzwerkprotokollen, und ist aufgrund seines Alters gut getestet. Dadurch ist es aber nicht so einfach erweiterbar, auch ist die Einarbeitungszeit relativ groß. Aufgrund des Alters entsprechen speziell die graphische Oberfläche und die Visualisierungswerkzeuge nicht mehr dem Stand der Technik. Modelle werden in einer Skriptsprache mit dem Namen „TCL“ erstellt. Es läuft ohne Probleme unter Linux/ Unix, es lässt sich aber mit Hilfe von Zusatzsoftware ebenfalls unter Windows installieren.

OMNeT++ ist ein neueres, objektorientiertes Simulationssystem. Es unterstützt nicht die Vielzahl von Netzwerkprotokollen wie ns-2, durch die Verwendung der INET-Erweiterung lässt sich aber eine gute Unterstützung der Internetprotokolle und von Ethernet sicherstellen. Die graphische Benutzeroberfläche und die Möglichkeiten zur Ergebnisanalyse bieten erheblich mehr Komfort und die Dokumentation ist leichter zugänglich. Modelle werden in einer einfachen, „NED“ genannten Sprache erstellt. Die primär unterstützten Betriebsysteme sind Linux- oder Unix-Varianten, es lässt sich aber auch recht einfach unter Windows nutzen.

Eine weitere Darstellung der Eigenschaften von OMNeT++ und ns-2 und ein Vergleich mit anderen Netzwerksimulatoren findet sich in der Masterarbeit von Luna-Vazquez [28].

Anhand der genannten Gründe erschien OMNeT++ das geeignete Werkzeug, die Simulation im Rahmen dieser Arbeit durchzuführen. Weitere Referenzen wie die Simulation eines PROFINET-Netzwerkes

mit OMNeT++ durch Jasperneite [20] und die genannte Masterarbeit unterstützten diese Entscheidung zusätzlich.

4.5 Darstellung von OMNeT++

Nach der Darstellung der Motive, die zur Wahl von OMNeT++ führten, folgt nun die Vorstellung der konkreten Repräsentation der einzelnen Simulationselemente im OMNeT++ System.

In Abbildung 55 sieht man das Hauptfenster von OMNeT++ unmittelbar nach dem Laden eines Netzwerkes. Direkt unter der Menüleiste im oberen Teil des Fensters befinden sich die Bedienelemente zur Ablaufsteuerung der Simulation. Hierüber erfolgt der Start und auch gegebenenfalls der Abbruch eines Simulationslaufs. Darunter erfolgt die Ausgabe der aktuellen Simulationszeit und Ereignisanzahl und weitere dynamischen Zustandparameter. Zwischen diesem Angaben und den grauen und gelben Bereich befindet sich eine Zeitleiste, die die zukünftigen Ereignisse nochmals graphisch wiedergibt. Der große, graue Bereich rechts unten dient zur Präsentation von Statusmeldungen der Simulationselemente, im gelben Bereich lässt sich das statische Simulationsmodell und die anstehenden Ereignisse zusätzlich in einer Listendarstellung visualisieren.

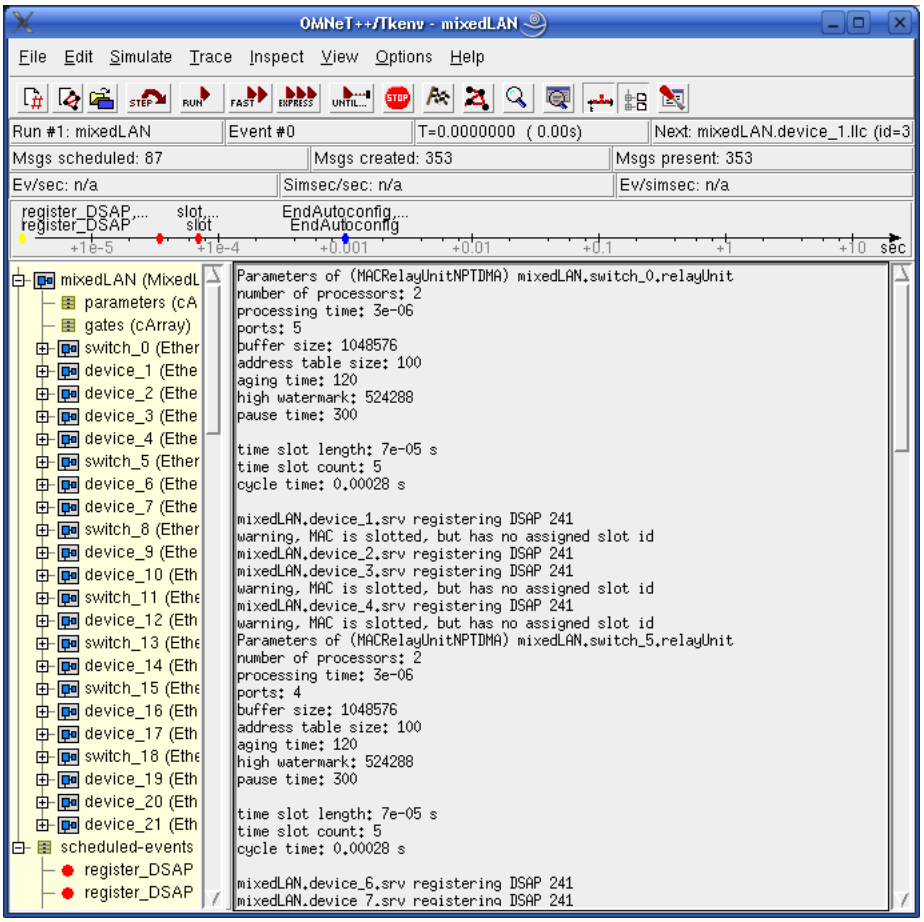

Abbildung 13: Hauptfenster OMNeT++

Abbildung 14 stellt nun die Visualisierung der statischen Struktur des Netzwerkes dar. Auf ihr lässt sich außerdem der Ablauf der Simulation verfolgen. Die Hervorhebung jeweils aktiver Elemente erfolgt durch

einen roten Rahmen. Verbindungen, über die gerade kommuniziert wird, sind gelb eingefärbt. In dem gezeigten Bildschirmfenster ist ein kleines Testnetzwerk dargestellt, die PCs symbolisieren Echtzeitgeräte, die beiden Laptops Standard Ethernet Teilnehmer. Die Kommunikation der Geräte erfolgt über einzelne Switches, die mittels bidirektionaler Verbindungen miteinander verbunden sind. Über diese Verbindungen versandte Ethernetframes werden durch sich bewegende graphische Elemente repräsentiert, die aber hier nicht dargestellt sind. Zusätzlich zu den graphischen Rückmeldungen erfolgt eine ausführliche Protokollierung der Ereignisse und Statusänderungen des Modells im Hauptfenster.

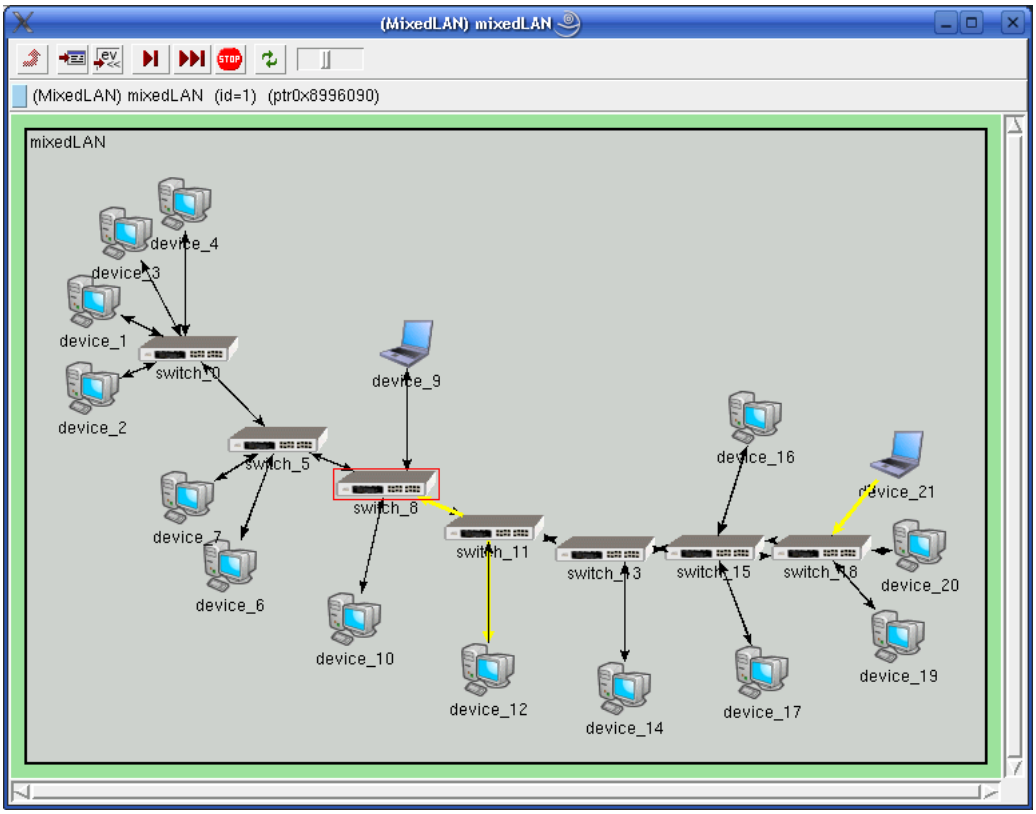

Abbildung 14: Netzwerkvisualisierung OMNeT++

In der Übersichtsdarstellung in Abbildung 14 lassen sich Netzwerkteilnehmer wie Switche oder Echtzeitgeräte detailliert betrachten. Hier kann man ausführlich den genauen Weg einzelner Frames nachvollziehen, der Status der Netzwerkschnittstellen und wichtiger Parameter wird ausgegeben.

Genauere Informationen über den Zustand von einzelnen, das Modellverhalten bestimmende Variablen lässt sich nach Anwählen der in dieser Ansicht visualisierten Elemente darstellen. In Abbildung 15 sei dies am Beispiel eines Switches und in Abbildung 16 für einen Teilnehmer des Echtzeitnetzwerks gezeigt. Hier kann die Arbeitsweise einzelner Komponenten detailliert beobachtet werden. Weiterhin ist es ist auch möglich, sich die Werte einzelner Variablen ausgegeben zu lassen.

Für eine weitere Wiedergabe und Erläuterung der graphischen Benutzerschnittstelle und der Bedienung des OMNeT++ Simulationssystems sei auf den Anhang A.3 verwiesen. Beispiele für die zur Modellierung verwendete Sprache und auch die Darstellung der Auswertungswerkzeuge finden sich in den folgenden Kapiteln.

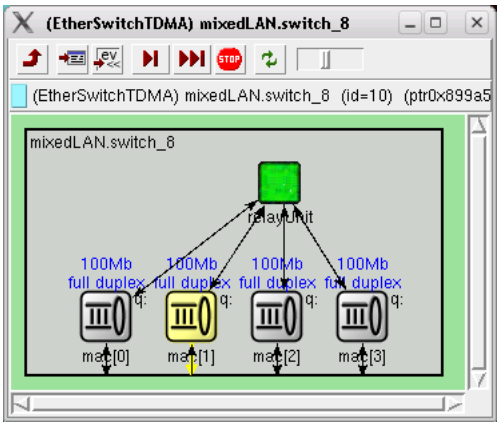

Abbildung 15: Detailansicht Switch OMNeT++

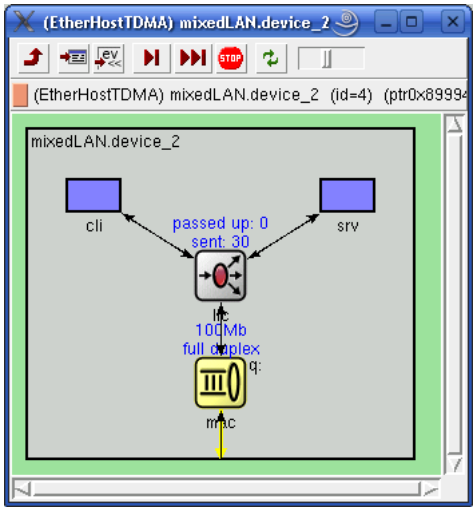

Abbildung 16: Detailansicht Netzwerkteilnehmer OMNeT++

5 Simulation des Echtzeit-Netzwerkes

Nach ausführlicher Diskussion der verschiedenen Realisierungsansätze für Echtzeit-Ethernet und den Grundlagen der diskreten Ereignissimulation schließt sich nun die Simulation des neu entwickelten RTE-Protokolls an. Zunächst erfolgt eine genauere Definition der Aufgabenstellung von Kapitel 1.2 und eine Aufgliederung in die zu leistenden Teilaufgaben. Anschließend lassen sich dann die einzelnen Anforderungen an die zu erstellenden Werkzeuge bzw. an die Simulation definieren. Nach dieser Analysephase folgt die Darstellung des gewählten Entwurfs in dem die gewählten Lösungsansätze weiter diskutiert und verfeinert werden. Sowohl die Analyse als auch der Entwurf werden dabei nach objektorientierten Techniken und Vorgehensweisen durchgeführt. Dieses Kapitel endet mit einer genauen Vorstellung der Implementierung. Dabei wird auf die Wiedergabe von Quellcode zugunsten einer ausführlichen Darstellung mit Hilfe von Diagrammen verzichtet. Die verwendete Notation entspricht dabei dem weitverbreiteten UML Standard, für eine Übersichtsdarstellung siehe [32].

5.1 Anforderungen an die Simulation

Nach der Wahl der Simulationsumgebung lässt sich die Aufgabenstellung nun genauer eingrenzen. Zunächst müssen allgemeine Anforderungen an das zu entwickelnde System spezifiziert, und eine Einteilung in einzelne Module vorgenommen werden. Diese einzelnen Komponenten lassen sich dann analog exakter definieren.

5.1.1 Anforderungen an das Gesamtsystem

Es ist ein Modell für den Simulator OMNeT++ oder das darauf aufsetzende INET Framework zu erstellen. Die Anforderungen an dieses Modell lassen sich den Arbeiten von Dopatka und der Projektgruppe entnehmen [5, 6, 39]. Weitere Designziele ergeben sich durch den vom Frameworks vorgegebenen Rahmen. So sollen sich die erstellten Module möglichst problemlos in das Simulationssystem einfügen, weiterhin ist eine Minimierung des Implementierungsaufwand beispielsweise durch Wiederverwendung innerhalb des gesamten Systems ebenfalls sehr sinnvoll. Natürlich soll die erstellte Simulation auch gültige Ergebnisse liefern, die sich einfach auswerten lassen.

Es sind zunächst Netzwerke zu erzeugen, die geeignet für die Simulation sind. Dabei sollen sich sowohl kleine Netze für Testzwecke, als auch sehr große für die abschließenden Simulationen generieren lassen. Für jedes dieser Netzwerke lässt sich dann ein eindeutiger zeitlicher Ablaufplan berechnen. Diese Berechnung erfolgt über ein externes Werkzeug, das seine Ergebnisse in einem eigenen Datenformat zur Verfügung stellt. Das Format der Netzwerke muss deshalb ebenso von diesem Werkzeug unterstützt werden. Um eine Simulation in OMNeT++ vornehmen zu können, ist anschließend eine Konvertierung dieser Daten notwendig. Er ist also eine Anpassung der Eingabedaten, die sich in Semantik und Syntax von dem Zielformat unterscheiden, vorzunehmen. Jetzt lässt sich die Simulation durchführen, die dabei entstehenden Daten erfordern eine gründliche Auswertung.

Anhand der im vorhergehenden Abschnitt genannten Anforderungen lässt sich eine Einteilung der einzelnen Teilaufgaben vornehmen. Um die Abhängigkeiten zwischen den einzelnen Modulen möglichst gering zu halten und funktional zusammengehörige Tätigkeiten nicht zu zergliedern, erscheint eine

Aufteilung in vier einzelne Arbeitspakete sinnvoll. Diese Einordnung wird in Abbildung 17 als UML-Aktivitätsdiagramm vorgestellt.

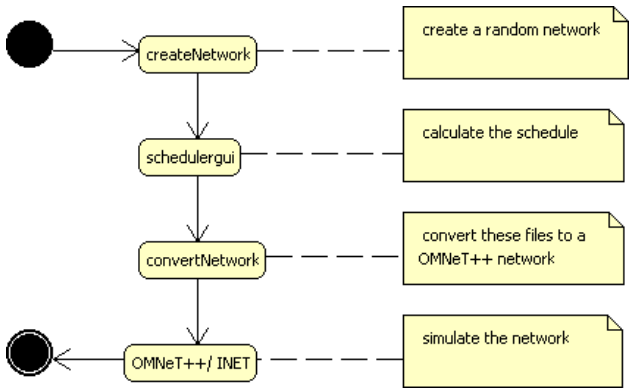

Abbildung 17: Aktivitätsdiagramm Gesamtablauf Simulation

Dieses Diagramm ist, wie auch alle folgenden, in Englisch verfasst, um eine Einheitlichkeit mit dem ebenfalls in dieser Sprache formulierten Programmcode zu erreichen. Anhand dieses Ablaufplanes lässt sich nun eine weitere Spezifizierung des zu erstellenden Systems vornehmen. Da es sich bei dem Programm „schedulergui“ um ein externes Programm handelt, dessen Implementierung zudem zum Zeitpunkt der Diplomarbeit schon weit fortgeschritten war, ist es wenig sinnvoll hierzu Anforderungen zu erfassen. Auf alle anderen Komponenten wird im Folgenden einzeln eingegangen.

5.1.2 Anforderungen an „createNetwork“

Das Werkzeug zum Erstellen der Netzwerke muss zunächst eine große Bandbreite von Netzen erzeugen können, es soll also flexibel parametrierbar sein. Die generierten Netzwerke müssen „wirklich“ zufällig aufgebaut sein und der Spezifikation der Projektgruppe [39] entsprechen. Eine gute Erzeugung von Zufallszahlen ist also essentiell. Es ist eine tragfähige und ausbaufähige Architektur vorzusehen und eine geeignete Datenstruktur zur internen Repräsentation des Modells zu entwerfen. Um den Aufwand überschaubar zu halten ist zu evaluieren, inwieweit der Einsatz von externen, frei verfügbaren Komponenten sinnvoll ist. Das Werkzeug soll einfach bedienbar sein, den Benutzer mit erläuternden Programmausgaben zur Seite stehen und ebenfalls die in [6] genannten unterschiedlichen Kommunikationsmodelle unterstützen.

Nach diesen Vorüberlegungen kristallisieren sich mehrere sinnvolle Eingabeparameter heraus. Zuerst natürlich die **Netzwerkgröße**, hier erscheint ein Bereich von zehn bis 100.000 Teilnehmer als absolut ausreichend. Dann die Einflußfaktoren für die Topologie und die Kommunikation, für die die Verwendung von Prozentwerten geeignet ist. Um die Form des Netzwerks zu beeinflussen, ist die sogenannte **Geschwisterchance** entscheidend. Sie beeinflusst ob ein Knoten weitere Geschwister hat, wobei ein Wert von null eine Bustopologie und die Verwendung von eins ein als Stern aufgebautet Netz erzeugt. Die beiden nächsten Einflussgrößen, **Verbindungschance** und **Verbindungsreichweite** legen analog die Wahrscheinlichkeit fest, mit der ein Knoten zu einem anderen eine Kommunikationsbeziehung aufbaut, und wie weit dieser Partner entfernt ist. Die Modifikation der **nicht Echtzeitchance** erlaubt das Einfügen von gewöhnlichen, nicht echtzeitfähigen Teilnehmern in das Netz. Über den **Verkehrsmodus**

lassen sich die möglichen Kommunikationsstrukturen Master, Multi-Master und eine zufällige Verteilung auswählen. All diese Parameter sind optional; es sind sinnvolle Defaultwerte vorgesehen. Die Angabe eines **Dateinamen** für das erzeugte Netz ist obligatorisch. In der normalen Betriebsart wird der **Zufallsgenerator-Startwert** aus der aktuellen Zeit und Datum gebildet. Um wiederholbar das gleiche Netzwerk zu erzeugen lässt er sich aber auch beim Start angeben. Die Bedienung dieses Werkzeuges ist im Anhang in Kapitel A.1 ausführlich dargestellt.

5.1.3 Anforderungen an „convertNetwork“

Bei dem Werkzeug zum Konvertieren des Netzwerkes, des Ablaufplanes und notwendiger Hilfsdaten (sog. Kommunikationslinien) steht in erster Linie eine einfache Umsetzung im Fordergrund. Eine Lösung beispielsweise durch ein Skript ist also der Erstellung eines weiteren Programms vorzuziehen. Fehler sollen sicher gehandhabt werden, es muss weiterhin die notwendige Flexibilität für eine einwandfreie Übersetzung der Daten gewährleistet sein. Auch hier ist eine einfache Bedienung vorzusehen, wenn möglich sollen keine zusätzlichen Angaben vom Benutzer erforderlich sein. Notwendige Eingaben sind die Dateinamen für das **Netzwerk**, den **Schedule** und die **Kommunikationslinien**. Die Namen der Ausgabedateien lassen sich dann daraus ableiten.

5.1.4 Anforderungen an das Simulationsmodell

Die zu erstellenden Modellkomponenten sollen sich gut in die bestehende Struktur des Simulationssystems einbinden. Dies erleichtert nicht nur die Implementierung und Wartung, sondern ermöglicht es bestehende Teile des Simulationssystems wiederzuverwenden. Weiterhin lassen sich dadurch die vorhandenen Möglichkeiten zur Benutzerinformation und Ergebnissicherung mitnutzen. Die zu verwirklichende Funktionalität lässt sich wiederum [5, 6] und der Dokumentation der Projektgruppe [39] entnehmen. Es ist eine gute Unterstützung des Standard Ethernet Protokolls anzustreben, dies wurde aber ebenso wenig wie der genaue Grad der Echtzeit-Unterstützung im Vorfeld exakt festgelegt.

Nach Überlegungen im Vorfeld und praktischen Versuchen innerhalb des Frameworks erscheint eine Anlehnung an die in Kapitel 3.2.2 vorgestellte PROFINET Struktur am sinnvollsten. Die Switche enthalten hier die komplette Intelligenz und kümmern sich um die Einhaltung des Zeitplans. Wie im IRT Modus dieses Verfahrens folgt die Verarbeitung des nicht Echtzeit-Verkehrs nach dem RT-Slot. Innerhalb eines Echtzeit-Zeitschlitz ankommender „gewöhnlicher“ Verkehr muss gegebenenfalls einige Zeit gepuffert werden. Der Fokus der Implementierung liegt zunächst auf einer guten Unterstützung des Echtzeitbetrieb unter eventueller Vernachlässigung von praktischen Problemen wie die Verarbeitungszeit in Switches, Datenlaufzeiten oder ähnlichem. Abbildung 18 stellt den angedachten Zeitschlitzaufbau des Simulationsmodells vor.

Natürlich soll in dieser Diplomarbeit nicht einfach PROFINET simuliert werden, zumal Jasperneite in [20] schon eine entsprechende Simulation vorgenommen hat. Der durch dieses Modell verwirklichende Ansatz umfasst daher erhebliche Leistungsverbesserungen. Als erste Maßnahme ist die optimierte Berechnung des Kommunikation-Ablaufplans mit Hilfe spezieller Algorithmen herauszustellen. Weiter können die Switche nicht belegte Zeitschlitze, obwohl sie innerhalb des Echtzeitbereichs liegen, für die Kommunikation mit nicht Echtzeitanforderungen nutzen. Diese beiden Maßnahmen versprechen eine

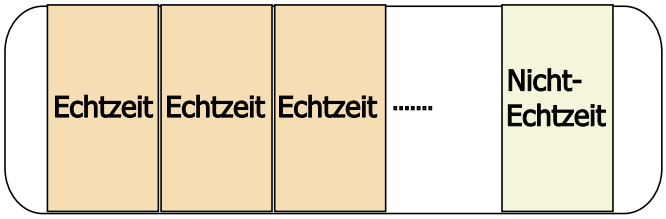

Abbildung 18: Entwurf Zeitschlitzaufbau des Simulationsmodells

erheblich bessere Auslastung des Übertragungsmediums und damit eine bessere Leistung sowohl im Echtzeit- als auch im nicht Echtzeitbetrieb. Dazu ist jedoch eine Anpassung der MTU[28] der einzelnen Netzwerkschnittstellen, oder eine Segmentierung größerer Rahmen im nicht Echtzeitbetrieb notwendig.

5.2 Entwurf und Implementierung der Simulation

Nun gilt es, die gefundenen Anforderungen in ein konkretes Design umzusetzen. Zusätzlich zu den im vorherigen Kapitel genannten Anforderungen ist jetzt die Berücksichtigung weiterer Faktoren wie beispielsweise der technischen Gegebenheiten notwendig. Die Darstellung einzelner Entwurfsentscheidungen und der konkreten Implementierung erfolgt anhand der im vorherigen Kapitel abgesteckten Einteilung in einzelne Module. Der Entwurf und die Implementierung erfolgen hier nach einem iterativen Prozess, d.h. zunächst verwirklicht man grundlegende Funktionalität, die dann nach und nach ausgebaut wurde. Erreichte Arbeitsergebnisse werden durch Tests überprüft. Genauere Information über die verwendeten Werkzeuge und natürlich über die Einrichtung der Arbeitsumgebung finden sich in Kapitel 5.3.2.

5.2.1 Entwurf von „createNetwork“

Schon zu Beginn des Designs erscheint nur eine Lösung mittels eines Toolkits, dass nahezu alle notwendige Unterstützungsfunktionen bereitstellt, sinnvoll. Eine Verwirklichung mit Hilfe eines Shellskripts, oder durch die Verwendung von verschiedenen Hilfsbibliotheken ist zu aufwendig. Aufgrund der guten Erfahrungen, die der Autor in früheren Projekten mit dem Toolkit Qt der Firma Trolltech machen konnte [40], wurde diese Bibliothek ausgewählt. Die freie Verfügbarkeit von Qt als Opensource, die Plattformunabhängigkeit und der Einsatz dieser Klassenbibliothek durch die Projektgruppe untermauerten diese Entscheidung zusätzlich. Aus diesem Entschluss ergibt sich dann die Implementierungssprache C++.

Einarbeitung und Prototypenerstellung Nach Einrichtung der Arbeitsumgebung mit Qt werden zunächst einige Prototypen für eine graphische Benutzeroberfläche mit Hilfe des Qt Designers erstellt. Diese erhalten dann im Anschluß erste Funktionen, um XML zu erzeugen und abzuspeichern. Dies dient auch der Wiedereinarbeitung in C++ und dem Kennenlernen der bereitgestellten XML Funktionen von Qt. Nach kurzer Zeit stellt sich dabei aber heraus, dass die Verwirklichung einer ansprechenden graphischen Oberfläche zu viel Zeit gekostet hätte. Ein textuelles Interface auf der Konsole ist zudem besser für häufig wiederholte Testläufe geeignet. Deshalb wurde die Idee, eine graphische Oberfläche zu

[28] Abkürzung für Maximum Transmission Unit

realisieren, nicht weiter verfolgt. Nach Portierung der bis dahin erreichten Ergebnisse ist aber erst ein weiterer Ausbau der Infrastruktur an notwendig. Diese iterative Herangehensweise war typisch für die Entwicklung dieses Werkzeuges, die endgültige Struktur entstand durch fortwährende Veränderungen bzw. Abstrahierung und gezieltes Refactoring.

Pseudo-Zufallszahlengenerierung Nach einigen Versuchen mit kleineren Testfällen stellt sich heraus, dass die in der C-Bibliothek integrierte Funktion zum Erzeugen von Zufallszahlen nicht den gesetzten Anforderungen entspricht. Die erzeugten Sequenzen sind nicht gleichmäßig verteilt, dies führt zu einem ungenügenden, „unausgeglichenen“ Netzaufbau[29]. Es muss also entweder ein eigener Pseudozufallszahlengenerator entworfen, oder ein externer eingebunden werden. Die Nutzung eines externen Bibliothek erscheint aber sinnvoller, auch da keinerlei Erfahrungen mit dem Entwurf eines solchen Generators vorhanden sind. Nach kurzer Recherche fällt die Wahl auf „MersenneTwister“ [29], ein Generator mit einer extrem langen Periode, der hochgradig gleichverteilte Folgen bereitstellt. Dadurch lassen sich die aufgetrenen Probleme beheben. Die von den Autoren bereitgestellte Version ist außerdem frei in eigenen Projekten nutzbar.

Der Zugriff auf den Generator erfolgt zunächst direkt. Um Probleme aufgrund eines nicht richtig initialisierten Generators oder der Verwendung von mehreren PRNGs[30] zu vermeiden, erscheint eine Änderung der Architektur notwendig. Mersenne Twister wird deshalb in einer Klasse gekapselt. Der Zugriff über eine abstrahierte Schnittstelle hilft zudem, die generelle Architektur flexibel zu halten. Durch den Einsatz des Singleton-Patterns kann man ausserdem sicherstellen, dass immer nur mit einem Generator gearbeitet wird, und eindeutige IDs wirklich auch über die gesamte Programmlebensdauer eindeutig bleiben.

Erzeugung der Netze Nun muss eine Datenstruktur für die interne Verarbeitung des Netzes ausgewählt werden. Die durch das Toolkit Qt bereitgestellten Strukturen wie Liste oder Array hätten Anpassungsmassnahmen benötigt, da die geplante Netzwerksstruktur mit ihnen nicht ohne weiteres zu verwirklichen ist. Die richtige Repräsentation der Daten ist wichtig genug, um die Benutzung einer externen Datenstruktur zu rechtfertigen. Diese Datenstruktur soll eine bekannte und leicht benutzbare API implementieren. Mit der „STL-like template tree class“ von Peeters [35] lässt sich eine gute Grundlage für die Netzgenerierung schaffen. Da diese Klasse freie Software ist und ein Interface ähnlich der C++ Standard-Template-Library besitzt, ist sie ohne Probleme in das Projekt einzubinden.

Nachdem die Grundlagen für eine tragfähige Architektur gelegt waren, folgt eine ausführliche Analyse der XML-Spezifikation der Projektgruppe. Dieses Format ist in [39] ausführlich dokumentiert, und auch im Kapitel 5.2.3 exemplarisch dargestellt. Die von dem Werkzeug „schedulergui“ benötigten Netzwerke gliedern sich in drei Teile. Zunächst erfolgt die Deklaration der Switches und gewöhnlichen Netzwerkgeräten, also der Hardware. Dann folgen die Verbindungen zwischen den einzelnen Teilnehmern und schließlich die auszutauschenden Nachrichten. Es erscheint sinnvoll, auch die Erzeugung der Netze anhand dieser vorgegebenen Struktur zu realisieren.

Für die Generierung der Netzwerke wird ein dreistufiger Prozess entworfen. Der erste Schritt ist die Erzeugung eines zufälligen Netzes als beliebigen Baum. Da die Bestimmung einzelner Knotentypen,

[29] Beispielsweise waren der Hauptteil der Geschwisterknoten nur in einem Bereich des Netzes angeordnet und nicht gleichmäßig verteilt.

[30] Eine Abkürzung für Pseudorandom number generator.

also Switch oder normaler Teilnehmer nicht während der Erzeugung möglich ist, erfolgt dies in einem separaten Durchlauf[31]. In dieser Phase lassen sich auch bestimmte Metadaten, die das Werkzeug der Projektgruppe benötigt, setzen. Anhand dieser Datenstruktur lassen sich nun weiter die Verbindungen und ausgetauschten Nachrichten ableiten, die in Listen zwischengespeichert werden. Die Verbindungen ergeben sich dabei direkt aus der Baumstruktur. Die Erzeugung der Nachrichten ist zufallsgesteuert, wobei natürlich die Eingabeparameter die Erzeugung beeinflussen. Der entworfene Prozess ist in Abbildung 19 noch einmal schematisch dargestellt.

