Bachelorarbeit

Fabian Kempf

Simulation von AFDX-Netzwerken basierend auf Rate-Constrained Traffic für Time-Triggered Ethernet in OMNeT++

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Fabian Kempf

Simulation von AFDX-Netzwerken basierend auf Rate-Constrained Traffic für Time-Triggered Ethernet in OMNeT++

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Korf
Zweitgutachter : Prof. Dr. Meisel

Abgegeben am 23. August 2011

Thema dieser Bachelorarbeit

Simulation von AFDX-Netzwerken basierend auf Rate-Constrained Traffic für Time-Triggered Ethernet in OMNeT++

Stichwörter

Echtzeit Ethernet, Simulation, OMNeT++, Rate-Constrained,Time-Triggered Ethernet, TTTech, Switch, AFDX, Virtual Link

Kurzzusammenfassung

Diese Arbeit befasst sich mit der diskreten, eventbasierten Simulation einer Echtzeit-Ethernet Lösung in der OMNeT++ Simulationsumgebung. Als Grundlage dient dabei die Rate-Constrained Traffic Klasse des von TTTech entwickelten Time-Triggered Ethernet Protokolls, das drei unterschiedliche Arten des Netzwerk-Traffics über den gleichen Nachrichtenkanal überträgt. Hierbei werden die Konzepte des AFDX-Protokolls beschrieben, auf denen der zu implementierende Modus basiert. Nach dem Entwurf und der Implementierung der benötigten Komponenten wird das Verhalten des Modells in unterschiedlichen Szenarien validiert und anschließend kleinere Netzwerke analysiert.

Topic of this Bachelorsthesis

Simulation of AFDX-Networks based on Rate-Constrained Traffic for Time-Triggered Ethernet in OMNeT++

Keywords

Real-time Ethernet, Simulation, OMNeT++, Rate-Constrained,Time-Triggered Ethernet, TTTech, Switch, AFDX, Virtual Link

Abstract

This thesis deals with a discrete eventbased simulation of a Realtime-Ethernet solution in the OMNeT++ simulation environment. As base principle serves the Rate-constrained traffic-type. It belongs to the Time-Triggered-Ethernet protocol developed by TTTech, which provides three message types over the same network. The implemented mode is based upon the AFDX-Protocol, which concepts will be described. After the design and implementation of the required components the performance of the model becomes validated in different scenarios and analysed in smaller networks afterwards.

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die durch ihre Unterstützung zum Gelingen dieser Bachelorarbeit beigetragen haben.

Mein besonderer Dank gilt Herrn Prof. Dr. Franz Korf für die Förderung des RTEthernet-Projekts, ohne das diese Arbeit nicht entstanden wäre und die stets konstruktive Kritik.

Ebenfalls möchte ich mich bei der RTEthernet-Gruppe der HAW für die konstruktiven Gespräche und die Diskussionsbereitschaft bedanken, durch die mir immer wieder neue Anstöße und Ideen für die Bearbeitung des Themas gegeben wurden.

Des Weiteren möchte ich mich bei Till Steinbach und Hermand Dieumo Kenfack bedanken, die für die Simulationsgrundlage dieser Arbeit sorgten und in Diskussionen die Lösung von Problemstellungen unterstützten.

Außerdem bedanke ich mich bei meinen Eltern und meiner Freundin Nadine für die Unterstützung während des Studiums und der Entstehungszeit dieser Arbeit.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Der Leistungszuwachs der IT in einem Automobil ist durch einen ständig wachsenden Prozess gekennzeichnet. Bis zu 70 dedizierte Kleincomputer, auch Steuergeräte genannt, überwachen Funktionen des Fahrzeugs und unterstützen den Fahrer durch Fahrerassistenzsysteme (vgl. Saad (2003), S.03). Zudem finden Informationssysteme und Unterhaltungselektronik immer mehr Einlass in ein heutiges Mittelklassefahrzeug. Somit haben wir es mit einem Zusammenschluss unabhängiger Computer zu tun, die sich für den Benutzer, in diesem Fall den Fahrzeugführer, als ein System präsentieren. Dies wird verteiltes System genannt (vgl. Tanenbaum und van Steen (2007)). Das Kommunizieren zwischen den Systemen wird heutzutage immer entscheidender. Während zum einen die Übertragungsgeschwindigkeit bei Entertainmentsystemen wie Videostreams eine Rolle spielt und zunehmend auch Front- und Heckkameras im Automobil eingesetzt werden, gibt es auf der anderen Seite Anforderungen an die Aktualität der Daten. Somit ist es in Zukunft zwingend notwendig zu wissen, wann das Bild der Frontkamera den Passanten auf dem Zebrastreifen erfasst.

Noch wichtiger als bei unterstützenden Systemen –wie der Personenerfassung– ist die genaue Zeitvorgabe von Datenübertragungen bei sicherheitskritischen Technologien. Als Beispiel lässt sich “X-by-Wire” nennen. Hierbei handelt es sich um eine Technologie, bei der Steuerbefehle über Datenleitungen übertragen werden und somit die mechanische Übertragung ersetzt wird. Dies ist im Flugzeug längst Standard und heißt “Fly-by-Wire”. In modernen Flugzeugen –wie dem A380– wird im Kabinenbereich ein ethernetbasierter Ansatz gewählt. Ethernet bezeichnet den am weitest verbreiteten Standard für lokale Netze (vgl. Institute of Electrical and Electronics Engineers (2005)) und sowohl die technische Planung, als auch die Instandhaltung sind vergleichsweise kostengünstig im Gegensatz zu anderen Bussystemen (vgl. GE Fanuc Intelligent Platforms). Allerdings gab es vor ca. 30 Jahren, als Ethernet entwickelt wurde, noch keine zeitkritischen, sicherheitsrelevanten Anwendungen, die ein deterministisches, also berechenbares Ethernet benötigten.

Ziel ist es somit, ein Netzwerk aufzubauen, was bestimmte Zeiten einhält und eine Datenkontrolle besitzt, um eine Echtzeit-Kommunikation im Automobil zu ermöglichen. Time-triggered Ethernet (TTE) (Steiner (2008)) besitzt einen solchen Ansatz und erweitert

das Standard-Ethernet mit berechnbaren Elementen. Hier werden drei verschiedene Trafficarten definiert, die über das gleiche Netzwerk mit unterschiedlicher Priorität laufen (vgl. Mikolasek u. a. (2008)). Einerseits können normale Ethernetframes verschickt werden, wodurch die Skalierbarkeit nach außen hin gewahrt bleibt, andererseits sind aber auch zeitbeziehungsweise bandbreitengesteuerte Nachrichten zwischen den Netzteilnehmern austauschbar. Eine Besonderheit bei TTE ist die Kompatibilität zu dem in modernen Flugzeugen etablierten Avionics Full-Duplex Switched Ethernet (AFDX). In der Industrie gibt es zusammenfassend einen fortschreitenden Einsatz von Echtzeit-Ethernet, der als Austausch von alten Feldbussystemen dient. Um diese ohne große Anschaffungskosten für neue Hardware zu testen und Leistungsdaten spezifischer Szenarien zu erlangen, ist Simulation eine sehr nützliche Technik (Jasperneite u. a. (2004)). Die Simulation muss dabei so weit wie möglich die realen Bedingungen widerspiegeln, so dass im Idealfall gleiche Ergebnisse erzielt werden, wobei die Simulationszeit kürzer als die Realzeit ist. Am Ende dient die Simulation der kostengünstigen Untersuchung komplexer TT-Ethernet-Netzstrukturen.

1.2 Bisheriger Simulationsstand

Als Basis für die Entwicklung wurde die eventbasierte OMNeT++ Simulationsumgebung (OMNeT++ Community (b)) und das dazugehörige INET Framework (OMNeT++ Community (a)) genutzt. Aufgrund der Eignung zur Simulation von Kommunikations- und ereignisbasierten Netzwerken wurde diese Entwicklungsumgebung gewählt. Außerdem gibt es mit dem INET-Framework ein implementiertes Ethernetmodell, das als Grundbaustein dient. Die bisher realisierte Simulation in OMNeT++ umfasst einen Switch, der von Till Steinbach im Verlauf seines Masterstudiums entworfen und implementiert wurde (Steinbach (2011)). Dieser ist in der Lage, die Konfigurationsdateien, die sich an das XML-Schema von TTTech halten, einzulesen und Time-triggered (TT) Nachrichten gemäß dieser Anweisungen an den richtigen Port zum gesetzten Zeitpunkt weiterzuleiten. Zudem kann der normale Ethernetverkehr seine Ziele erreichen, wenn kein Schedule zum Übertragen von höher priorisierten TT-Nachrichten ansteht.

Das Endsystem wurde von Hermand Dieumo Kenfack (Dieumo Kenfack (2010)) ebenfalls während seines Masterstudiums entworfen. Im Endsystem ist es genau wie im Switch möglich, TT-Schedules in den Konfigurationsdateien zu erstellen und diese in einem entworfenen Netzwerk zu visualisieren. Hierzu wurde eine Applikation erstellt, die zu gewissen Zeitpunkten Nachrichten erzeugt und dann über die TT-Programmierschnittstelle konform zum Schedule verschickt. In einer Konfigurationsdatei von OMNeT++ kann zudem festgelegt werden, ob die Applikation fähig sein soll, Time-triggered (TT) oder Best-Effort (BE) Nachrichten zu senden und/oder zu empfangen.

1.3 Zielsetzung und Abgrenzung

Ziel dieser Arbeit ist es, die bisherige Simulation in OMNeT++ um den Rate-Constrained-Traffic-Modus zu erweitern, sodass dieser Nachrichtentyp gemäß der Konfigurationsdateien verschickt, weitergeleitet und empfangen werden kann. Hierzu werden Teile aus der ARINC Spezifikation 664 Part 7 (Aeronautical Radio Incorporated (2009)) umgesetzt und der Switch sowie das Endsystem angepasst. Anschließend wird in der Validierung die Funktionalität der geänderten Komponenten festgestellt. Wenn das Grundgerüst zum Versenden aller drei Nachrichtentypen fertiggestellt ist, soll eine Anwendung fähig sein, eine Datei einzulesen und zu anderen Netzteilnehmern zu verschicken. Ein kleineres Netzwerk zeigt die Funktions- und Leistungsfähigkeit der gesamten Simulation.

Die Synchronisation, die auf dem IEEE 1588-Protokoll basiert und zu einer gleichen Zeitbasis zwischen den Systemen im Netzwerk führt, wird hier nicht weiter behandelt, weil der Rate-Constrained Traffic Modus keiner Zeitsynchronisation bedarf. Durch den Ablauf der Simulation innerhalb eines Systems und die daraus resultierende selbe Zeitbasis müssen keine zusätzlichen Nachrichten zur Synchronisation verschickt werden. Der bereits integrierte Time-Triggered Traffic Modus bezieht sich auf diese “globale” Zeit.

1.4 Inhaltlicher Aufbau der Arbeit

In Kapitel 2 werden Grundlagen über die in dieser Arbeit benutzen Konzepte von TTEthernet und dem AFDX-Protokoll aufgeführt. Zunächst wird erklärt, warum TTTech TTethernet entwickelt, und wie die drei unterschiedlichen Trafficarten von TTEthernet, nämlich Time-Triggered-, Rate-Constrained- und Best-Effort-Traffic, funktionieren. Anschließend sollen die Konzepte von AFDX erläutert werden. Dazu gibt es Beispiele, die das Verhalten verdeutlichen. Im letzten Teil des Grundlagenkapitels kommt es zur Vorstellung der Simulationsumgebung OMNeT++, in der die Simulation dieser Arbeit durchgeführt wurde.

Der Hauptteil beginnt in Kapitel 3. Designentscheidungen werden hier erklärt und Konzepte für die entworfenen Komponenten mit Hilfe von Aktivitätsdiagrammen verdeutlicht. Dabei wird zuerst das Endsystem vorgenommen, weil hier mehr Veränderungen nötig sind als im Switch. Die neuen Funktionen und Komponenten des Switches werden anschließend bearbeitet. Hier wird jedoch überwiegend auf die bisherigen Konzepte von Till Steinbach (Steinbach (2011)) zurückgegriffen, die das Weiterleiten von Nachrichten bereits ermöglichen.

2 Grundlagen

In diesem Kapitel werden die Grundlagen besprochen, die wichtig sind, um zu verstehen, wie Time-triggered Ethernet und das AFDX Protokoll funktionieren und zusammenhängen, und warum TTTech diese Form des Ethernets entwickelt. Dazu wird im Time-Triggered Ethernet Teil (2.1) die Etherneterweiterung TTEthernet behandelt. Hier findet man neben einer Definition von Echtzeitfähigkeit, die Beschreibung der Nachrichtenklassen Time-Triggered- (TT), Rate-Constrained- (RC) und Best-Effort-Traffic (BE). Im Avionics Full-Duplex Switched Ethernet Part (2.2) wird ein Überblick über das AFDX-Protokoll gegeben und der Zusammenhang zu Rate-Consrained hergestellt. Im dritten und letzten Absatz (2.3) wird die eingesetzte Simulationsumgebung OMNeT++ beschrieben, die Arbeitsweise mit ihr und die gegebenen Möglichkeiten im Rahmen dieser Arbeit.

2.1 Time-Triggered Ethernet

Der heutige Standard für drahtgebundene Vernetzungstechnik ist Standard-Ethernet. “Jedes netzwerkfähige Produkt - vom einfachsten 10-Euro-Switch bis zum digitalen Videorecorder - besitzt heute einen Fast-Ethernet-Anschluss für 100Mbit/s” (vgl.Rech (2007)). Allerdings verfügt Ethernet in seiner ursprünglichen Form nicht über eine Echtzeitfähigkeit. Das heißt, dass Nachrichten, die über ein ethernetbasiertes Netzwerk gehen, nicht zu einer garantierten Zeit beim Empfänger eintreffen. Sollen die Systeme also echtzeitfähig miteinander kommunizieren, dürfen Ereignisse, getriggert durch das Eintreffen von Nachrichten, nur innerhalb eines vordefinierten Zeitraums ausgelöst werden.

Ein Airbag im Automobil beispielsweise muss sich typischerweise bei einem Unfall je nach Geschwindigkeit zwischen $10ms$ und $40ms$ explosionsartig aufblasen. Wenn er zu früh aufgeht oder sich zu spät öffnet, kann dies einen nicht beabsichtigten Effekt auf den zu schützenden Fahrzeuginsassen haben. Somit ist es wichtig, dass die Nachricht zur Ereignisauslösung genau im Intervall eintrifft und nicht –wie häufig falsch angenommen– so schnell wie möglich.

Um Ethernet diese Echtzeitfähigkeit zu verleihen, wurde Time-Triggered Ethernet (TTE) von TTTech entwickelt. Ursprünglich entstand dies in der Real-Time Systems Group (vgl.