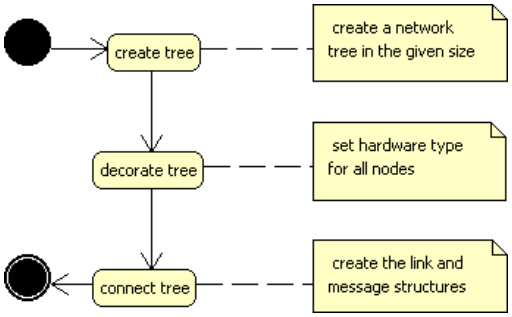

Abbildung 19: Aktivitätsdiagramm Netzwerkerzeugungsprozess createNetwork

Erzeugung von XML Die Generierung des XMLs aus dem Netzwerk ist nun nicht mehr schwierig. Die im vorherigen Schritt entstanden Datenstrukturen lassen sich einfach durchlaufen, und in der richtigen Syntax in einer Datei ablegen. Die gute Implementierung des XML Document Object Model durch Qt vereinfacht diese Arbeit zusätzlich. Die XML-Erzeugung und die Methoden zum Dateizugriff werden in eigenen Klassen gekapselt, um die Modularität der Applikation sicherzustellen.

Es zeigt sich aber in diesem Schritt, dass Spezifikationen von Dateiformaten sehr genau sein müssen. Viele implizite Annahmen werden oft nicht dokumentiert, dies kann zu Problemen bei der Interoperabilität führen. Als Beispiel wurden die Ids der einzelnen Netzwerkteilnehmer vom Autor als beliebige Zufallszahlen angenommen, die sonst keinerlei Bedeutung haben. Innerhalb des „schedulergui“-Werkzeuges treten dadurch Probleme auf, da sie dort zum Zugriff auf bestimmte Datenstrukturen vorgesehen sind. Diese und weitere Mißverständnisse können durch Zusammenarbeit aber schnell ausräumt werden.

5.2.2 Implementation von „createNetwork“

Nach der ausführlichen Vorstellung der Designentscheidungen folgt nun die Erläuterung der konkreten Implementation dieses Werkzeugs. In Abbildung 20 ist die Klassenstruktur von „createNetwork“ aufgeführt. Um die Übersicht zu bewahren, sind in diesem UML-Diagramm die Methoden ausgeblendet. Diese werden in den folgenden Abbildungen 21 und 22 ausführlich dargestellt.

Gut zu erkennen ist hier das Singleton-Pattern bei der `Utility` Klasse, durch den der Zugriff auf `mersenneTwister` ermöglicht wird. Es wurde realisiert über eine statische Zugriffsmethode, die beim ersten Aufruf den privaten Konstruktor aufruft. Ausserdem initialisiert sie den PRNG mit einen übergebenen Wert, oder erzeugt ihn aus Uhrzeit und Datum. Darüber hinaus bietet sie Hilfsfunktionen zur Variablenüberprüfung, Zufallszahlenerzeugung und MAC-Adressengenerierung an.

[31] Dieser Vorgang wird in Anlehnung an den Compilerbau mit „decorate tree“ bezeichnet.

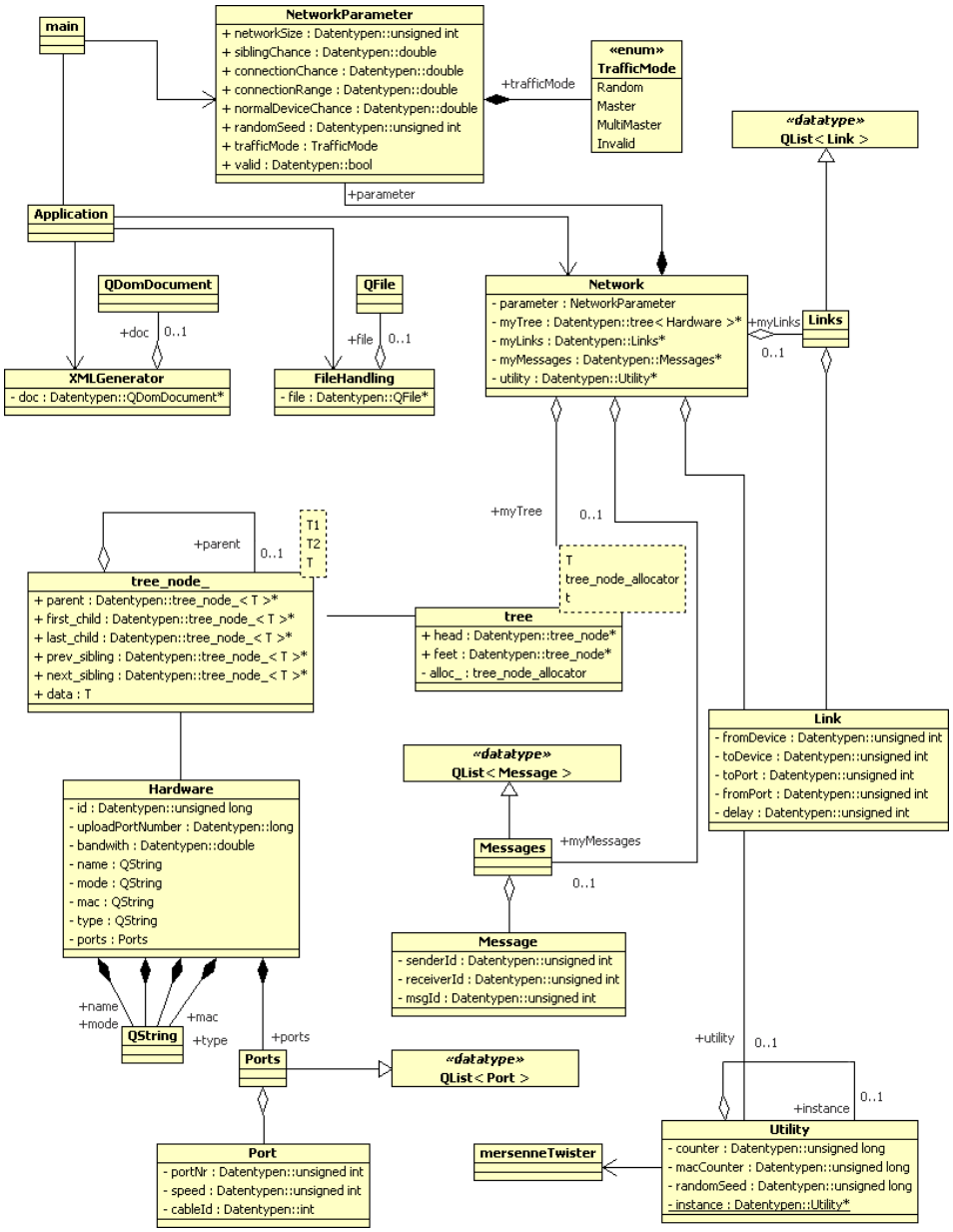

Abbildung 20: Klassenstruktur createNetwork Werkzeug

Weiter interessant ist die Instanzierung von der Templateklasse `tree` durch `Hardware`. Die Klassen `Links` und `Messages` erben von einer QList, die Qt Implementierung einer doppelt verketteten Liste, dadurch lässt sich die benötigte Funktionalität sehr einfach und effizient bereitstellen. Auch hier sorgt der Template-Mechanismus für eine typsichere Datenhaltung. Die eigentliche Benutzerführung befindet sich in der `main`-Methode, die dann ein Objekt vom Typ `Application` und `NetworkParameter` erzeugt, die die eigentliche Funktionalität kapseln. Somit ist eine Veränderung der Bedienung, beispielsweise durch Bereitstellung einer graphische Benutzeroberfläche, sehr einfach möglich. Die Klasse `Application` benötigt zur Erzeugung eines Netzwerkes nur einen Dateinamen und die Parameter.

Die einzelnen Funktionen des PRNGs und der `tree`-Klasse sind ebenfalls in der Detailansicht aus Gründen der Übersichtlichkeit nicht aufgeführt.

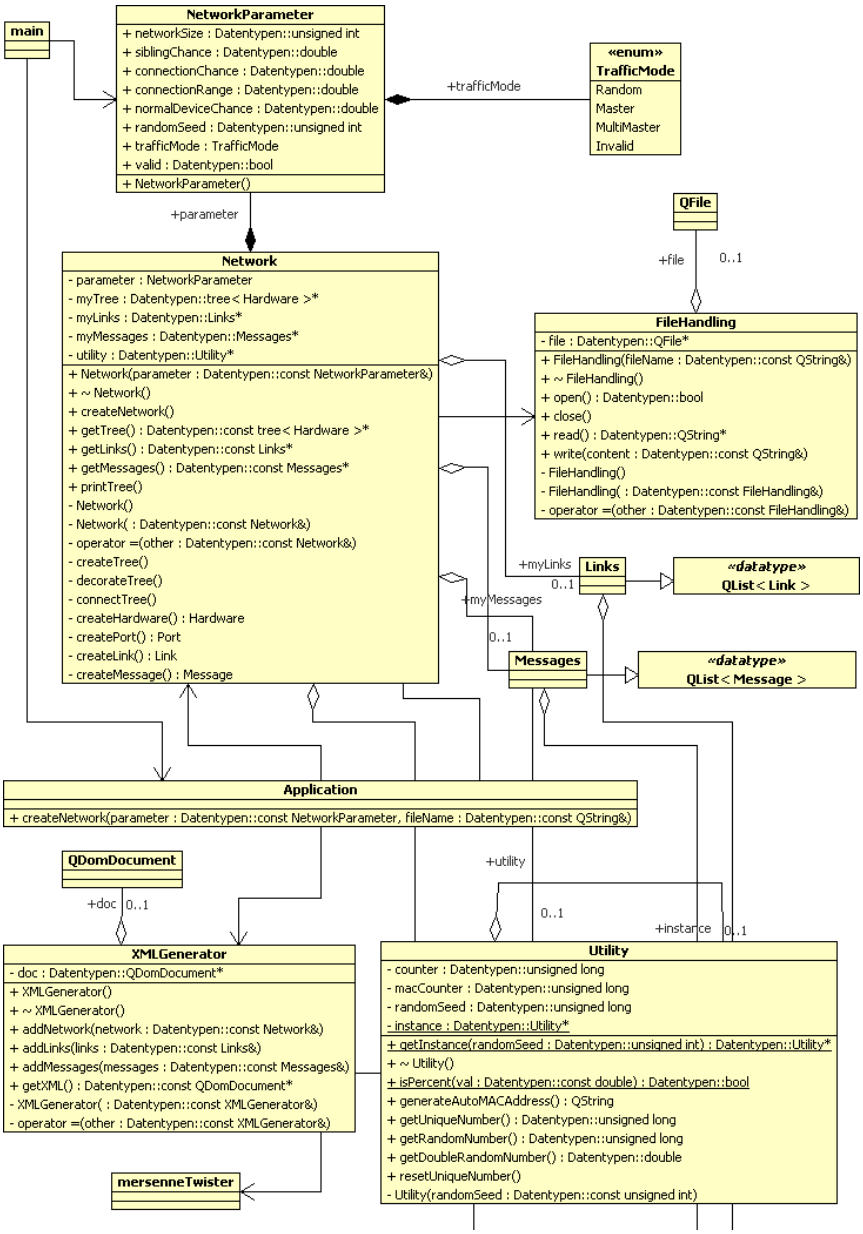

Abbildung 21: Detailansicht Klassenstruktur createNetwork Teil 1

Die Klassen `Hardware`, `Port`, `Link` und `Message` symbolisieren die entsprechenden Elemente innerhalb der zu erzeugenden XML-Struktur. Die für die spätere Berechnung des zeitlichen Ablaufplan nicht notwendigen Parameter erhalten Zufallswerte, bzw. entfallen innerhalb der Ausgabedatei komplett. `XMLGenerator` und `Filehandling` nutzen die von Qt angebotenen Funktionen QFile und QDomDocument zur Erledigung ihrer Aufgaben. Der `XMLGenerator` erzeugt den XML-Baum innerhalb des Speichers und gibt ihn als const-Objekt, das nicht verändert werden kann, weiter. Aus diesen Daten erzeugt das `Application` Objekt einen String und speichert ihn über die `FileHandling` Klasse ab.

Die Klasse `Network` stellt ein zentrales Element in der Anwendung dar. Die Methoden `createTree()`, `decorateTree()` und `connectTree()` stellen die im vorherigen Kapitel aufgeführten einzelnen Prozessschritte zur Erzeugung des Netzwerkes dar. Alle erzeugten Datenstrukturen werden gleichermaßen nur als const-Objekte an `XMLGenerator` übergeben, so dass auch hier die Unveränderlichkeit der

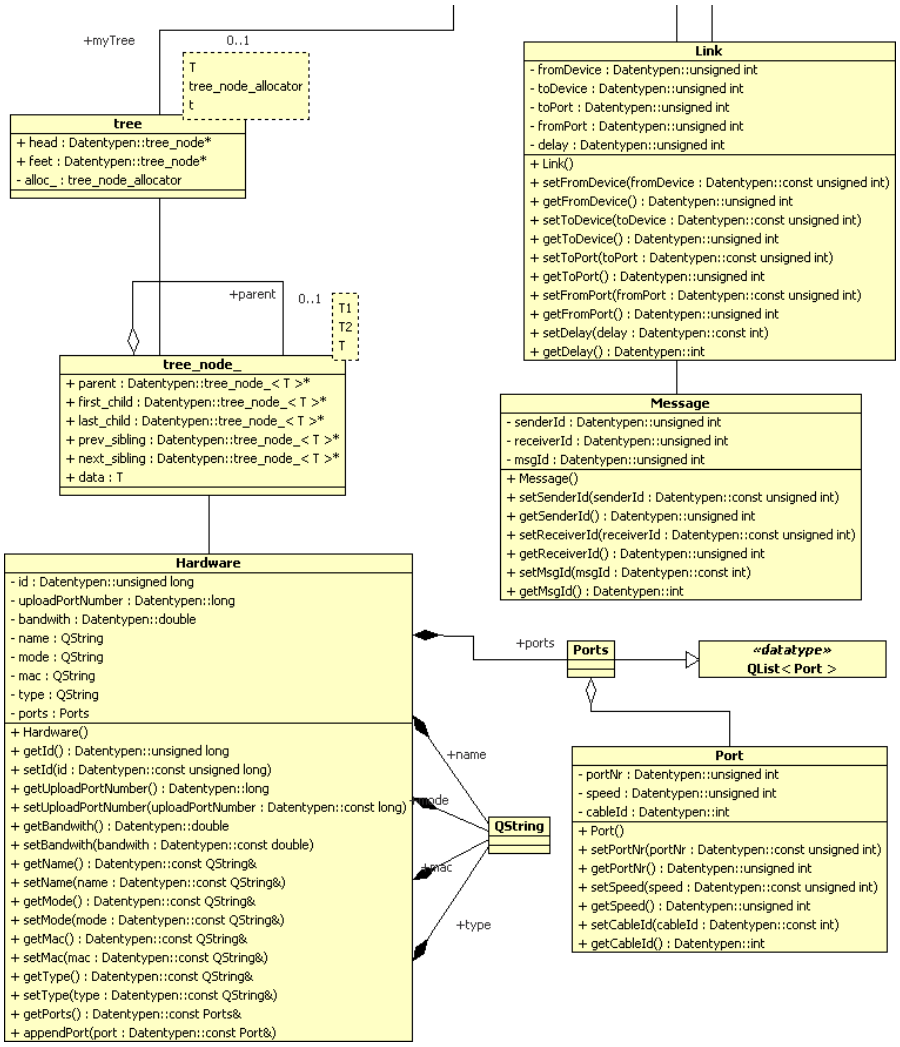

Abbildung 22: Detailansicht Klassenstruktur createNetwork Teil 2

Daten gewährleistet ist. Das Sicherstellen dieser sogenannten „const-correctness“ wirkt sich positiv auf die Leistung des Programms aus, außerdem können damit Fehler leichter erkannt werden. Die Erzeugung von Objekten zum einfügen in das erstellte Netzwerk erfolgt über separate Hilfsmethoden wie beispielsweise `createHardware()`, um Änderungen einfacher durchführen zu können.

Bei Klassen, die dynamisch erzeugte Datenstrukturen enthalten, wie `Network` oder `FileManagement` wird der Copy-Konstruktor, der operator „=“ und der Default Konstruktor als privat definiert. Dadurch ist kein Zugriff mehr auf diese normalerweise immer vom Kompiler generierten Funktionen mehr möglich. Durch diese Maßnahme lässt sich eine falsche Verwendung der Programmierschnittstellen oder eventuelle automatische Typkonvertierungen, durch die falsche Ergebnisse entstehen könnten, unterbinden.

Der erstellte Quellcode ist dem der Diplomarbeit beigefügten Datenträger zu entnehmen, dessen Struktur in Kapitel B.2 des Anhangs erläutert wird. Die entstandene Anwendung umfasst etwa 1500 Zeilen C++ Code. Zur Steuerung von Übersetzung und Link-Prozess kommt qmake, das Make von Qt zum Einsatz. Bei der Entwicklung traten keine größeren Komplikationen auf. Kleinere Schwierigkeiten machten nur die bei C++ üblichen Probleme wie ungültige Zeiger und die Feinheiten der manuellen Speicherverwaltung.

5.2.3 Entwurf von „convertNetwork“

Wie schon in Kapitel 5.1.3 dargestellt, steht bei diesem zu entwerfenden Werkzeug eine möglichst einfache Umsetzung im Fordergrund. Eine Option ist die Lösung durch ein Programm oder Skript, welches die Quelldatei zur Konvertierung einliest und sie anhand von DOM oder SAX in das Ausgabeformat umwandelt. Dieser Ansatz erfordert relativ viel Aufwand, allerdings kann eine erneute Einarbeitung entfallen, da beispielsweise die Qt XML Werkzeuge schon aus der Implementierung von createNetwork bekannt sind. Eine andere Herangehensweise an das Problem ist die Verwendung von Transformationssprachen, die speziell dafür geeignet sind, XML Dokumente in ein anderes Format zu überführen. XSLT ist der bekannteste Vertreter dieser Sprachen. Die Nutzung dieses Konzepts erfordert die Einarbeitung in eine neue Sprache, verspricht aber ingesamt kürzere Entwicklungszeiten.

Werkzeugauswahl und Einarbeitung Nach Recherchen und der Erstellung von kleineren Testfällen fällt die Wahl dann auf XSLT. Diese Entscheidung wird weiter dadurch unterstützt, dass auf der Entwicklungsplattform schon dafür geeignete Werkzeuge installiert sind. Diese Transformationssprache basiert auf XML, die eigentliche Umwandlung des Dokuments wird von einem XSLT-Prozessor vorgenommen. Die Ausgangsdateien können sowohl als XML, oder auch in einem beliebigen Textformat ausgegeben werden. Die Funktionsfähigkeit der vorhandenen Werkzeuge und der allgemeine Arbeitsablauf wird durch die Erstellung von kleineren Protoypen sichergestellt.

Die Simulationsumgebung OMNeT++ unterstützt die „NED“-Sprache, die Verwendung von XML ist aber gleichermaßen möglich. Die Definition von Netzwerken erfolgt in NED, für einfache Parameterdefinitionen wird ein Format benutzt, was allgemein als „INI“-Datei bekannt ist. Dieses wird auch als als universelles Konfigurationsformat innerhalb von OMNeT++ und dem INET Framework benutzt. Die Nutzung von XML ist dann für komplexere Eingabewerte sinnvoll.

Darstellung der zu konvertierenden Formate Um eine genauere Vorstellung von der zu leistenden Aufgabe zu bekommen, werden die einzelnen zu konvertierenden Quellen und die Zielformate beispielhaft aufgeführt. Zunächst ist die Quellsprache des „createNetwork“-Werkzeugs zu betrachten, also das Format welches die Projektgruppe erwartet. Das folgende Listing stellt einen Echtzeitswitch mit zwei Ports dar, an die zwei Echtzeitgeräte angeschlossen sind. Das erste Gerät kommuniziert dabei mit dem zweiten Gerät. Die Formatdefinition wurde unverändert von der Projektgruppe übernommen, auf genauere Informationen zu dieser Sprache wird daher an dieser Stelle verzichtet.

```
<NetStDTD>
  <Switches>
    <Hardware Mode="f" UploadPortNumber="-1" Type="RTSwitch" Id="0"
    BandWidth="800000000" Mac=" AAA1BBC23FE" Ports="5" >
      <Ports>
        <Port PortNr="1" Speed="800000000" CableId="-21" />
        <Port PortNr="2" Speed="800000000" CableId="-22" />
      </Ports>
    <Hardware Mode="f" UploadPortNumber="0" Type="RTDevice" Id="1"
    BandWidth="800000000" Mac="AAA1BBC23FF" Ports="1" >
      <Ports>
```

```
        <Port PortNr="0" Speed="800000000" CableId="-26" />
      </Ports>
    </Hardware>
    <Hardware Mode="f" UploadPortNumber="0" Type="RTDevice" Id="2"
    BandWidth="800000000" Mac="AAA1BBC240" Ports="1" >
      <Ports>
        <Port PortNr="0" Speed="800000000" CableId="-27" />
      </Ports>
    </Hardware>
  <Links>
    <Link FromDeviceId="0" ToPort="0" ToDeviceId="1" Delay="0"
    FromPort="0" />
    <Link FromDeviceId="0" ToPort="0" ToDeviceId="2" Delay="0"
    FromPort="1" />
  </Links>
  <Messages>
    <Message MsgId="0" SenderId="1" ReceiverId="2" />
  </Messages>
</NetStDTD>
```

Man erkennt gut die Aufteilung der Definition in Hardware, Verbindungen und Nachrichten, wie zuvor schon in Kapitel 5.2.1 vorgestellt. Nun folgt die Darstellung einer Datei im NED-Format zur Definition von Netzwerkstrukturen für OMNeT++, hier ein einfaches Netzwerk mit zwei Teilnehmern.

```
import
  "EtherHost",
  "EtherHostTDMA",
  "EtherSwitchTDMA";
module MixedLAN
  submodules:
    switch_0: EtherSwitchTDMA;
      gatesizes:
        in[5],
        out[5];
      display: "i=switch2";
    device_1: EtherHostTDMA;
      display: "i=device/pc2";
    device_2: EtherHostTDMA;
      display: "i=device/pc2";
  connection:
    switch_0.out[0] --> delay 0us --> device_1.in;
    switch_0.in[0] <-- delay 0us <-- device_1.out;
    switch_0.out[1] --> delay 0us --> device_2.in;
    switch_0.in[1] <-- delay 0us <-- device_2.out;
endmodule
network mixedLAN : MixedLAN
endnetwork
simple Simple
endsimple
```

In dieser Datei sind die gleichen zwei Geräte wie im vorherigen Beispiel dargestellt. Diese Sprache unterstützt, wie im Beispiel zu erkennen ist, also nur die Definition von Netzwerkteilnehmern und ihrer

Verbindungen. Also müssen die Kommunikationsanforderungen anders repräsentiert werden. Hier benutzt INET, wie schon angesprochen, ein sogenanntes INI-Format. In der folgenden Darstellung ist die Definition der Kommunikationsbeziehungen eines Beispielnetzwerks abgebildet.

```
**.device_2.cli.destStation = "device_20"
**.device_19.cli.destStation = "device_3"
**.device_6.cli.destStation = "device_16"
**.device_12.cli.destStation = "device_14"
```

Das Gerät mit der ID „device_2“ innerhalb des Netzwerks „**.“ sendet Nachrichten an den Netzwerkteilnehmer mit ID „device_20“. Abschließend wird die Schedule Definition für das erstellte Modell innerhalb des INET Frameworks vorgestellt, diese ist wiederum als XML formuliert.

```
<?xml version="1.0" encoding="UTF-8"?>
<schedule timeSlotCount="3">
  <device id="device_2" slotId="0"/>
  <device id="device_3" slotId="0"/>
  <switch id="switch_0">
    <timeSlot id="0">
      <connection out="4" in="1"/>
    </timeSlot>
    <timeSlot id="1">
      <connection out="2" in="4"/>
    </timeSlot>
  </switch>
</schedule>
```

Am Anfang der Datei erhalten die einzelnen Geräte, hier „device_2“ und „device_3“ ihren Zeitschlitz zugeteilt. Anschließend erfolgt die Definition der einzelnen Schedules für die Switche. Genauere Informationen über diese Konfigurationsdaten und wie sie sich auf die Arbeitsweise der Geräte auswirken finden sich in Kapitel 5.2.5. Abschließend folgt nun noch die Darstellung der Schedule-Definition und der sogenannten Kommunikationslinien, wie sie das Werkzeug schedulergui der Projektgruppe generiert.

```
<schedules description="Description" name="schedules" >
  <distributor id="0" name="" >
    <timeSlot id="0" >
       <communicationLine ID="0" />
    </timeSlot>
    <timeSlot id="1" >
      <communicationLine ID="1" />
    </timeSlot>
  </distributor>
</schedules>
```

Für eine genauere Erläuterung der Struktur dieser beiden Datenformate sei auf die Dokumentation der Projektgruppe [39] verwiesen.

```
<communicationLines description="Description" name="communicationLines" >
  <communicationLine id="0" >
    <distributor Id="0" name="" >
      <inputPort>1</inputPort>
     <outputPort>4</outputPort>
    </distributor>
  </communicationLine>
  <communicationLine id="1" >
    <distributor Id="1" name="" >
      <inputPort>0</inputPort>
      <outputPort>3</outputPort>
    </distributor>
  </communicationLine>
</communicationLines>
```

Vorgehen und Architekturentscheidungen Nach ausführlicher Analyse der Quell- und Zielformate kann die Konzeption des Werkzeugs fortgesetzt werden. OMNeT++ unterstützt die Konvertierung von NED-Netzwerkbeschreibungen aus einer XML-Struktur mittels eines eigenen Werkzeuges. Da die Netzwerke relativ komplex aufgebaut sind, ist es sinnvoll, die Konvertierung mit Hilfe dieses Zwischenschrittes vorzunehmen. Die Transformierung von einem XML-Format in ein anderes erscheint einfacher realisierbar. Da zu diesem Zeitpunkt die Spezifikation der Schedules und Kommunikationslinien von der Projektgruppe noch nicht abgeschlossen war, ist eine klare Trennung der einzelnen Konvertierungsschritte notwendig. Eine Aufteilung empfiehlt sich aber ohnehin aus Gründen der Modularisierung.

Auch die Entwicklung dieses Werkzeuges erfolgt anhand eines iterativen Vorgehensprozess. Die recht einfache Transformation der Netzwerke ermöglicht das Sammeln von Erfahrungen, die in die späteren Entwicklungsphasen mit eingehen. Anhand der XML-Spezifikation der Projektgruppe und einigen Beispieldokumenten lässt sich eine erste Version eines XSLT-Skriptes verwirklichen. Nach und nach können komplexere Netzwerke konvertiert und innerhalb des INET-Frameworks getestet werden. Als dieser Bestandteil des Konverters stabil funktionierte, steht als nächster Schritt die Transformation von Kommunikationsbeziehungen auf dem Programm.

Da die Generierung von Textdateien sich nur unwesentlich von der XML-Erzeugung unterscheidet, ist dieser Arbeitsschritt recht schnell abgeschlossen. Für die Schedule-Beschreibungen muss nun erstmals ein neues, eigenes Dateiformat definiert werden. Anfangs erscheint ein ähnliches INI-Format wie für die Kommunikationsanforderungen als geeignet. Schnell zeigt sich aber, dass diese Syntax zu wenig Ausdrucksmöglichkeiten erlaubt, sie die Entwurfsfreiheit zu sehr einschränkt. Da OMNeT++ innerhalb des Modells auch XML-Parameter einlesen kann, lässt sich hier eine andere Möglichkeit finden. Dieser Teil der Konvertierung erfordert den größten Aufwand, da hier Informationen aus mehreren unterschiedlichen Quellen benötigt werden. Weiter treten bestimmte Probleme oder Grenzfälle erst bei größeren Netzwerken oder bestimmten Topologien auf.

5.2.4 Implementierung von „convertNetwork“

Um dem Benutzer den manuellen Aufruf der einzelnen Konvertierungsschritte zu ersparen, wird ein Shellscript entworfen, welches ausserdem eine Benutzerführung zur Verfügung stellt. Die Struktur des

„convertNetwork“-Werkzeugs ist in Abbildung 23 dargestellt. Die Verarbeitung der Netzwerke beginnt mit der Validierung der Benutzereingaben, wobei bei falschen oder fehlenden Parametern eine Meldung ausgegeben wird. Die Benutzerausgaben sind in Kapitel A.2 abgebildet. Der erste Schritt, ist wie im vorherigen Kapitel aufgeführt, die Transformation des Netzwerks, gefolgt von den Verarbeitung der Kommunikationsbeziehungen.

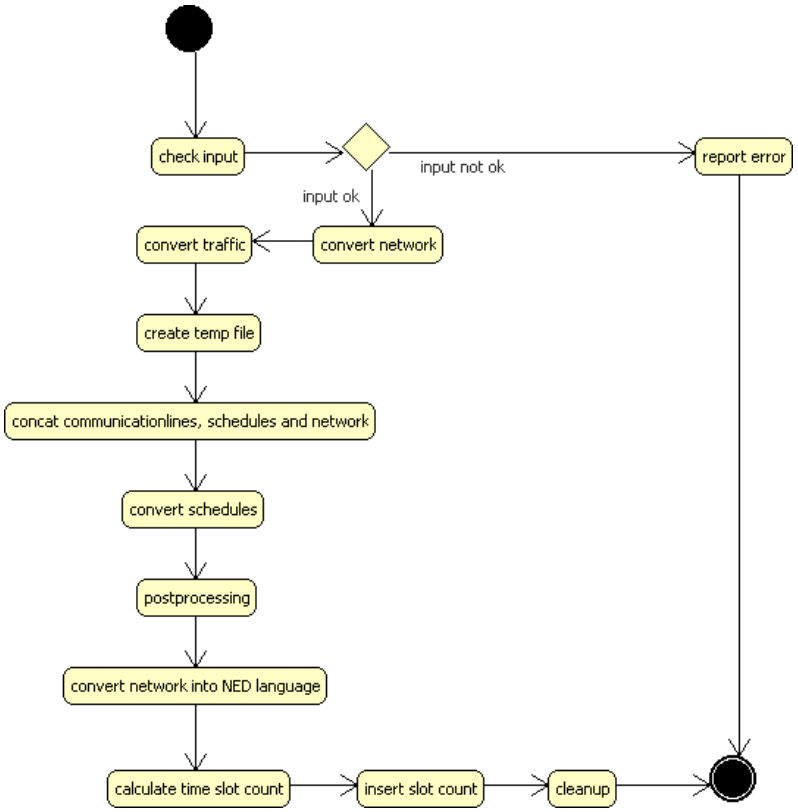

Abbildung 23: Aktivitätsdiagramm convertNetwork Werkzeug

Für die Generierung der von der Simulation benötigten Schedules ist es erforderlich, die verschiedenen Eingabedateien wie Kommunikationslinien, Schedules und das Netzwerk in einer temporären Datei zusammenzufassen. Die Postprozessing-Phase entfernt die durch die Konvertierung entstandenen Leerzeichen und Zeilenumbrüche mit dem Streameditor „sed“[32]. Nun wird durch ein OMNeT++ eigenes Werkzeug das XML-Netzwerk in eine NED-Darstellung umgewandelt. Für die einwandfreie Funktion des Modells ist nun noch die Zeitschlitzanzahl von Bedeutung. Diese lässt sich mit Hilfe des Textfilters „grep“ und des Werkzeugs „sed“ berechnen und in den zeitlichen Ablaufplan einfügen. Abschließend ist die nicht mehr benötigte Datei zu löschen. Während der Verarbeitung erfolgt eine ständige Rückmeldung an den Benutzer, zudem bekommt er zum Schluss die neu erzeugten Dateien mitgeteilt.