Real Time Systems Group (RTS)) an der TU Wien im Jahr 2004. TTTech übernahm die Konzepte von TTEthernet und bietet diese in erweiterter Form kommerziell in der Automobilindustrie an. Die Aufgabe von TTE ist es, unterschiedliche Kommunikationsarten über den gleichen Weg laufen zu lassen. Nachrichten mit niedriger priorisiertem Inhalt sollen nebenläufig zu sicherheitskritischen Nachrichten übertragen werden, ohne diese in ihrer Echtzeitfähigkeit einzuschränken. Zusätzlich soll es möglich sein, genügend Bandbreite für jede registrierte Applikation zur Verfügung zu stellen, damit Videostreams nicht plötzlich anhalten, wenn andere Applikationen versuchen, Daten zu schicken. Neben den registrierten Anwendungen dürfen aber auch nichtregistrierte an den Konversationen teilnehmen, die nur über das Ethernetprotokoll miteinander kommunizieren. Dies kann wie im normalen Ethernetbetrieb jederzeit und frei skaliert werden. (vgl. Steiner (2008))

Im Folgenden wird beschrieben wie Time-Triggered Ethernet versucht, den verschiedenen Kriterien gerecht zu werden.

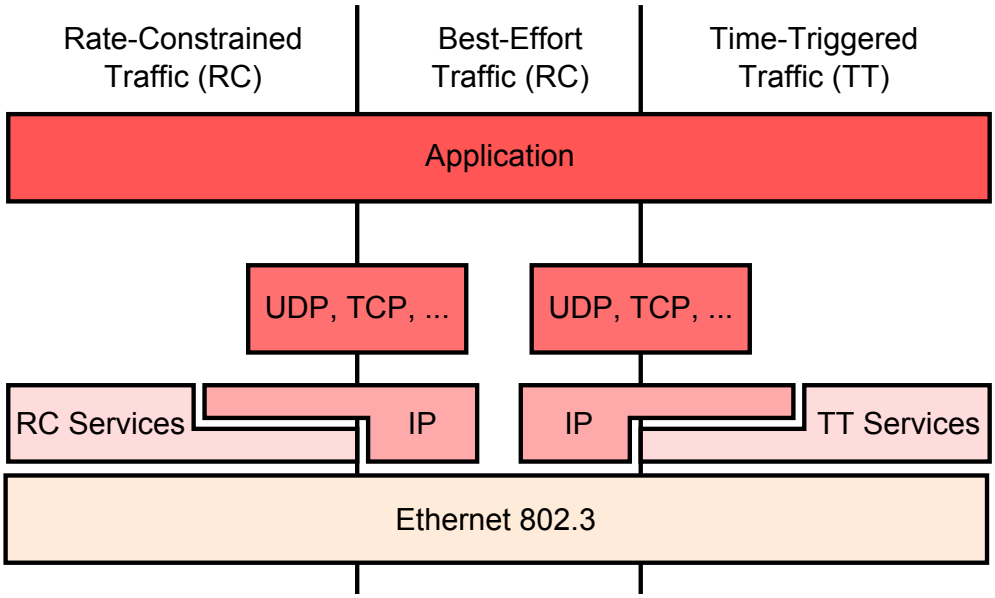

Abbildung 2.1: *Die eingeführte Serviceebene gibt der Ethernetschicht Echtzeitfähigkeit. Im Applicationlayer wird festgelegt, über welche Nachrichtenklassen gesendet wird. (Steiner u. a. (2009))*

Wie in Abbildung 2.1 zu sehen ist, definiert TTE einen Servicelayer, der direkt auf der Ethernet 802.3 Schicht aufbaut und somit zeitgesteuerte Kommunikation oberhalb der unteren beiden Layer (OSI-Layer 1-2) erlaubt. Nachrichten aus höher gelegenen Protokollen, wie zum Beispiel dem Transmission Control Protocol (TCP) oder User Datagram Protocol (UDP), können somit problemlos zeitgesteuert übertragen werden, ohne den Header der Nachrichten zu verändern.

TTEthernet definiert drei verschiedene Nachrichtenklassen, die je nach Bedarf für verschiedene Aufgaben genutzt werden können.

2.1.1 Time-Triggered-Traffic (TT)

Diese zeitgesteuerten Nachrichten werden von Anwendungen genutzt, die eine sehr hohe Zuverlässigkeit erfordern. Die Nachrichten haben die höchste Priorität im Netzwerk und werden an vorbestimmten Zeitpunkten versand. Dazu synchronisieren die teilnehmenden Systeme ihre interne Clock über Protocol Control Frames (PCFs), wodurch eine Genauigkeit von unter einer Mikrosekunde (vgl. Steiner (2008)) erreicht wird. Diese Zeit wird durch ein deterministisches System, das offline konfiguriert werden muss, garantiert.

TT-Nachrichten werden somit in zeitkritischen Systemen, wie Brake- oder Steer-by-Wire Systemen, eingesetzt.

2.1.2 Rate-Constrained-Traffic (RC)

Rate-Constrained Nachrichten sind im Gegensatz zu Nachrichten aus der TT-Traffic-Klasse nicht an vordefinierte Zeitpunkte zum Verschicken gebunden. Jeder RC-Sender im Netzwerk realisiert eine Traffic Shaping Funktion, die eine Verzögerung zwischen zwei RC-Nachrichten mit selber ErkennungsID (siehe Virtual Link (2.2.1)) auslöst. Im Switch wird diese Verzögerung überprüft, sodass Nachrichten, die zu schnell nacheinander eintreffen, verworfen werden. Somit steht jedem Endsystem eine gewisse Bandbreite zur Verfügung, wodurch es die anderen Systeme nicht beeinträchtigt.

Der Rate-Constrained-Traffic-Modus unterstützt zu großen Teilen die Konzepte vom AFDX-Protokoll, das im Abschnitt 2.2 beschrieben wird.

2.1.3 Best-Effort-Traffic (BE)

Best-Effort Nachrichten werden wie normale Ethernetframes behandelt. Sie sind die am geringsten priorisierten Nachrichten von TTEthernet und werden nur transferiert, falls keine TT oder RC Nachrichten zum Versenden anstehen. Diese Klasse nutzt somit die Restbandbreite des Netzwerkes aus, und es ist nicht gewährleistet, ob ihre Pakete ankommen. Ethernet kann vollständig auf Best-Effort Nachrichten abgebildet werden, wodurch ein fest konfiguriertes TTEthernet Netzwerk nach außen hin vergrößert werden kann und trotzdem seine Echtzeitfähigkeit beibehält.

2.2 Avionics Full-Duplex Switched Ethernet

Das Avionics Full-Duplex Switched Ethernet (AFDX), das im ARINC-664 Part 7 (Aeronautical Radio Incorporated (2009)) spezifiziert wurde, stellt ein geschlossenes Computernetzwerk dar und basiert auf dem IEEE 802.3 Ethernet Standard (Institute of Electrical and Electronics Engineers (2005)). AFDX wurde von der Firma Aeronautical Radio, Incorporated (ARINC) spezifiziert. Die Firma entwickelt seit 1929 Standards für Flugzeugelektronik ((Fluehr, 2010, S. 2ff.)) und hat unter anderem den klassischen Datenbus für Verkehrsflugzeuge im Protokoll ARINC-429 definiert.

Bei ARINC-429 handelt es sich um ein point-to-multi-point Netzwerk mit unidirektionalen Verbindungen. Aus diesem Grund muss jeder Empfänger direkt mit dem Sender verbunden sein, und die physikalisch gelegten Verbindungen sind nur in einer Richtung verwendbar. Somit können die einzelnen Controller nur mittels umfangreicher Kabelbäume miteinander verdrahtet werden, wobei an einen Sender bis zu 20 Empfänger anschließbar sind.

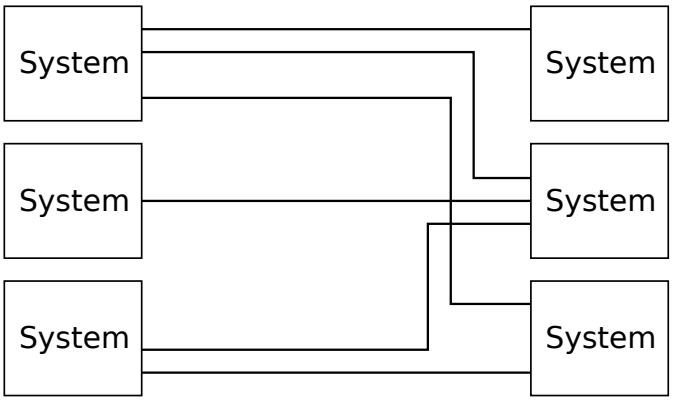

Abbildung 2.2: *ARINC-429 Systeme benötigen mehrere Anschlüsse zu verschiedenen Systemen, um mit diesen zu kommunizieren.*

AFDX hingegen stellt ein sternförmiges Netzwerk dar, bei dem jedes System über einen Switch mit anderen Systemen verbunden ist, wie man es beim Standardethernet vorfindet. Um die Kompatibilität zum vorherigen Standard zu gewährleisten, wurden Virtual Links (2.2.1) eingeführt. Diese bilden unidirektionale logische Verbindungen auf verschiedenen Übertragungsmedien ab. (vgl. Bob Pickles (2006))

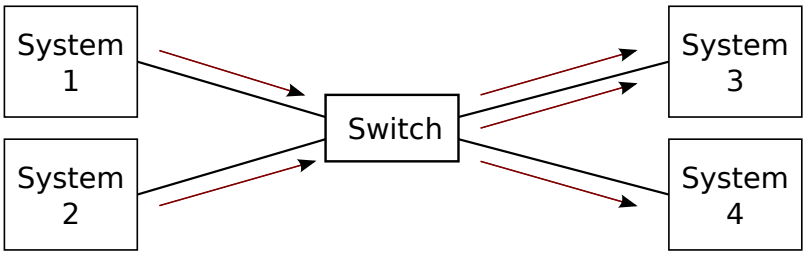

Abbildung 2.3: *Ein ethernetbasiertes Netzwerk, in dem jedes System mit einem Kabel verbunden ist. Die Pfeile bilden unidirektionale logische Verbindungen ab. System 3 besitzt zwei Pfeile in die gleiche Richtung, weil System 1 sowie System 2 logisch mit diesem verbunden sind.*

Abbildung 2.3 zeigt ein Netzwerk mit vier Systemen, die jeweils mit dem Switch über ein Kabel verbunden sind. Jedes dieser Systeme besitzt logische Verbindungen, die durch rote Pfeile gekennzeichnet sind.

2.2.1 Virtual Link

Virtual Links (VLs) sind logische, unidirektionale Verbindungen von einem Endsystem zu einem oder mehreren anderen. Sie stellen also eine 1-zu-1 oder 1-zu-n Beziehung dar. Anders als in einem herkömmlichen Switch werden AFDX-Frames nicht über die herkömmliche Ethernet-Destination-Address gerouted, sondern über die 16 Bits lange Virtual Link ID.

48 bits	
Constant Field 32 bits	Virtual Link Identifier 16 bits
xxxx xx11 xxxx xxxx xxxx xxxx xxxx xxxx	

Abbildung 2.4: *Unterteilung eines AFDX-Frames. Die vorderen 32 Bits beschreiben das Constant Field. Die letzten 16 Bits sind die Virtual Link ID, aus der die Zugehörigkeit entnommen wird. (Quelle: Aeronautical Radio Incorporated (2009), S. 17)*

Die Virtual Link ID, auch VLID genannt, beschreibt einen im Switch festgelegten Pfad und bestimmt, auf welchen Ports die einkommenden AFDX-Frames weitergeleitet werden. Dies wird zuvor in einer Konfigurationsdatei für jeden Switch und jedes Endsystem vom Netzwerkadministrator festgelegt.

Das Constant Field, also die ersten 32 Bits des AFDX-Frames, sollten für jedes AFDX-Endsystem gleich sein, um die Zugehörigkeit zum Netzwerk zu kennzeichnen. Um zu zeigen, dass es sich um eine Gruppen- und lokal administrierte Adresse handelt, sind das Least Significant Bit (LSB) des zweiten Bytes und das Bit davor auf eins gesetzt. (Aeronautical Radio Incorporated (2009))

Abbildung 2.5 veranschaulicht die Funktionalität von VLs.

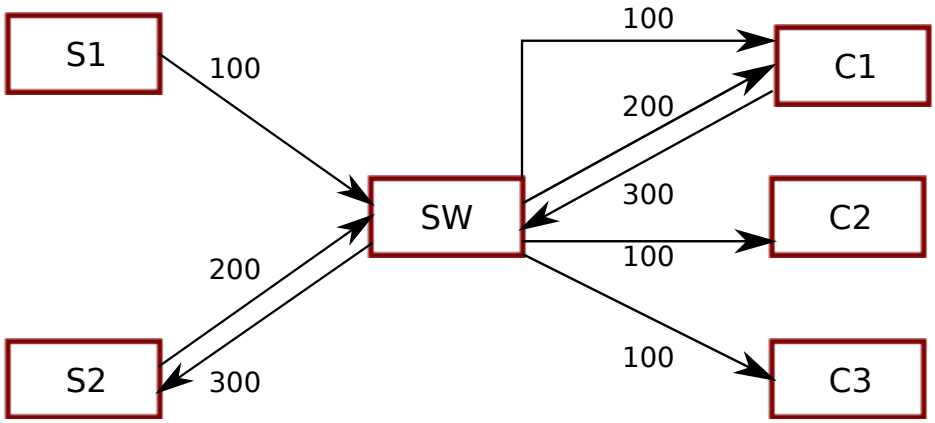

Abbildung 2.5: *Fünf Endsysteme an verschiedenen Ports des Switches kommunizieren über drei verschiedene Virtual Links miteinander.*

Abbildung 2.5 zeigt, dass das Endsystem S1 an die Endsysteme C1, C2 und C3 Nachrichten senden kann, wenn der Virtual Link Identifier der Frames auf 100 gesetzt wurde. Hierbei kann es sich beispielsweise um einen Videoserver handeln, der über diesen Virtual Link an verschiedene Videoclienten Videobotschaften schickt. Das Endsystem S2 sendet über die VL-ID 200 an den Clienten C3 Nachrichten, der wiederum über die VL-ID 300 Anfragen beantworten kann. Durch die Einhaltung der Bandwidth Allocation Gaps (BAGs) (2.2.2), die der nächste Abschnitt beschreibt, ist garantiert, dass Botschaften vom Endsystem S2 über die VL-ID 200 an den Clienten C3 übertragen werden, ohne die Frames des Videostreams von S1 zu C1 zu verzögern oder zu unterbrechen.

2.2.2 Bandwidth Allocation Gap (BAG)

Bandwidth Allocation Gap, kurz BAG genannt, ist ein Parameter eines Virtual Links (2.2.1), der das minimale Intervall zwischen zwei Nachrichten, die zu einem VL gehören, und ihre maximale Länge beschreibt.