Für die XML-Transformation steht mit „xsltproc“ ein auf der Konsole arbeitendes Werkzeug zur Verfügung. Dieses Hilfsprogramm ist in den verbreiteten Bibliotheken libxml2 und libxslt enthalten. Innerhalb der XSLT-Skripte wird das Funktionale oder Deklarative Modell zur Programmierung benutzt. Dies erfordert ein anderes Vorgehen als in normalen Programmiersprachen: Man definiert hier Regeln, die auf alle passenden Muster angewandt werden. Dieses Programmiermodell erfordert einige Einarbeitung, ist aber erheblich mächtiger als ein prozedurales Vorgehen. Genauere Informationen über XML und XSLT

[32] Das Programm „sed“ ist ein verbreitetes Werkzeug zur Textmanipulation auf Linux- oder Unix-Systemen.

finden sich beispielsweie in der Referenz von Skonnard [49]. Da XSLT in der verwendeten Version bestimmte benötigte Funktionen nicht anbietet, oder sie nur mit sehr großen Aufwand zu realisieren sind, lagert man einige Berechnungen in das Skript aus. Als Einschränkungen lässt sich beispielsweise die Nutzung von Variablen nennen, die nur begrenzt möglich ist, auch Schleifen und bedingte Anweisungen sind nicht ohne Einschränkungen verwendbar.

Insbesondere für die Berechnung der Zeitschlitzanzahl und das Postprozessing ist die Verwendung eines Shellscriptes aber nicht optimal geeignet. Die dadurch notwendigen Hilfsprogramme lassen bei der Programmierung doch einige Wünsche offen, speziell die Syntax von „sed“ ist wenig eingängig. Eine Lösung in einer mächtigeren Skriptsprache wie Python oder Perl wäre im nachhinein betrachtet besser geeignet gewesen, dadurch ließe sich außerdem die Plattformunabhängigkeit problemlos gewährleisten. Durch die gute Unterstützung der Verarbeitung von Stringvariablen durch diese Sprachen entfiele unter anderem die Notwendigkeit, eine temporäre Datei anzulegen.

Die Geschwindigkeit der erstellten XSLT-Anweisungen war zunächst nicht ausreichend, die Konvertierung der Daten benötigte ab einer Netzwerkgröße von 1000 Teilnehmern mehrere Stunden. Durch verschiedene Optimierungsmaßnahmen, wie die möglichst direkte Formulierung von XPath-Abfragen zur Adressierung von Knoten und deren Zwischenspeicherung, könnte sie aber um mehr als das vierzigfache gesteigert werden. Der Implementierungsquellcode ist der dieser Arbeit beigelegten CD-ROM zu entnehmen, siehe auch Kapitel B.2 im Anhang.

5.2.5 Entwurf des Simulationsmodells

Wie im Kapitel 5.1.4 dargestellt, sollen sich die zu entwerfenden Komponenten möglichst gut in das Simulationssystem eingliedern. Dem begrenzten Zeitrahmen steht der Wunsch nach einer möglichst kompletten und realitätsnahen Simulation gegenüber. Zusätzlich muss noch eine Einarbeitung in das Framework vorgenommen werden. Nach Vorüberlegungen lassen sich zwei gegensätzliche Herangehensweisen an die Entwurfsaufgabe identifizieren, die nachfolgend diskutiert werden. Um effektiv innerhalb des Frameworks arbeiten zu können, muss zusätzlich eine intensive Einarbeitung stattfinden.

Einarbeitung Da keinerlei Erfahrungen mit dem Einsatz und der Implementierung von Simulationssystemen vorlagen, ist eine umfangreiche Einarbeitung notwendig. Sowohl OMNeT++ als auch das INET-Framework bringen eine umfangreiche Dokumentation und zahlreiche Beispiele mit. Insbesondere das Durcharbeiten des OMNeT++ „Tic-Toc“-Tutorial war sehr effektiv, um die Arbeitsumgebung besser kennenzulernen. Da der Entwurf des Simulationsmodells zeitlich nach den Hilfswerkzeugen geplant war, waren schon genügend Erfahrungen mit C++ und dem Entwicklungswerkzeugen auf der Zielplattform vorhanden.

Vorgehen und Architekturentscheidungen Beim ersten Ansatz ist die Simulation nur mit Hilfe des OMNeT++ Frameworks zu realisieren. Die zusätzliche Einarbeitungszeit in das INET-Framework kann deshalb entfallen, es sind mit wenig Aufwand erste Ergebnisse zu erzielen. Durch die Konzentration auf die wesentlichen Anteile, das Echtzeit-Protokoll, lässt sich dieses Modell zudem leichter verstehen und gegebenenfalls modifizieren. Ein wesentlicher Nachteil dieses Vorgehens ist der erhebliche Aufwand der

bei der Erweiterung mit normaler Ethernetfunktionalität entsteht. Es müssen zahlreiche, schon im INET-Framework enthaltene Komponenten neu implementiert werden. Eine Eigenentwicklung ist zudem nie so stabil und gut getestet wie die im Framework enthaltenen Komponenten.

Da es unklar war, ob eine eigene Implementation der relevanten Ethernetsimulationen sich überhaupt in der begrenzten Zeit durchführen ließe und sich zudem schon die Projektgruppe mit einer ähnlichen Aufgabe beschäftigte, wurde die alternative Vorgehensweise, die Nutzung von INET gewählt. Also beginnt eine Einarbeitung in die INET-Erweiterung, mit dem Ziel möglichst viel der vorhandenen Funktionen zu übernehmen und die Integration der neuen Echtzeitmodule so einfach wie möglich zu machen. Die ausführliche Dokumentation von OMNeT++ [52] und die vielen mit dem INET-Framework mitgelieferten Beispiele erleichtern diese Phase erheblich.

Nach der ersten Einarbeitung findet eine Identifikation der Bereiche statt, in denen Erweiterungen bei INET notwendig waren. Ein zusätzlicher Abgleich wird mit den in Kapitel 5.1.2 vorgestellten Werkzeug zur Erzeugung einer Netzwerkstruktur vorgenommen. Die drei gefundenen Submodule „Echtzeitswitch“, „Echtzeitteilnehmer“ und „Echtzeit-Verkehrsgenerator“ lassen sich dann einzeln weiter spezifizieren.

Echtzeitswitch Um Echtzeitnachrichten ungestört zustellen zu können, muss der Switch eine Unterscheidung der verschiedenen Frames vornehmen. Dazu teilt er die empfangenen Daten in zwei Kategorien ein. Echtzeitnachrichten, also Rahmen die innerhalb einen Echtzeitslots an dem richtigen Port ankommen, werden sofort zugestellt. Nicht Echtzeitframes müssen auf eine Gelegenheit zum Senden warten, wenn sie in einem Echtzeitslot ankommen. Sofern sie innerhalb eines freien Zeitsschlitz eintreffen, ist es möglich, sie direkt weiterzuleiteten. Falls die Zieladresse unbekannt ist, wird sie wie bei normalen Switches über einen Broadcast ermittelt. Diese werden aber nur innerhalb eines nicht-Echtzeitslot versendet, auch sie müssen gegebenenfalls warten. Abbildung 24 zeigt die entworfene Logik nochmals im Überblick.

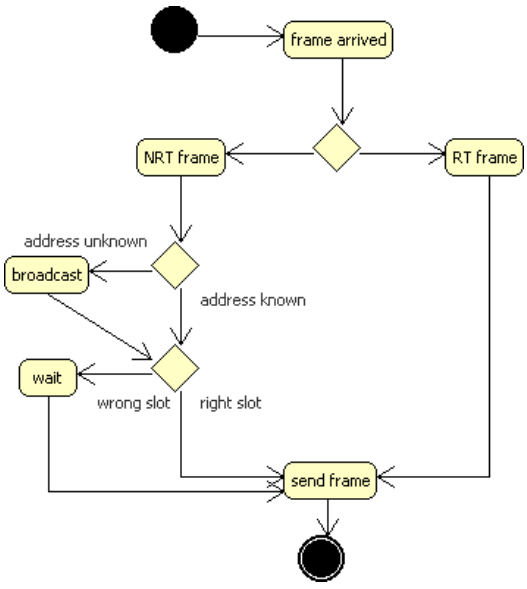

Abbildung 24: Aktivitätsdiagramm Echtzeitswitch

Um die angesprochenen Unterscheidungen treffen zu können, muss der Switch Informationen über die Zeitschlitzanzahl innerhalb des Netzes, die Lage der nicht Echtzeitslots und natürlich den zeitlichen

Ablauf besitzen. Das Werkzeug „convertNetwork“ bereitet diese Daten auf und stellt sie in der in Kapitel 5.2.3 auf Seite 48 vorgestellten Form zur Verfügung. Die Weiterleitung von Daten soll möglichst schnell erfolgen, insbesondere im Echtzeitfall ist es sinnvoll die Daten direkt, ohne Auswertung des Rahmenkopfs, weiterzuleiten.

Echtzeitteilnehmer Die Verhaltenslogik der Echtzeitteilnehmer unterscheidet zunächst die vom Netzwerk empfangenen und die von einer höheren Schicht weitergeleiteten Nachrichten. Die Verarbeitung der Frames aus dem Netzwerk erfolgt wie bei gewöhnlichen Netzwerkteilnehmern. Beim Senden ist die Einhaltung der Slotgrenzen wie bei dem Echtzeitswitch sehr wichtig. Sofern die Nachrichten innerhalb des Zeitschlitzes ankommen, der dem Teilnehmer zugeteilt wurde, lassen sie sich direkt senden. Andernfalls muss auf den richtigen Zeitpunkt gewartet werden. Das Aktivitätsdiagramm in Darstellung 25 fasst das angestrebte Verhalten noch einmal zusammen.

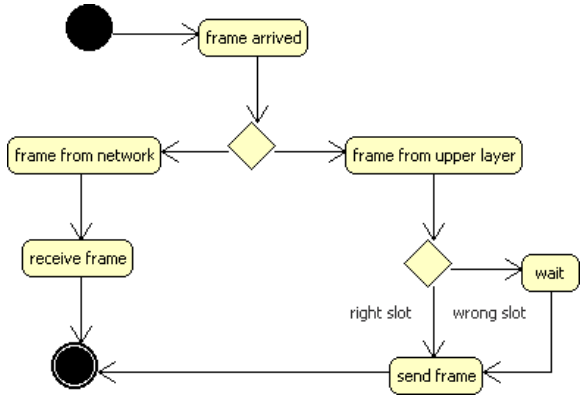

Abbildung 25: Aktivitätsdiagramm Echtzeitteilnehmer

Genau wie die Switche benötigen die Echtzeitteilnehmer ebenfalls Informationen über die Konfiguration des Netzwerkes. Für sie reicht allerdings die Zeitschlitzanzahl und der eigene Echtzeitslot aus. Genauere Informationen über die Implementierung dieses Mechanismus zur Parametrierbarkeit sind im nachfolgenden Kaptitel aufgeführt.

Echtzeit-Verkehrsgenerator Die vorhandenen Verkehrsgeneratoren innerhalb des INET-Frameworks stellen nur ein Client-Server Verkehrsmodell bereit. Für Modellierung des Echtzeitverkehrs wird aber ein Erzeuger-Verbraucher Modell benötigt. Der Verkehrsgenerator muss also ohne Unterbrechung Nachrichten zu einem oder mehreren vorgegebenen Zielen senden. Diese Nachrichten empfängt dort eine Senke, die sie gegebenenfalls weiterverarbeitet. Da sich diese Funktionalität durch einfache Modifikationen der vorhandenen INET-Infrastruktur erzeugen lassen, entfällt hier eine weitere Spezifikation des Verhaltens. Für die Eingabe der Ziele wird ebenso auf existierende Funktionen zurückgegriffen, die Syntax der Parameterübergabe ist in Kapitel 5.2.3 auf Seite 48 dargestellt.

5.2.6 Implementierung des Simulationsmodells

Die im vorherigen Kapitel dargelegten Entwurfsideen und Verhaltensvorstellungen sind nun an die vorgegebene Struktur des INET-Frameworks anzupassen. Zunächst wird die grobe Struktur der einzelnen

Module erläutert, anschließend folgt eine detaillierte Beschreibung des genauen Aufbau der wichtigsten Simulationselemente. Der Quellcode des erstellten Simulationsmodell ist der Arbeit als CD-ROM beigefügt.

Echtzeitswitch Die Abbildung 15 auf Seite 35 stellt den Aufbau eines Echtzeitswitches innerhalb des INET-Systems dar. Die genaue Modulbezeichnung innerhalb von INET ist „EtherSwitchTDMA“, das für Ethernet Switch mit TDMA[33] Verhalten steht. Er besteht, wie alle Ethernetswitche innerhalb des Frameworks, aus einer Relayunit und mehreren Modulen die zuständig für die Medium Access Control (Medienzugriffskontrolle) sind, den MACs. Die Defininition dieses Moduls, wie auch aller anderen INET-Komponenten erfolgt mit Hilfe der Sprache NED. Die MACs, angesiedelt auf der OSI-Schicht 2 [34], bestimmen, zu welcher Zeit ein Teilnehmer auf das physikalische Medium zugreifen darf und sorgen unter anderem für die Einhaltung des CSMA/CD Algorithmus[35]. Für den Echtzeitswitch lassen sich die die vorhandenen Standard-Ethernet-MACs nutzen, dadurch können die notwendigen Änderungen minimiert werden. Die Leistung kann durch die Implementierung eines verbesserten „cut-through-Mechanismus“, siehe auch Kapitel 2.4.8, noch erheblich gesteigert werden.

Die Relayunit sorgt für die eigentliche Funktionalität des Switches. Sie leitet die empfangenen Rahmen nach den ihr bekannten Adressen oder Regeln weiter und sorgt gegebenenfalls für eine Adressauflösung mit Hilfe eines Broadcast. Im Echtzeitfall stellt sie zudem die notwendige Funktionalität für die Einhaltung der Zeitschlitze bereit. Die Implementierung findet unter Berücksichtigung der vorhandenen Infrastruktur statt, durch den Einsatz von Vererbung und der Verwendung von polymorphen Funktionen konnte eine elegante Lösung gefunden werden. Das Klassendiagramm in Darstellung 26 stellt die sich ergebene Struktur im Kontext des Frameworks dar. Bei der Beschreibung im Fließtext wird aus Gründen der Übersichtlichkeit auf eine Darstellung der Funktionsparameter verzichtet.

Die Klasse `MACRelayUnitNPTDMA` stellt die für den Echtzeitbetrieb relevanten Funktionen bereit, das normale Ethernet-Verhalten ergibt sich wie schon angesprochen durch die Vererbung von `MACRelayUnitNP`. Zur Verdeutlichung werden nun die implementierten Funktionen genauer vorgestellt. Die Methode `initialize()` nimmt nach dem Start des Simulators notwendige Initialisierungen vor. Dazu gehört das Einlesen der zum Switch gehörenden Ablaufpläne aus der XML Spezifikation und die Zwischenspeicherung dieser Daten in einer leichter und schneller handhabbaren Struktur. Als eine solche Datenstruktur dient ein spezieller Vector, in dem Referenzen auf Objekte vom Typ `IOHolder` abgelegt sind. Weiter erfolgt die Initialisierung von notwendigen Zeitparametern, wie die Länge eines Zeitschlitzes, der Interframe Gap und der sogenannter „safetyInterval“, ein Sicherheitsabstand. Der Nutzer bekommt die Werte dieser Parameter zusätzlich angezeigt. Der Sicherheitsabstand ist notwendig, um Ungenauigkeiten bei den Berechnungen aufzufangen[36] und dient als Reserve aufgrund der Berücksichtigung des Interframe Gaps innerhalb der MACs und für die Verarbeitung der Rahmen innerhalb der Switche. Er soll mindestens genauso groß wie die Verarbeitungszeit der Switche zuzüglich des Interframe Gaps sein.

$$t_{safetyInterval} = t_{switchProcessing} + t_{IFG}$$

[33] Abkürzung für Time Division Multiple Access, auf deutsch Zeitmultiplex-Verfahren.
[34] Genauer teilen sie sich die Schicht 2 mit dem Logical Link Control.
[35] Der aber nur bei Halbduplex-Verbindungen genutzt wird.
[36] OMNeT++ benutzt für die Zeit Gleitkommawerte, dadurch treten unvermeidlich Berechnungsfehler auf.

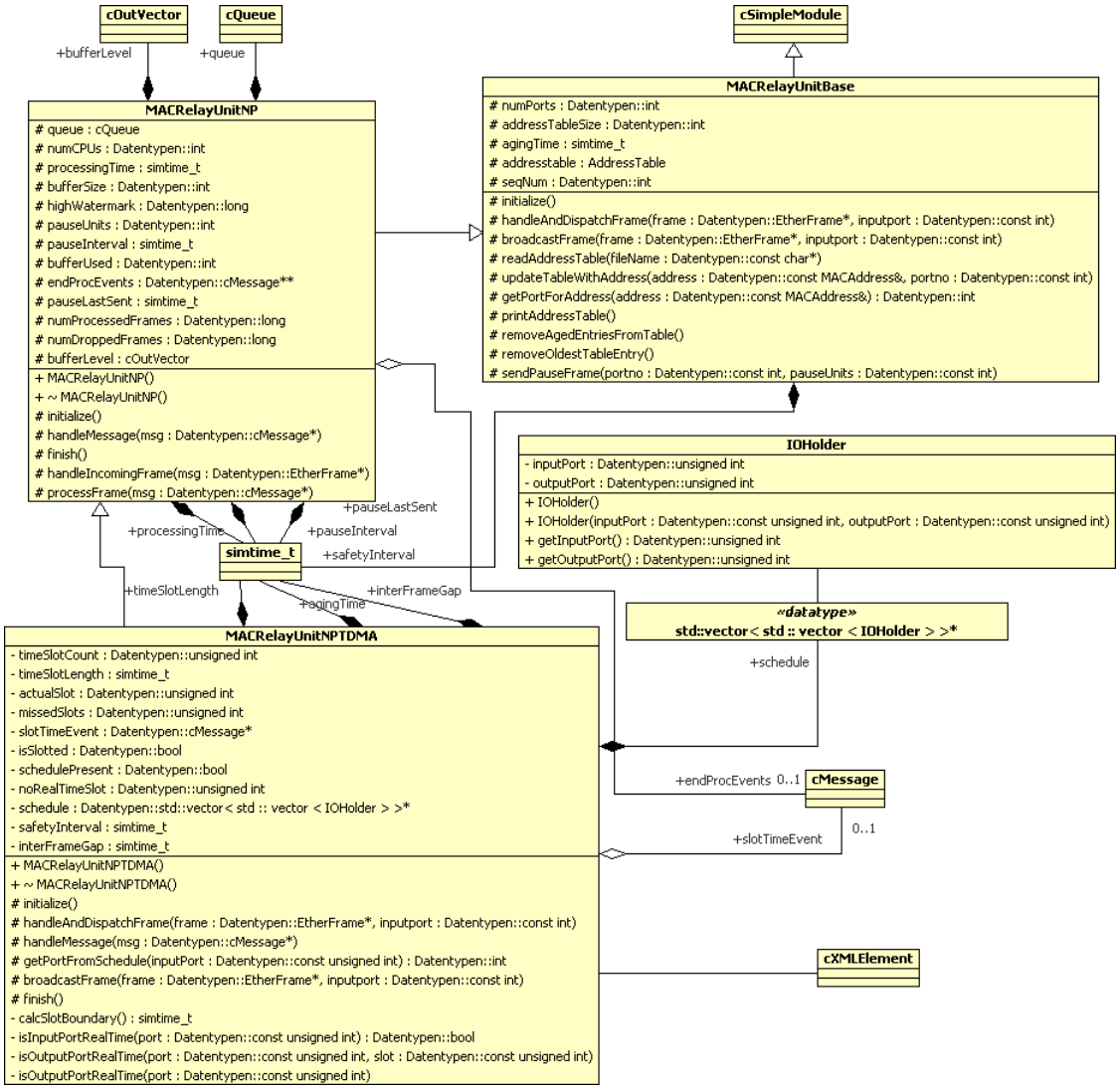

Abbildung 26: Klassendiagramm MACRelayUnitNPTDMA INET Framework

Im Abschnitt „Zeitschlitzaufbau“ auf Seite auf Seite 63 findet sich ein Beispiel für die Auslegung dieses Sicherheitsabstands. Abschließend wird das erste Zeitslot-Ereignis erzeugt, über diesem auf Ereignissen basierenden Mechanismus erfolgt später im Programmablauf die Inkrementierung des aktuellen Zeitschlitzes.

Die Basisklasse `MACRelayUnitNP` speichert empfangene und momentan nicht zu verarbeitene Rahmen in einer Queue. Bei Überlast sendet sie ein Pausesignal[37], um die angeschlossenen Teilnehmer zum drosseln des Datensendens zu veranlassen und verwirft gegebenenfalls auch Daten. `MACRelayUnitBase`, wiederum als Basisklasse, verwaltet als wichtigste Funktion die Adresstabelle des Switchs. Beide Klassen führen über ihre Aktivitäten Statistik und informieren den Benutzer über wichtige Ereignisse. `MACRelayUnitBase` erbt von `cSimpleModule`, einer Klasse in OMNeT++, die die grundlegenden Simulationsfunktionen bereitstellt.

Über die Hilfsfunktion `getPortFromSchedule()` lässt sich zu einem gegebenen Eingangsport der Ausgangsport aus dem Ablaufplan ermitteln. Die Methoden `isInputPortRealTime()` and `isOut-`

[37] Eigentlich ruft sie die `sendPauseFrame()` Methode in der Basisklasse auf.

`putPortRealTime()` ermöglichen die Überprüfung des Echtzeitstatus einer Eingangs- oder Ausgangsschnittstelle, um Fehler richtig einzuordnen und Echtzeitdaten von nicht Echtzeitrahmen zu unterscheiden. Die Funktion `finish()` speichert statistische Daten über den Simulationsablauf, unter anderem die Zahl der durch Fehler oder sonstige Probleme verpassten Zeitschlitze. Für die gewöhnliche Ethernet-Funktionalität stehen die Methoden von `MACRelayUnitNP` aufgrund der Vererbung zur Verfügung. Die zur genaueren Beschreibung der sonstigen Funktionen notwendigen Hintergrundinformationen sind in Abbildung 27 als Zustandsdiagramm dargestellt.

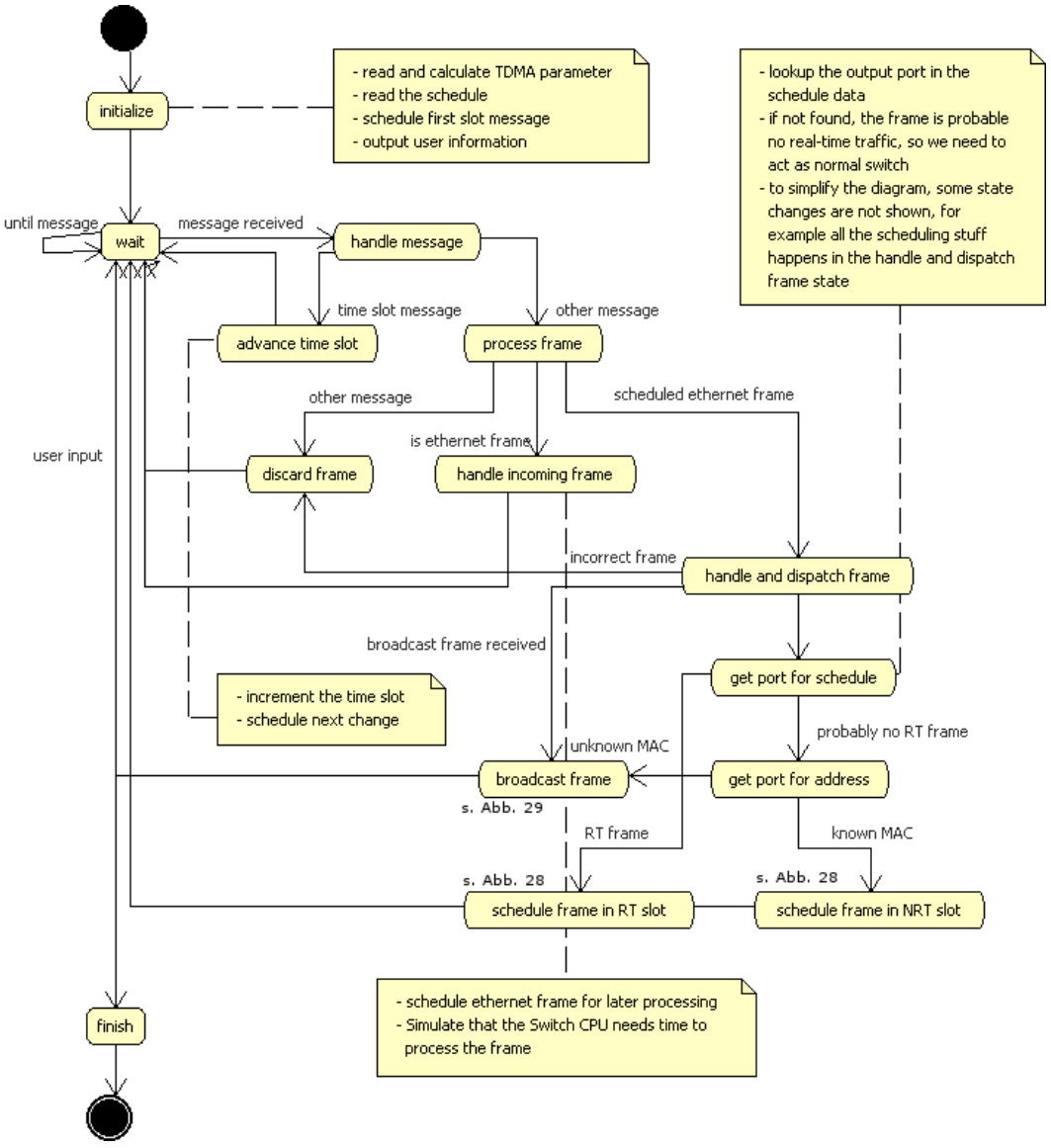

Abbildung 27: Zustandsdiagramm MACRelayUnitNPTDMA im INET Framework

In den MACs erfolgt zunächst eine Umwandlung der Rahmen in OMNeT++-Nachrichten, die die Relayunit empfängt. Die Verarbeitung dieser Messages findet in der Funktion `handleMessage()` in den Klassen `MACRelayUnitNP` und `MACRelayUnitNPTDMA` statt. Wenn es sich um eine Nachricht mit dem Typ „slotTimeEvent“ handelt, wird der aktuelle Zeitschlitz inkrementiert, und der nächste Slotwechsel vorbereitet. Weitere erlaubte Nachrichten sind gültige Ethernetframes, oder schon empfangene und zur späteren Verarbeitung vorgesehene Rahmen, andere werden verworfen.

Die weitere Verarbeitung der Daten findet in der Funktion `processFrame()` in `MACRelayUnitNP` statt. Erstmals ankommene Ethernetrahmen bekommen dort eine freie Switch-CPU zugeteilt bzw. müssen in einer Warteschlange auf Zuteilung warten. Nach dem Verstreichen der Verarbeitungszeit kommen sie wieder in der Funktion `handleMessage()` an. Nun lassen sie sich erneut über `prozessFrame()` an die Methode `handleAndDispatchFrame()` in der Klasse `MACRelayUnitNPTDMA` weiterleiten. Hier findet erstmals eine Einteilung der verschiedenen Daten in Echtzeit- und nicht Echtzeit-Anforderungen statt.

Echtzeitdaten können nun direkt weitergeleitet werden, nicht Echtzeitrahmen müssen auf eine Sendegelegenheit warten. Diese Priorisierung wird von der Switch-Relayunit vorgenommen. Broadcasts erfordern eine gesonderte Behandlung, wobei sie die gleiche Priorität wie nicht Echtzeitdaten erhalten. Da die Kompatiblität zu normalen Ethernet gewährleistet bleiben soll, sind sie unverzichtbar für die Auflösung von unbekannten Adressen[38]. Der Ausgangsport für Echtzeitdaten lässt sich über die Methode `getPortFromSchedule()` ermitteln. Bei nicht Echtzeitrahmen findet diese Funktion natürlich kein Ziel, dann erfolgt die Portauflösung über `getPortforAddress()` innerhalb der Klasse `MACRelayUnitBase`. Nicht bekannte Adressen generieren hier nun einen Broadcast, der in einem nicht Echtzeitslot gesendet wird.

Um eine gute Auslastung der einzelnen Zeitschlitze ohne zu viele Übertragungsfehler zu erreichen, ist eine robuste Ablauflogik erforderlich. Diese muss auf die einzelnen unterschiedlichen Zustände bei der Übertragung angemessen eingehen. In Abbildung 28 und 29 ist das implementierte Ablaufschema für die Zustellung von Echtzeit- und nicht Echtzeitdaten bzw. Broadcasts als Zustandsdiagramm dargestellt.

Für die Berechnung des Ausgangsports, der sogenannten „slot boundary“ und der aktuellen Zeit ist die Initialisierungsphase zuständig. Die Variable „slot boundary“ repräsentiert die Simulationszeit abgerundet auf den Anfang des jeweiligen Zeitschlitzs. Die Berechnung dieser Größe ist in der nachfolgenden Gleichung dargestellt.

$$t_{slot\,boundary} = t_{slot\,length} \cdot \lfloor \frac{t}{t_{slot\,length}} \rfloor$$

Mit Hilfe dieses Werts ist die Verzögerungszeit der Rahmen einfacher zu berechnen. Nach dem nun diese Berechnungen abgeschlossen sind, lassen sich die Daten entsprechend der aktuellen Zeit und dem Zielslot versenden.

Da die einzelnen Teilnehmer unsynchronisiert senden und auch die einzelnen Ablaufpläne der Switche nicht zu einander synchronisiert sind, muss nun eine Fallunterscheidung vorgenommen werden. Im besten Fall kommt der Echtzeitrahmen im richtigen Zeitschlitz an und er passt auch noch in die verbleibende Zeit, bevor der nächste Slot anfängt. Dann kann der Switch ihn direkt weitersenden. Falls die Daten nicht mehr passen, müssen sie einen vollen Zyklus warten. Die Verarbeitung von nicht Echtzeitframes erfolgt analog hierzu, wenn ihr Ausgangsport in dem aktuellen Slot frei ist. Falls die NRT-Daten[39] nicht mehr in den Zeitschlitz passen, oder sie innerhalb eines belegten Echtzeit-Zeitschlitzs ankommen, müssen sie bis zum nächsten freien Slot warten. Die Verarbeitung von Broadcasts, die die gleiche Priorität wie nicht Echtzeitdaten besitzen, erfolgt nach dem gleichen Prinzip. Allerdings wird der Datenversand hier immer

[38] Eine Alternative wäre natürlich die Verwendung von statischen MAC-Adresseinträgen in den Switchen.
[39] Eine Abkürzung für den Begriff „no real-time“.