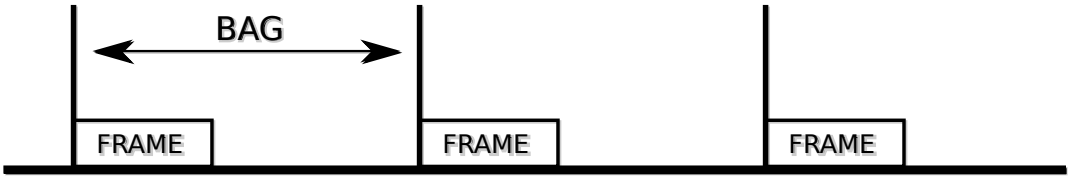

Abbildung 2.6: *Die Bandwidth Allocation Gap (GAP) lässt das erneute Senden eines Frames erst nach Ablauf der Zeit wieder zu. (Quelle: Aeronautical Radio Incorporated (2009), S.11)*

BAG-Values werden laut ARINC-664 Standard in Millisekunden angegeben und betragen zwischen 1ms und 128ms. Der Standard definiert die BAG-Values in Zweierpotenzen, welches bei dem angegebenen Intervall 8 verschiedene Werte ergibt. Durch die maximale Länge eines Frames von 1.518 Bytes ergibt sich für einen VL mit einem BAG-Value von 1ms die maximale Transmission von 1.518.000 Bytes pro Sekunde. Die maximale Transmission bei einem BAG-Value von 128ms beträgt 11.859 Bytes. Bei mehreren VLs muss also vom Netzwerkadministrator sichergestellt werden, dass die Prioritäten der einzelnen VLs Anwendungen nicht in den Hintergrund stellen und die Bandbreite für die jeweilige Applikation ausreicht. Im Vergleich zu Ethernet sind diese Übertragungsraten extrem langsam, jedoch verlangt das AFDX-Protokoll Determinismus für mehrere sicherheitskritische Systeme. Bei TTEthernet sind Werte ab $1\mu s$ für den BAG-Value vorgesehen. Dies ist gerade beim Automobil sehr wichtig, da dort die Zyklen der sicherheitskritischen Systeme wesentlich geringer ausfallen als im Flugzeug. Damit ist eine Erweiterung zum AFDX-Standard gegeben, die ihn vollends unterstützt.

2.2.3 Virtual Link Scheduling

Jeder sendende Port eines AFDX-Systems steht über ein oder mehrere VLs mit anderen Endsystemen in Verbindung. Wenn der Port beziehungsweise das Endsystem mehrere VLs besitzt, muss entschieden werden, welcher VL als nächstes das Recht zu senden hat. Hierfür ist das Virtual Link Scheduling (VLS)) zuständig. Es entscheidet welcher VL als nächstes aus seiner Nachrichtenqueue über den Port eine Nachricht versenden darf. Hierzu muss die BAG und die maximale Länge, die ein Frame haben kann, beachtet werden.

Abbildung 2.7 hält ein Beispiel mit drei verschiedenen Virtual Links bereit. Diese werden mit den VL-IDs 100, 200 und 300 markiert. Jeder dieser Virtual Links hat eine BAG (2.2.2), also eine Wartezeit, in der er nicht schicken darf. Die BAG des Virtual Links 100 ist auf 16 eingestellt. Somit wird ermöglicht, alle 16 ms Frames über diesen VL zu versenden. Der VL 200 kann mit einer BAG von 32 ms nur halb so schnell versenden, während der VL mit der

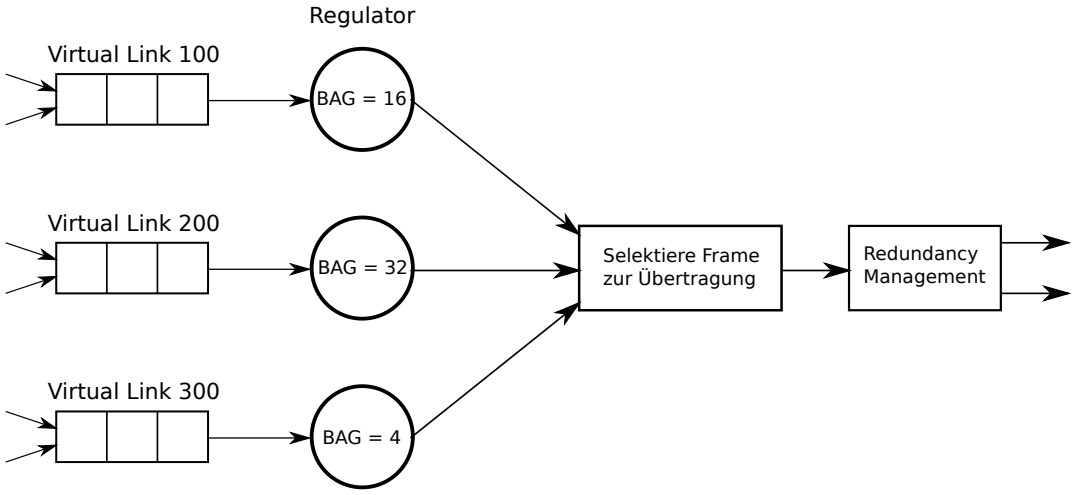

Abbildung 2.7: *Drei Virtual Links mit unterschiedlichen BAG-Values versuchen Nachrichten zu verschicken. Das Virtual Link Scheduling ist dafür verantwortlich, welcher Frame als nächstes gesendet wird. (Quelle: Condor Engineering Incorporated (2005), S.39)*

300er ID alle 4 ms Nachrichten verschicken darf. Die Berechnung, welcher VL als nächstes senden kann, findet anschließend statt und muss die Parameter, BAG, Priorität und Länge der Nachricht beachten. Durch das Redundanzmanagement (2.2.4) werden die Pakete –in entsprechender Zahl der Kanäle– an die unteren Layer weitergeleitet.

2.2.4 Redundanzmanagement

Ein AFDX-System besteht aus zwei voneinander unabhängigen Netzwerken um beispielsweise den Ausfall eines Switches kompensieren zu können und die Latenz so gering wie möglich zu halten. Daher senden angeschlossene Endsysteme ihre Pakete immer an beide Netzwerke, sodass an den empfangenden Endsystemen zwei Kopien der Pakete ankommen. Diese Aufgaben wird dem Redundanzmanagement zugeteilt. Um zu erkennen, ob das Paket bereits eingetroffen ist und somit verworfen werden kann, wird die 1-Byte Sequenz Nummer des AFDX-Frames überprüft.

Zuvor überprüft das empfangende Endsystem anhand der Sequenz Nummer, ob sich die ankommenden Pakete in der richtigen Reihenfolge („Integrity Check“) befinden. Ist dies nicht der Fall, können je nach Endsystem die Pakete verworfen oder an eine höher liegende Ebene weitergeleitet werden. Abbildung 2.8 fasst den Vorgang zusammen.

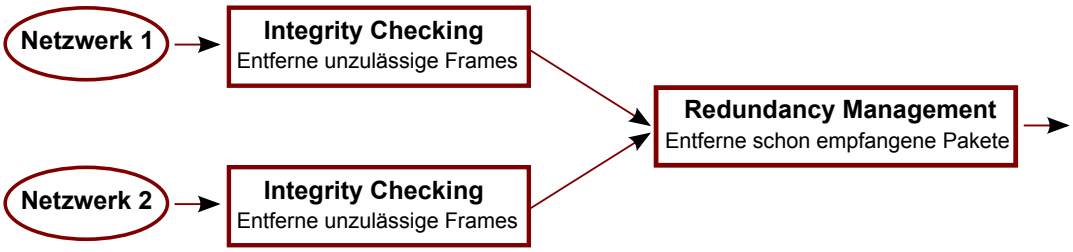

Abbildung 2.8: *Redundanzmanagement eines AFDX-Endsystems (Quelle: Condor Engineering Incorporated (2005), S. 26)*

2.2.5 Jitter

Obwohl Nachrichten in einem AFDX-Netzwerk bandbreitenlimitiert sind, kann in einem Endsystem mit mehreren VLs Jitter entstehen. Jitter nennt man den Effekt, der durch die Berechnungen des Schedules und der Zeitkontrolle im Endsystem entsteht (vgl. Bob Pickles (2006), S.5). Zudem entsteht er, wenn eine Nachricht über einen Virtual Link verschickt werden kann, weil die dazugehörige BAG abgelaufen ist, jedoch ein anderer Frame gerade übertragen wird. Somit verschiebt sich der gesamte Schedule des VLs um die restliche Übertragungsdauer des derzeitig gesendeten Frames.

In Beispiel 2.9 kommen sich die Pakete nur beim ersten Verschicken in die Quere. Der gesamte Schedule der roten Pakete wird um die Übertragungsdauer des ersten gelben Paketes verschoben.

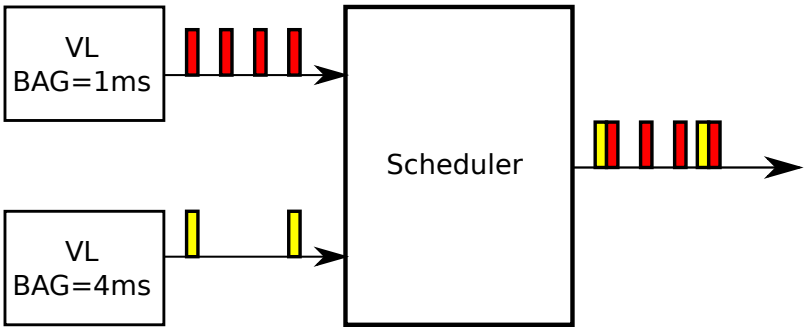

Abbildung 2.9: *Die rot gekennzeichneten Pakete werden später als die gelben gesendet, weil sie nicht zeitgleich übertragen werden können. Die zeitliche Verzögerung, die durch die Berechnung im Scheduler entsteht, nennt sich Jitter. (Quelle: Bob Pickles (2006), S.5)*

2.3 Grundlagen Simulationsumgebung

2.3.1 OMNeT++

Objective Modular Network Testbed in C++ (OMNeT++ (OMNeT++ Community (b))) ist eine diskrete, eventbasierte Simulationsumgebung zur Modellierung von Netzwerken und parallel arbeitenden, beziehungsweise verteilten Systemen. Die Umgebung ist Open-Source und für akademische sowie private Zwecke unter der GNU General Public License (GPL) nicht kommerziell verwendbar (Varga (2001)). OMNeT++ bietet eine Eclipse-basierte Entwicklungsumgebung mit einer grafischen Benutzerführung. Die Programmierung von Modulen findet in C++ statt, kann jedoch auch in anderen Programmiersprachen, wie beispielsweise Java, durchgeführt werden.

Module sind die Komponenten von OMNeT++. Sie können Protokolle wie das TCP darstellen, oder es kann sich um eine Hardwarekomponente wie einen Switch handeln. Somit wird eine Komponentenstruktur unterstützt, die das Wiederverwenden von Modulen fördert. Gut strukturierte Module sind einfach mit anderen Modulen verwendbar und können über Gates mit diesen verbunden werden. Nachrichten sind über Gates an andere Module verschickbar und werden Messages genannt. Abbildung 2.10 zeigt ein Netzwerk mit zwei Modulen, die über Gates miteinander verbunden sind.

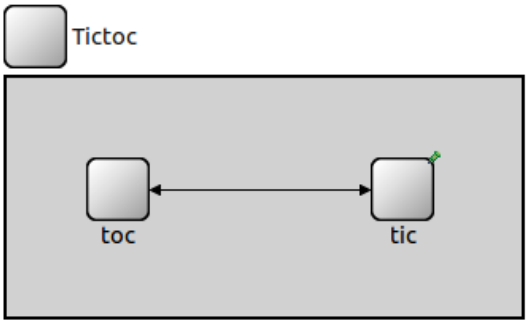

Abbildung 2.10: *Ein kleines OMNeT++ Netzwerk. Die Module Tic und Toc schicken sich gegenseitig Messages über Gates.*

Die Anweisungen der einzelnen Module wird in Network Description (NED)-Files angelegt. In der grafischen Benutzeroberfläche 2.10 kann per Drag-And-Drop ein Netzwerk zusammengestellt werden. Jederzeit kann dabei in die textuelle Ansicht der NED-Datei gewechselt werden, um weitere Anpassungen vorzunehmen. Spezifische Eigenschaften und Parameter sind in der *omnetpp.ini* eines Projekts zu setzen. Hierüber ist es ebenfalls möglich, Durchläufe mit verschiedenen Parametern zu beschreiben. Der folgende Codeabschnitt zeigt den Code der NED-File des TicToc-Beispiels.

```
network Tictoc
{
    submodules:
        tic: Txc;
        toc: Txc;
    connections:
        tic.out --> {delay = 100ms;} --> toc.in;
        tic.in <-- {delay = 100ms;} <-- toc.out;
}
```

Das Netzwerk hat zwei Submodule vom gleichen Typ Txc. Diese werden im Connectionsabschnitt der NED-Datei miteinander verbunden. Die Übertragungszeit wird dabei auf $100ms$ festgesetzt. Durch die explizite Angabe einer Richtung können sowohl unidirektionale Verbindungen, als auch Verbindungen, die in zwei Richtungen verwendbar sein sollen, angelegt werden.

Die eigentliche interne Logik der Komponenten steckt in den C++ Klassen. Wenn Nachrichten über ein input-Gate ein Modul erreichen, wird vom Simulations-Kernel die *handleMessage*-Funktion der dazugehörigen Klasse aufgerufen. Andere Module können Nachrichten über eine einfache *send*-Funktion verschicken, die als Übergabeparameter die Nachricht und eine ID für das Gate, auf der sie verschickt werden soll, besitzt. Außerdem kann ein Modul eigene Nachrichten zu bestimmten Zeitpunkten erzeugen. Dies ist wichtig, um beispielsweise einen Timer zu implementieren, der einen eigenen Zyklus besitzt. Diese Ereignisse werden über den *scheduleAt*-Funktionsaufruf ausgelöst. Dieser benötigt die zuvor berechnete Simulationszeit, an der die Message eintreffen soll, als Parameter.

2.3.2 INET Framework

Das INET Framework (OMNeT++ Community (a)), das als Grundlage für die Simulation dient, ist eine open-source Zusammenstellung verschiedener Internet-Protokolle für die OMNeT++ Simulationsumgebung. Dazu zählen unter anderem das Transmission Control Protocol (TCP), das Internetprotokoll (IP) und Ethernet. Durch den intensiven Einsatz von Vererbung und den modularen Aufbau der Komponenten, können die vorhandenen Elemente um Echtzeit-Eigenschaften erweitert werden. Ziel ist es die vorhandene Struktur des Frameworks zu übernehmen und das Standard-Ethernet um die TTEthernet-Logik zu erweitern. Dazu dient sowohl der Ethernet-Switch, als auch der Ethernet-Host als Grundlage für die jeweiligen TT-Ethernet-Komponenten.

3 Konzept und Architektur der TTEthernet-Komponenten

Dieses Kapitel befasst sich mit dem Konzept der Simulation. Hier werden Designentscheidungen zur konkreten Implementierung der Komponenten gefällt, die für die spätere Netzwerksimulation nötig sind.

Am Anfang wird der TTEthernet Protokollstack (3.1) behandelt und gezeigt, auf welcher Ebene Veränderungen durchgeführt werden müssen, um die Endsysteme mit AFDX zu erweitern.

Der darauf folgende Abschnitt beschäftigt sich mit den Veränderungen der einzelnen Komponenten, damit das Endsystem fähig ist, Nachrichten vom Rate-Constrained (2.1.2) Traffic Type zu versenden. Anschließend soll der TTE-Switch die RC Nachrichten AFDX-konform an das per Virtual Link (2.2.1) festgelegte Endsystem weiterleiten, sodass bei der BAG-Überprüfung (2.2.2) auf Empfängerseite die Nachrichten nicht zu schnell hintereinander ankommen und somit eingehalten werden.