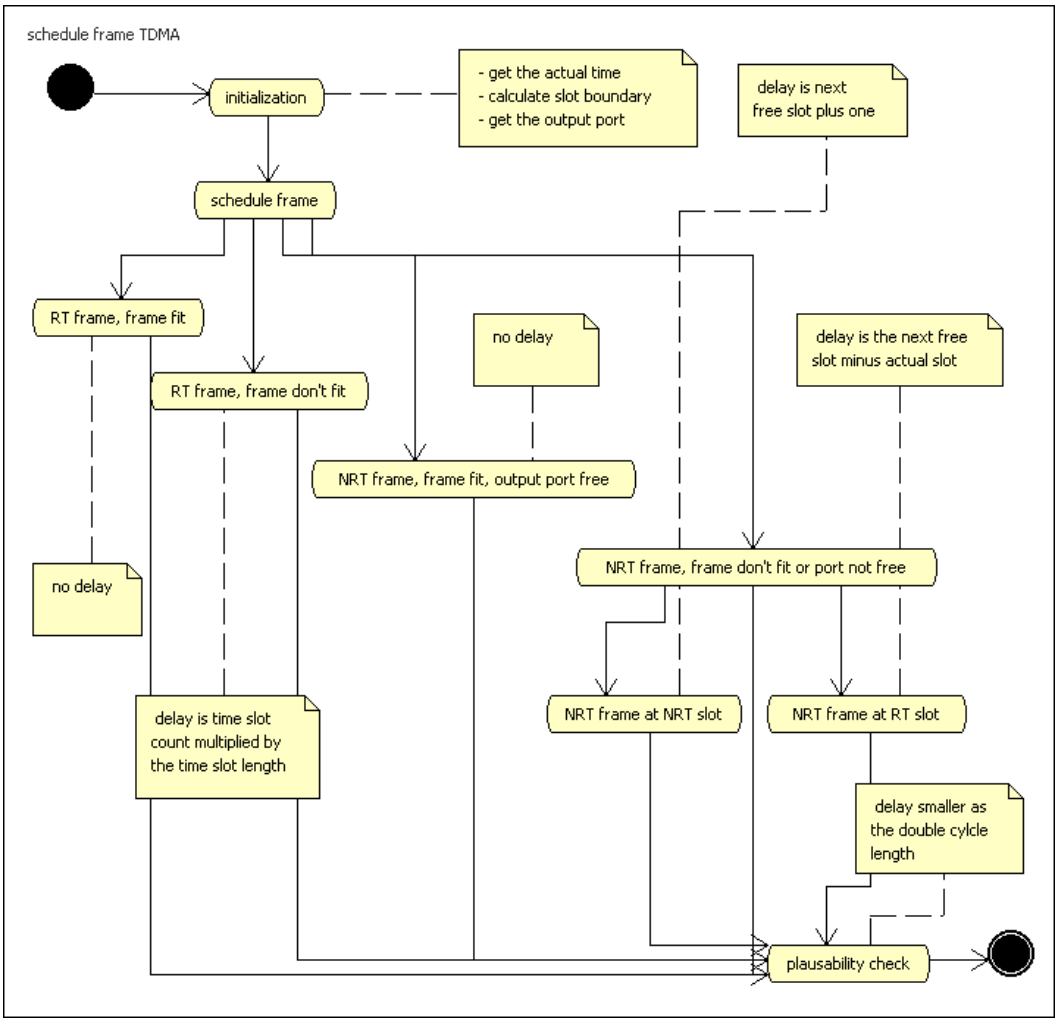

Abbildung 28: Zustandsdiagramm „schedule frame“ MACRelayUnitNPTDMA im INET Framework

innerhalb des NRT-Slots vorgenommen, da Broadcasts zu selten auftreten, so dass diese Optimierung wenig sinnvoll[40] ist.

Echtzeitteilnehmer Die Abbildung 16 auf Seite 35 zeigt den Aufbau eines Echtzeitteilnehmers innerhalb von INET. Er besteht aus dem MAC, einem Modul namens Logical Link Control, einem Verkehrsgenerator und einer entsprechenden Senke. Die LLC teilt sich die OSI-Schicht 2 mit der Medium Access Control und sorgt für eine Datensicherung auf der Verbindungsebene. Hier in diesem Einsatzfall wird diese Funktionalität aber nicht benötigt[41], sie sorgt nur für Anbindung, Multiplexing und Demultiplexing der Datenströme der Verkehrsgeneratoren. Der Generator und die Senke sind innerhalb INETs mit „srv“ und „cli“ bezeichnet, der MAC hat die Bezeichnung „EtherMACTDMA“.

Die Zeitmultiplex-Funktionalität des Echtzeitgerätes wird innerhalb des MACs verwirklicht. Dadurch sind keinerlei Änderungen in den anderen Komponenten des Teilnehmers vorzunehmen. Dies verringert den Implementierungsaufwand erheblich und steigert zudem die Stabilität und Wartbarkeit des Gesamtsystems, da dadurch eine zu starke Kopplung der einzelnen Module vermieden wird. Die Umsetzung des

[40] Die damit zu erreichenden Gewinne stehen in keinen Verhältnis zu dem Arbeitsaufwand.
[41] Sie ist auch nicht innerhalb des INET-Framework implementiert.

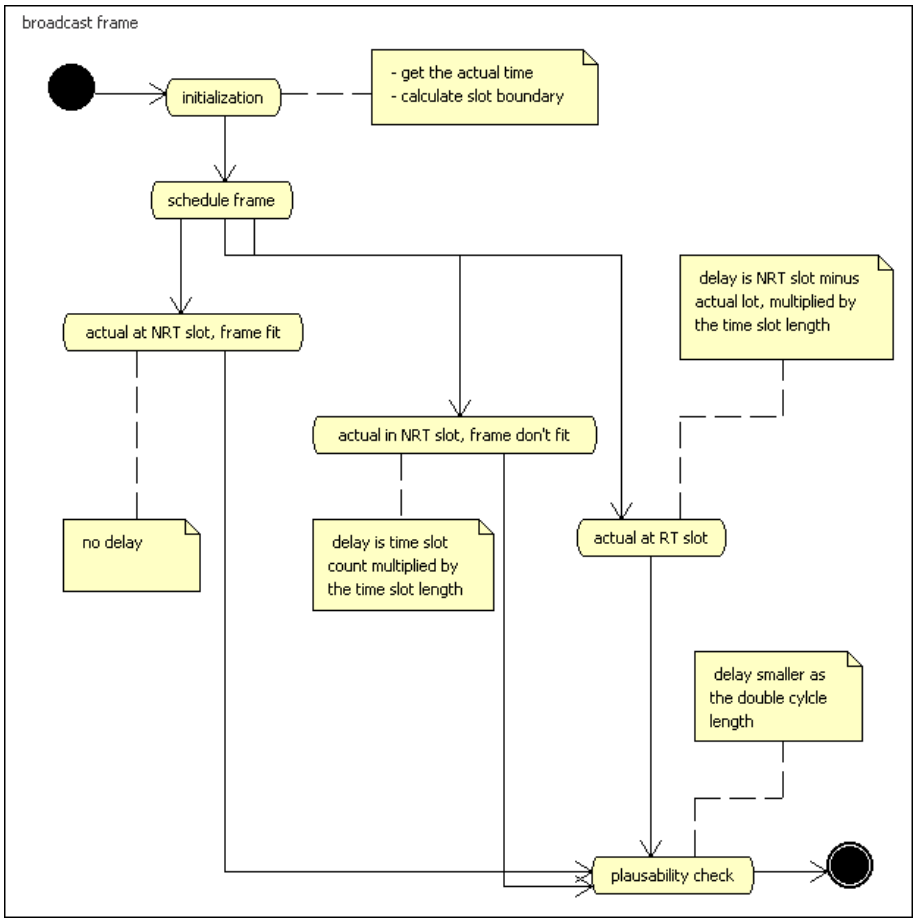

Abbildung 29: Detailansicht Zustandsdiagramm „broadcast frame“ MACRelayUnitNPTDMA

Zeitmultiplexverfahrens soll möglichst die Kompatibilität zu gewöhnlichen Ethernet nicht beeinträchtigen. Nach dem IEEE Standard 802.3 ist es erlaubt, den Interframe Gap, der zwischen zwei Rahmen liegen muss, länger auszulegen:

> „Note that interFrameSpacing is the minimum value of the interframe spacing. If necessary for implementation reasons, a transmitting sublayer may use a larger value with a resulting decrease in its throughput.“ [25]

Dieser Weg wird für die Implementierung gewählt, da er ausserdem die geringsten Änderungen an dem vorhandenen Simulationsmodell verspricht. Abbildung 30 stellt die Struktur des Echtzeit-MACs eingebettet in die vorhanden Klassen des INET-Frameworks als Klassendiagramm vor.

Die Klasse `EtherMACTDMA` stellt die Echtzeitfunktionen zur Verfügung. Sie erbt von `EtherMAC` alle benötigte Standard-Ethernetfunktionalität, nur die Methoden mit veränderten Funktionen werden überschrieben. Notwendige Initialisierungen nimmt die Methode `initialize()` vor. Dazu gehören das Einlesen des zugeteilten Zeitmultiplex-Slots und die Berechnung der schon bei der Beschreibung des Echtzeitswitches vorgestellten Parameter. Zusätzlich interessant ist die Variable „safety slot“, die auf ein tausendstel der Zeitschlitzlänge gesetzt wird und auf die berechneten Zeiten addiert wird.

$$t_{safetySlot} = \frac{t_{slot}}{1000}$$

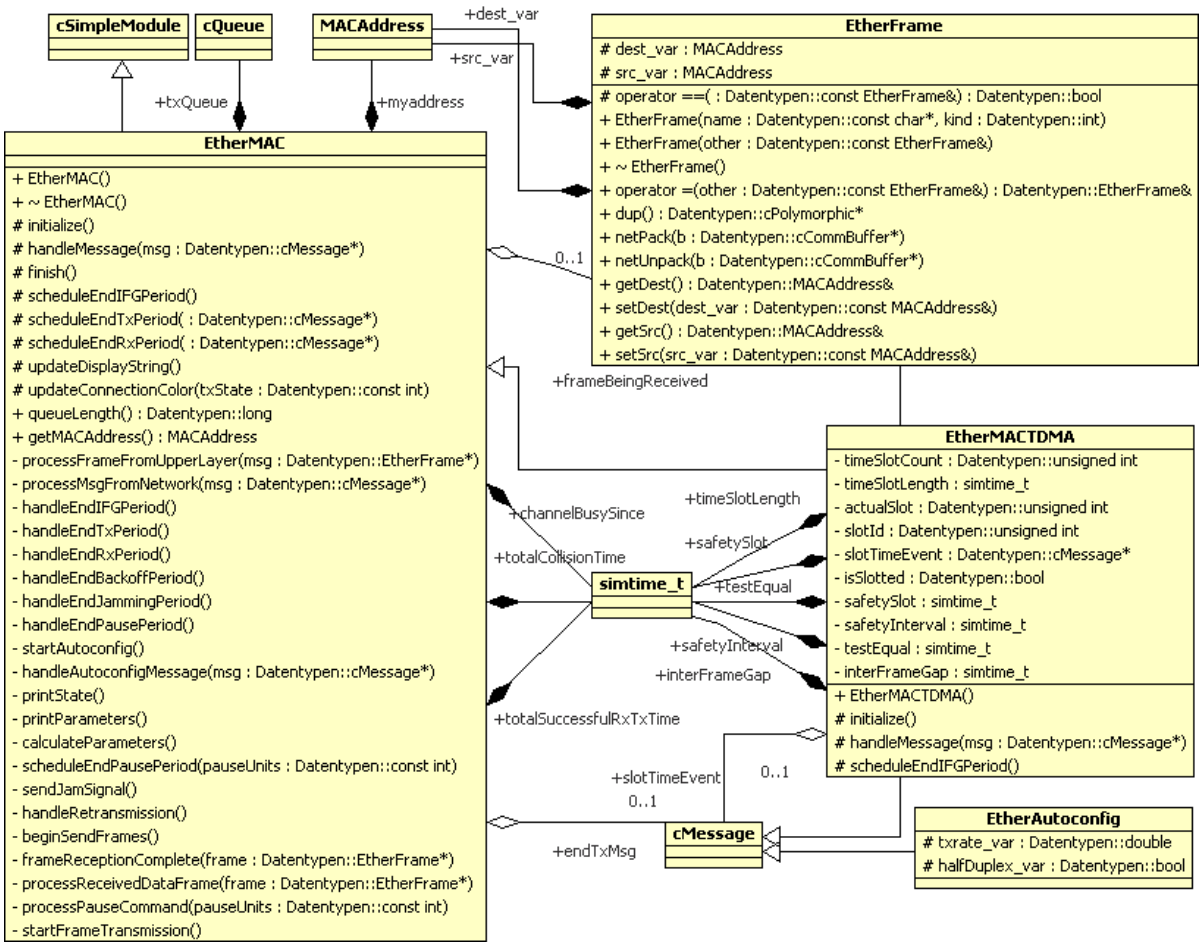

Abbildung 30: Klassendiagramm EtherMACTDMA im INET Framework

Hiermit lässt sich eine einwandfreie Funktion des Simulationssystems in bestimmten Grenzfällen[42] sicherstellen und die Logik des Echtzeitswitches lässt sich einfacher auslegen. Die Zeit „testEqual“ definiert den Wert, ab dem zwei Zeiten als gleich gelten. Diese Hilfsvariable ist notwendig, da das Simulationssystem intern mit Gleitkommawerten rechnet, die sich nicht sinnvoll vergleichen lassen. Beide Maßnahmen werden nur aufgrund von Unzulänglichkeiten bzw. den speziellen Anforderungen des Simulationssystems benötigt, in einem realen System können sie entfallen.

Die Basisklasse `EtherMAC` nutzt gleichermaßen eine Queue, um die Sendeanforderungen zu puffern. Als wichtige Funktionen lassen sich weiter die Autokonfiguration, über die die Ethernet-Autonegotiation[43] simuliert wird und die allgemeine Unterstützung von Sende-, Empfangs- und anderen Ethernetoperationen nennen. Sie hält eine Referenz zu der eigenen `MACAdresse`, und verarbeitet `EtherFrames`, die abgeleitet von der OMNeT++ Standard-Nachricht `cMessage` sind. Auch zur Umsetzung der Autokonfiguration und zum Signalisieren des Ende von Empfangs- und Sendevorgängen benutzt das Simulationsmodell gewöhnliche Nachrichten. Durch Vererbung von `cSimpleModule` erfolgt hier wie bei dem Echtzeitswitch die Anbindung zum Simulationssystem. Zur besseren Verdeutlichung des Zusammenspiels der einzelnen Klassen bezieht sich die folgende Erläuterung auf Abbildung 31, die ein Zustandsdiagramm des Verhaltens des RT-MACs zeigt.

[42] Falls keine CPU-Verarbeitungszeit des Echtzeitwitches angegeben ist.
[43] Ein Aushandeln der Geschwindigkeit der Netzwerkschnittstellen.

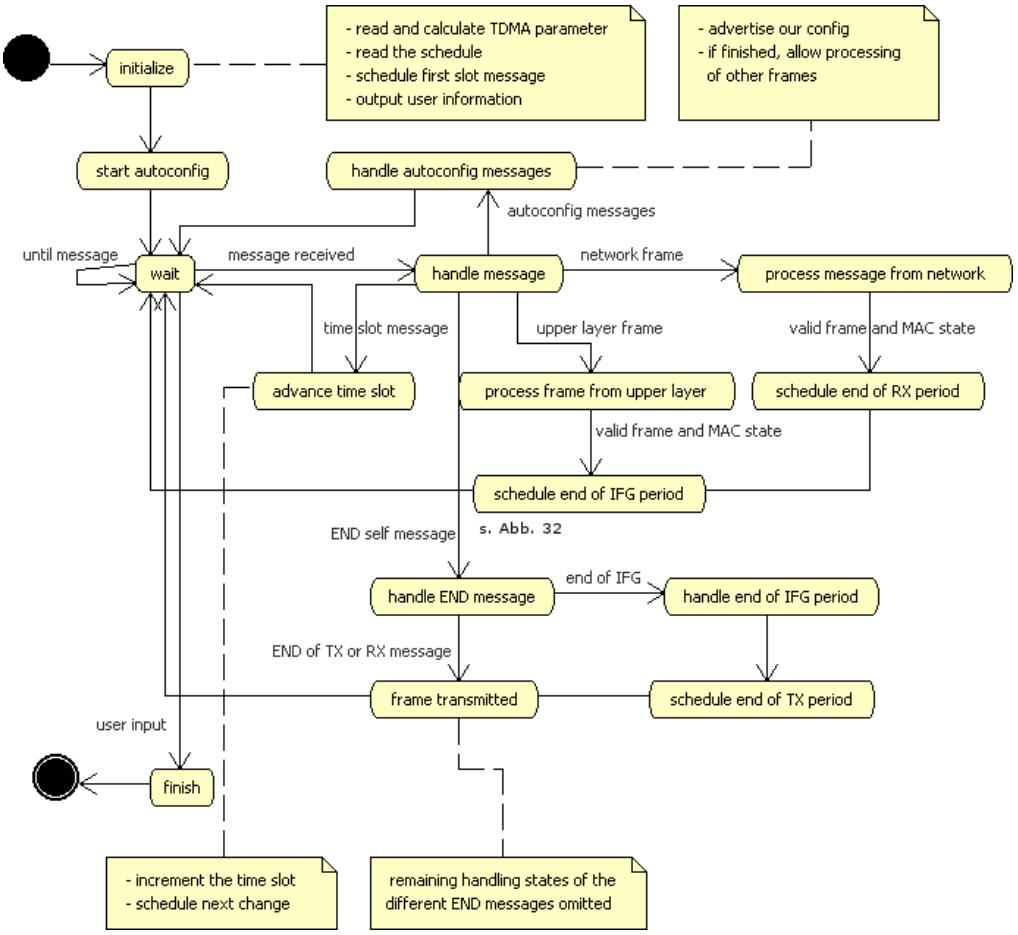

Abbildung 31: Zustandsdiagramm EtherMACTDMA im INET Framework

Nach dem Ablauf der oben beschriebenen Initialisierungsphase beginnt unmittelbar mit dem Start der Simulation die Autokonfiguration des Netzwerks. Dazu generiert jeder MAC Autoconfig-Nachrichten und empfängt im Gegenzug die von anderen Teilnehmern gesendeten Messages. Für die Verarbeitung dieser Konfigurationsnachrichten und auch aller anderer Nachrichten ist die Funktion `handleMessage()` zuständig. Nach dem die Autokonfiguration abgeschlossen ist, lässt diese Methode in `EtherMAC` auch andere Daten passieren. Die Inkrementierung der Zeitschlitze erfolgt analog wie im Echtzeitswitch über die Verarbeitung von Nachrichten in `EtherMACTDMA`. Die Verarbeitung von ankommenen Daten unterscheidet sich nach der Quelle der Daten. Rahmen von oberen Schichten müssen versendet, Ethernetframes vom Netzwerk empfangen werden. Für die Verarbeitung von Daten aus oberen Schichten ist die Methode `processFrameFromUpperLayer()`, für die aus dem Netzwerk empfangenen Rahmen die Funktion `processMsgFromNetwork()`, die sich beide in der Klasse `EtherMAC` befinden, zuständig. Beim Empfangsprozess ist zum Erreichen der Echtzeitfunktionalität keine Veränderung notwendig. Das Ende des Empfangsprozesses signalisiert ein Ereignis, das `scheduleEndRXPeriod()` erzeugt. Diese Ereignisse verarbeiteten dann die dazugehörigen Methoden, hier beispielsweise `handleEndRXPeriod()`. Diese einzelnen „Ende-Funktionen“ sind aber aus Gründen der Übersicht nicht im Zustandsdiagramm dargestellt.

Wie schon erläutert, modifizieren die MACs zum Durchsetzen des Zeitmultiplexverfahrens die Länge des Interframe Gaps. Die Methode `processFrameFromUpperLayer()` in `EtherMAC` ruft dazu

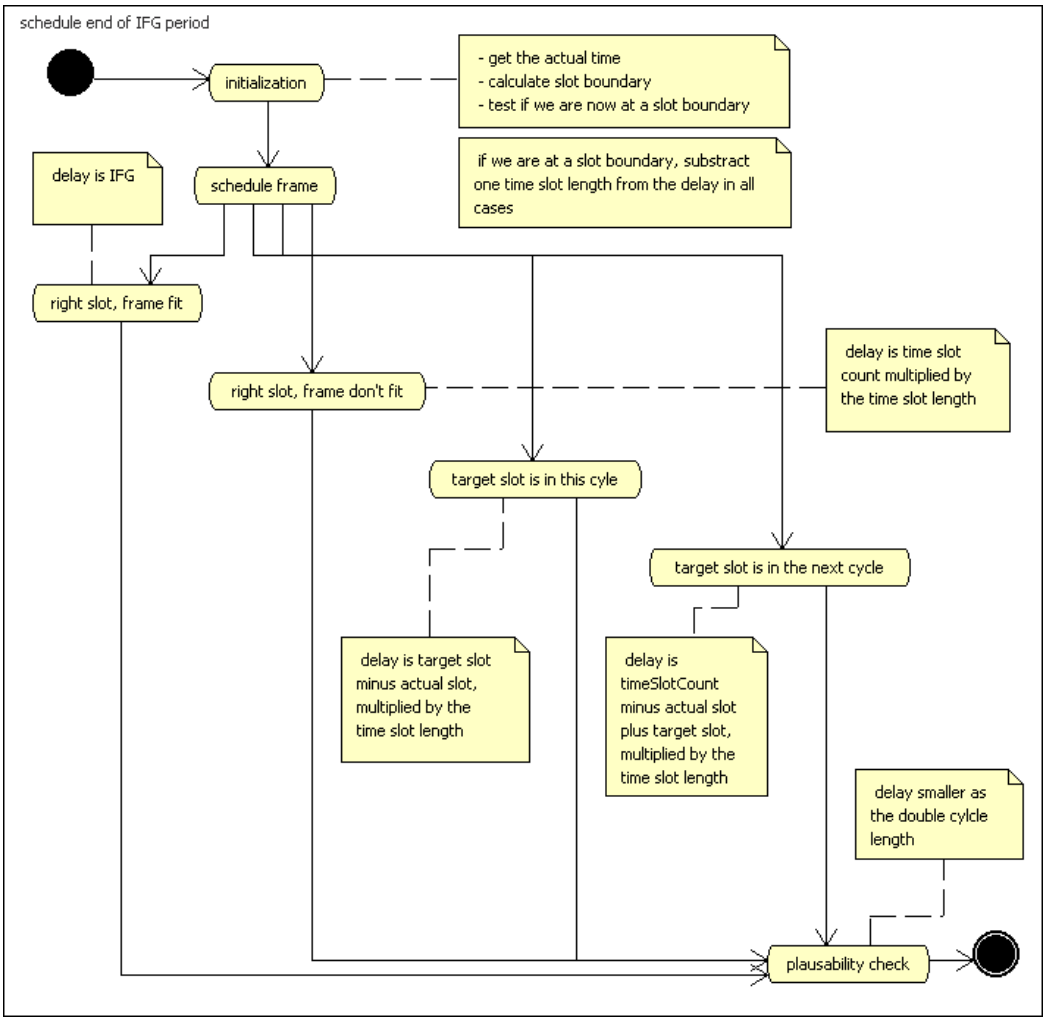

Abbildung 32: Detailansicht Zustandsdiagramm „schedule end IFG period“ EtherMACTDMA

die überladene Funktion `scheduleEndIFGPeriod()` in `EtherMACTDMA` auf. Die nach Ablauf des Interframe Gaps aufgerufene `handleEndIFGPeriod()` Prozedur lässt über `scheduleEndTXPeriod()` die Zeit für das ordnungsgemäße Senden des Rahmens berechnen. Nach Ablauf dieses Zeitintervalls übernimmt `handleEndTXPeriod()` den Abschluss des Sendevorgangs und aktualisiert ebenso wie die RX-Methode die interne Statistik.

In Abbildung 32 wird nun die interne Logik der Funktion vorgestellt, die für das Einhalten des zeitlichen Ablaufplans verantwortlich ist. Wenn ein Rahmen von einer höheren Schicht innerhalb des richtigen Zeitschlitzs ankommt, kann er direkt nach Verstreichen des Interframe Gaps versendet werden, sofern er noch in den verbleibenen Zeitinterval hinein passt. Trotz Verwendung des Zeitmultiplexverfahrens ist dieser Wert wichtig, um direkt nacheinanderfolgende Rahmen im Empfänger sicher zu unterscheiden. Falls das Ende des Ethernetframe ausserhalb des aktuellen Zeitschlitzes liegt, muss der MAC einen vollen Zyklus warten. In den beiden anderen zu behandelnen Fällen kommen die Daten entweder vor oder nach dem Ziel-Zeitschlitz an. Hier ist dann die Differenz der beiden Zeitschlitze oder fast ein ganzer Zyklus zu warten.

Aufgrund der Eigenschaften des Simulationssystem sind nun noch zwei Grenzfälle zu betrachten. Wie in Kapitel 4.2 dargestellt, können in OMNeT++ Ereignisse nur als diskrete Punkte auf der Zeitachse auf-

treten. Bei zeitgleich auftretenen Ereignissen kann der Simulator sie beliebig anordnen. OMNeT++ führt sie nach Erzeugungszeitpunkt geordnet aus. Deshalb kommt es manchmal vor, dass die Simulationszeit schon zum nächsten Zeitschlitz vorgerückt ist, das Signal, das den internen Slotzähler innerhalb des Moduls vorrückt, aber noch nicht empfangen worden ist. Dadurch erfolgt die Zustellung des Rahmens zu spät. Dies wird mit einer entsprechenden Abfrage abgefangen. Ein ähnliches Problem entsteht, wenn der Zielzeitschlitz in dem nächsten Zyklus liegt, und wiederum der Sendevorgang an einer Slotgrenze startet. Hier muss eine Slotlänge zu der Verzögerung des Rahmens dazu addiert werden.

Zeitschlitzaufbau Zusammenfassend ist hier nochmals der sich ergebene Zeitschlitzaufbau in Abbildung 33 dargestellt. In diesem Fall ist nun beispielhaft ein Zyklus bestehend aus drei Echtzeit- und einem nicht Echtzeitslot aufgeführt. Der NRT-Zeitschlitz muss aus Sicherheitsgründen immer vorhanden sein, damit jeder Switch nicht Echtzeitverkehr befördern kann, auch wenn der Zyklus voll ist.

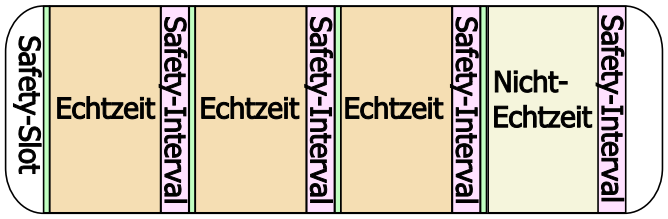

Abbildung 33: Zeitschlitzaufbau des Simulationsmodells

Bei einem Zeitschlitz von 100 μs und einer Verarbeitungszeit von 3 μs innerhalb des Switchs muss der safetySlot 100 ns, und der safetyInterval[44] 3,96 μs lang sein. Der effektiv bei diesem Fall für die Datenübertragung nutzbare Zeitschlitz ist dann:

$$\begin{aligned} t_{slotEffektiv} &= t_{slot} - t_{safetySlot} - t_{safetyInterval} \\ &= 95,94\,\mu s \end{aligned}$$

Die Gleichungen zur Berechnung des safetySlots und des safetyIntervals finden sich im Kapitel 5.2.6 auf Seite 54 bzw. Seite 60. Bei der Auslegung der Zeitschlitze und der Sicherheitsintervalle ist eine Abwägung zwischen der zu erreichenden Leistung und der gewünschten Sicherheit des Systems vorzunehmen. Ein großer Zeitschlitz ermöglicht eine sichere Übertragung von mehreren Rahmen, auch wenn die Teilnehmer nicht synchronisiert senden. Bei einem kleinen Zeitschlitz muss dieser genau getroffen werden, um eine ausreichende Leistung zu ermöglichen. Die Anforderungen an die Teilnehmer sind hier also höher.

Probleme bei der Entwicklung Bei der Umsetzung des Simulationsmodells treten zahlreiche Probleme auf, die im Vorfeld so nicht vorhergesehen werden konnten. Insbesondere die Entwicklung der Logik zur Berechnung der jeweiligen Verzögerungszeiten zum Einhalten der Zeitschlitze erfordert die Erstellung von zahlreichen Testfällen. Durch die Beschaffenheit des Frameworks entstehen zusätzliche Komplikationen. Einige Fehler treten teilweise nur sehr selten oder erst nach längerer Laufzeit auf. Die

[44] Bei einem IFG für ein 100 MBit/s Netzwerk.

große Anzahl von Ereignissen[45] und die vielen unterschiedlichen Zustände der Module innerhalb des Simulationssystems erschweren die Fehlersuche erheblich. Durch die Verwendung des INET-Frameworks treten sehr viel Ereignisse in nicht selbst erstellten Simulationselementen auf, diese müssen erst mit Diagnoseausgaben versehen werden.

Der komplette Arbeitsgang für die vollständige Simulation eines Modells umfasst mehrere verschiedene Schritte. So ist zunächst ein Netzwerk zu erzeugen und für dieses dann mit dem Werkzeug der Projektgruppe die zeitlichen Ablaufpläne zu berechnen. Weiter müssen dann diese Daten konvertiert werden; erst im letzten Schritt lässt sich die Simulation testen. Hier gefundene Fehler müssen auf die einzelnen Werkzeuge zurückgeführt und gegebenenfalls mit der Projektgruppe kommuniziert werden. Anschließend ist es notwendig, die Behebung des Fehlers durch Wiederholung aller Schritte zu verifizieren. Dieses Vorgehen erschwert und verzögert die Entwicklung der Simulation beträchtlich.

Durch Verwendung eines iterativen Ansatzes bei der Implementierung dieser Komponente lassen sich diese Probleme aber bewältigen. Zunächst konzentriert sich die Entwicklung auf ein einfaches Echtzeitnetzwerk mit nur wenigen Geräten, mit eingeschränkter Ausnutzung der Zeitschlitze. Als dieses dann größenteils fehlerfrei läuft, kommen nicht Echtzeitteilnehmer dazu. Nach Abschluss der Implementierung wird das Netzwerk vergrößert, die Zeitmultiplexparameter und die Einstellungen der Verkehrsgeneratoren angepasst. Parallel dazu lassen sich die Schwierigkeiten an den Zeitschlitzgrenzen nach und nach beheben, die Slots optimal ausnutzen und eine fehlerfreie Simulation erreichen.

5.3 Anmerkungen zur Simulation

Abschließend wird kurz auf die Beschränkungen der Simulation eingegangen und die für die Erstellung der Arbeit verwendete Arbeitsumgebung vorgestellt.

5.3.1 Einschränkungen der Simulation

Zunächst erfolgt eine Darstellung der Verbesserungsmöglichkeiten der Werkzeuge „createNetwork“ und „convertNetwork“, dieser Abschnitt schließt mit der Diskussion der Einschränkungen des erstellten Simulationsmodells. Gründe für die aufgeführten Begrenzungen sind zum einen die begrenzte Zeit die für die Entwicklung zur Verfügung stand, zum anderen Eigenschaften der verwendeten Werkzeuge bzw. des Simulationssystems. Die Behebung der hier aufgezeigten Defizite sollte aber problemlos möglich sein.

Einschränkungen von „createNetwork“ Die von „createNetwork“ erstellten Netzwerke besitzen aufgrund von Implementierungsentscheidungen immer eine linienähnliche Struktur. Um wirkliche Baumtopologien zu erstellen, ist deshalb Nachbearbeitung notwendig. Eine Ausnahme stellen die Grundformen Stern und Bus dar, deren Struktur keinerlei Nachbehandlung bedarf. Sonstige erzeugte Netze sind immer eine Mischform dieser beiden Topologien. Des Weiteren kommt es bei Verwendung eines hohen Wahrscheinlichkeitsfaktors für die Verbindungen zu Generierung von doppelten Verkehrszuteilungen[46]. Dies führt dann später im Simulationsmodell zu Problemen, die erstellten zeitlichen Ablaufpläne passen dann

[45] So erzeugt die Simulation von 1 ms in einem mittelgroßen Netzwerks mit 100 Teilnehmer über 11.000 Ereignisse.
[46] Eine doppelten Verkehrszuteilung liegt vor, wenn ein Verkehrsgenerator an mehrere Ziele senden soll.

nicht zu dem simulierten Netzwerk. Abhilfe lässt sich hier durch eine spätere Ausfilterung dieser Verbindungen schaffen. Letztlich bestehen bei der erstellten Benutzeroberfläche einige Defizite, insbesondere bei der Flexibilität der Parameterübergabe. Durch die Bereitstellung einer graphischen Oberfläche ließe sich das Werkzeug sicherlich angenehmer zu bedienen als durch das jetzt vorhandene textuelle Interface.

Einschränkungen von „convertNetwork“ Das Werkzeug „convertNetwork“ unterstützt nur eine Untermenge der von der Projektgruppe spezifizierten Netzwerkkomponenten. Zusätzlich zu den eingangs aufgeführten Ursachen begründet sich dies aus der zum Zeitpunkt der Werkzeugkonzeptionierung unvollständigen XML-Spezifikation der Projektgruppe. Die jetzige Benutzerschnittstelle informiert den Nutzer nicht ausführlich genug über eventuell bei der Datenverarbeitung aufgetretene Fehler. Die aktuelle Geschwindigkeit, insbesondere des für die Konvertierung der Switch-Ablaufpläne zuständigen Moduls, ist für die Verarbeitung von großen Netzwerken mit über 1000 Teilnehmern nicht optimal geeignet. Hier macht sich der große Aufwand bemerkbar, da die von der Projektgruppe bereitgestellten Daten erst nach umfangreichen Konvertierungen nutzbar sind. Die Umwandlung eines Netzwerks dieser Größenordnung benötigt etwa drei Minuten auf einem Rechner mit 1,6 GHz Taktfrequenz. Weiterhin lässt sich vermuten, dass aus der Implementierung des Kommandozeilenwerkzeugs „xsltproc“ resultierende Defizite ab dieser Größe der Eingabedaten zusätzlich zu den Problemen beitragen.

Einschränkungen der Simulationskomponenten Durch den Umfang der Implementierung ergeben sich bei den erstellten Komponenten der Simulation die meisten Verbesserungsmöglichkeiten. Zunächst unterstützen die Netzwerke nur den Vollduplex-Betrieb, die Verwendung von Hubs und sonstigen Halbduplex Teilnehmern ist also nicht möglich. Der erzeugte Echtzeitverkehr ist immer unidirektional, nicht Echtzeitkommunikation muss allein aus praktischen Gründen[47] bidirektional verlaufen. Die zeitlichen Ablaufpläne der Switches müssen im Halbduplex-Modus berechnet werden, Vollduplex Weiterleitung wird nicht unterstützt. Der Verkehrsgenerator für den RT-Betrieb unterstützt nur ein Ziel, die Konfiguration von mehreren Zielen führt zu Fehlern. Diese äußern sich in verpassten Zeitschlitzen aufgrund eines unangepassten zeitlichen Ablaufplans. Gleichermaßen können die Echtzeitgeräte nur einen Zeitschlitz nutzen, auch dies verhindert die Übertragung an mehrere Ziele. Die Fragmentierung von großen NRT-Rahmen, die nicht in den Zeitschlitz passen, wird nicht unterstützt.

Innerhalb der ersten Milisekunde der Simulationszeit blockiert die Autokonfiguration der Netzwerkkomponenten sämtliche Sendevorgänge. Da die Verkehrsgeneratoren so einfach wie möglich konzipiert werden sollten, wurde auf eine analoge Verzögerung des Sendevorgangs verzichtet. Die eventuell aufgefüllte Warteschlange des Geräte-MACs wird aber im Anschluss problemlos abgearbeitet. Durch die Verwendung von normalen MACs in den Switches kommt es bei hohen Füllständen der MAC-Warteschlange zu Fehlern, da Zeitschlitzgrenzen überschritten werden können. Dies rührt daher, dass diese Medienzugangsschichten nicht die Zeitmultiplex-Disziplin einhalten, sondern den Inhalt ihrer Warteschlange bei Gelegenheit[48] einfach senden. Dieses Problem entsteht aber nur bei sehr hoher Auslastung. Echtzeit-MACs wären an dieser Stelle schon aufgrund der Vorgaben der Projektgruppe nicht einsetzbar, weiterhin hätte sich der Aufwand der Implementation erheblich erhöht.

[47] Um den Switchen die Möglichkeit zu geben, die MAC-Adressen der Kommunikationspartner zu lernen.
[48] Ein einmal gestarteter Sendevorgang wird eventuell über Zeitschlitzgrenzen fortgesetzt.

Alle Netzwerkkomponenten sind durch Nutzung der Simulationsuhr immer miteinander synchronisiert, es ist also kein expliziter Synchronisationsmechanismus beispielsweise durch Verwendung des IEEE 1588 Protokolls notwendig. Der immer vorhandene nicht Echtzeitslot ist nicht getrennt von den Echtzeitzeitschlitzen parametrierbar, weiterhin ist im Moment nur die Unterstützung eines NRT-Slots möglich. Die Übertragungsverzögerung in Leitungen wird ebenso wie die Verarbeitungszeit innerhalb der Switche nicht explizit bei der Synchronisation der Teilnehmer berücksichtigt. Diese Werte finden im Moment ihre Berücksichtigung in dem Safety-Intervall. Eine Implementierung ohne diesen Sicherheitsabstand müsste eine Anpassung des zeitlichen Ablaufplans innerhalb jedes Netzwerkgeräts vornehmen, um dann gegebenenfalls „vor dem Takt“ senden zu können.

Die Weiterleitung von Broadcasts im nicht Echtzeitfall erfolgt nicht nach dem optimierten Algorithmus wie die Versendung von normalen NRT-Nachrichten. Die in [6] von Dopatka dargestellten Varianten des Echtzeitbetriebs mit Hilfe von Polling-Verfahren konnten ebenso wenig wie die Unterstützung eines speziellen, von der Projektgruppe skizzierten Echtzeitprotokoll umgesetzt werden. Die gewählte Variante verspricht aber gegenüber Polling die besere Performance. Größere Leistung ließe sich durch die Deaktivierung des IFGs in reinen Echtzeit-Netzwerken erreichen, auch diese Erweiterung ist noch nicht realisiert. Für die Modellierung des nicht Echtzeitverkehrs, der im Gegensatz zum RT-Fall nicht nach festen Regeln abläuft, wird auf die vom Simulationssystem angebotenen Wahrscheinlichkeitsfunktionen zurückgegriffen, es wurden keine eigenen Funktionen implementiert.

Die Geschwindkeit des Simulationssystems und speziell der Netzwerkvisualisierung sinkt bei großen Netzen mit über 1000 simulierten Geräten stark ab. Durch Deaktivierung der Darstellung lässt sich zwar noch eine Simulation vornehmen, durch den hohen Aufwand der ereignisbasierten Simulation verläuft die Berechnung dieser Netzwerke aber um Größenordnungen langsamer als bei kleinen Netzwerken[49]. Da aber gleichermaßen die Konvertierung des Netzwerkes und die Berechnung der Schedules ab dieser Größe erheblich länger dauert, fällt diese Einschränkung des Simulators nicht so sehr ins Gewicht.

Durch die Sende-Optimierung im nicht Echtzeitfall funktioniert die Fehlerberichterstattung nicht mehr einwandfrei. Beim Empfang von NRT-Daten im ersten Switch und im optimierten Weiterleitungsfall bei allen weiteren Switches werden Meldungen über verpasste Zeitschlitze protokolliert, die ungültig sind. Der reine Echtzeitbetrieb ist aber davon nicht beeinträchtigt. Diese Einschränkung ließe sich durch Nutzung eines eigenen Ether-Typs im Header der Echtzeit-Ethernetrahmen (wie bspw. bei PROFINET) beheben. Dann ließe sich auch ein für das REAL-System spezifische Protokoll entwerfen, mit dem eine sichere Trennung von Echtzeit- und nicht Echtzeit-Betrieb möglich wäre.

5.3.2 Einrichtung der Arbeitsumgebung

Alle die Diplomarbeit betreffenden Arbeiten wurden mit freier Software[50] erstellt. Als einzige Ausnahme ist OMNeT++ zu nennen, das lediglich frei verfügbar für akademische Zwecke ist. In der Tabelle 8 ist die verwendete Software mit der jeweiligen Versionsnummer aufgeführt.

Die unmittelbar für die Simulation benötigten Werkzeuge sind auch auf der CD enthalten, wie im Anhang in Abschnitt B.4 dargestellt. Für die komfortable und wiederholbare Installation der notwendigen

[49] Bei einem Rechner mit 1,6 GHz Taktfrequenz sinkt die Berechnungsgeschwindigkeit von 30 ms bis unter 0,5 ms-10 μs Simulationszeit/s .

[50] Frei im Sinne von OpenSource Software.

Name	Version	Kommentar
Linux OpenSUSE System	9.2, 10.0	Linux Betriebssystem
OMNeT++	3.2	Simulationssystem
INET Framework	20060330	Netzwerkprotokollunterstützung
GNU Compiler Collection	4.0	Kompiler für C und C++
KDevelop	3.3	Entwicklungsumgebung
Eclipse	3.2	Entwicklungsumgebung
Qt Toolkit	4.1	C++ Klassenbibliothek
libxml/ libxslt	2.6/ 1.1	XSL Verarbeitung
Subversion	1.3	Versionsverwaltung
LyX	1.4	LaTeX Textverarbeitungsprogramm
Umbrello	1.5	UML Editor
KOffice	1.6	KDE Office-Paket
TCL, TK, BLT	8.4, 2.4	TCL Skriptsprache und Erweiterungen
graphviz, giftrans, libjpeg, ImageMagick	-	Hilfsprogramme zur Grafikkonvertierung

Tabelle 8: Einrichtung der Arbeitsumgebung

Software wurden Skripte geschrieben, die das Entpacken, Kompilieren und Installieren vornehmen. Diese Hilfsprogramme (siehe auch Kapitel B.3), die in der „bash“ Skriptsprache verfasst sind, erleichterten die Installation der Werkzeuge, da insbesondere für die Kompilierung von OMNeT++ und LyX zahlreiche Bibliotheken als Abhängigkeiten erforderlich sind. Die manuelle Installation dieser Bibliotheken war notwendig, da auf dem gestellten Arbeitsplatz kein Administrationszugriff verfügbar, und zudem die verwendete Linuxdistribution veraltet war. Mit diesen Maßnahmen und der Verwendung einer Versionsverwaltung konnte eine optimale und einheitliche Umgebung an mehreren Arbeitsplätzen geschaffen werden. Für eine genaue Erläuterung des Installationsprozesses sei auf die mitgelieferte Dokumentation der einzelnen Programme bzw. Bibliotheken verwiesen.

Bei der verwendeten Hardware handelte es sich um einen AMD Athlon XP2000 (1,6 GHz Prozessortakt) und einen Intel P4 mit einer Taktfrequenz von 3,4 GHz. Beide Maschinen waren mit jeweils 1 GByte Arbeitsspeicher ausgestattet, was die Simulation des Netzes als auch die Auswertung der Daten, die im folgenden Kapitel dargestellt wird, erleichterte.

6 Durchführung der Simulation

Nach der ausführlichen Darstellung des entwickelten Systems folgt nun die Durchführung der Simulation. Dazu werden zunächst die unterschiedlichen Kriterien dargestellt, anhand derer die Szenarien konzeptioniert sind. Weiter werden allgemeine Simulationsparameter vorgestellt, und der Aufbau der einzelnen Szenarien genauer erläutert. Dieses Kapitel endet mit der Präsentation und Bewertung der Simulationsergebnisse.

6.1 Konzeption der Szenarien

Bevor die Simulation von komplizierteren Netzen vorgenommen werden kann, ist zunächst die allgemeine Funktionsfähigkeit des Modells zu überprüfen. Dazu ist ein kleineres Netzwerk besser geeignet, da sich so die zu betrachtenen Zustände des Simulationsmodells minimieren lassen. Eventuelle Unregelmäßigkeiten lassen sich hier leichter überprüfen und gefundene Fehler besser beheben. Anschließend erfolgt die Simulation eines mittelgroßen Netzes, dass sich näher an der Realität orientiert. Mit der Untersuchung eines großen Netzwerkes soll schließlich sich der Belastungsgrenze der erstellten Simulation angenähert werden.

Von den drei in Kapitel 5.1.2 dargestellten Kommunikationsmodellen ist zunächst nur die „eins zu eins“ Betriebsart interessant. Bei Netzen im Masterbetrieb besteht nur wenig Möglichkeit zur Parallelisierung, da alles Daten über eine Verbindung bzw. einen Switch übertragen werden müssen. Dies gilt in geringeren Maß auch für die Multi-Master Kommunikation. Da die Vorteile des entwickelten Systems insbesondere bei der besseren Auslastung der Netzwerkverbindungen gezeigt werden sollen, wird im Folgenden nur zufällig verteilter Verkehr simuliert.

Die Erstellung aller vorgestellten Szenarien erfolgt zunächst mit dem Werkzeug „createNetwork“. Diese werden dann nachbearbeitet, also doppelte Nachrichten entfernt, die Kommunikationsstruktur ausgeglichen und gegebenenfalls die Topologie angepasst. Anschließend müssen mit der Anwendung „schedulergui“ der Projektgruppe die zeitlichen Ablaufpläne zunächst erstellt und dann mit dem Werkzeug „convertNetwork“ konvertiert werden. Die Simulationsmodelle einschließlich aller notwendigen Parameter sind auf dem der Arbeit beigelegten Datenträger enthalten, siehe auch Kapitel B.5.

Auf die Beeinflussung von Simulationsparametern wie Taktsynchronität oder Sendezeitpunkte durch Jitter, beispielsweise nach einer Normalverteilung, wurde aus verschiedenen Gründen verzichtet. Diese Modulation hätte zusätzliche Rechenleistung benötigt und die Anpassung der Simulationsparameter nochmals erschwert. Weiterhin ist eine Betrachtung der Wahrscheinlichkeit von Fehlfunktionen anhand der Verteilungsfunktion und dem Erwartungswert auch ausserhalb des Simulationssystems möglich. Darauf wird im Rahmen dieser Arbeit aber verzichtet.

6.1.1 Allgemeine Simulationsparameter

Ein umfangreiches Simulationsframework wie INET besitzt natürlich zahlreiche Parameter, die den Ablauf der Simulation beeinflussen. Da diese für alle Szenarien gleich sind, werden sie hier an zentraler Stelle einmal eingeführt. Teilweise handelt es sich um vorgegebene Standardwerte des Frameworks,

andere Werte ergeben sich direkt aus den Implementierungsentscheidungen in Abschnitt „Zeitschlitzaufbau“ auf Seite 63. Die Tabelle 9 stellt die gewählten Einstellungen vor.

Parameter Netzwerkteilnehmer	Parameter Switch	Zeitmultiplex Parameter
Länge MAC Queue 1000	Buffergröße 1 MByte	Nachrichtenlänge 43, bzw. 1497 Bytes
	2 Switch-CPUs, Verarbeitungszeit 3 μs	Länge des safety interval 3,96 μs
normaler MAC Modus	cut-through MAC Modus	Länge des safety slot $\frac{slot\ time}{1000}$
promiscous deaktiviert	promiscous deaktiviert	
duplex modus und autoconfig aktiviert	duplex modus und autoconfig aktiviert	
	Kapazität Adressentabelle 1000, Gültigkeit 120 s	

Tabelle 9: Allgemeine Simulationsparameter

Die Netzwerkteilnehmer besitzen eine Eingangswarteschlange mit der Kapazität von 1000 Rahmen und unterstützen die Autokonfiguration der Geschwindigkeitseinstellungen. Sie arbeiten in der „normalen“ Betriebsart, unterstützen sowohl Halbduplex- als auch Vollduplex-Verbindungen und der Promiscous-Modus ist deaktiviert. Bei den Switchen ist der cut-through Modus aktiviert, ansonsten sind sie analog konfiguriert. Zusätzlich ist bei ihnen noch die Weiterleitung von Daten von Interesse. Innerhalb des Switches arbeiten zwei CPUs, die für die Auswertung eines Rahmen 3 μs benötigen. Die Verarbeitungszeit ist damit identisch zu PROFINET. Bei kurzzeitiger Überlast können Daten in einem 1 MByte großen Puffer zwischengespeichert werden. Insgesamt kann der Switch 1000 Adressen lernen; nach 120 s verlieren alte Einträge ihre Gültigkeit. Diese Parameter sind bei handelsüblichen Switchen gebräuchlich.

Die Nachrichtenlänge lässt sich direkt aus der minimalen bzw. maximalen Rahmengröße von Ethernet ableiten. Da für die Anbindung der Verkehrsgeneratoren an die MAC-Schicht LLC-Rahmen benötigt werden, die 17 Bytes lang sind, ergibt sich eine Nachrichtenlänge von 43 bzw. 1497 Bytes. Für die anderen Parameter sei auf das schon genannte Kapitel verwiesen. Die genaue Form der Konfigurationsdateien wird in Abschnitt A.4 vorgestellt.

6.1.2 Aufbau des Szenarios „kleines Netzwerk“

Bei diesem Szenario handelt es sich um ein einfaches Netzwerk, um die grundsätzliche Funktionalität der Simulation zu zeigen. Es besteht aus sieben Switches, an die insgesamt 15 Teilnehmer angeschlossen sind. Von diesen Geräten sind zwei gewöhnliche, also nicht echtzeitfähige Netzwerkteilnehmer, die durch Notebooks graphisch symbolisiert werden. Die Topologie des Netzes ist in Abbildung 34 dargestellt, Tabelle 10 stellt die Kommunikationsbeziehungen des Szenarios vor.

Sender	Empfänger
device_2	device_20
device_19	device_3
device_6	device_16
device_12	device_14
device_21	device_9

Tabelle 10: Kommunikationsbeziehungen Szenario „kleines Netzwerk“

Anhand dieses Netzwerks erfolgt nun die Untersuchung von verschiedenen Testläufen. Die allgemeine Funktionalität des Simulationsmodells wird anhand eines Betriebs des Netzwerks mit mittlerer Auslas-

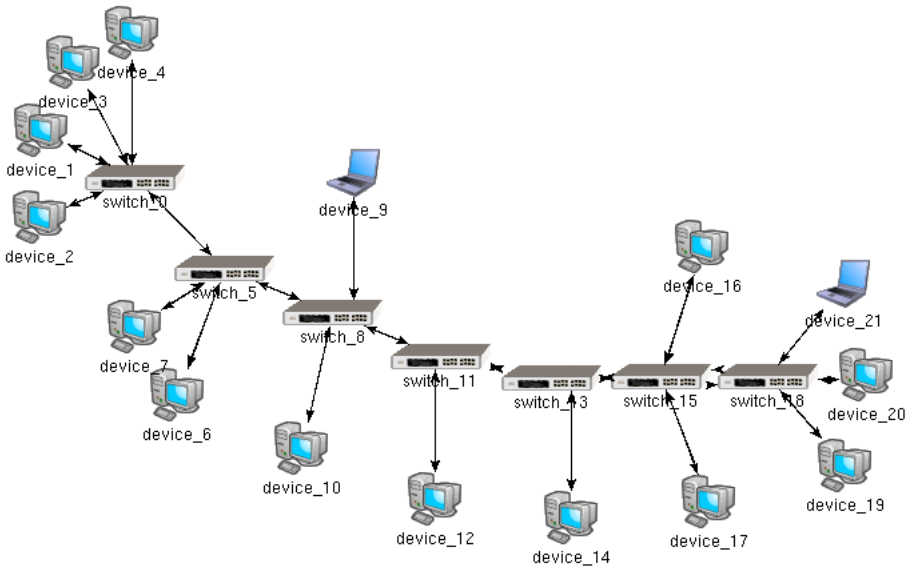

Abbildung 34: Darstellung Simulationmodell Szenario „kleines Netzwerk“

tung gezeigt. Dann erfolgt die Simulation eines Laufs mit der maximal möglichen Performance. Die letzten drei Testfälle analysieren den Betrieb mit normalen Ethernetrahmen der maximalen Länge und zwei Fehlerfälle. Es werden zwei unterschiedliche Fehlerklassen betrachtet, der Betrieb des Netzes mit einer zu geringen Zykluszeit und die Überlastung des Netzes durch eine zu kleine Nachrichtenperiode. Die gewählten Simulationsparameter für die gerade genannten Testläufe, die empirisch ermittelt wurden, stellt Tabelle 11 dar. Die Nachrichtenperiode bezeichnet die Dauer zwischen zwei Sendevorgängen der Verkehrsgeneratoren. Da dieses Netzwerk vier Echtzeit- und einen nicht Echtzeit-Zeitschlitz nutzt, beträgt die Zykluszeit der einzelnen Läufe das fünffache der Zeitschlitzlänge.

Testlauf 1	Testlauf 2	Testlauf 3	Testlauf 4	Testlauf 5
Zeitschlitzlänge 31 μs	Zeitschlitz 5,4 μs	Zeitschlitz 125 μs	Zeitschlitz 5 μs	Zeitschlitz 35 μs
Nachrichtenperiode 35 μs	Nachrichtenper. 27 μs	Nachrichtenper. 133 μs	Nachrichtenper. 36 μs	Nachrichtenper. 30 μs
Zykluszeit 155 μs	Zykluszeit 27μs	Zykluszeit 625 μs	Zykluszeit 25 μs	Zykluszeit 165μs
Mittler Betrieb	Maximale Performance	gemischer Betrieb	Zeitschlitz zu klein	Überlastung

Tabelle 11: Simulationsparameter Szenario „kleines Netzwerk“

Da pro Sekunde bei diesem Szenario über 110 Mb an Ergebnisdaten[51] anfallen, sich also längerer Testläufe weder sinnvoll auswerten[52], noch der Arbeit beifügen lassen, wird hier für die graphische Darstellung jeweils nur eine Simulation von 1 s vorgenommen. Die erzielten Daten wurden bei Bedarf aber durch längere Testläufe bestätigt, darauf wird gegebenenfalls im Text hingewiesen.

6.1.3 Aufbau des Szenarios „realer Betrieb“

Hier soll ein mittelgroßes Netzwerk mit ca. 100 Kommunikationspartnern simuliert werden. Der Aufbau soll sich zum einen an der Automatisierungspyramide in Kapitel 2.3.1 orientieren, um sich mehr der

[51] Dabei handelt es sich um Vektordaten, die die Erstellung von Graphen erlauben.

[52] Sobald die Eingabedaten die Größe des Arbeitsspeichers überschreiten, ist die graphische Auswertung nur noch sehr langsam möglich.

Realität anzunähern. Weiterhin ist eine verwertbare Leistungsbewertung für Abschnitt 3.4 zu finden. Auf den Einsatz von NRT-Teilnehmern wird aus Performancegründen und den Erfahrungen aus dem ersten Szenario verzichtet, deshalb erfolgt nur die Simulation eines Testlaufs.

Im unteren Teil des Netzes sind ca. 200 Teilnehmer in vier Gruppen angeordnet, die jeweils mit einem anderen Partner kommunizieren. Der Aufbau dieser Subnetze entspricht in etwa dem von Szenario „kleines Netzwerk“, sie sind also als Linie angeordnet. In der nächsten Ebenen fragt jeweils ein Rechner die Daten aus den untergeordneten Netzen ab. An oberster Stelle der Hirarchie werden die einzelnen Werte der vier mittleren „Steuerrechner“ in einem Gerät konzentriert. Die einzelnen Geräte sind dabei als Stern miteinander verbunden. Abbildung 35 zeigt den sich ergebenen Aufbau im Überblick.

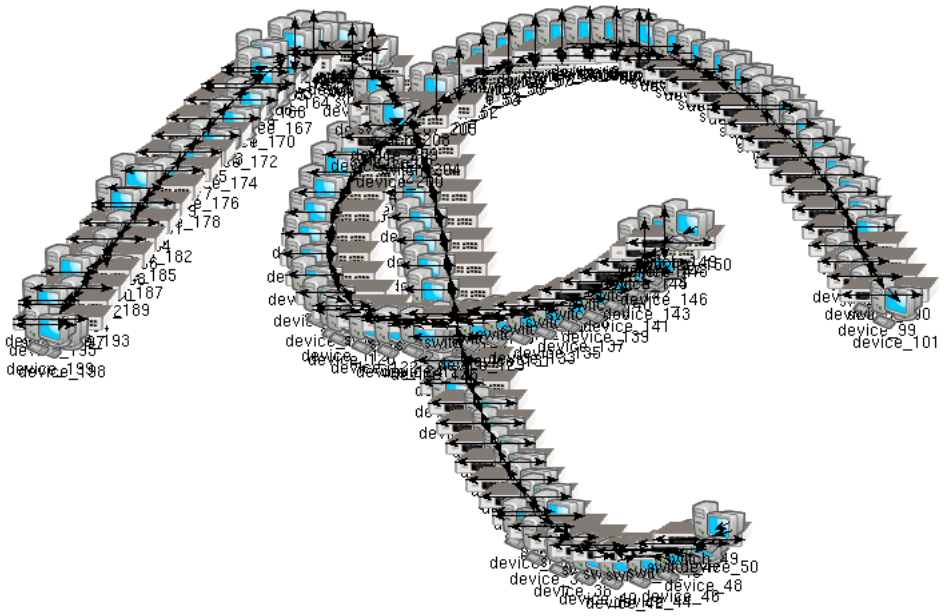

Abbildung 35: Darstellung Simulationsmodell Szenario „realer Betrieb“

Man erkennt deutlich die vier Linien, die den Hauptteil des Netzes bilden. Die zentralen Switches, die die Verbindung der einzelnen Stränge untereinander herstellen und die fünf an sie angeschlossenen Teilnehmer sind in der Mitte der Darstellung angeordnet. Diese sind aufgrund von Einschränkungen der Visualisierung nicht richtig zu erkennen. Deshalb ist in Abbildung 36 nochmals die entworfene Topologie vereinfacht dargestellt. Der zentrale Verbindungsswitch in der vereinfachten Darstellung ist in der Mitte der aus OMNeT++ entnommenden Abbildung angeordnet. Tabelle 12 stellt die Simulationsparameter dieses Szenarios dar. Bei der Auslegung der Zeitschlitzlänge wurde darauf geachtet die Zykluszeit klein zu halten, um eine verwertbare Leistungsaussage zu erhalten. Diese Netz nutzt insgesamt 35 Zeitschlitze.

Testlauf 1
Zeitschlitzlänge 30 μs
Nachrichtenperiode 750 μs
Zykluszeit 1,05 ms
reiner Echtzeit-Betrieb

Tabelle 12: Simulationsparameter Szenario „realer Betrieb“

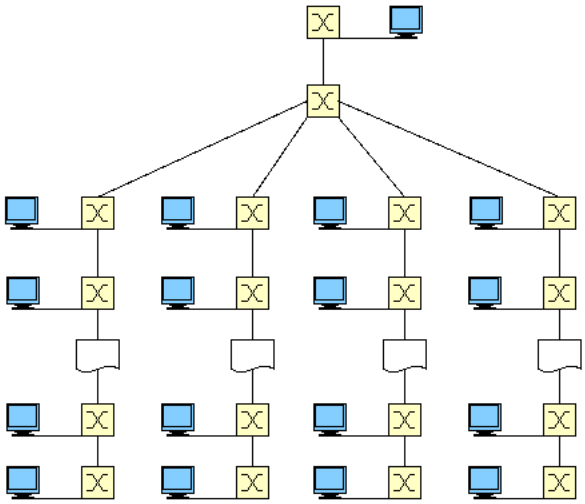

Abbildung 36: Vereinfachte Darstellung Simulationsmodell „realer Betrieb“

6.1.4 Aufbau des Szenarios „Lastgrenze“

Abschließend ist ein Netzwerk zu untersuchen, das die Belastungsgrenzen des erstellten Modells innerhalb des Simulationsframework zeigt. Das erzeugte Netzwerk mit 1000 Teilnehmern besteht aus mehreren sternförmigen Subnetzen die anhand einer Linienstruktur miteinander verbunden sind. Da es sich bei dieser Topologie annähernd um eine Linienstruktur handelt, wird hier aus Platzgründen auf eine vereinfachte Darstellung verzichtet. Abbildung 37 zeigt grob den Aufbau des Netzes, die relativ kleine und unübersichtliche Darstellung wird durch die hohe Anzahl der Teilnehmer verursacht.

Abbildung 37: Darstellung Simulationsmodell Szenario „Lastgrenze“

Bei der generierten Kommunikation handelt es sich um 164 Anforderungen, die zufällig verteilt sind, darin enthalten sind fünf NRT-Nachrichten pro Zyklus. Die Kommunikationsziele sind alle im ersten Drittel des Netzwerkes angeordnet, so dass sowohl lokaler als auch netzwerkübergreifender Verkehr

entsteht. Insgesamt enthält der zeitliche Ablaufplan 104 Echtzeit- und einen nicht Echtzeit-Zeitschlitz. In Tabelle 13 werden die Simulationsparameter aufgeführt.

Testlauf 1	Testlauf 2
Zeitschlitzlänge 50 μs	Zeitschlitzlänge 150 μs
Nachrichtenperiode 1 ms	Nachrichtenperiode 3 ms
Zykluszeit 5,25 ms	Zykluszeit 15,75 ms
nur Echtzeit	gemischter Betrieb

Tabelle 13: Simulationsparameter Szenario „Lastgrenze“

Bei diesem Szenario steht in erster Linie das Verhalten bei dieser hohen Auslastung im Vordergrund, insbesondere sollen im NRT-Fall die Auswirkungen der Broadcasts auf die Netzstabilität untersucht werden. Auch hier wurden die Simulationsparameter durch Versuche ermittelt. Auf die Darstellung der Kommunikationsbeziehungen wird hier aus Platzgründen verzichtet.

6.2 Simulationsergebnisse

Bei der Auswertung der Simulation erfolgt zunächst eine ausführliche Betrachtung der Hochlaufphase[53]. Dann wird der normale Betrieb eingehend untersucht und die korrekte Funktionsweise sowohl des normalen RT-, als auch der nicht Echtzeit-Betriebsart gezeigt. Weiter wird auf auftretene Fehler und deren Ursachen eingegangen. Die Untersuchung der einzelnen Szenarien erfolgt natürlich auch unter dem Gesichtpunkt der möglichen Leistung, um einen Vergleich zu den in Kapitel 3.2 vorgestellten RTE-Verfahren zu erlauben.

Bestimmte Werte, die typischerweise stark schwanken, so wie die Füllstände von Warteschlangen oder Zwischenspeicher, werden durch Bildung des arithmetischen Mittelwerts geglättet. Anderfalls wäre eine sinnvolle und ansprechende Präsentation der Ergebnisse nicht möglich gewesen. Die Rohdaten der Simulationen sind auf dem der Arbeit beigefügten Datenträger enthalten, wie in Abschnitt B.6 erläutert.

Die Auswertung wird mit Hilfe der von OMNeT++ zur Verfügung gestellten Hilfsmittel vorgenommen. Für die Erstellung von Graphen kommt „plove“ zum Einsatz, dass Vektordaten verarbeitet. Das Werkzeug „scalars“ hingegen erlaubt die Aufbereitung von Daten, die als skalare Werte vorliegen. Nahezu alle Ergebnisse lassen sich durch entsprechende Konfiguration des Simulationssystems gewinnen, in einigen Fällen mussten zusätzliche Möglichkeiten zur Datenerfassung geschaffen[54] werden.

6.2.1 Ergebnisse des Szenarios „kleines Netzwerk“

Testlauf 1 Zunächst ist der Füllstand der Sendewarteschlange in den Echtzeitgeräten von Interesse. Während der Autokonfiguration des Netzwerks ist keine Übertragung von Nachrichten erlaubt. Da die Verkehrsgeneratoren aber schon bei Start der Simulation zu senden beginnen, füllen sich der Puffer der Echtzeitgeräte-MACs. Nach dieser Phase, die nach 1 ms endet, wird diese Warteschlange dann abgearbeitet. Abbildung 38 stellt die Hochlaufphase als Auschnittsvergrößerung dar.

[53] Unter Hochlaufphase versteht man in diesem Kontext die Zeit, bis sich ein stabiler Betriebszustand eingestellt hat. Ein Teil dieser Zeit ist die Autokonfigurationsphase.

[54] Durch Änderung und Erweiterung des Simulationssystems.