3.1 TTEthernet Protokollstack

Der TTEthernet Protokollstack erweitert den Standard-Ethernet Protokollstack um die in den Grundlagen (2.1) beschriebenen Echtzeiteigenschaften. Wie in der Visualisierung 3.1 zu sehen ist, besteht der Stack aus drei Schichten des OSI-Referenzmodells (vgl. Boger (1999), S. 44), deren Implementation unbedingt notwendig sind. Auf unterster Ebene dient die Bitübertragungsschicht (Physical Link Layer), der Übertragung von Datenbits über ein physikalisches Medium. Diese kann direkt aus dem Standard-Ethernetmodell übernommen werden.

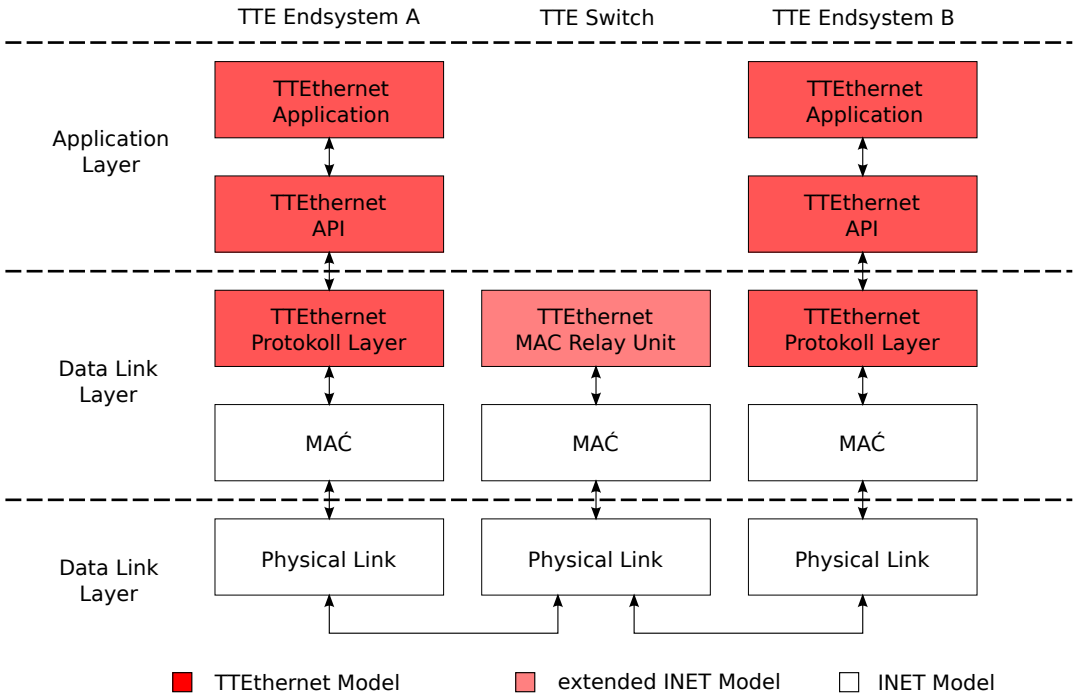

Abbildung 3.1: *Time-Triggered-Ethernet Protokollstack.*

Direkt über der Bitübertragungsschicht liegt die Sicherungsschicht (Data Link Layer). Hier werden die Datenbits der Übertragungsschicht zu maximal 1518 großen Datenframes zusammengefasst. Sie teilt sich in zwei verschiedene Subschichten auf. Die eine ist die MAC-Subschicht, die direkt mit der MAC-Subschicht des Ethernetprotokollstacks vom INET Framework übereinstimmt, die andere ist die Time-Triggered-Ethernet Protokollschicht. Hier wird die Echtzeitfähigkeit und somit die Erweiterung zu Standard-Ethernet verwirklicht. Diese Schicht wird genauer in Abschnitt 3.3 in der Komponentenmodellierung des Endsystems besprochen.

Die Applikationsebene ist die dritte und letzte Schicht, die in unserem Modell zum Tragen kommt. Auch dieser Layer ist in zwei Sublayer aufgeteilt. Und zwar die TTEthernet Application Programming Interface (API) und eine Applikationsschicht. Die API wurde von TTTech zur Verfügung gestellt und ist eine festgelegte Schnittstelle zur Initialisierung des Protokollstacks und um Nachrichten von der Applikationsebene aus zu verschicken und zu empfangen. Die Applikationsebene dient dazu, in der Simulation die TTEthernet API aufzurufen und Beispielprogramme zu verwirklichen, die über die Simulation und Time-Triggered-Ethernet miteinander kommunizieren. Durch das gleiche Interface, das auch in “echten” Anwendungen Verwendung findet, braucht es keine Portierung von Anwendungen aus der Simulation in echte Systeme.

Protokolle aus anderen Schichten sind kompatibel zu Time-Triggered Ethernet, weil TTE nur eine Erweiterung ist und nichts an der Schichtstruktur verändert. Somit können beispielsweise TCP-Verbindungen und UDP-Nachrichten, also Protokolle aus Schichten, die höher als Layer-2 liegen, über TTE aufgebaut und verschickt werden.

3.2 Konfiguration der Simulationskomponenten

Die Konfiguration der einzelnen Komponenten eines Netzwerks wird von TTEthernet über ein XML Format definiert. Sie wird *Network-Configuration* genannt und enthält die gesamte Beschreibung jeglicher Komponenten bis hin zum Scheduling. Wie genau die XML-Files eingelesen werden, ist der Masterarbeit von Till Steinbach zu entnehmen (vgl. Steinbach (2011), S. 24). Für die vorliegende Arbeit ist es wichtig zu wissen, in welchen Dateien die Parameter zur konkreten Beschreibung der Komponenten liegen. Jede Komponente hat dabei seine eigene *Device Specifikation*, in der gerätespezifische Parameter festgelegt werden, um beispielsweise die BAG der Virtual Links, die TT-Schedules und die Prioritäten zu setzen. Die Virtual Links werden zuvor in der *Virtual Link Map* festgelegt. Dazu gehört neben dem Namen des VLs die VL-ID. In der *System Specifikation* werden die Topologieanforderungen an das System gesetzt. Um die Vorstellung einer XML-Datei zu bekommen, gibt es hierzu ein Beispiel über die Festlegung der BAG in einer beispielhaften *Device Specifikation*.

Listing 3.1: *Definition der BAGs in einer Device-Specification der Network-Configuration*

```
1  ...
   <bagAccounts>
3      <incomingBagAccount
               name   ="RC_44_in"
5              bag    ="100us"
               jitter="0us"/>
7       <outgoingBagAccount
               name   ="RC_44_out"
9              bag    ="200us"
               jitter="0us"/>
11 </bagAccounts>
   ...
```

Das Listing 3.1 zeigt, wie der BAG-Value eines Virtual Links gesetzt wird. Hierbei ist zu beachten, dass diese Bagaccounts später von mehreren VLs genutzt werden können.

Mit dem TTEPlan Tool von TTTech soll es möglich sein, Netzwerke zu erstellen und daraus deren Konfigurationen zu erhalten. Leider wurde das Programm von TTTech bis zur Fertigstellung dieser Arbeit nicht ausgeliefert, sodass die Dateien durch ein Script erstellt werden mussten.

3.3 Komponenten des Simulationsmodells

Dieses Kapitel befasst sich mit den unterschiedlichen Komponenten, die nötig sind, damit das Simulationsmodell fähig ist, die drei verschiedenen Trafficarten Time-Triggered, Rate-Constrained und Best Effort gemäß der Spezifikation zu benutzen. Die Aufteilung der Komponenten im Endsystem wurde von Hermand Dieumo in Teilen seines Masterstudienganges realisiert. Hier bedarf es einiger Anpassungen, und es mussten Designentscheidungen gefällt werden, damit Rate-Constrained Nachrichten gemäß ARINC-664 Spezifikation verschickt und empfangen werden können. Der TTE-Switch wurde von Till Steinbach während seiner Masterarbeit angefertigt. Dieser ist genau wie das Endsystem in der Lage, TT und BE Nachrichten weiterzuleiten. Auch hier sind einige Entscheidungen zu treffen um die Simulation sowohl um den RC-Modus zu erweitern als auch die Performance zu verbessern.

Im Folgenden wird das Endsystem unterhalb der *TTEthernetAPI* und die Veränderungen, die auf der *TTE-Protokollebene* gemacht werden müssen, beschrieben. Anschließend werden Anforderungen an einer Beispielapplikation erklärt, die das Testen der Funktionen in der Komponentenvalidierung ermöglicht. Danach kommt es zur Switchmodellierung.

3.3.1 TTEthernet-Endsystemmodellierung

Die Komponenten, aus denen das Endsystem besteht, werden in Abbildung 3.2 visualisiert. Hierbei sind die drei OMNeT++ Module genApp, videoControServer und videoControClient im Moment zu vernachlässigen. Der Hauptteil des Endsystems wurde in zwei logische Blöcke unterteilt.

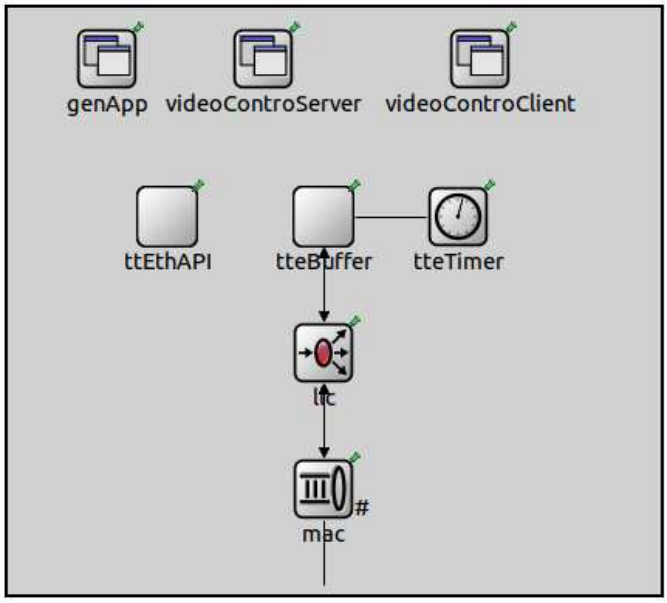

Abbildung 3.2: *Die Komponenten eines Endsystems. Der TTEBuffer enthält Nachrichtenqueues für die verschiedenen Trafficmodi und wird über den TTETimer getriggert. Anwendungen –wie der videoControServer– können über die TTEthAPI Nachrichten verschicken und somit das erweiterte Ethernet nutzen.*

Auf der einen Seite haben wir ein Modul, das *TTEBuffer* genannt wird. Der *TTEBuffer* hat seinen Namen aufgrund der Speichereigenschaften beim Versenden von Nachrichten eines TTE-Endsystems erhalten. Er enthält Queues und Buffer zum Zwischenspeichern von Nachrichten aus allen drei Trafficklassen. Außerdem ist er für das Scheduling verantwortlich und entscheidet somit, über welchen Virtual Link als nächstes gesendet wird.

Durch die drei Nachrichtenklassen ist eine Prioritätensteuerung wichtig, die festlegt, was als nächstes gesendet werden kann. Diese kann sowohl *preemptive* (höher priorisierte Nachrichten unterbrechen die Übertragung von niedriger priorisierten Nachrichten) als auch *non-preemptive* sein. Da die Simulation am Ende mit höchstem Datendurchsatz laufen soll, unterbrechen höher priorisierte Nachrichten die derzeitige Übertragung von Nachrichten nicht,

sondern werden in den nächsten Schedule eingebunden. So wird das erneute Senden von bereits teilweise verschickten Nachrichten vermieden.

Die Prioritätensteuerung gilt sowohl zwischen den Klassen mit der Regel *TT* > *RC* > *BE*, als auch innerhalb der RC Traffic-klasse. Bei mehreren RC Virtual Links können Prioritäten mit den Werten von eins bis acht vergeben werden, wobei Queues, in denen sich Nachrichten befinden, von höher priorisierten Virtual Links bevorzugt werden. Diese müssen allerdings noch ihre BAG einhalten, wodurch auch Nachrichten aus RC-VLs mit niedrigeren Prioritätswerten die Möglichkeit zum Nachrichtenversand eingeräumt werden.

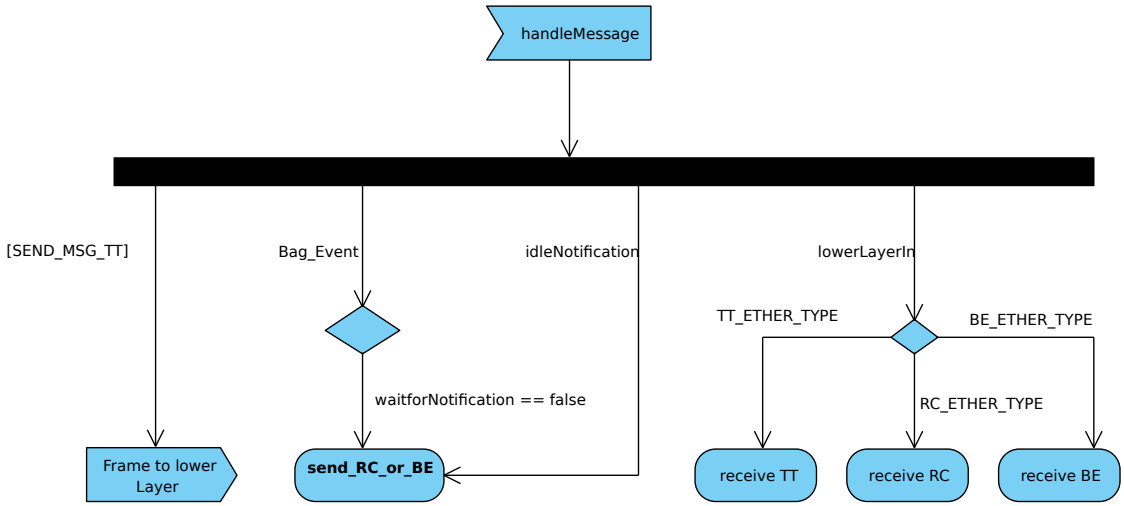

Abbildung 3.3: *Der TTEBuffer stellt die unterschiedlichen Queues für Rate-Constrained und Best-Effort bereit, außerdem einen Buffer für jeden Time-Triggered Virtual Link. Über das handleMessage erfahren Nachrichten, die von einem höheren oder niedrigeren Layer stammen, unterschiedliche Behandlung.*

Abbildung 3.3 zeigt das Aktivitätsdiagramm beim Eintreffen einer Nachricht aus höherem oder niedrigerem Layer des *TTEBuffers*. Die *handleMessage* Funktion entscheidet, je nachdem welcher Parameter in der Nachricht enthalten ist, was genau gemacht wird. Auf der rechten Seite ist das Empfangen einer Nachricht abgebildet. Die Nachricht wurde über den *lowerLayerIn*-Eingang empfangen und trägt einen Parameter, der erkennen lässt, ob es sich um eine TT, RC oder BE Nachricht handelt. Bei Erhalt einer Critical-Traffic (CT)-ID im Frame werden die bisher bekannten Überprüfungen des Empfangs durchgeführt und anschließend in dem dazugehörigen Buffer eingereiht. Bei BE-Nachrichten entfällt die Überprüfung und es wird gleich mit der Einreihung in die BE-Queue fortgefahren. Später können sich Applikationen über sogenannte *Callback-Functions* für das Eintreffen bestimmter Nachrichtentypen

eintragen. Die passende Funktion wird dann von der *TTEthernetAPI* aufgerufen, sobald eine Nachricht in einem Buffer des Endsystems eintrifft.

Neben dem *lowerLayerIn* gibt es eine weitere Nachricht, die von einer unteren Ebene hochgeschickt werden kann. Es handelt sich um die *idleNotification*, die vom MAC Layer an die oberen Schichten gesendet wird. Diese teilt mit, dass zur Zeit keine weiteren Frames zum Versenden anstehen. Die *idleNotification* dient somit dem erneuten Starten des Schedulers, der über das Aufrufen einer Funktion RC oder BE Nachrichten verschicken kann, und sie wird genau einmal nach jeder übertragenen Nachricht im *TTEBuffer* empfangen. Die Funktion *send_RC_or_BE* wird ebenfalls aufgerufen, wenn die BAG einer RC Nachrichtenqueue abgelaufen ist. Jeder RC-VL hat eine gewisse Zeit, in der er keine Nachrichten versenden kann. Wenn die *idleNotification* empfangen wird, bevor eine BAG fertig zum Verschicken ist, wartet das System solange, bis ein RC-VL wieder senden kann. Dieses Trigger wird durch das *BAG_Event* ausgelöst. Somit gibt es drei verschiedene Wege, Nachrichten aus der RC oder BE Queue zu holen:

- Die BAG eines VLs läuft ab und löst ein *BAG_Event* aus.
- Eine *idleNotification* teilt dem *TTEBuffer* mit, dass zur Zeit keine Übertragung stattfindet.
- Eine TT-Nachricht wurde verschickt.

Abschließend veranlasst der *SEND_MSG_TT* Parameter das sofortige Versenden einer TT-Nachricht. Hierbei wird davon ausgegangen, dass sich die Funktion ‘*send_RC_or_BE* an den Schedule der TT-VLs hält.

Als zweite wichtige Komponente bestimmt der *TTETimerEndsystem* den Schedule des *TTE-Buffers*. Um das Endsystem auf maximale Bandbreite zu trimmen, musste auf eine strikte Trennung vom Schedule im Timer verzichtet werden. Der Timer, von Hermand Dieumo implementiert, sendet *TRUE* oder *FALSE* an den *TTEBuffer*, wenn Platz für RC oder BE Nachrichten zwischen zwei TT-Schedules ist. Abbildung 3.4 zeigt dieses Verhalten.

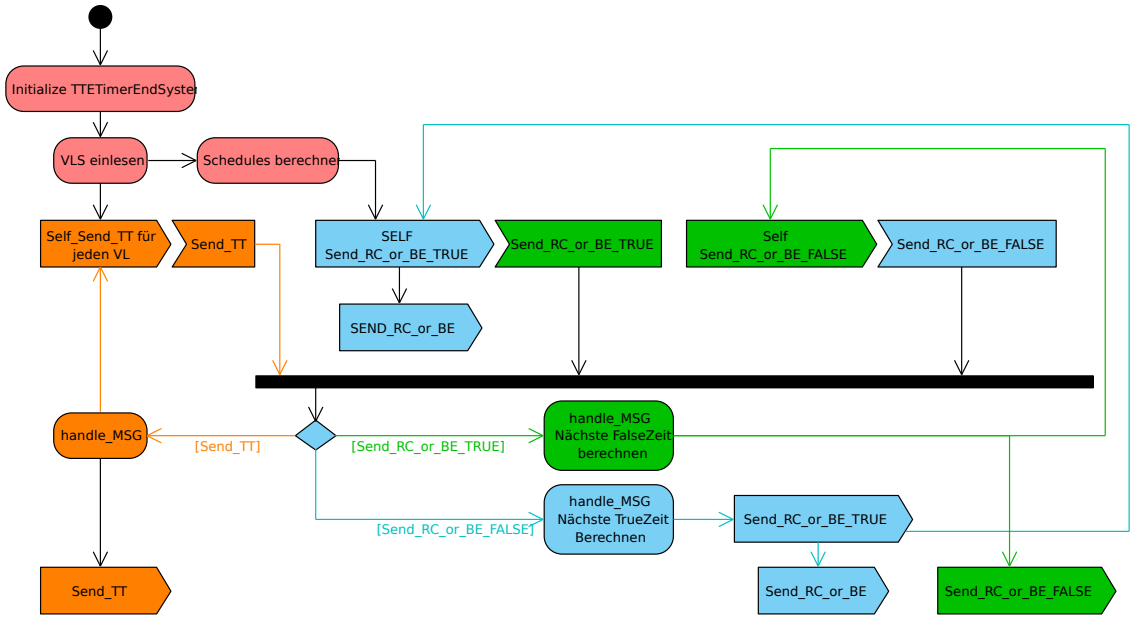

Abbildung 3.4: *Der Timer des Endsystems ist nur noch für den Schedule von Time-Triggered Nachrichten zuständig. Der Schedule von RC und BE Frames wurde in den TTEBuffer verlagert.*

Der orangefarbene Zyklus in Abb. 3.4 veranlasst den Timer, immer wenn ein TT-Schedule ansteht, eine *Send_TT* Nachricht an den *TTEBuffer* zu senden. Somit kann der *TTEBuffer* eine Nachricht aus dem zugehörigen TT-Buffer holen und sofort verschicken. Der grüne Zyklus folgt im Wechsel zum hellblauen. Hier wird berechnet, wann die nächste TT-Schedulezeit ansteht, und ein Event wird erzeugt. Immer wenn Platz vor dem nächsten TT-Schedule ist und somit Nachrichten der anderen beiden Klassen Zeit bekommen, wird ein TRUE, gefolgt von einem *SEND_RC_or_BE* an den *TTEBuffer* gesendet. Somit weiß die Komponente nun, dass genügend Zeit zum Versenden einer RC- oder BE-Nachricht bleibt und wird aufgefordert, dies zu tun. Dieser Vorgang wird nach dem roten *Schedules berechnen* im hellblauen Zyklus erreicht.

Durch die Selfmessage *Send_RC_or_BE_TRUE* des *TTETimerEndsystem* kommt die Komponente in den grünen Zyklus. Hier wird die nächste False-Zeit berechnet, in der nur TT-Nachrichten eines bestimmten TT-VLs sendbar sind. Anschließend wird eine Nachricht an den *TTEBuffer* verschickt, die zu dem berechneten False-Datum dort eintrifft. Dies verhindert weiteres Senden von RC- oder BE-Nachrichten. Abschließend wird das Modul durch eine Self-Message in den hellblauen Zyklus zurückgeführt.

Diese Lösung hat einen großen Nachteil. Durch die Vorberechnung ist es nicht mehr möglich,

kleine Nachrichten zu versenden, obwohl diese noch vor dem nächsten TT-Schedule über das Netzwerk gehen könnten.

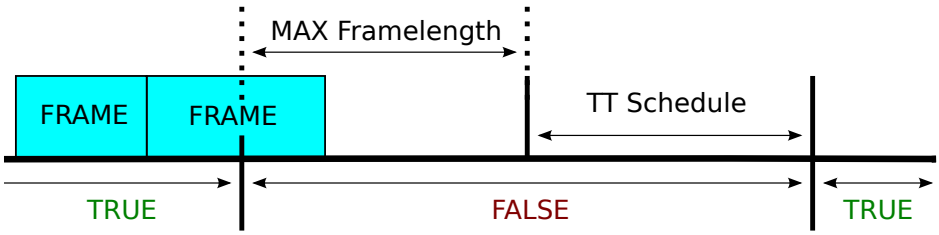

Abbildung 3.5: *Frames können während eines TRUEs gesendet werden. Im "False"-bereich werden kleine Pakete, die zwischen den letzten Frame und den nächsten TT-Schedule passen, nicht transmittiert.*

Abbildung 3.5 zeigt, dass der Platz vor dem TT Schedule nicht mehr genutzt werden kann. Bei der Berechnung muss die maximale Framelänge beachtet werden, weil sonst der Schedule beim Schicken einer TT-Nachricht verhindert wäre. Somit ist dies nicht der Ansatz, der zur maximalen Übertragungsbandbreite führt.

Um die Bandbreite zu maximieren, muss eine Veränderung durchgeführt werden. Und zwar findet die zukünftige Berechnung für Rate-Constrained und Best-Effort Nachrichten direkt im *TTEBuffer* statt. Somit wird die bisherige Trennung von Schedule und Bufferpool aufgebrochen, und in beiden Komponenten gibt es eine Teilberechnung. Der *TTETimerEndsystem* ist jetzt nur noch für den Versand der TT-Nachrichten verantwortlich. Dazu werden alle TT-VLs in einer Tabelle gespeichert und der Reihe nach, wie sie in einem Zyklus vorkommen, sortiert. Beim Versenden eines TT-Events wird auf das nächste Ereignis in der Tabelle verwiesen. Dadurch verfällt das bisherige Durchsuchen der VLs nach dem nächsten Schedule. In der Simulation reicht einmaliges Sortieren der TT-VLs, während der Initialisierungsphase.

In Abbildung 3.6 wird der *TTETimerEndsystem* so dargestellt, wie er zu implementieren ist. Die Virtual Links werden aus der Konfiguration des jeweiligen Devices eingelesen und sortiert. Anschließend stößt die *NEW_CYCLE_MSG*, eine Selfmessage des Timers, für alle TT-VLs eine *TIMER_MSG* zu den richtigen Zeitpunkten an. Trifft eine *TIMER_MSG*-Nachricht ein, wird die in der Tabelle verwiesene VLID an den *TTEBuffer* übertragen und auf den nächsten Eintrag in der Tabelle der VLIDs gezeigt.

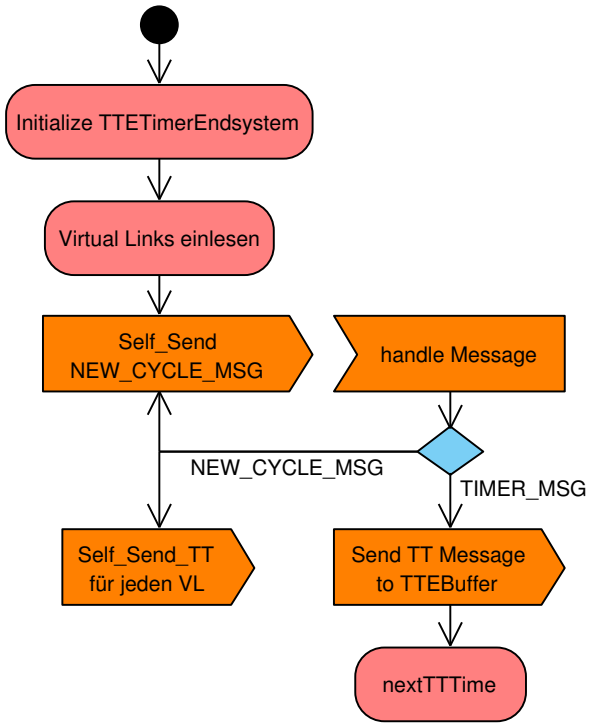

Abbildung 3.6: *Pro Zyklus wird für jeden TTSchedule eine Nachricht zum richtigen Zeitpunkt versendet. Die NEW_CYCLE_MSG stößt einen neuen Zyklus an, während beim Eintreffen der TIMER_MSG ein TT-Schedule ausgelöst wird.*

Rate-Constraint Virtual Link Scheduling

Das Virtual Link Scheduling, das im Grundlagenteil 2.2.3 beschrieben wird, findet in der *send_RC_or_BE* Funktion statt. Das Aktivitätsdiagramm in Abbildung 3.7 zeigt den groben Schedule.

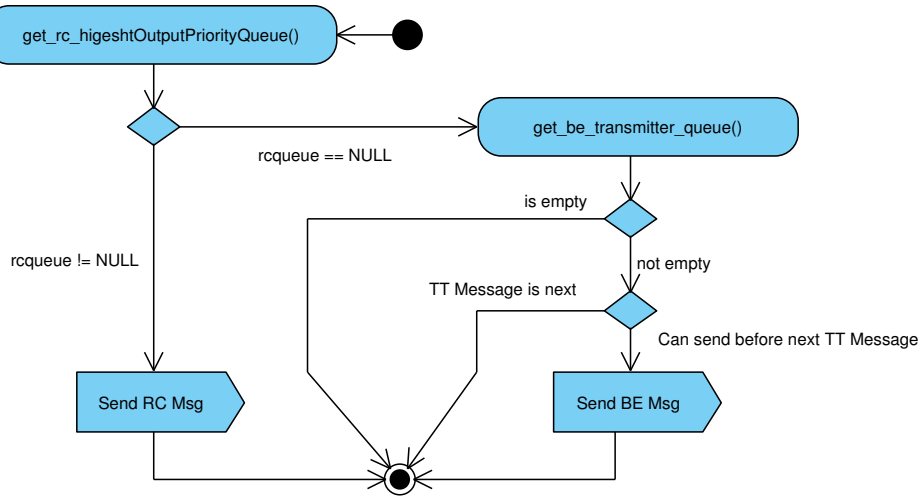

Abbildung 3.7: *Diese Funktion wird aufgerufen, wenn gerade keine Time-Triggered Nachrichten zum Schedule anstehen. Sie überprüft, ob Nachrichten der Rate-Constrained Traffic Klasse in einer Queue zwischengespeichert wurden. Falls dies nicht der Fall ist, wird geschaut, ob noch Best-Effort Nachrichten zum Versenden bereit stehen. Somit werden hier die Prioritäten der unterschiedlichen Traffic-Modi von Time-Triggered-Ethernet umgesetzt.*

Zunächst überprüft die Funktion *get_rc_highestOutputPriorityQueue*, welcher RC-VL als nächstes senden kann. Die genaue Funktionalität wird später erläutert. Nach Aufruf der Funktion gibt es zwei Möglichkeiten. Entweder wurde eine RC-Queue, in der Nachrichten enthalten sind und deren BAG abgelaufen ist gefunden, oder keine RC-Queue wurde zurückgegeben.

Beim ersten Fall wird die nächste Nachricht aus der RC-Queue entnommen und nach Überprüfung, ob die Nachricht nicht den nächsten TT-Schedule behindert, an das Netzwerk verschickt. Eine weitere Möglichkeit wäre es hier dies für alle Nachrichten der RC-Queue zu überprüfen. Allerdings würde hierbei die Reihenfolge der Frames verändert werden, und die Simulation läuft durch das ständige Durchsuchen der Buffer um ein Vielfaches langsamer, gerade wenn diese voll sind. Somit bedarf dieser Ansatz keiner weiteren Beachtung.