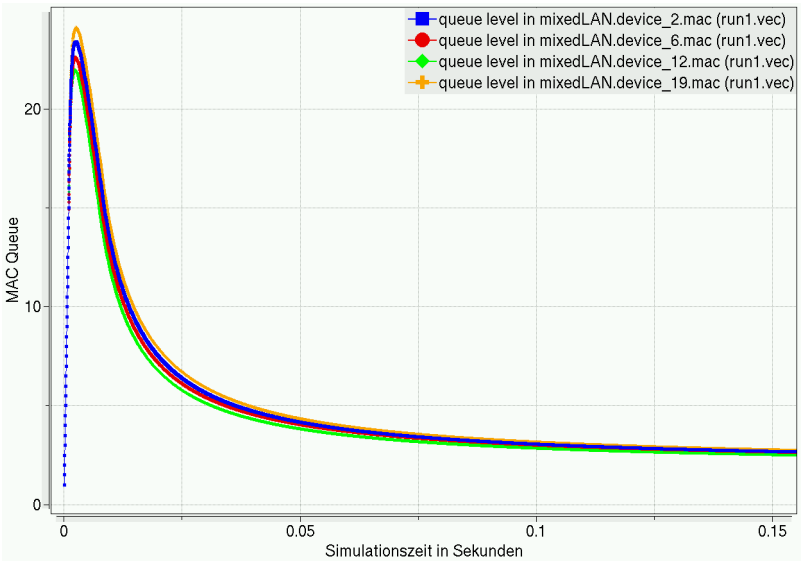

Abbildung 38: Szenario „kleines Netzwerk“, Testlauf 1, Rahmen in Warteschlange MAC Echtzeitgeräte

Der Füllstand der MACs steigt nach kurzer Zeit einen Wert von 22-24 Rahmen an und fällt dann ab 0,1 s auf 2,5-2,7 Rahmen ab. Nach einer Sekunde ist ein Wert von 1,98-2,1 Rahmen erreicht, der weiter abfällt, dieser Graph ist aus Platzgründen hier nicht dargestellt. In der Realität ist natürlich ein möglichst geringer Füllstand der Warteschlangen anzustreben. Dies ist hier in der Simulation aufgrund der unsynchronisierten Verkehrsgeneratoren nicht möglich. Für eine genauere Darstellung der sich durch die Verwendung der implementieren Simulationskomponenten ergebenden Einschränkungen sei auf den Abschnitt „Einschränkungen der Simulationskomponenten“ auf 65 verwiesen.

Analog zu den MACs der Echtzeitgeräte verhalten sich die Medienzugriffschichten der Switche. Die Länge der Warteschlangen steigt ab dem Ende der Autokonfiguration auf ca. 1,2 Rahmen an und wird dann langsam abgearbeitet. Nach einer Sekunden konvergieren die einzelnen Queues auf einen Wert von 0,9 Rahmen, der die optimale Auslastung des Gesamtsystems zeigt. Die mittlere Wartezeit eines Rahmens in den Switches beträgt also eine Zykluszeit. Dieser Hochlaufvorgang ist in Darstellung 39 abgebildet. MAC 0 von Switch 18 ist in der Legende verdeckt. Der MAC 1 und 3 des Switch 18, und MAC 2 des Switch 15 sind hier nicht dargestellt, da auf sie nur Störungen vom Hochlaufvorgang durchschlagen.

Interessant ist in diesem Graphen insbesondere das Verhalten von MAC 1 des Switchs 13, hier olivgrün dargestellt. Zur genaueren Untersuchung dieses Vorgangs wurde in Abbildung 40 nochmals eine Vergrößerung des interessanten Zeitraumes vorgenommen. Nach einem kurzzeitigen Maximum von 1,2 Rahmen sinkt die Warteschlangenlänge schnell auf 0,56 Rahmen ab, und nähert sich dann schließlich den anderen Werten an. Dieses Verhalten folgt daraus, dass an diesem Switch nur eine Datensenke angeschlossen ist, wie im Folgenden erläutert wird.

Zunächst wird die Warteschlange ausgehend von Switch 11 nach Ende der Autokonfiguration aufgefüllt. Solange der Puffer noch gut gefüllt ist, werden diese so schnell wie möglich versendet, auch teilweise unter Vernachlässigung der Zeitmultiplexdisziplin (s. auch Abschnitt „Einschränkungen der Simulationskomponenten“ auf S. 65). Während dieser Abarbeitung kommen nun nur noch wenige Daten an, und der Pufferlevel sinkt. Nach einiger Zeit verringert sich der mittlere Warteschlangenstand und es stellt sich ein ausgeglichener Zustand ein. Bei den anderen Switches, an denen auch Datenquellen vorhanden

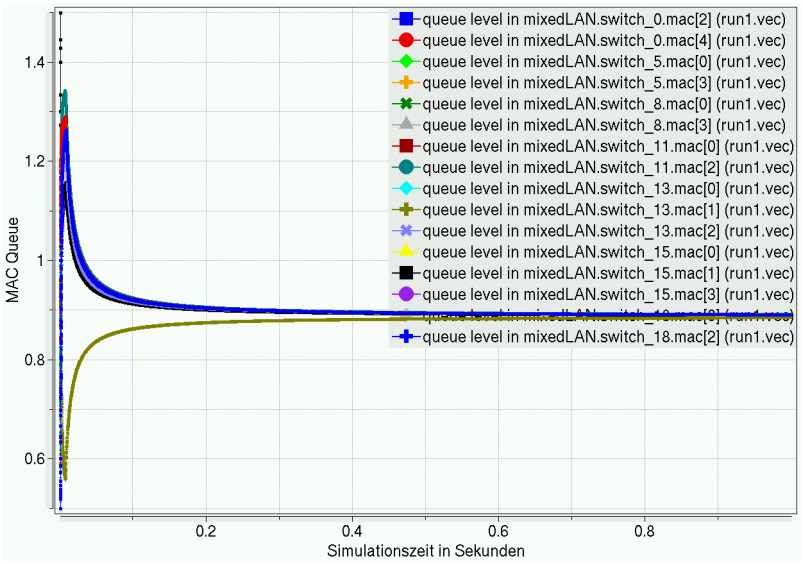

Abbildung 39: Szenario „kleines Netzwerk“, Testlauf 1, Rahmen in Warteschlange MAC Switches

sind, läuft derselbe Vorgang ab. Hier übertragen die MACs der Echtzeitgeräte anfangs die vorhandenen Daten direkt nacheinander, auch hier stellt sich nach einiger Zeit ein Gleichgewichtszustand ein.

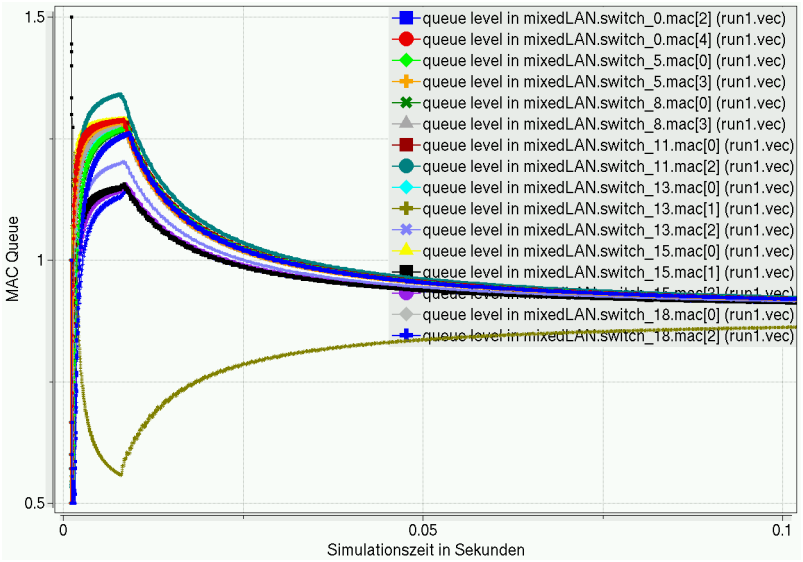

Abbildung 40: Szenario „kleines Netzwerk“, Testlauf 1, Detail Rahmen in Warteschlange MAC Switches

Das Ende des Start- oder Hochlaufvorgangs lässt sich gut an der Anzahl der verworfenen Rahmen in den Geräten ablesen. In Abbildung 41 sind die an falsche Ethernetgeräte zugestellten Rahmen aufgeführt. Die Hochlaufphase für diesen Testlauf dauert also ca. 8 ms, danach werden keine Rahmen mehr verworfen. Dies zeigt die einwandfreie Funktion des Echtzeitbetriebs, alle zeitlichen Grenzen und Ablaufpläne der Switches werden eingehalten. Da bei allen vier MACs die gleiche Anzahl an Fehlern auftreten, fallen die vier Kurven zusammen.

Nach Darstellung der Startphase wird nun die genaue Funktionsweise der Switches und Echtzeitgeräte untersucht. Dazu stellt Darstellung 42 die Anzahl der gesendeten Rahmen der Echtzeitgeräte und eines Switch vor. Bei diesem Graph wurde ein Vergrößerung bis auf Mikrosekundenebene vorgenommen, um

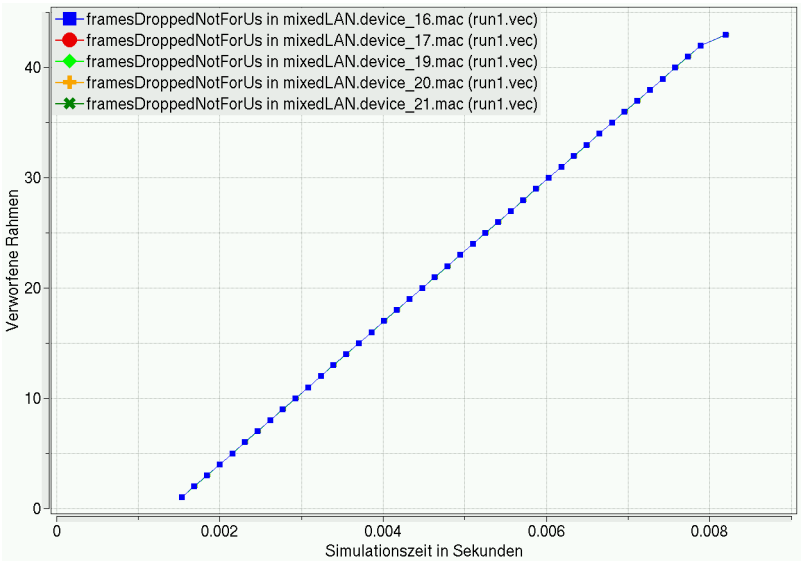

Abbildung 41: Szenario „kleines Netzwerk“, Testlauf 1, Verworfene Rahmen Hochlaufphase Echtzeitgeräte

die notwendige Genauigkeit zu erreichen. Wie in Abschnitt „Rahmenaufbau“ auf Seite auf Seite 12 dargestellt, beträgt die Sendedauer eines Rahmens der minimalen Länge 5,76 μs. Bei einer Zeitschlitzlänge von 31 μs passen dann vier oder fünf Rahmen in einen Slot, wie in der Grafik sichtbar ist. Sehr gut ist auch die Einteilung in verschiedene Zeitschlitze zu erkennen, die einzelnen Geräte senden jeweils abwechselnd.

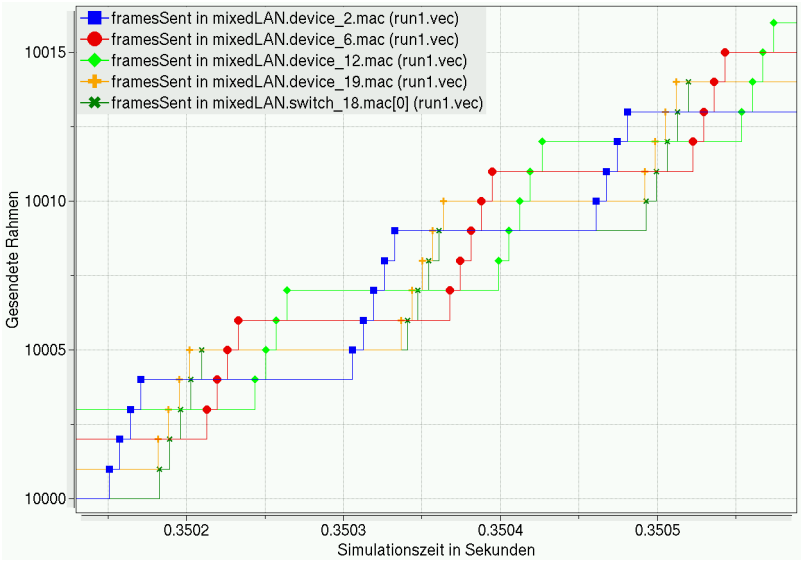

Abbildung 42: Szenario „kleines Netzwerk“, Testlauf 1, Gesendete Rahmen Echtzeitgeräte

Bei Device 19 wurde zusätzlich der Uplink-Switchport mit abgebildet (als orange und olivgrün in dem Graph dargestellt), um die Bearbeitungszeit eines Rahmens zu zeigen. Nach 3 μs ist der empfangene Rahmen im Switch ausgewertet, und wird nach Ablauf des IFGs mit 0,96 μs im gleichen Slot weitergesendet. Nach Slot 4, hier grün dargestellt, folgt jeweils eine kurze Pause, da bei diesem Testlauf kein nicht Echtzeitslot genutzt wird.

Auf eine Präsentation des Füllstand der Switch-Zwischenspeicher wurde verzichtet, da er über die gesamte Simulationsdauer die Hälfte eines Rahmens beträgt. Dies ist natürlich durch den Zeitmultiplex begründet, es werden einfach nicht mehr Daten an die Switches weitergeleitet. Verpasste Zeitschlitze treten bei diesem Lauf nicht auf, da der reine Echtzeitbetrieb nach Ende der Hochlaufphase fehlerfrei funktioniert. Diese Konfiguration wäre durchaus in der Praxis für die Steuerung einer kleinen automatisierten Anlage geeignet.

Die einzelnen Echtzeitgeräte übertragen ihre Daten jeweils mit einer Rate von 1,75 MByte/s, die Auslastung des Netzes beträgt also ca. 56%.

$$\begin{aligned} A &= \frac{4 \cdot 1,75 \frac{MByte}{s}}{12,5 \frac{MByte}{s}} \\ A &\approx 0,56 \end{aligned}$$

Die minimale Übertragungszeit einer Nachricht lässt sich anhand der Verzögerungen auf der Übertragungsstrecke innerhalb des Netzes berechnen. Für die längste Strecke von Gerät 2 nach 20 ergibt sich beispielsweise:

$$\begin{aligned} t_{transfer} &= (n_{CPU} + t_{CPU} + n_{MAC}) \cdot t_{IFG} \\ &= 7 \cdot 3\,\mu s + 8 \cdot 0,96\,\mu s \\ &= 28,68\,\mu s \end{aligned}$$

Dieser Minimalwert wird aber in der Simulation nicht erreicht, da die zeitliche Reserve zur Slotgrenze zu gering dimensioniert ist. Um diesen Wert zu erreichen müssten also die Zeitschlitze größer dimensioniert werden, um genügend Reserve für die maximale Übertragungszeit zu beeinhalten. Bei geringfügigen Verzögerungen in der MAC-Schicht, oder verzögertes Senden des Verkehrsgenerator muss eine komplette Zykluszeit gewartet werden. Da diese schon 155 μs beträgt, lässt sich eine mittlere Übertragungszeit von 185 μs als realistische Größe annehmen. Bei Einbeziehung der Verzögerungen der Verkehrsgeneratoren, die ja unabhängig von den Echtzeit-MACs ihre Daten erzeugen, ergibt sich im schlechtesten Fall eine Ende-zu-Ende Verzögerung von 340 μs. Diese Abschätzungen wurden durch Beobachtungen innerhalb der Simulation überprüft.

Testlauf 2 In Lauf 2 steht die maximale mögliche Leistung des Simulationssystem, ohne dass es zu einer Überlastung kommt, im Vordergrund. Sowohl die Zykluszeit, als auch die Sendeperiode der Verkehrsgeneratoren ist deshalb so gering wie möglich ausgelegt. Die minimale Zeitschlitzlänge ergibt sich direkt aus der Nachrichtenlänge, die Nachrichtenperiode ist von der Konfiguration des Netzes abhängig. Hier lässt sie sich direkt aus der Anzahl und Länge der Zeitschlitze ermitteln, wie nachfolgend gezeigt:

$$\begin{aligned} t_{min} &= (n_{RT} + n_{NRT}) \cdot t_{slot} \\ &= (4 + 1) \cdot 5,4\,\mu s \\ &= 27\,\mu s \end{aligned}$$

Bei den gewählten Simulationsparametern ist mit einer größeren Warteschlange in den Echtzeit-MACs zu rechnen, was auch in der nachfolgenden Abbildung zu beobachten ist. Nach Abbildung 43 wächst der Puffer nach dem Start bei RT-Geräten auf über 36 Rahmen. Man erkennt deutlich, dass die Puffer der Echtzeitteilnehmer-MACs sich an der Grenze zur Überlastung bewegen. Die Medienzugangsschicht aller Switches ist mit 0,8 bzw. 0,5 Rahmen wiederum wenig ausgelastet. Die Rahmen halten sich ca. 200 μs in der Teilnehmer-Warteschlange auf[55], bevor sie überhaupt übertragen werden können. Dieser Testlauf stellt also keine empfohlende Betriebsart des Echtzeitnetzwerkes dar.

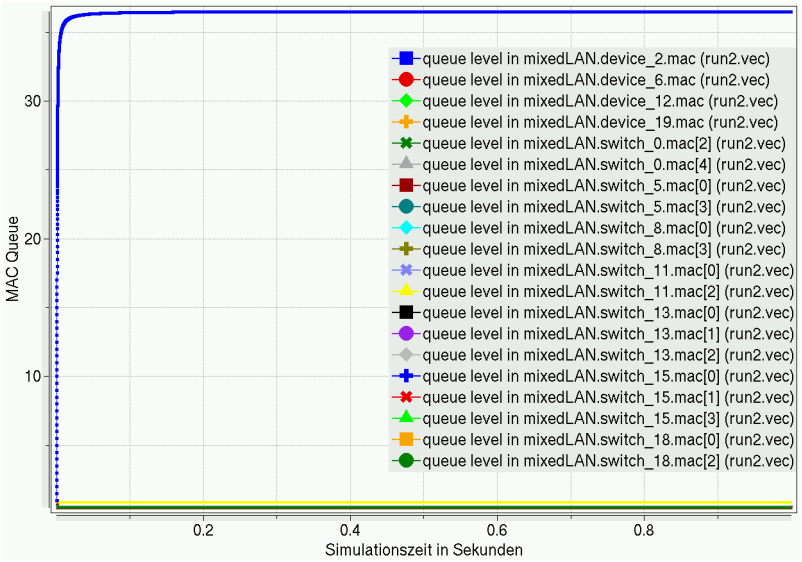

Abbildung 43: Szenario „kleines Netzwerk“, Testlauf 2, Rahmen in Warteschlange MAC Echtzeitgeräte und Switches

Wie in Darstellung 44 abbgebildet, passt nun nur jeweils ein Rahmen in einen Zeitschlitz. Diese Einstellung stellt also wirklich annähernd die maximale Leistung des Netzes dar. Eigentlich ist die Sendeperiode sogar einige Mikrosekunden zu klein, da die Warteschlangenlänge minimal innerhalb des Simulationszeitraums ansteigt. Die erreichte Genauigkeit soll aber für die Zwecke der Arbeit genügen, zumal eine solche Konfiguration in der Praxis ohnehin zu gefährlich wäre.

Zusätzlich ist auch hier wieder ein Switch-MAC zu Vergleichszwecken abgebildet. Der Übertragungsvorgang von MAC 0 des Switch 18 startet ca. 1 μs später als die des MACs 19, da er erst noch den IFG abwarten muss, bevor er senden darf. Im Unterschied zu Testlauf 1 ist hier der Zeitschlitz zu klein, als dass eintreffende Rahmen direkt bearbeitet und versendet werden könnten. Auch bei geringerer Auslastung benötigt die Weiterleitung eines Rahmens bei dieser Konfiguration also immer vollen Zyklus.

Testlauf 3 Nachdem die einwandfreie Funktion des Simulationssystems im RT-Betrieb gezeigt wurde, ist nun das Verhalten des Netzes mit hinzugefügten nicht Echtzeit-Teilnehmern interessant. Hier treten zum ersten Mal häufigere Broadcasts und verpasste Zeitschlitze auf, die durch die im Abschnitt „Einschränkungen des Simulationskomponenten“ auf Seite 65 dargestellten Probleme verursacht werden. Durch die Einführung des NRT[56]-Verkehr ändern sich die Eigenschaften des Gesamtsystems erheblich. Zunächst verlängert sich die Hochlaufphase, wie Abbildung 45 zeigt. Die auftretenen Schwankungen

[55] Genauer auf $36 \cdot 5,4\,\mu s = 194,4\,\mu$s .
[56] Eine Abkürzung für den Begriff „no real-time“.

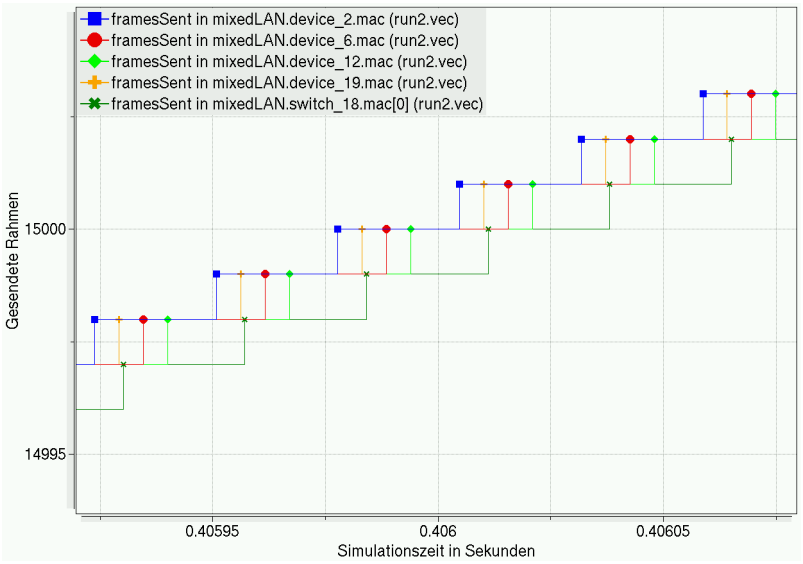

Abbildung 44: Szenario „kleines Netzwerk“, Testlauf 2, Gesendete Rahmen Echtzeitgeräte

klingen erst nach gut einer Sekunde vollständig ab, wie bei den um den Wert 1 angeordneten Kurvenzügen schön zu sehen ist.

Die Auslastung der einzelnen Switch-Warteschlangen ist, trotz einer relativ langen Sendeperiode, erheblich höher als in den vorherigen Testläufen. Dies wird durch die großen nicht Echtzeit-Nachrichten mit maximaler Ethernet-Rahmenlänge verursacht. Switch 0 ist teilweise, Switch 5 komplett aus Gründen der Übersichtlichkeit nicht dargestellt, sie besitzen einen Füllstand der Warteschlangen von 0,5 bis 0,7 Rahmen. Der MAC 0 von Switch 18 ist kurzzeitig am höchsten belastet, er stellt den Uplink zum nächsten Switch dar.

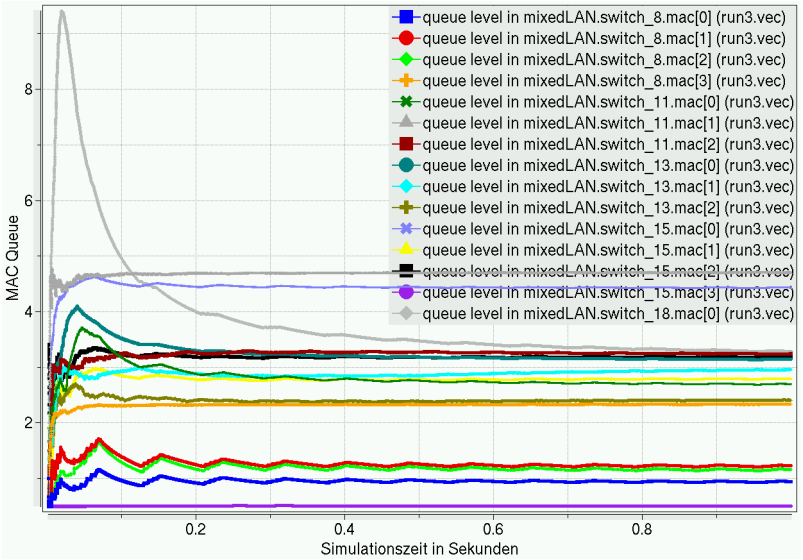

Abbildung 45: Szenario „kleines Netzwerk“, Testlauf 3, Rahmen in Warteschlange MAC Switches

Bei Verringerung der Sendeperiode der Verkehrsgeneratoren auf 130 μs vergrößert sich die Dynamik des Systems noch. Darstellung 46 zeigt den sich ergebenen Graph. Die Frequenz der Schwankungen ist mit Erhöhung der Sendefrequenz niedriger geworden, dafür treten größere Ausschläge auf. Diese

Simulationsparameter wären für die Praxis zu gefährlich, da der entstehende Jitter aufgrund starken Veränderungen der Warteschlangen zu stark schwanken würde. In der beschriebenen Abbildung ist MAC 0 des Switch 18 aus Darstellungsgründen[57] nicht enthalten, seine Warteschlange erreicht einen Spitzenwert von 14,5 Rahmen.

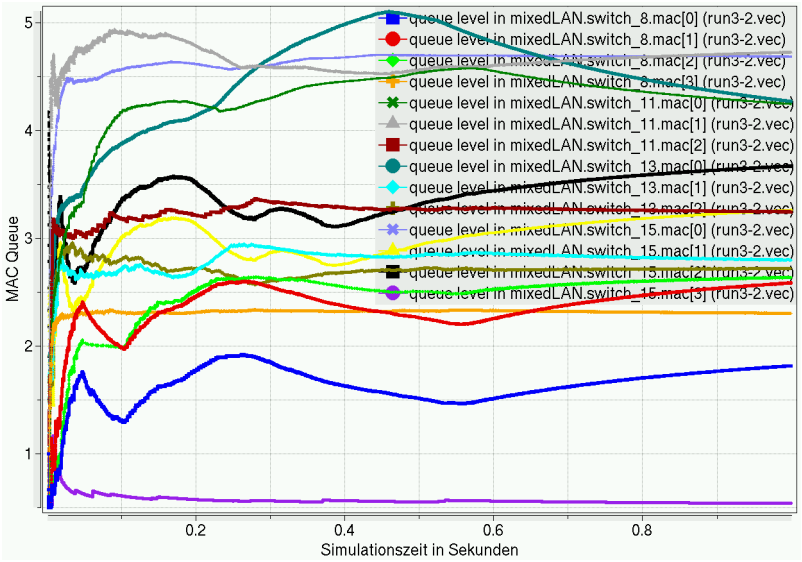

Abbildung 46: Testlauf „kleines Netzwerk“, Szenario 3, 130 μs Sendeperiode, Rahmen in Warteschlange MAC Switches

Bei dieser hohen Auslastung der MAC-Warteschlangen treten noch zahlreiche Fehler auf. So beträgt die Ende-zu-Ende Verzögerung der NRT-Teilnehmer 279 μs bzw. für die Gegenrichtung 679 μs. Dieser große Unterschied deutet auf häufige Verzögerungen durch Broadcast-Stürme hin. Einige MACs verwerfen auch wirklich innerhalb einer Sekunde ca. 15.000 Rahmen. Diese hohe Fehleranzahl werden durch die nicht richtig angepassten Switch-Medienzugriffsschichten verursacht, wie schon in Testlauf 1 erwähnt. Teilweise „rutschen“ nicht Echtzeitdaten auch in einen RT-Zeitschlitz, und werden zu einem falschen Ziel übertragen. Es ist also eine möglichst kleine MAC-Queue empfehlenswert, auch um die Nachrichtenlatenz gering zu halten.

Bei gemischten Betrieb mit einer vergrößerten Sendeperiode von 250 μs entstehen sehr viel weniger Broadcasts. Die Sendequeue der MACs ist mit 0,5 bis 1,3 Rahmen erheblich weniger ausgelastet. Mit diesem Parameter sinkt die Ende-zu-Ende Verzögerung auf 167 μs bzw. 223 μs. Die Graphen, die sich bei dieser Betriebsart ergeben, werden hier aus Platzgründen nicht dargestellt, die Rohdaten sind aber der Arbeit beigefügt. Durch eine Überarbeitung der Switch-Medienzugriffsschicht ließen sich die verbliebenen Fehler wahrscheinlich noch eliminieren.

Die einzelnen Echtzeitgeräte übertragen bei dieser Geschwindigkeit ihre Daten jeweils mit einer Rate von 0,47 MByte/s, die NRT-MACs mit 10,88 MByte/s bzw. 8,59 MByte/s in der Gegenrichtung. An den nicht Echtzeitgeräten kommen die Daten aber nur mit einer Rate von 8,98 MByte/s bzw. 5,14 MByte/s

[57] Auf die Verwendung eines logarithmischen Maßstabs wurde verzichtet, um die Vergleichbarkeit zu den anderen Graphen nicht zu beeinträchtigen.

an, da Fehler auftreten. Die sich daraus ergebene Auslastung des Netzes beträgt ca. 87 %.

$$\begin{aligned} A &= \frac{4 \cdot R_{RT} + R_{NRT}}{R_{Netz}} \\ &= \frac{4 \cdot 0,47 \frac{MByte}{s} + 8,98 \frac{MByte}{s}}{12,5 \frac{MByte}{s}} \\ A &\approx 0,87 \end{aligned}$$

Hier begrenzen wohl die Broadcasts die Leistung der Echtzeitgeräte, weshalb die Auslastung im gemischten Falls wohl geringfügig kleiner ist. Dies begründet sich aus denen in Abschnitt „Einschränkungen der Simulationskomponenten“ auf Seite 65 dokumentierten Einschränkungen. Dazu gehört beispielsweise die nicht nicht ausreichende Differenzierung der Echtzeit- von den nicht Echtzeit-Daten durch die fehlende Protokollunterstützung.

Abschließend wird der genaue Zeitschlitzaufbau des gemischten Betriebs in Abbildung 47 vorgestellt. Man erkennt deutlich den langen Sendevorgang der NRT-Rahmen und die vergleichsweise schnelle Übertragung der Echtzeitdaten. Weiterhin ist die Sendeoptimierung im nicht Echtzeitfall zu sehen, bei Switch 18 beispielsweise stehen einen Echtzeit-Zeitschlitz vier durch die NRT-Übertragung belegte Slots gegenüber. Switch 15 besitzt zwei durch RT-Verkehr belegte Slots, aber auch er nutzt die anderen für die NRT-Datenübertragung. Die nicht genutzten Resourcen des Echtzeitbetriebs stehen also in vollen Umfang für die Übermittlung der nicht Echtzeitdaten zur Verfügung.