Wenn der zweite Fall eintritt, also keine RC-Queue senden kann, erfolgt eine Überprüfung der Best-Effort-Queue. Sind Nachrichten vorhanden, wird die Bedingung, ob die Nachricht keinen TT-Schedule behindert, ausgewertet und wird bei erfolgreichem Test verschickt.

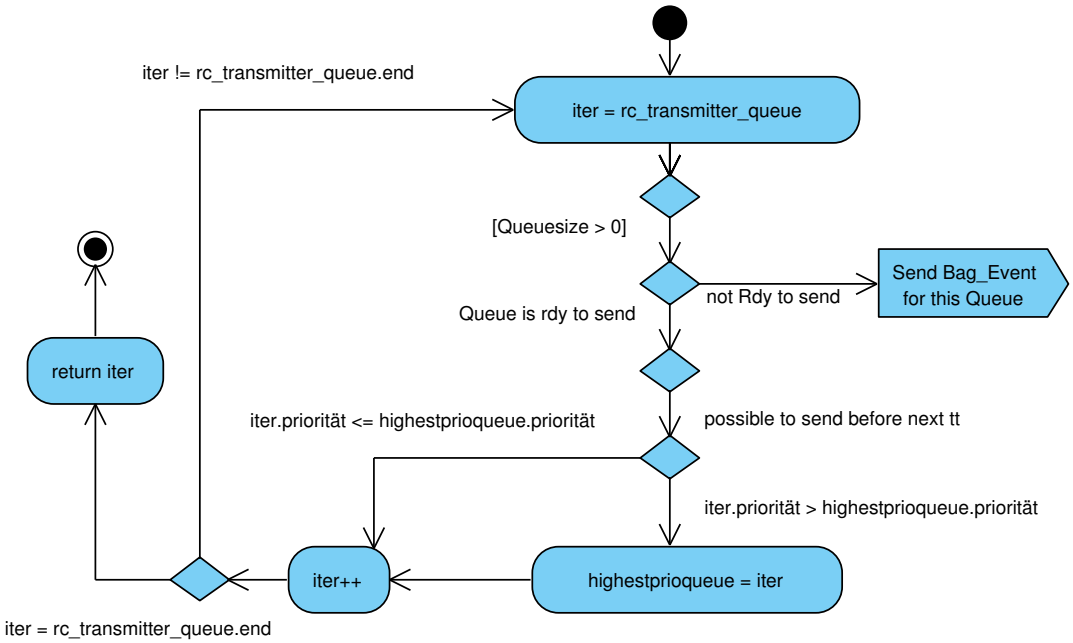

Abbildung 3.8: *Dieses Aktivitätsdiagramm zeigt das Scheduling über mehrere Rate-Constrained Queues hinweg. Hier wird entschieden, welcher Virtual Link –gemäß seiner Priorität– als nächstes senden kann.*

Abbildung 3.8 lässt die Entscheidung über den nächsten RC-VL erkennen, der senden darf. Durch eine Iteration über jeden VL hinweg erfolgt ein Vergleich mit der Queue, die bisher zum Senden ausgewählt wurde. Nach dem Test, ob sich überhaupt Nachrichten in der Queue befinden, gibt es eine Abfrage zur BAG. Bei Nichtablauf wird die nächste Möglichkeit zum Verschicken auf diesem VL berechnet und anschließend eine Self-Message, die zum passenden Zeitpunkt auslöst, gesendet. Wenn die Priorität des VLs nun höher als die des bisher markierten VLs ist, wird dieser als nächster Schedule genommen. Bei gleicher Priorität von zwei Virtual Links entscheidet ein Zeitstempel, der beim Senden gesetzt wird. Dieser bietet dem VL, der länger nicht gesendet hat, Vorzug.

Applikation zur Validierung

In Abbildung 3.2 wurde das Endsystem in OMNeT++ visualisiert. Im bisher vernachlässigten *genApp*-Modul, verbirgt sich eine Applikation. Diese greift über das *TTEthernet API* auf die beschriebenen Konzepte zum Versenden und Empfangen von Nachrichten zu. Die *genApp* soll in der Lage sein, alle drei Trafficarten TT, RC und BE zu behandeln und kann über ein Netzwerk mit anderen *genApps* oder weiteren Applikationen kommunizieren. Die Parameter können später direkt über die *omnetpp.ini* gesetzt werden. Somit legt man für jede Applikation im Netzwerk fest, welcher Nachrichtentyp gesendet und empfangen wird. Die Konfiguration für einen Videoserver, der ausschließlich RC-Nachrichten versenden kann, würde demnach wie folgt aussehen:

```
**.videoserver.genApp.sendTTMsg = false
**.videoserver.genApp.sendRCMsg = true
**.videoserver.genApp.sendBEMsg = false
**.videoserver.genApp.sendToRCID = 43
```

Jede Applikation dieser Art kann über einen RC-VL senden und über beliebig viele Nachrichten empfangen. In dem Videoserverbeispiel verschickt die Applikation Nachrichten über die VLID 43. Soll die Anzahl der zu sendenden VLs in einem Endsystem erhöht werden, kann dies direkt durch eine Anpassung der Applikation oder durch das Schreiben einer neuen geschehen.

Die Beispielanwendung liest während der Simulationszeit eine Datei aus und verschickt ihren Inhalt in Textform. Das Einlesen passiert periodisch und kann ebenfalls über die omnetpp.ini eingestellt werden. Dazu bestimmt der Parameter *rcWaitTime* die Pause bis zum erneuten Einlesen. Somit kann die Datei auch während der Simulationsdauer verändert werden, und die Auswirkungen sind direkt einsehbar. Falls die Datei länger als die maximale Framelänge ist, wird diese durch die Applikation auf mehrere Frames aufgeteilt.

3.3.2 TTEthernet-Switchmodellierung

Der Switch wurde im Wesentlichen von Till Steinbach (Steinbach (2011)) während seines Masterstudiums erarbeitet. Dieser Abschnitt dient somit dazu, einen Überblick über die wichtigsten Teile des Moduls zu erhalten und die durchgeführten Erweiterungen zu verstehen.

Der TTEthernet-Switch basiert wie alle Ethernet-Switche im INET-Framework auf dem MACRelayUnit-Interface (siehe Abbildung 3.9).

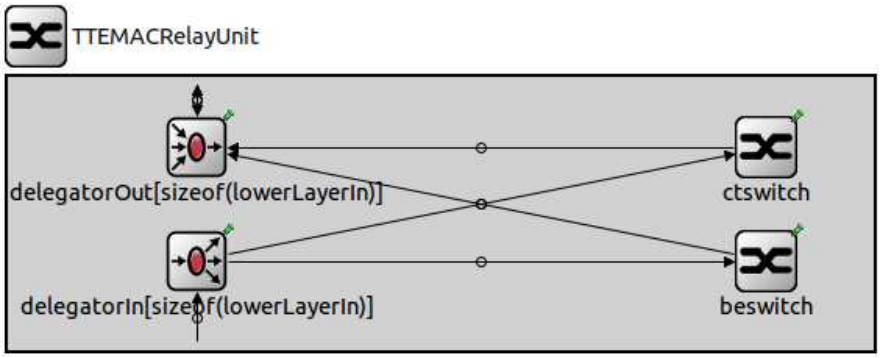

Abbildung 3.9: *Zwei MACRelayUnit-Module für critical- und non-critical-Traffic sind mit n Delegatoren der einzelnen Ports verbunden. Der DelegatorOut ist für das Virtual-Link Scheduling zuständig.*

Die *TTEMACRelayUnit* beinhaltet zwei Module für die unterschiedlichen Trafficarten. Nachrichten, die über die RC- oder TT-Virtual Links in dem Switch eintreffen, werden in der *ctswitch* MACRelayUnit verarbeitet, die das korrekte Empfangen und die Weiterleitung an den richtigen Port veranlasst. Best-Effort Nachrichten werden nicht als “critical traffic” angesehen und dementsprechend an die MACRelayUnit *beswitch* geschickt. Die weitere Verarbeitung dieser Nachrichten basiert auf der INET-Implementierung.

Nachrichten können den Switch nur über die Delegatoren erreichen oder verlassen. Dazu hat jeder Port seinen eigenen *DelegatorIn* und dementsprechend auch einen *DelegatorOut*. Der darunterliegende MACLayer, der einen Zugriff auf die physikalischen Ports ermöglicht, wird an dieser Stelle nicht weiter betrachtet, weil er aus dem INET-Framework stammt. Kommt eine Nachricht auf einem Port des Switches an, überprüft der *DelegatorIn* die Zieladresse aus dem Ethernet-Header des Frames. Handelt es sich um ein CT-Frame, beauftragt der *DelegatorIn* den *ctswitch*, der die Nachricht weiter bearbeitet. Abbildung 3.10 zeigt die beiden Komponenten, die zur Überprüfung der Nachrichten dienen. Die *Clock* liest die TT-Schedules aus der Konfiguration des Switches und löst beim Versenden einer Nachricht jeweils einen TT-Schedule aus. Sie stellt somit das Pendant zum *TTETimerEndsystem* dar.

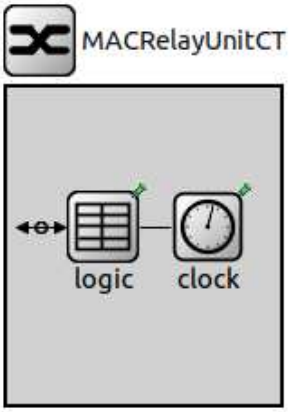

Abbildung 3.10: *Die MACRelayUnitCT enthält die logic zum Empfangen von Dateien. Durch den Timer können die Schedules der TT-Nachrichten berechnet werden*

Die *TTELogic* hat die selbe Funktionalität wie die Empfangsseite des *TTEBuffers* im Endsystem. Hier werden Nachrichten überprüft, ob sie ihren vorgesehenen Schedule einhalten oder nicht. Für TT-Nachrichten gilt dabei das Intervall, in dem sie ankommen dürfen. Bei RC-Nachrichten muss hier zukünftig eine Überprüfung der BAG stattfinden. Ist ein zu schnelles Eintreffen auf einem Virtual Link festgestellt worden, muss die Nachricht an dieser Stelle verworfen werden. Bei Einhaltung der BAG wird die Nachricht an den *DelegatorOut* weitergeleitet.

Die Empfängerseite des*TTEBuffers* übernimmt der *DelegatorOut*. An dieser Stelle wird entschieden, welche Nachricht als nächstes über den Port des Delegators den Switch verlässt. Hier findet das Virtual Link Scheduling wie im Endsystem 3.3.1 (2.2.3) statt.

4 Komponentenvalidierung

Die Validierung dieser Arbeit besteht aus diversen funktionalen Black-Box-Tests, die eine einwandfreie Funktionsweise der Module testet. Hierzu ist es wichtig, dass die einzelnen Simulationsergebnisse eine Übereinstimmung mit den Spezifikationen erzielen. Ausgangspunkt der folgenden Tests soll ein Netzwerk mit einem Sender, einem Empfänger und einem Switch, der die beiden verbindet, sein (Abb. 4.1).

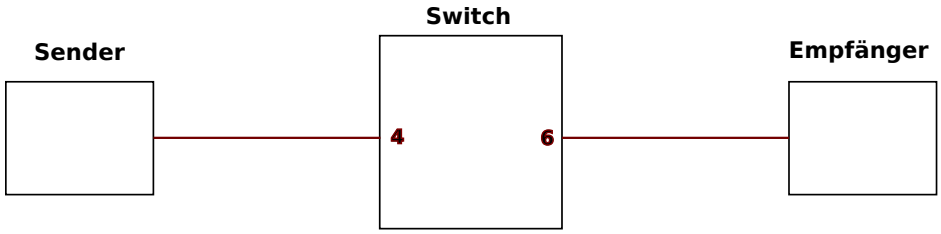

Abbildung 4.1: *Ein kleines Netzwerk zum Testen der Funktionalität. Es besteht aus einem Endsystem zum Senden von Dateien, einem Switch und einem Zielendsystem.*

Der Sender ist dabei an den Port-4 des Switches angeschlossen, während der Empfänger über Port-6 erreicht werden kann. Beide Endsysteme sind die in Kapitel 3.3.1 vorgestellte Applikation zum Einlesen und Verschicken einer Datei. Die Datei besteht aus 2162 Bytes und muss in zwei Frames in der Anwendungsebene fragmentiert werden, damit die maximale Länge von 1518 Bytes einer Ethernet Nachricht nicht überschritten wird. Zur Überprüfung der Simulation dient die Auswertung des *Output Skalar Files*. Es gibt Auskunft über die Anzahl empfangener und verschickter Nachrichten aller drei Trafficarten zu jedem Device. Zudem wird die Gesamtanzahl an Nachrichten, die gesamte Übertragung in Bytes und weitere Kenngrößen aufgezeichnet.

4.1 Einhaltung der BAG beim Switch

Das Testen der BAGS im Switch unterteilt sich in das Senden und das Empfangen. Um zu testen, ob sich das Versenden von Nachrichten im Switch wie zu erwarten verhält, ist das

bisher bereits erwähnte Netzwerk mit einem Sender, einem Switch und einem Empfänger zu betrachten (Abb. 4.2).

4.1.1 Versand

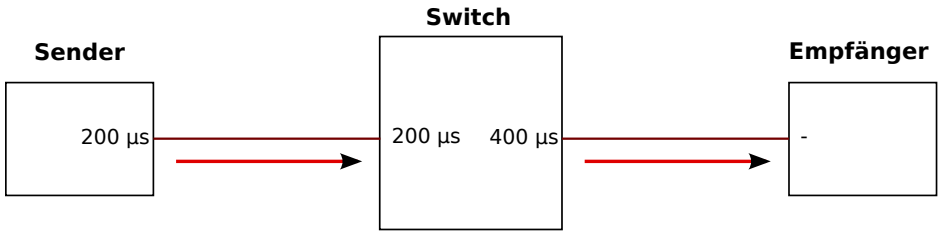

Abbildung 4.2: *Der Switch leitet die Nachrichten zu langsam weiter. Es kommt zu einer Anstauung der Daten, und das Zielendsystem erhält die Daten langsamer als vorgesehen.*

Der Sender verschickt Nachrichten in einem Abstand von $200\mu s$ an den Switch. Die Nachrichten, die auf dem Virtual Link im Switch ankommen, werden allerdings erst alle $400\mu s$ weitergeleitet. Somit kommt es unweigerlich zu einer Anstauung von Nachrichten im Switch. Die Empfangsseite sowohl im Switch als auch im Empfängerendsystem ist bei diesem Test zu vernachlässigen.

Bei einer Simulationsdauer von einer Sekunde wäre zu erwarten, dass $\frac{1s}{200\mu s} = 5000$ Frames im Switch eintreffen. An den Empfänger werden im Switch $\frac{1s}{400\mu s} = 2500$ dieser Frames unter Einhaltung der BAG vom Switch weitergeleitet. Dieses gilt es jetzt in der Simulation zu testen.