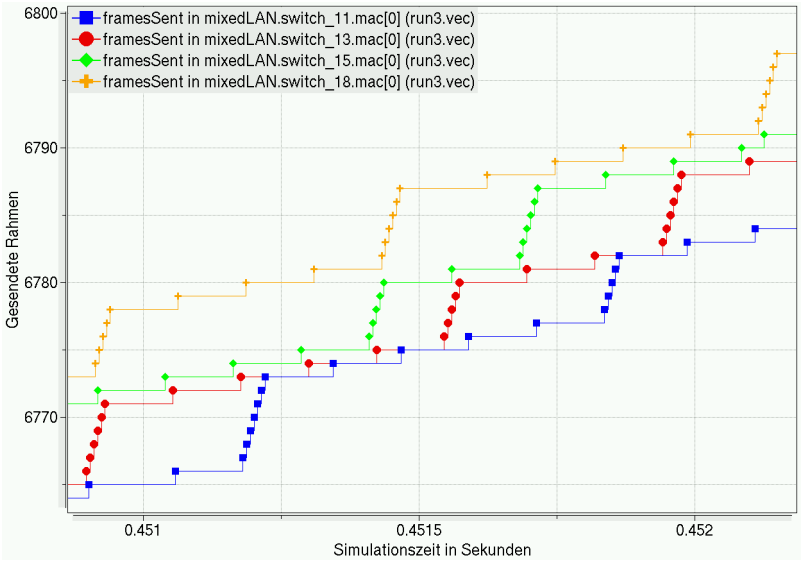

Abbildung 47: Szenario „kleines Netzwerk“, Testlauf 3, Gesendete Rahmen Switches

Testlauf 4 Hier ist das Verhalten bei zu klein gewählten Zeitschlitz zu betrachten. Da das Zeitmultiplex nicht mehr einwandfrei funktionieren kann, werden die Grenzen der einzelnen Zeitschlitze nicht eingehalten. Einige wenige Daten werden mit Hilfe von Broadcasts noch zugestellt, davon werden aber die meisten verworfen; eine funktionierende Datenübertragung ist nicht mehr gewährleistet. Der MAC des Teilnehmer 16 verwirft beispielsweise über 30.000 Rahmen pro Sekunde, Switch 15 meldet über 13.000 verpasste Zeitschlitze. Auf die Darstellung eines Graphen wird hier verzichtet, da es bei diesen einfachen Fehlerzustand nicht notwendig ist. Da Überschreitung der Zeitschlitzgrenzen vom System

nicht als gesonderter Fehler gemeldet wird und auch von den MACs der Switche keine Überprüfung der Zeitschlitzgröße vorgenommen wird, ist ein ausreichender Sicherheitsabstand einzuplanen.

Testlauf 5 Testlauf 5 untersucht die Auswirkungen einer Überlastsituation, wie sie bei der Wahl einer zu geringen Sendeperiode entsteht. Hier kommt es mit der gewählten Konfiguration nach ca. 0,2 s zu einem automatischen Abbruch der Simulation, da ein Pufferüberlauf der MAC-Warteschlangen auftritt. Da eine Darstellung des Kurvenverlauf keine neuen Erkenntnisse mit sich bringt, wird auch hier darauf verzichtet. In der Realität treten Überlastsituationen allerdings erheblich langsamer und eventuell unbemerkt auf. Aus diesem Grund sollte ein genügender Sicherheitsabstand vorgesehen werden.

Zusammenfassung Wie im ersten Testlauf gezeigt, funktioniert die erstellte Simulation im Echtzeitfall einwandfrei. Die maximale Leistung ist mit einer Zykluszeit und Sendeperiode von 27 μs nach Lauf 2 sehr gut. Aber auch eine mittlere Auslastung des Netzes erlaubt mit einer Zykluszeit von 155 μs und einer mittleren Verzögerung von ca. 200 μs einen Einsatz innerhalb der höchsten Echtzeitklasse nach IEC (s. auch Tabelle 2 auf Seite 4). Für einen Einsatz in der Praxis können hier natürlich trotz der vorgenommenen Simulationen keine Aussagen getroffen werden, auch der auftretene Jitter ist noch durch Versuche mit realer Hardware zu evaluieren.

Beim nicht Echtzeitbetrieb besteht noch einiger Verbesserungsbedarf, dies lässt sich aus den Ergebnissen von Testlauf 3 deutlich erkennen. Trotzdem wurde auch hier die sichere Funktionsweise des Systems dargestellt. Die Auslastung des Netzes ist mit 59% in RT-Fall, und mit 87% im NRT-Betrieb insbesondere im Vergleich zu einem reinen Polling Verfahren relativ gut. Auch hier besteht sicherlich noch die Möglichkeit zu Optimierungen.

Die Sendeperiode lässt sich, falls nur ein Rahmen in einen Zeitschlitz passt, direkt aus der Zykluszeit ableiten. Dies wurde bei Testlauf 2 dargestellt. Andernfalls ist die Zykluszeit durch die mittlere Rahmenanzahl pro Slot zu teilen, falls aufgrund der Netzkonfiguration mehrere Nachrichten der Verkehrsgeneratoren in einen Zeitschlitz passen. Für Testlauf 1 ergibt sich danach eine minimale Sendeperiode von:

$$\begin{aligned} t_{min} &= \frac{(n_{RT} + n_{NRT}) \cdot t_{slot}}{\frac{t_{slot}}{t_{Rahmen}+t_{IFG}}} \\ &= \frac{155\,\mu s}{\frac{31\,\mu s}{5{,}76\,\mu s+0{,}96\,\mu s}} \\ &= 33,6\,\mu s \end{aligned}$$

Der gewählte Parameter von 35 μs stellt aber eindeutig einen besser geeigneten Wert dar, auch die daraus nach obiger Gleichung sich ergebene mittlere Rahmenanzahl von ca. 4,4 erscheint nach Abgleich mit Abbildung 42 auf Seite 77 realistischer.

Abschließend wurde auf das Fehlerverhalten des Systems bei Nichteinhaltung des Zeitmultiplexes und Überlast kurz eingegangen. Um Fehler zu vermeiden und ein robusteres Systemverhalten zu erreichen, sind ausreichende zeitliche Sicherheitsabstände vorzusehen, auch bei der Auslastung des Netzes ist eine großzügige Reserve einzuplanen.

6.2.2 Ergebnisse des Szenarios „realer Betrieb“

Im letzten Szenario wurde die allgemeine Funktionsweise des Netzes ausführlich mit Hilfe von Graphen gezeigt. Für dieses Szenario wird darauf zugunsten einer Darstellung mittels Balkendiagrammen verzichtet. Von besonderen Interesse ist hier die Leistungsfähigkeit in einem typischen, an reale Automatisierungsnetze angelehnten Szenario. Zunächst betrachten wir die in der Mitte der einzelnen Linien angeordneten Switches, um den Verkehr dort zu untersuchen. Die hier auftretene Kommunikation ist in Abbildung 48 dargestellt. An die mittleren Switch-Ports ist jeweils ein Netzwerkteilnehmer angeschlossen, dies erkennt man auch an den geringen Datenübertragungsraten von nur ca. 83 Kbyte/s. An Port 1 des Switches 175 senden zwei Teilnehmer ihre Daten, deshalb ergibt sich hier die doppelte Rate. Der Port 0 stellt immer die Verbindung nach „oben“ dar, die letzte MAC-Schnittstelle ist die Anbindung an den unteren Teil der Linie. Bei Switch 125, Port 0 ergibt sich mit 0,97 MByte/s die höchste Datenrate, die anderen liegen im Bereich von 0,5 MByte/s bis 0,75 MByte/s. Die Füllstände der MAC-Warteschlangen der Echtzeitteilnehmer betragen nach 1 Sekunde ca. 0,8 Rahmen, die der Switches liegen in dem Bereich von 0,5 bis 0,8 Rahmen. Damit liegt auch hier die mittlere Wartezeit eines Rahmens in den Echtzeitgeräten unter einer Zykluszeit.

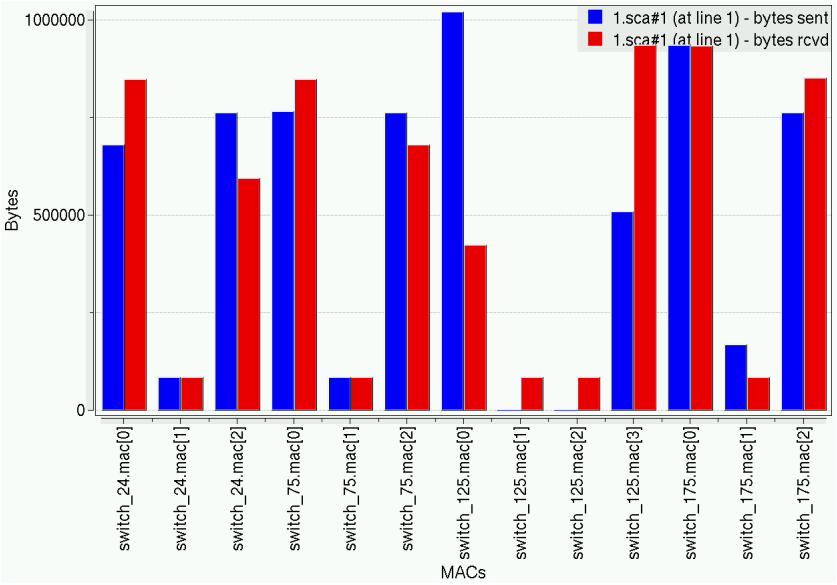

Abbildung 48: Szenario „realer Betrieb“, Testlauf 1, Gesendete und empfangene Daten in Linien

Nach Betrachtung des Linienverkehrs ist nun der Datenaustausch zwischen den einzelnen Linien zu untersuchen. Abbildung 49 stellt diesen Fall dar. Die Switche 204, 205, 206 und 207 sind die Verbindungen der einzelnen Linien, über sie wird der Querverkehr transportiert. Switch 208 verbindet diese Switche untereinander. Die Netzwerkteilnehmer 200 bis 204 senden ihre Daten an Gerät 209, und sind an die mittleren Ports der aufgezählten Switches angeschlossen. Die maximale Datenrate dieser betrachteten MACs liegt hier bei ca. 1,5 MByte/s, die anderen Schnittstellen befördern 0,75 MByte/s bis 1,25 MByte/s.

Wenn man von den die einzelnen Datenraten nun die Summe bildet, lässt sich annähernd die Auslastung des Netzes berechnen. Zunächst zusammengefasst für die Linien, dort ergibt sich eine Auslastung von

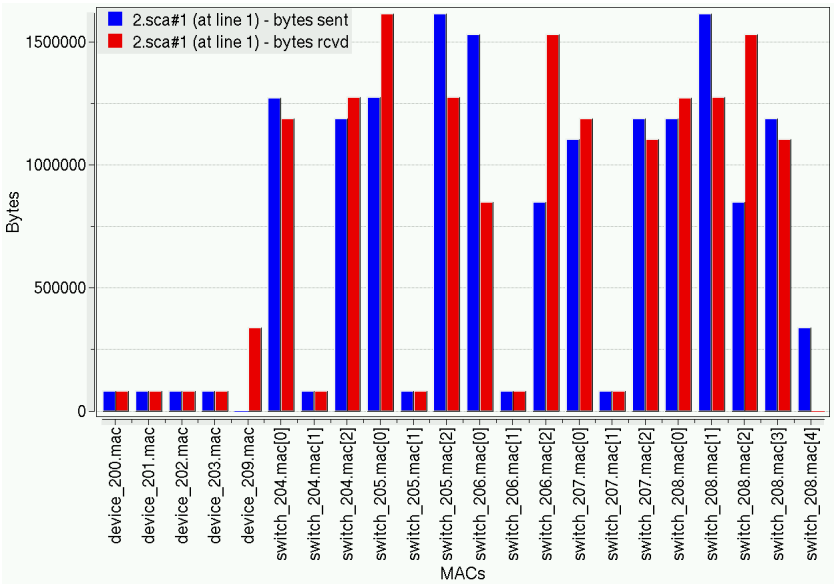

Abbildung 49: Szenario „realer Betrieb“, Testlauf 1, Gesendete und empfangene Daten zwischen Linien

89%. Die Netzresourcen werden also durch die vier einzelnen Linien gut ausgenutzt.

$$A = \frac{R_{RT-Linie}}{R_{Netz}} = \frac{11,19\frac{MByte}{s}}{12,5\frac{MByte}{s}} \approx 0,89$$

Als ergänzender Wert ist nun noch die Auslastung im Switch 208, der den Querverkehr der einzelnen Linien aufnimmt, zu betrachten. Auch hier ergibt sich ein gutes Ergebnis von 79%.

$$A = \frac{R_{RT-Quer}}{R_{Netz}} = \frac{9,9\frac{MByte}{s}}{12,5\frac{MByte}{s}} \approx 0,79$$

Da aufgrund der Netzwerkgröße nur Stichproben der Verkehrsauslastung genommen werden konnten, stellen diese Werte natürlich nur Mittelwerte dar.

Zusammenfassung Wie innerhalb dieses Szenarios gezeigt, funktioniert die Simulation auch in dieser an die Realität angelehnte Konfiguration einwandfrei. Die einzelnen Netzwerklinien und auch ihre Verbindungen untereinander sind gut ausgelastet. Abschließend ist noch der Gewinn, der sich durch die Aufteilung in einzelne Linien ergibt, zu betrachten. Im schlechtesten Fall benötigt jede Kommunikation zwischen zwei Geräten auch einen Zeitschlitz, wie in Szenario „kleines Netzwerk“ gesehen. Durch die Unterteilung des Netzes, wie in diesem Szenario, lassen sich nun Sendevorgänge parallel durchführen. Der daraus resultierende Gewinn sei definiert als die Anzahl der Kommunikationen durch die Anzahl der Zeitschlitze, und beträgt hier ca. 2,49. Dies stellt einen sehr guten Wert dar, dies zeigt auch der Vergleich mit dem nächsten Szenario.

$$G = \frac{n_{communication}}{n_{slot}} = \frac{87}{35} \approx 2,49$$

6.2.3 Ergebnisse des Szenarios „Lastgrenze“

Testlauf 1 In diesem Lauf ist die Funktionsfähigkeit bei reinen RT-Betrieb zu untersuchen. Aufgrund der hohen Anzahl an Zeitschlitzen in diesem Szenario muss die Sendeperiode erheblich größer gewählt werden. Deshalb tritt so gut wie kein Überschwingen des MAC Warteschlangen-Füllstands in der Hochlaufphase auf. Darstellung 50 zeigt den sich ergebenen Graphen für die Echtzeitgeräte. Nach gut 0,1 s konvergieren die einzelnen Füllstände auf ca. 2,7 Rahmen, aufgrund der großen Anzahl (154) von MACs in diesem Netz wird hier die Legende nicht dargestellt.

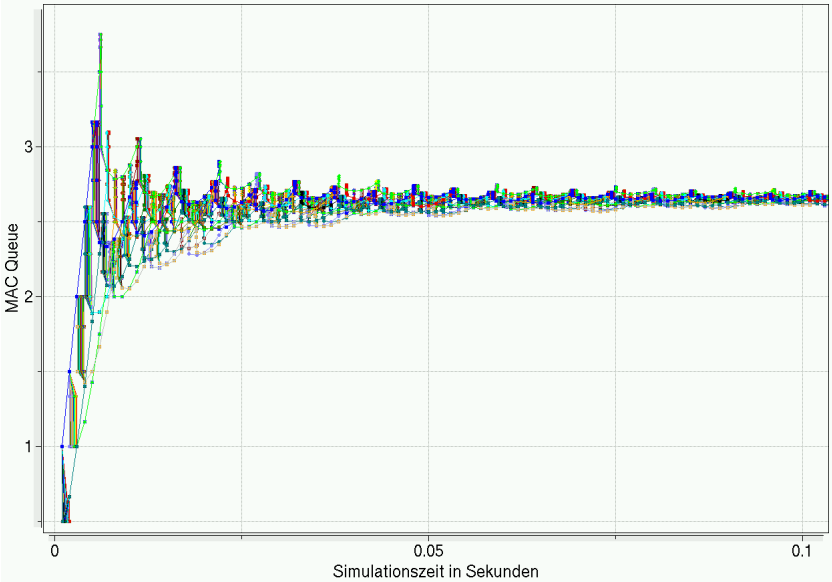

Abbildung 50: Szenario „Lastgrenze“, Testlauf 1, Rahmen in Warteschlange MAC Echtzeitgeräte

Gleiches gilt auch für die Medienzugriffsschicht der Switches, wie Abbildung 51 wiedergibt. Aufgrund der sehr viel größeren Zykluszeit erkennt man hier deutlich die einzelnen Sendevorgänge der angeschlossenen Echtzeitgeräte, die die Queue-Füllstände erhöhen. Der Graph stellt insgesamt 169 Kurvenzüge dar, auch hier ist aus Gründen der Übersichtlichkeit die Legende nicht abgebildet.

Abschließend ist noch die Funktion des Zeitmultiplex zu überprüfen. Wie in Darstellung 52 als Ausschnitt wiedergebenen, werden die einzelnen Zeitschlitze sauber eingehalten. Die aufgetretenen Lücken entstehen zum einen durch den NRT-Slot und zum anderen durch einzelne Verkehrsanforderungen, die zur Simulationszeit aufgrund von Fehlern entfernt werden mussten. Diese Probleme resultierten aus Fehlern in der Berechnung der zeitlichen Ablaufpläne durch das Werkzeug „schedulergui“ und wurden erst nach Abschluss dieser Simulation behoben.

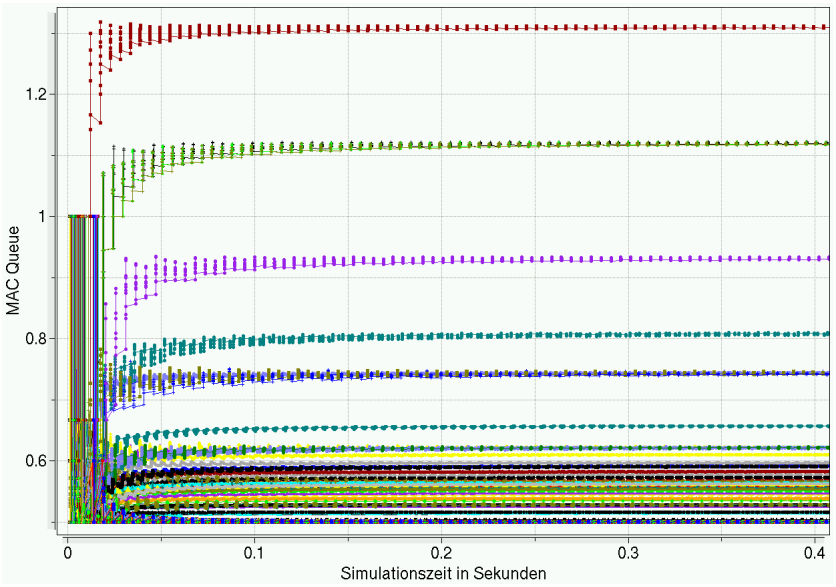

Abbildung 51: Szenario „Lastgrenze“, Testlauf 1, Rahmen in Warteschlange MAC Switches

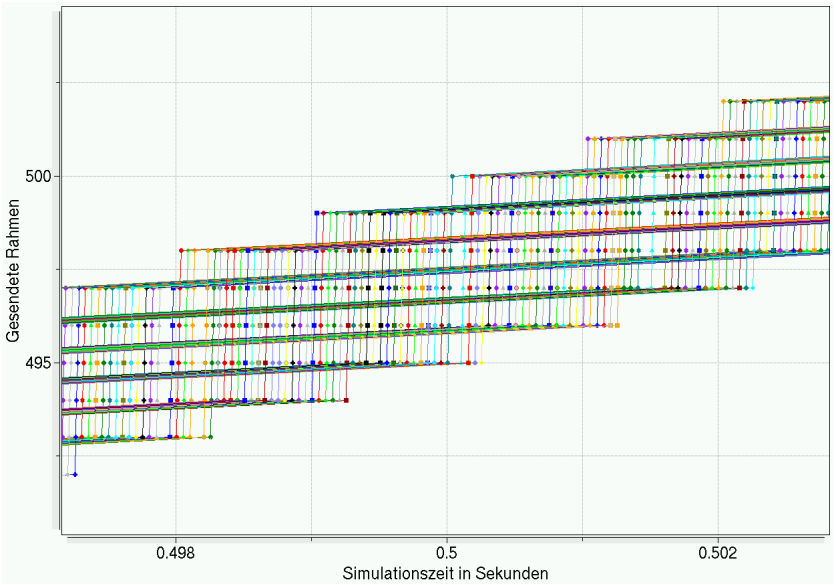

Abbildung 52: Szenario „Lastgrenze“, Testlauf 1, Gesendete Rahmen Echtzeitgeräte

Trotz der erheblichen Größe dieses Netzwerks kommt es in der simulierten Konfiguration nach Ende der Hochlaufphase zu keinerlei Schwingungen oder sonstige Unregelmäßigkeiten. Es werden keine Zeitschlitze verpasst, oder Rahmen aufgrund falscher Weiterleitung verworfen. Auch bei diesem Szenario funktioniert das Simulationsmodells im Echtzeit-Betrieb einwandfrei.

Testlauf 2 Nun ist das Verhalten des erstellten Modells bei gemischten Betrieb zu betrachten. Durch die Einflüsse des nicht Echtzeitverkehrs verändern sich die Eigenschaften des Modells, es wird wesentlich dynamischer. Es entstehen starke Schwankungen in den Füllständen der MAC Warteschlangen, wie bei Szenario „kleines Netzwerk“, Testlauf 3 beobachtet. Die große Anzahl der Broadcasts begrenzt den Netzwerkdurchsatz auch für die RT-Teilnehmer, dies begründet sich aus den in Abschnitt „Einschränkun-

gen der Simulationskomponenten“ auf Seite 65 dokumentierten Einschränkungen. Abbildung 53 stellt den Füllstand der 164 MAC Warteschlangen dar.

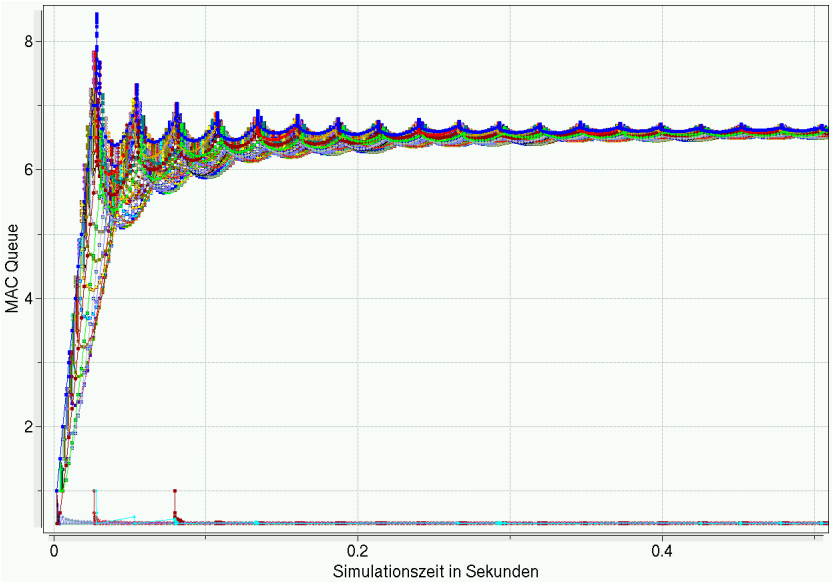

Abbildung 53: Szenario „Lastgrenze“, Testlauf 2, Rahmen in Warteschlange MAC Echtzeitgeräte

Trotz der verlängerten Sendeperiode der Verkehrsgeneratoren ist der Füllstand in diesem Lauf viel höher. Die zu beobachtenen Schwingungen klingen erst sehr viel später ab, auch dieses Verhalten deckt sich mit den Beobachtungen in dem ersten Szenario. Nach ca. einer halben Sekunde stellt sich ein stabiler Wert von ca. 5,6 Rahmen ein. Analog dazu ist in Abbildung 54 das Verhalten der Switch-Warteschlangen aufgeführt. Aufgrund der Broadcastsstürme im Netz sind hier Daten von 1037 MAC-Queues zu betrachten; man erkennt deutlich das dadurch verursachte chaotische Verhalten des Systems.

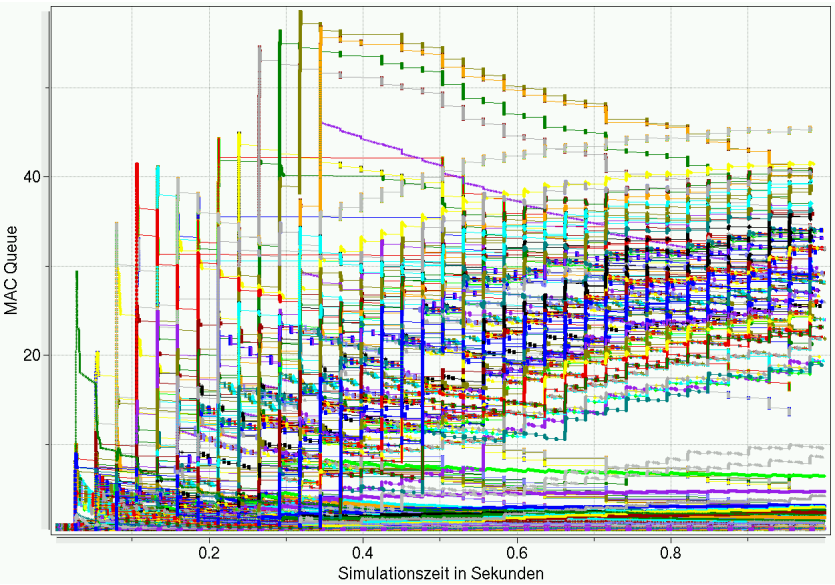

Abbildung 54: Szenario „Lastgrenze“, Testlauf 2, Rahmen in Warteschlange MAC Switches

Die Ende-zu-Ende Verzögerung schwankt deshalb auch recht stark, sie variiert bei den verschiedenen NRT-Teilnehmern von 0,1 ms bis über 100 ms. Auf weitere Untersuchungen, etwa des Durchsatzes, wur-

de verzichtet, da es sich bei diesem Testlauf nicht um eine in der Praxis sinnvoll einsetzbare Betriebsart handelt.

Zusammenfassung Wie in Testlauf 1 gezeigt, funktioniert der Echtzeitbetrieb auch bei dieser Netzwerkgröße ohne Probleme. Bei zusätzlich hinzugefügten NRT-Verkehr kommt es zu Funktionsstörungen, wie sie ähnlich auch im Szenario „kleines Netzwerk“ auftreten. Die dort getroffenen Aussagen zum Verbesserungsbedarf gelten hier analog. Die Simulationsumgebung als auch das Werkzeug zur graphischen Auswertung kommen bei der verwendeten Hardware an ihre Leistungsgrenzen, einige Male kam es auch zu Programmabstürzen. Die Simulation ist bei Darstellung der Vorgänge innerhalb des Netzwerkes nicht mehr sinnvoll zu bedienen, durch Deaktivierung dieser Funktion oder Verwendung der textbasierten Oberfläche lassen sich diese Probleme aber mindern. Bei der Simulation fallen erhebliche Datenmengen von 390 MB pro Sekunde an, die die Auswertung erschweren. Durch die große Anzahl von Ereignissen ist die Fehlersuche sehr aufwendig. Dieses Szenario stellt also wirklich eine Belastungsgrenze des Modells für die verwendete Hardware und die Durchführung der Simulation dar.

Der Gewinn, der sich durch die Parallelisierung der Kommunikationsanforderungen erreichen lässt, ist hier im Vergleich zu Szenario „realer Betrieb“ geringer. Dies lässt sich durch den Netzaufbau begründen, der weniger Möglichkeiten zur Parallelisierung bietet.

$$G = \frac{n_{communication}}{n_{slot}} = \frac{164}{105} \approx 1,56$$

6.3 Beurteilung der Ergebnisse

Die erstellte Simulation funktioniert im reinen Echtzeitbetrieb einwandfrei. Die maximale Leistung bei kleinen Netzen ist sehr gut, die minimale Zykluszeit ist absolut konkurrenzfähig zu kommerziellen Verfahren. Ein Einsatz des REAL Systems für höchste Echtzeitanforderungen wäre möglich, wenn die Konfiguration des Netzes entsprechend ausgelegt wird, also möglichst kleine Subnetze gebildet werden. Im gemischen Betrieb mit nicht Echtzeitteilnehmern besteht noch Verbesserungspotential. Innerhalb dieser Betriebsart reagiert die Simulation sehr viel dynamischer und schlecht vorhersehbar, insbesondere verlängert sich die Hochlaufphase erheblich.

Die Aufteilung des Netzes in einzelne getrennte Abschnitte verbessert die Leistung erheblich, der Betrieb mit einer geringeren Zykluszeit wird möglich. Es lassen sich reale Gewinne durch die optimierte Berechnung der Ablaufpläne erreichen. Die Belastbarkeit der Simulation ist befriedigend. Die sehr große Zykluszeit, die bei so umfangreichen Netzen in der Realität aufträte, würde einen sinnvollen Einsatz für Echtzeitanwendungen aber auch erheblich erschweren. Um eine ansprechende Leistung und Funktionssicherheit zu erreichen, ist eine Segmentierung in kleinere Netze sinnvoll.

Da zum Zeitpunkt der Fertigstellung der Arbeit die Projektgruppe [39] noch nicht abgeschlossen war, konnte ein Abgleich der hier dargestellten Simulationsergebnisse mit ihrer Arbeit nicht stattfinden. Die hier dargestellten Resultate lassen sich nicht ohne weiteres in die Praxis übertragen, hier müsste zunächst durch weitere Arbeiten das Design des REAL-Systems verfeinert werden.

7 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wird ein Modell einer neuartigen Echtzeit-Ethernet Lösung für das Simulationssystem OMNeT++ entwickelt. Dafür müssen im Vorfeld verschiedene Echtzeit-Ethernetansätze in ihrer Funktionsweise untersucht und Grundlagen über das zu betrachtene Umfeld – Ethernet in der Automatisierungstechnik – recherchiert werden. Die einzelnen Echtzeitlösungen werden detailliert miteinander verglichen, weiterhin finden sie bei der Entwicklung des Modells Berücksichtigung. Zusätzlich zur Konzeption des Simulationsmodells ist der Entwurf und die Implementierung von verschiedenen Hilfswerkzeugen wie ein Netzwerkgenerator und ein Konverter notwendig. Anschließend lässt sich das Modell entwickeln und auf einwandfreie Funktionsweise überprüfen. Das Verhalten des Modells wird durch die Simulation von drei unterschiedlichen Netzwerk-Szenarien ausführlich untersucht. Anhand der ermittelten Ergebnisse ist die einwandfreie Funktion zu zeigen, des Weiteren lässt sich ein erster Vergleich mit den im Vorfeld betrachteten Systemen vornehmen.

Das entwickelte System bietet eine Leistung, die auch höchsten Echtzeitanforderungen genügt. Der gemische Betrieb mit zusätzlichen nicht Echtzeitteilnehmern ist möglich, hier besteht aber noch Optimierungspotential. Durch die optimierte Berechnung der zeitlichen Ablaufpläne lässt sich die Leistung gegenüber gewöhnlichen, auf Zeitmultiplex-basierenden Echtzeit-Ethernetlösungen erheblich erhöhen. Diese Arbeit bietet somit eine Ausgangsbasis für eine weitergehende Realisierung dieses neuartigen Systems.

Zunächst bestehen bei den Hilfswerkzeugen zur Erstellung und Konvertierung der Netze erhebliche Ausbaumöglichkeiten. Beim Werkzeug „createNetwork“ könnte die Generierung von Netzen verbessert werden. Bei der Benutzerschnittstelle bestehen bei beiden Hilfsprogrammen Optimierungspotential. Die Geschwindigkeit von „convertNetwork“ bei großen Netzwerken ist nicht ausreichend. Bei dem erstellten Simulationsmodell ließen sich durch die Unterstützung von Vollduplex-Verbindungen noch beträchtliche Leistungssteigerungen erreichen. Weitere Verkehrsmodelle wie Multicast würden zusätzliche Effizienzsteigerungen ermöglichen. Eine vollständige Umsetzung der in [6] vorgeschlagenen Varianten des Echtzeitbetriebs wäre denkbar, natürlich bestehen auch bei Parametrierung und Fehlerberichterstattung Erweiterungsmöglichkeiten. Für eine stabile Funktion des gemischten Betriebes müsste ein entsprechendes Echtzeit-Protokoll entwickelt und zusätzliche Erweiterungen vorgenommen werden. Für weitere Verbesserungsmöglichkeiten siehe auch Kapitel 5.3.1.

Nach Abschluss der Projektgruppe könnte eine Überprüfung der dargestellten Ergebnisse mit einem unabhängig entwickelten Modell erfolgen. Um weitere Erkenntnisse über Umsetzungsmöglichkeiten dieses Echtzeit-Ethernet Verfahrens zu erlangen, wäre die Entwicklung eines Switch-Prototypen in Hardware sinnvoll. Damit könnten die hier erzielten Ergebnisse verifiziert werden. Durch Umsetzung der vorgeschlagenen Erweiterungen ließen sich aber auch genauere Ergebnisse innerhalb der entwickelten Simulationsumgebung gewinnen.

A Beschreibung der Benutzerschnittstelle und des Simulators

Im Folgenden werden nun noch kurz die Benutzerschnittstellen der erstellten Hilfswerkzeuge und der Simulationsumgebung dargestellt. Weiterhin wird auf die Übersetzung und Anpassung des erstellten Quellcodes eingegangen.

A.1 Benutzerschnittstelle des „createNetwork“-Werkzeugs

Das Werkzeug createNetwork ist als Konsolenanwendung konzipiert. Nach dem Aufruf ohne Optionen erfolgt eine Ausgabe von Bedienungshinweisen und Informationen zum Copyright, die im Folgenden abgebildet sind.

```
Create random networks in the 'PG' format.
Usage: ./createNetwork [FILE] [OPTIONS] [RANDOM SEED]

The [FILE] specify the output filename, this is a mandatory value.

Valid [OPTIONS] are:
Network size, the minimal size is 10, maximal is 100000.
Default value for the network size is 100. The size of the
created network is always a odd value, for internal reasons.

The four following parameters are percent values, the minimal
value is therefore zero, the maximal one hundred.

Sibling chance specify the network topology, use zero for a bus
network, 100 for a star topology.
Default value for sibling chance is 50.

Connection chance specify the chance that a node is connected
to another node. 33 is the default value. A value of 100 percent
don't work with random traffic mode at the moment, high chances
leads also to multiple connections to the same receiver.

Connection range specify how far the other connection partner
are from a given node. 33 is a good start value, this is also
the default.

No real-time chance is the chance that a device is a normal (no
real-time) device. 0-5 percent are meaningful values, 0 is the
default. Network with no real-time devices are currently not
supported by the schedule calculation application.

Communication mode is the mode of the the generated communication.
The possible modes are 'random' (one to one) for a random network,
'master' for a master (one to many), and 'multimaster' (few to many)
```

```
structure. Random is the default.

[RANDOM SEED] specify the seed of the random generator. The same
seed leads to the same networks (if the options are the same).
Use a positive value from 0 to 1000000. Omit this value to use
a seed generated from the system time and date.

Examples:
Create a medium network with mixed topology (default values):
./createNetwork outputFile.xml

Create a big network with star topology, and few random connections:
./createNetwork outputFile.xml 10000 100 15 33 0 random

Create a default network with a given random seed:
./createNetwork outputFile.xml 34234

createNetwork version 21,
Copyright (C) 2006, Henning Westerholt,
E-Mail: 'hw at skalatan dot de'

createNetwork comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute
it under certain conditions.

This program uses additional free software:
- the STL-like templated tree class from Kasper Peeters
- MersenneTwister class from Makoto Matsumoto and Takuji
  Nishimura

Please refer to the header of the source code for the copyright
of certain files.
```

Nach einem Aufruf mit gültigen Parametern gibt das Programm eine kurze Information aus, um den Benutzer über die erfolgreiche Erzeugung des Netzwerkes zu informieren.

```
./createNetwork test-network.xml
Create random network in file test-network.xml, with size 100, sibling
chance 0.5, connection chance 0.33, connection range 0.33,
normal device chance 0 and a auto generated random seed.
Use a random communication structure.
```

A.2 Benutzerschnittstelle des „convertNetwork“-Werkzeugs

Das Werkzeug convertNetwork ist als Konsolenanwendung implementiert. Auch hier wird beim Aufruf ohne Parameter eine Hilfe ausgegeben.

```
./convertNetwork.sh
Convert a network description from the PG-format in the OMNeT++ format
Usage: ./convertNetwork.sh [NETWORK] [COMMUNIKATIONLINES] [SCHEDULE]
The input must be valid XML files in the 'PG'-Format.
```

Bei ordnungsgemäßem Aufruf erfolgt eine Rückmeldung über die erfolgten Aktionen an den Benutzer.

```
./convertNetwork.sh test-network.xml test-communicationlines.xml
test-schedule.xml
Convert a network description from the PG-format in the OMNeT++ format.
convert the network..
convert the traffic..
convert the schedules..
convert the network to ned format for OMNeT++..
The number of time slots in the schedule is 3.
Finish network converting, the generated files are:
test-out-network.xml, test-out-traffic.ini, test-out-schedule.xml.
```

A.3 Benutzerschnittstelle des Simulationssystems

Nun erfolgt noch eine kurze Einführung in die Oberfläche des Simulationssystems OMNeT++. Abbildung 55 zeigt das Hauptfenster des Simulators direkt nach dem Start.

Die für die Kontrolle des Simulationsablaufs wichtigen Aktionen lassen sich über den rot eingefärbten Bereich vornehmen. Hier kann die Berechnung gestartet und auch wieder beendet werden. Der grün gefärbte Abschnitt zeigt wichtige Simulationsparameter wie die aktuelle Zeit und das nächste Ereignis an. Im blauen Fenster stellt das Framework die Ausgaben der einzelnen Simulationselemente dar. Im gelben Bereich werden die nächsten Ereignisse zunächst oben als Zeitleiste und links als Liste dargestellt. Über den grau markierten Bereich können die verschiedenen Testläufe eines Szenarios zur Simulation ausgewählt werden. Für weitere Informationen zur Bedienung des Systems sei hier auf die Dokumentation des Frameworks [52] verwiesen.

A.4 Konfiguration des Simulationssystems

Die Konfiguration des Simulationssystems erfolgt über mehrere sogenannte „ini“-Dateien, und die Netzwerkbeschreibung. Nachfolgend sei dies für das erste Szenario dargestellt. Über die Datei **omnetpp.ini** werden allgemeine Simulationsparameter wie Anzahl und Aufbau der einzelnen Testläufe konfiguriert. Konkrete Eigenschaften des Systems, wie in Kapitel 6.1.1 dargestellt, lassen sich über **default.ini** verändern. In der Datei **.tkenvrc** werden Informationen über den Zustand des Simulationssysstems gespeichert, hier sind aber keine Änderungen erforderlich.

Die Netzwerkbeschreibung, die Verkehrsbeziehungen und zeitlichen Ablaufpläne lassen sich über die Dateien **network.ned**, **traffic.ini** und **schedule.xml** definieren. Für genauere Informationen über den Aufbau sei auf den Abschnitt „Darstellung der zu konvertierenden Formate“ auf auf Seite 46 verwiesen.

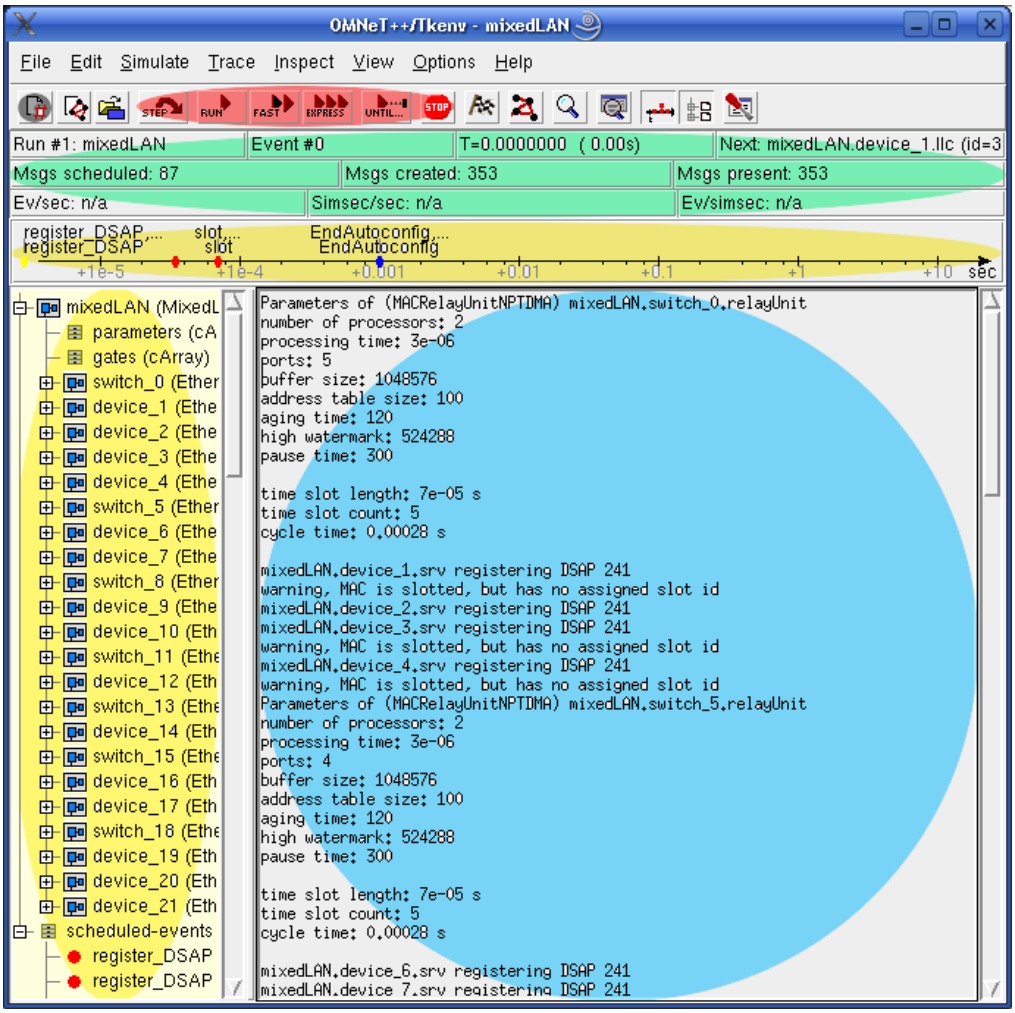

Abbildung 55: Hauptfenster OMNeT++

A.5 Beschreibung der Simulationselemente

Die im Rahmen der Simulation erstellten Module werden in Kapitel 5.2.6 in ihrem Verhalten ausführlich beschrieben. Für eine weitergehende Erläuterung der Bedeutung dieser Elemente, insbesondere ihre Berücksichtigung bei der Berechnung des zeitlichen Ablaufplans mit Hilfe der „schedulergui“, sei auf die Dokumentation der Projektgruppe [39] verwiesen.

B Beschreibung der beigefügten CD

Alle Ergebnisse dieser Arbeit finden sich auf dem beigefügten Datenträger. Im Folgenden wird kurz der Inhalt der einzelnen Verzeichnisse erläutert.

B.1 Diplomarbeit und Bibliographie

Die Arbeit befindet sich im Verzeichnis „thesis“, der Dateiname ist „westerholt-rte-thesis-2006.pdf“. Im selben Verzeichnis finden sich in der Datei „westerholt-rte-thesis-2006.bib“ die kompletten Quellenan-

gaben zu der Diplomarbeit. Diese Daten sind im verbreiteten BibTEX-Format gespeichert, und können direkt mit dem LATEX Textsatzsystem verwendet werden.

B.2 Quellcode der Implementierung

Innerhalb des Verzeichnis „source“ befindet sich der Quellcode der Implementierung. Es folgt ein Überblick über die programmierten Module.

B.2.1 createNetwork

Das Werkzeug „createNetwork“ ist im verbreiteten „tar“-Format in der Datei „createNetwork.tar“ archiviert. Die Tabelle 14 zeigt den Inhalt der Archivdatei.

application.cpp	filehandling.h	mersenneTwister.h	ports.h	utility.h
application.h	hardware.h	messages.h	README	xmlgenerator.cpp
createNetwork.pro	links.h	network.cpp	tree.hh	xmlgenerator.h
filehandling.cpp	main.cpp	network.h	utility.cpp	

Tabelle 14: Inhalt Archiv „createNetwork“

B.2.2 convertNetwork

Das Werkzeug „convertNetwork“ ist im verbreiteten „tar“-Format in der Datei „convertNetwork.tar“ archiviert. Die Tabelle 15 zeigt den Inhalt der Archivdatei.

convertNetwork.sh	network.xsl	README
schedule.xsl	traffic.xsl	

Tabelle 15: Inhalt Archiv „convertNetwork“

B.2.3 Simulationsmodule

Die Simulationsmodule sind im verbreiteten „tar“-Format in der Datei „INET-REAL.tar“ archiviert. Die Tabelle 16 zeigt auszugsweise den Inhalt der Archivdatei, zusätzlich ist aber noch der gesamte INET Quellcode in dem Archiv enthalten.

MACRelayUnitNP.h	EtherSwitchTDMA.ned	MACRelayUnitNPTDMA.cc	MACRelayUnitNPTDMA.ned
MACRelayUnitBase.cc	EtherSwitch.ned	MACRelayUnitBase.h	EtherHostTDMA.ned
EtherMACTDMA.h	EtherMAC.ned	EtherMACTDMA.ned	EtherFrame.msg
EtherMAC.cc	EtherMAC.h	EtherStreamApp.cc	EtherStreamApp.ned
EtherStreamSink.cc	EtherStreamSink.ned	EtherStreamSink.h	makemakefiles
MACRelayUnitNPTDMA.h	EtherMACTDMA.cc	EtherHost.ned	EtherStreamApp.h
inetconfig			

Tabelle 16: Inhalt Archiv „INET-REAL.tar“

Zusätzlich zu diesem Archiv ist der erstellte Quellcode noch als sog. „unified diff“ als Differenz zu der unveränderten INET-Version in der Datei „INET-REAL.diff“ abgelegt. Diese Datei kann mit dem Werkzeug „patch“ auf eine unveränderte INET-Version angewandt werden, um alle Veränderungen zu erhalten. In dieser Differenzdatei sind zusätzlich noch einige erste Entwürfe zu einem speziellen REAL-Protokoll und einem dazugehörigen Verkehrsgenerator enthalten.

B.3 Skripte

Im Verzeichnis „script“ sind verschiedene Hilfsskripte für die Erstellung des Simulationssystems und zum setzen notwendiger Umgebungsvariablen enthalten. Im Folgenden werden die einzelnen Skripte kurz aufgeführt.

- Entpacken, konfigurieren und erstellen des Simulationssystems OMNeT++
  - build-omnetpp.sh
  - build-INET.sh
- Entpacken, konfigurieren und erstellen des Simulationssystems ns2
  - build-ns2.sh
- Entpacken, konfigurieren und erstellen der für die Diplomarbeit benötigten Werkzeuge und Abhängigkeiten von OMNeT++
  - build-qt4.sh
  - build-da-tools.sh
  - build-omnetpp-deps.sh
- Setzen der Umgebungsvariablen für den Betrieb der Simulationsumgebungen
  - set-omnetpp-env-vars.sh
  - set-ns2-env-vars.sh
  - .bashrc
- Aufräumen der Installations- und Erstellungs-Verzeichnisse
  - clean-builddir.sh
  - clean-installdir.sh

B.4 Simulationssoftware

Im Verzeichnis „software“ finden sich die zum Erstellen des Simulationssystems verwendete Software. Für eine genaue Übersicht der einzelnen Versionen siehe auch Kapitel 5.3.2.

B.5 Erstellte Szenarien

Im Verzeichnis „simulation“ sind die drei Testszenarien im verbreiteten „tar“-Format in der Datei „simulation.tar“ archiviert. In diesem Archiv sind sowohl die für die Simulation benötigten Konfigurationsdateien, als auch die Rohdateien vor der Konvertierung durch das Werkzeug „schedulergui“ abgelegt.

B.6 Rohdaten der Simulationsergebnisse

Die Rohdaten der Simulationsläufe befinden sich im Verzeichnis „result“ in drei Unterverzeichnissen. Die Daten der einzelnen Testläufe sind aus Platzgründen im „gzip“-Format komprimiert.

C Erstellung und Anpassung des Simulationssystems

Abschließend erfolgt noch eine kurze Einführung in die Erstellung und Anpassung der Simulationsumgebung. Die notwendigen Arbeitsgänge werden nur skizziert, es wird davon ausgegangen, dass grundlegende Kenntnisse in der Linux-Systemadministration, C++ Programmierung und zu dem Aufbau von OMNeT++ vorhanden sind.

C.1 Erstellung des Simulationssystems

Zunächst muss die richtige Verzeichnisstruktur angelegt werden, wie im Ordner „script“ in der Datei README beschrieben ist. Die Software ist ins Verzeichnis „download“ zu kopieren. Gegebenenfalls müssen die Pfade in den einzelnen Skripten angepasst werden. Dann sollte sich durch Aufruf der Skripte „build-omnetpp-deps.sh“ und „build-omnetpp.sh“ das Simulationssystem erstellen und installieren lassen. Eine ausführliche Darstellung des Installationsprozesses findet sich im Handbuch des Frameworks [52] bzw. im Verzeichnis „Documentation“ des INET-Ordners.

Anschließend müssen durch den Aufruf des Skriptes „set-omnetpp-env-vars.sh“ die notwendigen Umgebungsvariablen gesetzt werden. Nun lässt sich das INET-Framework entpacken und der Patch anwenden. Das Framework muss konfiguriert werden, dieser Prozess wird in der Datei README beschrieben. Anschließend sollte es sich übersetzen lassen und die Beispiele im Verzeichnis „INET/Examples“ für eine erste Funktionsüberprüfung gestartet werden. Das Skript zum setzen der Umgebungsvariablen ist erneut nach einem Beenden der Sitzung aufzurufen, alternativ lässt sich aber die mitgelieferte „.bashrc“ für diese Initialisierung nutzen.

Nach der Überprüfung der korrekten Funktionsweise lassen sich die erstellten Simulationen testen. Gegebenenfalls können die entstandenen Ergebnisse mit den mitgelieferten Daten verglichen werden. Weitere Überprüfungen lassen sich durch die internen Tests von OMNeT++ und INET vornehmen. Hierfür sei aber auf die mitgelieferte Dokumentation dieser Werkzeuge verwiesen.

Die Werkzeuge „createNetwork“ und „convertNetwork“ sind nach Qt-Standard durch einen Aufruf von „qmake“ und anschließend „make“ zu kompilieren. Für eine Erstellung der „schedulergui“ sei auf die Dokumentation der Projektgruppe [39] verwiesen. Bei Problemen bei Erstellung oder Ausführung der Werkzeuge ist die Erfüllung der Anforderungen aus Kapitel 5.3.2 zu überprüfen.

C.2 Erstellung eines eigenes Szenarios

Um ein neues Szenario zu erzeugen, ist zunächst ein Netzwerkes mit „createNetwork“ zu generieren. Für dieses müssen nun die Kommunikationslinien und der synchronisierte Schedule mit Hilfe der „schedulergui“ berechnet werden. Diese Berechnungen sind im Halbduplex-Modus vorzunehmen. Die berechneten Daten und das Ursprungsnetzwerk sind in das Arbeitzverzeichnis des Werkzeugs „convertNetwork“ zu kopieren, anschließend lässt sich die Konvertierung durchführen.

Das Ergebnis des Konvertierungsschrittes ist in ein leeres Verzeichnis zu kopieren, es müssen zusätzlich noch die Konfigurationsdateien „default.ini“ und „omnetpp.ini“ hinzugefügt und gegebenenfalls angepasst werden. Nun kann die Simulation mit dem Befehl „INET“ gestartet werden. Bei Aktivierung von Vektor- und Skalar-Ausgabe können diese Daten anschließend mit den Werkzeugen „plove“ und „scalars“ visualisiert werden.

C.3 Anpassung des Simulationssystems

Die mitgelieferten oder auch selbst erstellten Szenarien lassen durch durch Veränderung der in Kapitel A.4 vorgestellten Dateien konfigurieren. Für eine genaue Bedeutung der einzelnen Parameter sei auf die Kapitel 5.2 und 6.1.1 verwiesen. Änderungen lassen sich anhand der Ausgabe beim Start des Simulationssystems kontrollieren. Sofern keine graphische Ausgabe vorgesehen ist, können die einzelnen Objekte auch inspiziert werden, um Einblick in die internen Variablen zu erhalten. Für Verhaltensänderungen der Simulation müssen natürlich Anpassungen am Quelltext der erstellten Module vorgenommen werden. Hierzu sei auf den mitgelieferten Programmcode verwiesen.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] *Das Kopf-an-Kopf-Rennen, Realtime-Ethernet-Lösungen im direkten Vergleich*. Elektrotechnik - Das Automatisierungs-Magazin, Oktober 2004.

[2] AAXEON INTERNETSHOP: *4-Port Ethernet Switch Industrie-tauglich*. `http://www.aaxeon.com/products/Productdetail.aspx?cate=3&modelno=LNX-500W`.

[3] BORMANN, ALEXANDER und INGO HILGENKAMP: *Industrielle Netze - Ethernet Kommunikation für Automatisierungsanwendungen*. Hüthig Verlag Heidelberg, 2006.

[4] CHAFFEE, MARK und BOB HIRSCHINGER: *EtherNet/IP Motion recieves top priority*. The industrial ethernet book, Mai 2006.

[5] DOPATKA, FRANK und ROLAND WISSMÜLLER: *Achieving Realtime Capabilities in Ethernet Networks by Edge-Coloring of Communication Conflict-Multigraphs*. IASTED International Conference on Parallel and Distributed Computing and Networks, Acta Press, Februar 2006.

[6] DOPATKA, FRANK und ROLAND WISSMÜLLER: *A Top-Down Approach for Realtime Industrial-Ethernet Networks using Edge-Coloring of Conflict-Multigraphs*. SPEEDAM International Symposium on Power Electronics, Electrical Drives, Automation and Motion, IEEE, Mai 2006.

[7] ENSTE, UDO: *Auf zu neuen Ufern*. Chemie Produktion, November 2002.

[8] ETHERCAT TECHNOLOGY GROUP: *EtherCAT Webseite*. `http://www.ethercat.org/`.

[9] ETHERCAT TECHNOLOGY GROUP: *EtherCAT - Technical Introduction and Overview*. `http://www.ethercat.org/pdf/english/EtherCAT_Introduction_en.pdf`, Juli 2005.

[10] ETHERNET POWERLINK STANDARDIZATION GROUP: *EPSG Webseite*. `http://www.ethernet-powerlink.org/`.

[11] ETHERNET POWERLINK STANDARDIZATION GROUP: *EPSG Brochure 2005*. `http://www.ethernet-powerlink.org/`, Oktober 2005.

[12] FELSER, MAX: *Ethernet Switche für Motion-Control*. Automate.now! Jahrbuch der Automatisierungstechnik, 2005.

[13] FELSER, MAX: *Real-Time Ethernet - Industry Prospective*. Proceedings of the IEEE, 93(6):1118–1128, Juni 2005.

[14] FELSER, MAX und THILO SAUTER: *Standardization of Industrial Ethernet - the next Battlefield?* `http://felser.ch/download/FE-TR-0403.pdf`, 2004.

[15] HILL, JÜRGEN: *Ethernet erobert die Produktion*. Computerwoche, Juli 2005.

[16] HMS INDUSTRIAL NETWORKS: *Technologiebeschreibung EtherNet/IP*. `http://www.hms-networks.de/Technologies/EthernetIP.shtml`.

[17] IEEE COMPUTER SOCIETY: *IEEE Std. 1588-2002, Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, November 2002.

[18] INTERNATIONAL ELECTROTECHNICAL COMMISSION: *Digital data communications for measurement and control - Part 1: Profile sets for continuous and discrete manufacturing relative to fieldbus use in industrial control systems*, März 2003.

[19] INTERNATIONAL ELECTROTECHNICAL COMMISSION: *Digital data communications for measurement and control - Part 2: Additional profiles for ISO/IEC 8802-3 based communication networks in real-time applications*, September 2005.

[20] JASPERNEITE, JÜRGEN und EHAB ELSAYED: *Investigations on a Distributed Time-triggered Ethernet Realtime Protocol Used by PROFINET*. 3rd International Workshop on Real-Time Networks, Juli 2004.

[21] JOST, MICHAEL und MARTIN ROSTAN: *EtherCAT-Implementierung: Möglichkeiten, Schnittstellen, Aufwand*. `http://www.ethercat.org/pdf/german/EtherCAT_Implementierung.pdf`, November 2004.

[22] KISZKA, J., N. HAGGE, P. HOHMANN und B. WAGNER: *RTnet - Eine Open-Source-Lösung zur Echtzeitkommunikation über Ethernet*. Telematik 2003, VDI-Berichte, (1785):55–64, jun 2003.

[23] KISZKA, JAN, BERNARDO WAGNER, YUCHEN ZHANG und JAN BROENINK: *RTnet - A Flexible Hard Real-Time Networking Framework*. 10th IEEE International Conference on Emerging Technologies and Factory Automation, IEEE, sep 2005.

[24] LABOR FÜR PROZESSDATENVERARBEITUNG DES STUDIENBEREICHS MECHATRONIK DER HOCHSCHULE REUTLINGEN: *Webseite Real-Time-Ethernet in der Industrieautomation*. `http://real-time-ethernet.de`, September 2006.

[25] LAN/MAN STANDARDS COMMITTEE OF THE IEEE COMPUTER SOCIETY: *IEEE Std 802.3-2002, Part3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*. `http://standards.ieee.org/getieee802/`, März 2002.

[26] LAN/MAN STANDARDS COMMITTEE OF THE IEEE COMPUTER SOCIETY: *IEEE Std 802.1Q-2003, Virtual Bridged Local Area Networks*. `http://standards.ieee.org/getieee802/`, März 2003.

[27] LAN/MAN STANDARDS COMMITTEE OF THE IEEE COMPUTER SOCIETY: *IEEE Std 802.1D-2004, Media Access Control (MAC) Bridges*. `http://standards.ieee.org/getieee802/`, Juni 2004.

[28] LUNA-VAZQUEZ, ISRAEL: *Implementation and Simulation of Routing Protocols for wireless sensor networks*. Diplomarbeit, Betriebssysteme und Verteilte Systeme Universität Siegen, März 2006.

[29] MATSUMOTO, MAKOTO und TAKUJI NISHIMURA: *Mersenne Twister PRNG Homepage*. `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html`.

[30] METCALFE, ROBERT M. und DAVID R. BOGGS: *Ethernet: Distributed Packet Switching for Local Computer Networks*. Communications of the ACM, 19(7):395–404, Juli 1976.

[31] ODVA: *ODVA Webseite*. `http://www.odva.org/`.

[32] OOSE INNOVATIVE INFORMATIK GMBH: *UML-Notationsübersicht*. http://www.oose.de/downloads/uml-notationsuebersicht.pdf.

[33] OPEN DEVICENET VENDOR ASSOC. AND CONTROLNET INTERNATIONAL: *EtherNet/IP specification, Volume 2: EtherNet/IP Adaption of CIP, Capther 1: Introduction to Ethernet/IP*. http://odva.org, 2006.

[34] PAGE, BERND, HANSJÖRG LIEBERT und ANDREAS HEYMANN: *Diskrete Simulation. Eine Einführung in Modula-2*. Springer, November 2001.

[35] PEETERS, KASPER: *Webseite zu tree.hh: an STL-like C++ tree class*. http://www.aei.mpg.de/~peekas/tree/.

[36] POPP, MANFRED: *Das PROFINET IO-Buch*. Hüthig Verlag Heidelberg, August 2005.

[37] POULSEN, KEVIN: *Slammer worm crashed Ohio nuke plant network*. SecurityFocus, August 2003.

[38] PROFIBUS INTERNATIONAL: *PROFIBUS International Webseite*. http://www.profibus.com/pi/.

[39] PROJEKTGRUPPE 'ERSTELLUNG EINES NETZWERK-SIMULATORS MIT JAVA UND C++': *Projektdokumentation*. Betriebssysteme und Verteilte Systeme Universität Siegen, Dezember 2006.

[40] PROJEKTGRUPPE 'INTERAKTIVE PROZESSKOMMUNIKATION IM MODELLBAU': *InProMo Abschlussbericht*. Technische Informatik Universität Siegen, Oktober 2005.

[41] RTNET: *RTnet Webseite*. http://www.rts.uni-hannover.de/rtnet/.

[42] SCHEITLIN, HANS: *Das industrial Ethernet und seine Anwendungsschichten - Teil 1: Ethernet und die Protokolle - ein Überblick*. Elektronik, (8):48–54, 2001.

[43] SCHNEIER, BRUCE: *Secret and Lies - IT Sicherheit in einer vernetzen Welt*. Wiley, dpunkt.Verlag, 2000.

[44] SCHWAGER, JÜRGEN: *Ethernet erreicht das Feld - Sechs Echtzeitvarianten im Vergleich - Teil 1*. Elektronik, (11):48–54, 2004.

[45] SCHWAGER, JÜRGEN: *Ethernet erreicht das Feld - Sechs Echtzeitvarianten im Vergleich - Teil 2*. Elektronik, (13):38–43, 2004.

[46] SIEMENS AG: *PROFIBUS - Das Multitalent für die Kommunikation in der Prozessindustrie*. https://pcs.khe.siemens.com/efiles/pcs7/pdf/00/prdbrief/kb_profibus_de.pdf, Mai 2006.

[47] SIEMENS AG - AUTOMATION AND DRIVES: *PROFINET Produktinformationen - Protokoll*. http://support.automation.siemens.com/WW/view/de/22055264.

[48] SIEMENS AG - AUTOMATION AND DRIVES: *PROFINET Produktinformationen - Zyklusverlauf*. http://support.automation.siemens.com/WW/view/de/22056476.

[49] SKONNARD, AARON und MARTING GUDGIN: *Essentiell XML - Quick Reference*. Addison-Wesley, 2003.

[50] STOREY, NEIL: *Safety Critical Computer Systems*. Prentice Hall, 1996.

[51] TANENBAUM, ANDREW S.: *Computernetzwerke*. Pearson Studium, 4. Auflage, 2003.

[52] VARGA, ANDRAS: *OMNeT++ - Discrete Event Simulation System, User Manual Version, 3.2*, März 2005.

[53] WIKIPEDIA, DEUTSCHSPRACHIGE: *Artikel Ethernet, Wikipedia*. `http://de.wikipedia.org/w/index.php?title=Ethernet&oldid=25086078`, Dezember 2006.

[54] WISMÜLLER, ROLAND: *Folien zur Vorlesung Rechnernetze II an der Universität Siegen*. `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/ws05/rn2/v04_4.pdf`, November 2005.

[55] WOLLSCHÄGER, MARTIN: *Folien zur Vorlesung 'Ethernet-basierte Systeme in der Industrie' an der TU Dresden*. `http://iai82110.inf.tu-dresden.de/lehre/Ethernet_in_Automation.pdf`, 2005.

Danksagung

Zunächst möchte ich mich bei meinen Betreuer Dipl. Inf. Frank Dopatka für die hervorragende Betreuung bedanken. Ohne die zahlreichen anregenden Diskussionen mit ihm wäre ich nicht in der Lage gewesen, die Arbeit in dieser Form fertig zu stellen. Des Weiteren geht mein Dank an Prof. Dr. Roland Wismüller, dass er es mir ermöglicht hat, in der Fachgruppe „Betriebssysteme und verteilte Systeme“ dieses Thema zu bearbeiten. Weiterhin möchte ich Alexandra Krumm und Benjamin Pieck, die mir während dieser Zeit zur Seite standen und die diese Arbeit mehrfach gelesen haben, meinen Dank aussprechen. Schlussendlich bin ich meiner Familie zu Dank verpflichtet, ohne ihre Unterstützung hätte ich das Studium der „Angewandten Informatik“ niemals abschließen können.