Beim Blick in das *Output Skalar File* zeigt die Tabelle beim *processed RC frames*-Eintrag einen Wert von 5000. Somit traf also wie erwartet die tatsächliche Frameanzahl im Switch ein. Im Endsystem, das die Nachrichten empfangen soll, gibt es den Eintrag *frames rcvd*. Dieser teilt die gesamte Anzahl an empfangenen Frames mit. Da zu diesem Zeitpunkt nur RC-Nachrichten verschickt werden, ist dies der zu überprüfende Wert. Er liegt bei 2500 und bestätigt das erwartete Ergebnis.

4.1.2 Empfang

Als nächstes gilt es, den Empfang im Switch zu testen. Hierzu werden die BAG-Parameter der bisherigen Versuchssimulation geändert. Das sendende Endsystem verschickt Nachrichten wie zuvor mit einem BAG-Value von $200\mu s$. Der Switch ist in der Lage, nur noch alle

$400\mu s$ Frames zu empfangen. Er kann jedoch die Nachrichten gleich weiterleiten, denn die BAG zum Versand beträgt nur $200\mu s$. Abbildung 4.3 fasst den Simulationsaufbau zusammen.

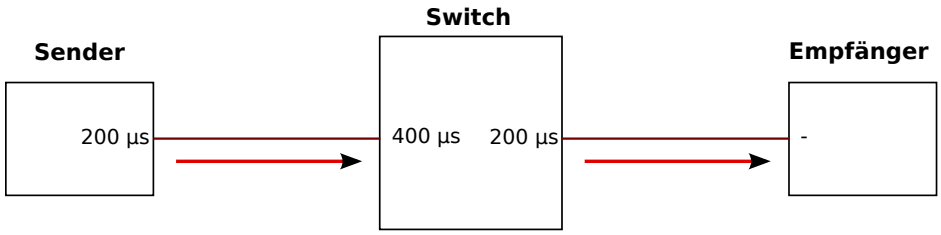

Abbildung 4.3: *Das Endsystem verschickt die Daten schneller, als der Switch sie empfangen darf.*

Auch bei diesem Test wird am Ende erwartet, dass 2.500 Nachrichten das Endsystem erreichen. Durch die Sendegeschwindigkeit von $200\mu s$ beim verschickenden Endsystem empfängt der Switch 5.000 Frames. Allerdings werden durch die empfangende BAG des RC-VLs tatsächlich nur $\frac{1s}{400\mu s} = 2500$ Frames davon im Switch zwischengespeichert. Diese leitet der Switch dann durch die wesentlich schnellere BAG von $200\mu s$ sofort weiter.

Das *Output Skalar File* zeigt auch bei diesem Test die gewünschten Ergebnisse an. Der *frames rcvd*-Eintrag vom Switch zeigt 4.999 empfangene Pakete im Switch an. Dies zeigt, dass die Pakete tatsächlich ankommen, die im $200\mu s$ Abstand gesendet worden sind. Der Eintrag *frames sent* von Port-6, der die versendeten Nachrichten anzeigt, erreicht einen Stand von 2.500. Somit wurden tatsächlich nur 2.500 Frames weitergeleitet. Um das Ergebnis zu überprüfen, klärt der RC-Buffer über die noch vorhandenen Nachrichten im Switch auf. Dieser steht auf 0 und zeigt, dass wirklich alle Nachrichten, die im Switch konform ihrer BAG ankamen, weitergeleitet wurden.

4.2 Einhaltung der BAG beim Endsystem

Im Endsystem brauchen wir nur einen Test, der validiert, dass die Empfangsseite konform arbeitet. Die sendende Seite wurde in den vorigen beiden Tests schon genutzt und erfährt auch hier wieder mit einer BAG von $200\mu s$ Nutzung. 5.000 Frames wurden jedes Mal verschickt, was auch zu erwarten war.

Um die BAG im Endsystem zu überprüfen, zeigt Abbildung 4.4 das bisher bekannte Netzwerk mit neuen BAG-Values an.

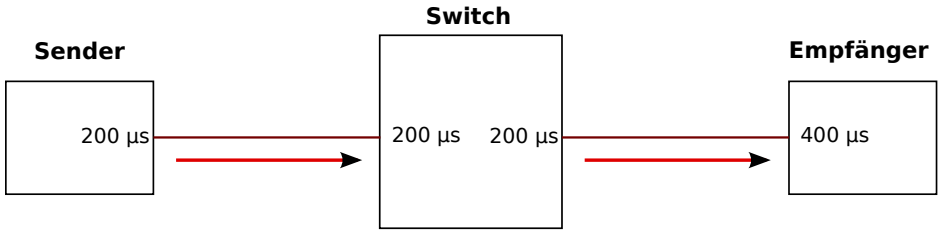

Abbildung 4.4: *Die BAG beim Zielendsystem ist zu hoch eingestellt. Jede zweite Nachricht wird verworfen.*

Der Switch hat dabei die gleichen Empfangs- wie auch Sendezeiten von $200\mu s$ und sollte daher alle Frames, die vom Endsystem ankommen, durchlassen. Im Empfänger steht die BAG auf einem Wert von $400\mu s$. Da dies wieder den doppelten Wert der anderen Systeme aufzeigt, ist hier eine Halbierung der Frames, die konform eintreffen, zu erwarten.

Bei der erneuten Überprüfung des *Output Skalar Files* wird einerseits die Anzahl der empfangenen Frames und andererseits die Anzahl der Frames, die jemals im RC-Buffer waren, getestet. Der erste Wert steht auf 4.999 erhaltenen Frames im Endsystem. Durch die BAG-Kontrolle wurden 2.500 Frames gelassen. Dieser Wert steht auch im *Received RC Messages*-Eintrag des Zielendsystems. Somit funktioniert das Endsystem auch wie erwartet.

4.3 Prioritätensteuerung bei Weiterleitung

Als abschließender Test der Validation wird die Prioritätensteuerung beim Scheduling im Switch vorgenommen. Dazu wird ein Netzwerk mit drei sendenden Endsystemen, einem empfangenden Endsystem und dem TT-Switch, der alle vier miteinander verbindet. Dabei werden die Nachrichten über drei verschiedene Virtual Links verschickt. Da jeder eine BAG von $200\mu s$ besitzt, läuft der Buffer im Switch für die Rate-Constrained Nachrichten unweigerlich voll. Somit liegt es an den Prioritäten der einzelnen Virtual Links, über welchen die nächste Nachricht verschickt werden darf. Abbildung 4.5 zeigt die Endsysteme, deren Verbindungen und die Virtual Links.

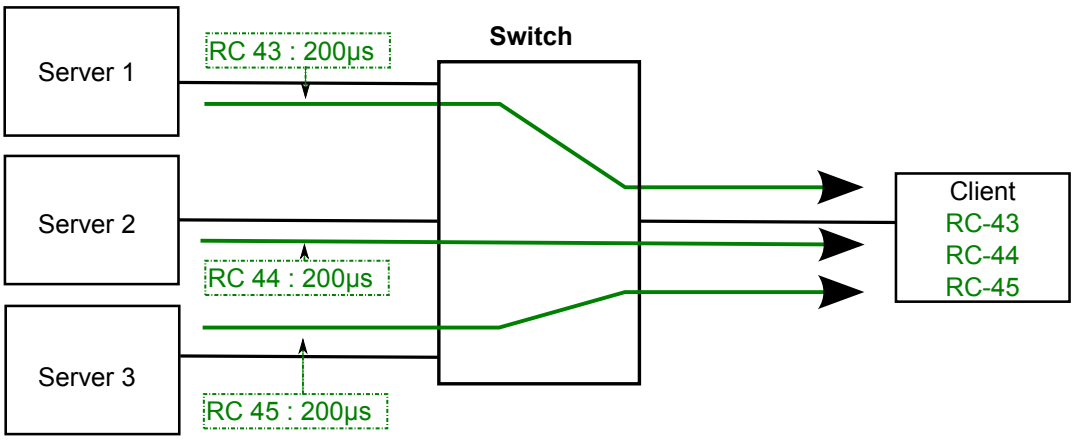

Abbildung 4.5: *Drei sendende Endsysteme verschicken Nachrichten über Virtual Links mit Prioritäten. Die geringen BAG-Values tragen zu einer hohen Übertragungsbandbreite der einzelnen Endsysteme bei.*

Zuerst werden gleiche Prioritäten bei allen Virtual Links vergeben. Dies zeigt, ob der VL, der am längsten nicht gesendet hat, bevorzugt wird. Ist dies der Fall, müsste jeder Virtual Link die gleiche Anzahl an Nachrichten im empfangenden System erreichen. Die Simulationszeigt beträgt $1s$. Beim Blick in das bereits früher verwendete *Output Skalar File* zeigt dies folgende Aufteilung der VL-Werte der 10142 RC-Frames im empfangenden Endsystem:

- “Received RC ID: 43” : 3381
- “Received RC ID: 44” : 3380
- “Received RC ID: 45” : 3381

Es werden also über jeden VL 3381 Nachrichten verschickt und keiner ist den anderen gegenüber bevorzugt.

Nun wird die Konfigurationsdatei insoweit verändert, dass die Prioritäten nun *VL-43* > *VL-44* > *VL-45* entsprechen. Jetzt zeigt sich ein anderes Bild und die empfangenen 10203 Frames verteilen sich den Prioritäten entsprechend auf die unterschiedlichen VLs:

- “Received RC ID: 43” : 3602
- “Received RC ID: 44” : 3460
- “Received RC ID: 45” : 3141

Durch die BAG der einzelnen VLs wird garantiert, dass die Gesamtbandbreite auf verschiedene Endsysteme aufgeteilt wird. Durch die sehr geringe BAG von $200\mu s$ ist ein deutlicher Abstand zwischen den Empfangszahlen der VLs festzustellen. Würde man die Bandbreite der einzelnen mit einem höheren BAG-Value limitieren, hätte die Frame-Übertragungsdauer einen nicht mehr so großen Einfluss auf das Ergebnis und die Anzahl der übertragenen Frames würde sich angleichen.

5 Simulation exemplarischer Netzwerke

In diesem Kapitel geht es darum, verschiedene Netzwerke in der Simulation laufen zu lassen. Hierbei wurde entschieden, ein Netzwerk mit einer größeren Last zu testen, bei dem sechs Endsysteme an einem Switch hängen. Dies wird im Abschnitt 5.1 durchgeführt. Zudem ist es interessant, inwieweit sich der entworfene TTE-Switch von einem Standard-Ethernet-Switch in der Performance unterscheidet. Um dieses herauszufinden, wird in Abschnitt 5.2 ein Vergleich zum Switch aus dem INET-Framework vorgenommen.

5.1 Netzwerk unter Last

Um den Switch und die Endsysteme in einem größeren Netzwerk zu simulieren, wird ein Netzwerk mit drei Sendern und drei Empfängern an sechs verschiedenen Ports des entwickelten TTEthernet-Switches entworfen. Hierbei macht sich das fehlende *TTE-Plan* bemerkbar, weil die Anpassungen der verschiedenen Konfigurationsdateien sehr umfangreich sind, und das Modell bei kleineren Abweichungen Fehler wirft. Dies tritt besonders auf, wenn Referenzen nicht richtig gesetzt wurden.

Abbildung 5.1 bildet das Netzwerk ab. Die schwarzen Linien kennzeichnen die Verbindung der Endsysteme mit dem Switch. Die Pfeile zeigen Time-Triggered und Rate-Constrained Virtual Links und den Verlauf von Best-Effort Frames im Netzwerk.

Grafisch wurden unterschiedliche Farben für die verschiedenen Trafficarten genutzt. Rot symbolisiert einen TT-VL. Hierbei handelt es sich um Nachrichten, die die Critical-Traffic-ID 100 besitzen. Somit kann *Server 1* diesen Nachrichtentyp an *Client 1* einmal pro Zyklus verschicken. Die grünen Pfeile entsprechen dem Kontrollfluss der RC-Virtual Links. Im Netzwerk hat jeder Server einen Virtual Link zum gegenüberliegenden Client. Jedoch werden die Nachrichten des VLs 43 ebenfalls an *Client 2* weitergeleitet, was einem Multicast entspricht. Zum Schluss bleiben noch die lilafarbenen, gestrichelten Pfeile übrig. Diese stellen die Wege der Best-Effort-Frames dar. Zusammenfassend ist also gesagt:

- *Client 1* empfängt: **TT-100** und **RC-43**
- *Client 2* empfängt: **RC-43**, **RC-44** und **BE** von *Server 3*

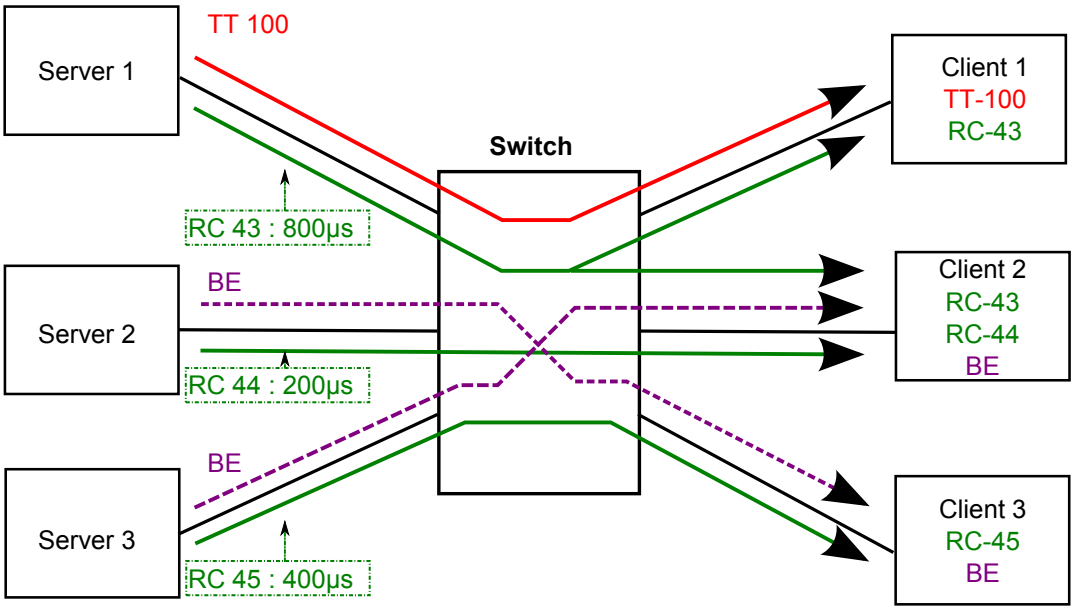

Abbildung 5.1: *Sechs Endsysteme sind mit einem Switch verbunden. Ein TT-Virtual Link in Rot, drei RC-Virtual Links in Grün und die Richtung von BE-Streams in Violett zeigen die Verbindungen an.*

- *Client 3* empfängt: **RC-45** und **BE** von *Server 2*

Als Endsystem kommt in diesem Netzwerk die bekannte Applikation, welche eine Datei einlesen und versenden kann, zum Einsatz. Dies wird ebenfalls zum Verschicken von RC-Nachrichten genutzt. *Server 1* ist im Stande, über den VL mit der ID 43 alle $800\mu s$ ein Frame zu verschicken. Das entspricht der Anzahl von $\frac{1s}{800\mu s} = 1250$ Nachrichten pro Sekunde. Der VL-44 verursacht die vierfache Last im Netzwerk mit $200\mu s$ und maximal erreichbaren 4000 Frames, während VL-45 versucht, im $400\mu s$ Abstand Frames zu versenden. Um das Netzwerk voll auszulasten, erzeugen die Best-Effort sendenden Endsysteme *Server 2* und *Server 3* alle $150\mu s$ Nachrichten und füllen damit ihre BE-Queue über die TTE-API. Dies entspricht einer Anzahl von 6667 erstellten Best-Effort-Nachrichten.

Der TT-Schedule des *Server 1* fängt nach genau $800\mu s$ des Zyklus an und endet nach $180\mu s$. Der Zyklus wiederholt sich alle $3ms$, wodurch dieser 333 mal pro Sekunde durchlaufen wird. Dies verlangt später nach einer Überprüfung.

Zudem hat der Switch die gleichen Werte für RC-VL-BAGs und TT-Schedules wie die Endsysteme in seiner Konfigurationsdatei, sodass die RC und TT Nachrichten dem Schedule entsprechend weitergeleitet werden können. Es sollen 10 Sekunden Realzeit simuliert werden. Die Geschwindigkeit, mit der die Simulation läuft, ist hierbei irrelevant. Anschließend

werden die empfangenen und gesendeten Frames der einzelnen Endsysteme miteinander verglichen und das Ergebnis ausgewertet.

Diagramm 5.2 zeigt die Anzahl der empfangenen Bytes von *Server 1-3* und die gesendete Anzahl an Bytes von *Client 1-3* an. Die gesendeten Daten von *Server 1* und *Server 3* entsprechen dabei *Client 1* und *Client 3*. Jedoch müsste *Client 2* durch zwei Virtual Links, über die es empfängt, deutlich mehr Frames empfangen haben.

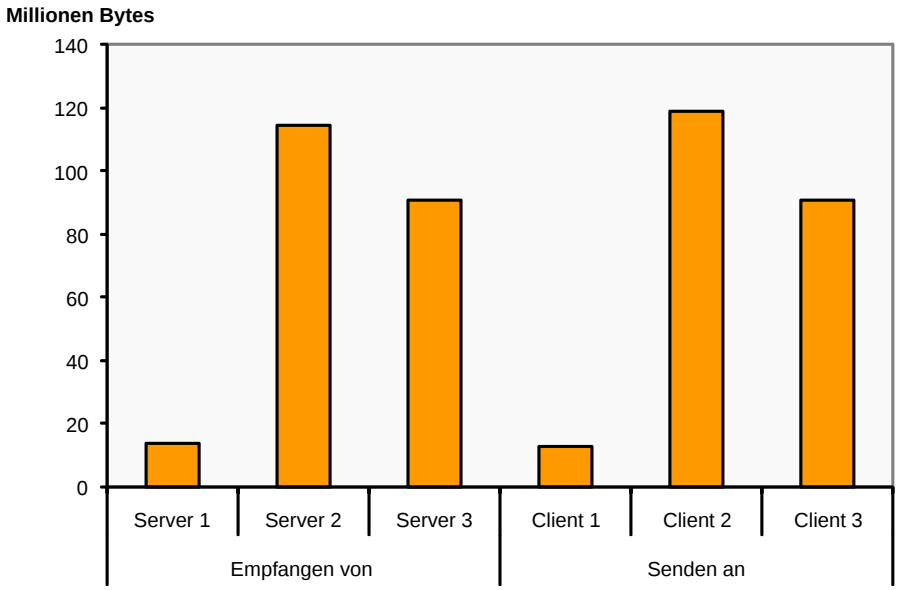

Abbildung 5.2: *Das Diagramm zeigt die Anzahl der empfangenen und gesendeten Bytes der verschiedenen Endsysteme an.*

Daher wird ein Blick in die Aufteilung der Frames in die unterschiedlichen Trafficarten geworfen und überprüft, wie sich die Frames im *Client 2* zusammensetzen. Abbildung 5.3 visualisiert die gesendeten und empfangenen Frames der Endsysteme nach Traffictyp gepaart.

Auf der linken Seite in Abbildung 5.3 sind die drei sendenden Endsysteme abgebildet. Alle 3333 TT-Frames (rote Säulen) werden übertragen, wodurch die bereits früher in diesem Kapitel erwähnte durchzuführende Überprüfung vorgenommen wurde. Die grünen Säulen zeigen die über RC-VL gesendeten und empfangenen Frames. Sowohl bei *Client 1* als auch bei *Client 3* stimmt die Anzahl der Frames mit der verschickten Anzahl der Server überein. Bei *Client 2* ist diese mit 53.333 RC-Frames ein wenig höher als die von *Server 2* versendeten RC-Botschaften. Jedoch werden hier auch die höher priorisierten Nachrichten von *Client*

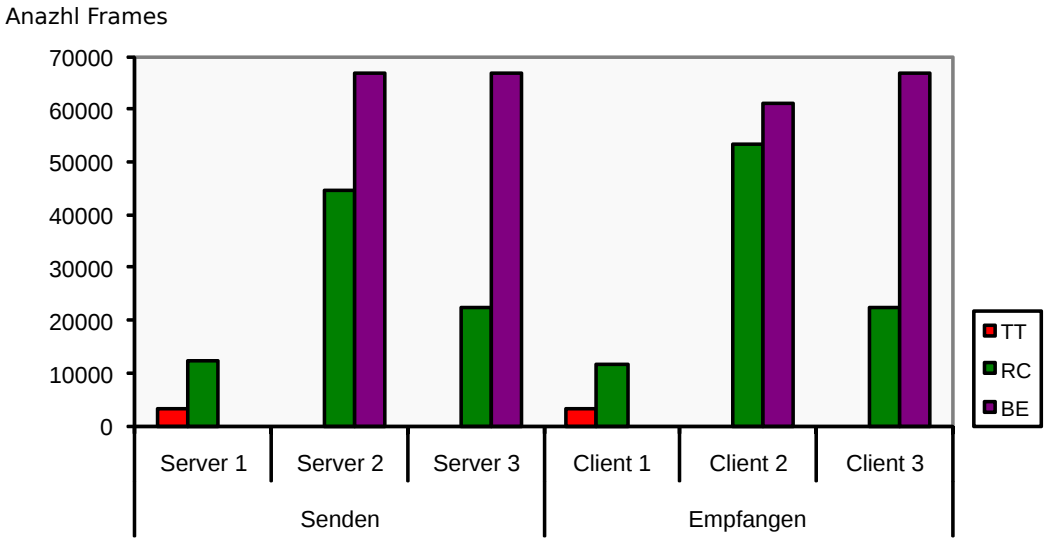

Abbildung 5.3: *Das Diagramm zeigt in unterschiedlichen Farben die Anzahl der gesendeten Frames der Server-Endsysteme, sowie die Anzahl der empfangenen Client-Endsysteme, nach Trafficart aufgeteilt. Die Ergebnisse sind erklärbar.*

1 empfangen. Wenn die ursprünglich verschickten Nachrichten der beiden Virtual Links 43 und 44 addiert werden, fällt auf, dass diese eigentlich höher sein müssten, nämlich 56667. Wenn die BAG eines Virtual Links abläuft, kann über diesen nicht sofort versendet werden, weil ein anderer VL oder Nachrichten aus der niedriger priorisierten Best-Effort-Queue noch auf dem Weg zum Empfänger sind. Somit verschiebt sich der Schedule ein wenig nach hinten und Bandbreite geht verloren. Dies fällt gerade bei vielen maximalen Frames, wie sie in der Simulation vorkommen, durch die lange Übertragungszeit ins Gewicht. Somit sind die Werte der RC-Übertragungen sehr gut, auch weil die BAGS sehr niedrig sind und mit $200\mu s$ an der Übertragungsdauer von maximalen Frames liegen. AFDX fängt normalerweise erst bei BAG-Values von 1 ms an und steigt dann quadratisch.

Im Weiteren ist die Performance des Switches zu testen und festzustellen, inwieweit sich dieser von einem Standard-Ethernet-Switch unterscheidet. Dieses wird im nächsten Kapitel vorgenommen.

5.2 Switchperformance

Um die Performance des TTEthernet-Switches zu testen, wurde dieser in der Simulation mit dem *EtherSwitch2* aus dem INET-Framework verglichen. Dazu wurde die schon bekannte Simulation aus der Validation mit einem sendenden und einem empfangenden Endsystem genommen, die jeweils mit einem Port des Switches verbunden sind. Um eine hohe Auslastung im Netzwerk zu erzeugen, schickt das sendende Endsystem Nachrichten aller drei Klassen über das Netzwerk und hat dazu einen TT-Virtual Link mit der ID 100, außerdem einen RC-VL, der die ID 43 trägt und fähig ist, alle $128\mu s$ Nachrichten durch seinen BAG-Value zu senden. Um die Simulation nicht weiter zu belasten und wirklich nur die Verarbeitungszeit des Schedules anzuschauen, bearbeitet das Zielendsystem die eintreffenden Nachrichten nicht. Dazu werden in der *omnetpp.ini* die passenden Werte des Empfängers auf false gesetzt, was zum Ignorieren auf der Empfangsseite führt.

Weiterhin ist zu beachten, dass beide Endsysteme die gleichen Parameter in beiden Simulationen behalten, und nur der Switch ersetzt wird. Somit kann eine Verfälschung des Ergebnisses ausgeschlossen werden.

Zudem ist die sendende Applikation angepasst worden. Sie liest nur ein einziges mal die Datei ein und verschickt immer 500 Byte große Pakete, um die Simulation nicht durch das ständige Einlesen zu verlangsamen. Dieses wird als *Fast Endsystem* bezeichnet. Der Performancegewinn zum bisher genutzten *Slow Endsystem* wird ebenfalls festgehalten. Als Testsystem dient eine Intel Core i5 CPU mit vier Kernen und einer Taktfrequenz von 2,67 GHz. Der Arbeitsspeicher fällt auf eine Größe von 4 GiB. Somit wurde die Simulation in einem heute üblichen Rechner getestet. Außerdem wurden jeweils 10 Sekunden Realzeit überprüft.

Das Säulendiagramm 5.4 zeigt die mögliche Simulationszeit pro Sekunde der einzelnen Simulationen an. Dabei wird das *Fast Endsystem*, wie bereits erwähnt, mit dem *Slow Endsystem* verglichen. Die erste Säule des Diagramms zeigt die Geschwindigkeit an dem INET-Ethernetswitch mit den schnelleren Endsystemen. Hier ist es ungefähr möglich, 0,404 Echtzeitsekunden in einer Simulationssekunde zu simulieren. Der TTEthernet Switch kann mit einer Durchschnittsgeschwindigkeit von 0,477 Sekunden pro Simulationssekunde also mehr in kürzerer Zeit simulieren. Dieses Bild zeigt sich auch bei dem langsameren Endsystem, wenn auch nicht ganz so deutlich. Dort beträgt der Simulationswert 0,089s und ist somit ein wenig höher als die 0,086 Realzeitsekunden pro Simulationssekunde des INET-Switches. Die Frage ist aber, ob der Switch jetzt tatsächlich schneller ist, was ungewöhnlich wäre, da umfangreichere Bedingungen überprüft werden müssen.

Um dies zu testen, wurde ein weiteres Diagramm 5.5 erstellt. Dieses visualisiert die empfangenen und gesendeten Nachrichten des Switches. Außerdem ist ein Durchschnittswert der pro Sekunde erstellten Messages dargestellt. Dies umfasst jede Art von Message und nicht

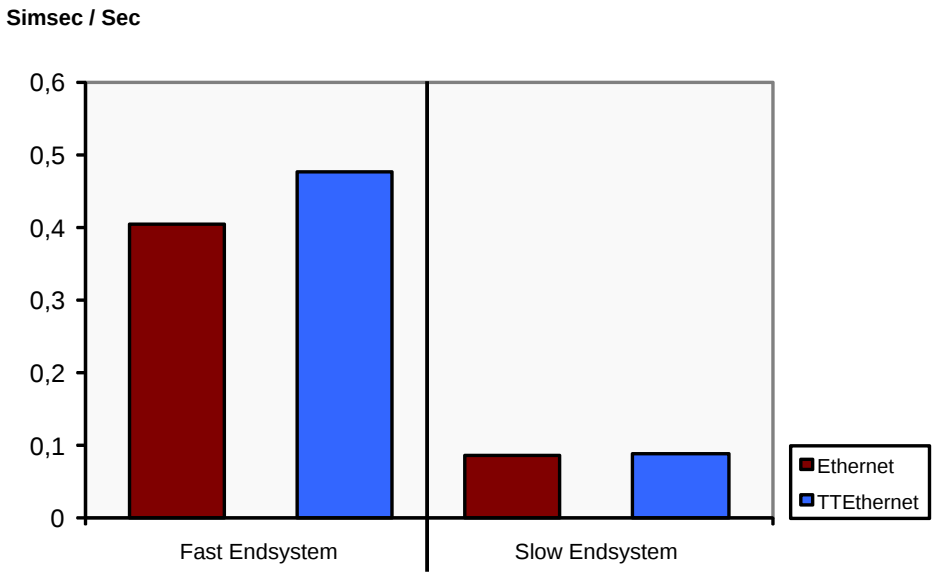

Abbildung 5.4: *Die Veränderung der Simulationszeit pro Sekunde bei unterschiedlichem Switch und Endsystem.*

nur die Frames, die über das Netzwerk übertragen werden. Aus diesem Grund ist die Zahl der erstellten Messages auch um ein Vielfaches höher.

Die Anzahl der empfangenen Nachrichten beträgt sowohl im INET-Switch als auch im TTE-Switch 143.119 beziehungsweise 112.603 im langsameren Endsystem. Dies kommt durch das limitierte Senden von exakt 500 Bytes pro Nachricht zu Stande. Im *Slow Endsystem* findet dagegen ein Wechsel von maximalen Frames mit 1.518 Bytes und ungefähr 600 Byte großen Frames statt. Während der INET-Switch die Frames alle direkt weiterleitet, fällt bei Überprüfung der versendeten Nachrichten im TTE-Switch auf, dass die Anzahl signifikant niedriger ist. Hier werden nur zirka 60 % der Frames übertragen. Dies liegt zum einem an dem BAG-Value von $400\mu s$ im TTE-Switch, aber zum anderen auch an dem TT-Schedule, in dem nur eine Nachricht verschickt wird und ein wenig Bandbreite bei jedem Zyklus verloren geht.

Am interessantesten zur Geschwindigkeitsüberprüfung ist die letzte Säule im Diagramm. Sie fasst -wie bereits erwähnt- die Menge der erstellten Messages pro Sekunde zusammen. In den 10 Sekunden Simulationszeit wurden 2.006.115 Messages erzeugt, was zu einer Auslastung von $\frac{2.006.115\ Messages}{\frac{10s}{0,404s}} = 81.047,04$ Messages pro Simulationssekunde führt. Wenn man diesen Wert mit der Simulation des TTE-Switches vergleicht, fällt auf, dass er sich kaum davon unterscheidet. Diese Auslastung liegt nämlich mit $\frac{1.676.618\ Messages}{\frac{10s}{0,477}} = 79.974,68$ Messages im selben Bereich wie der Ethernetswitch. Dies zeigt, dass die Si-

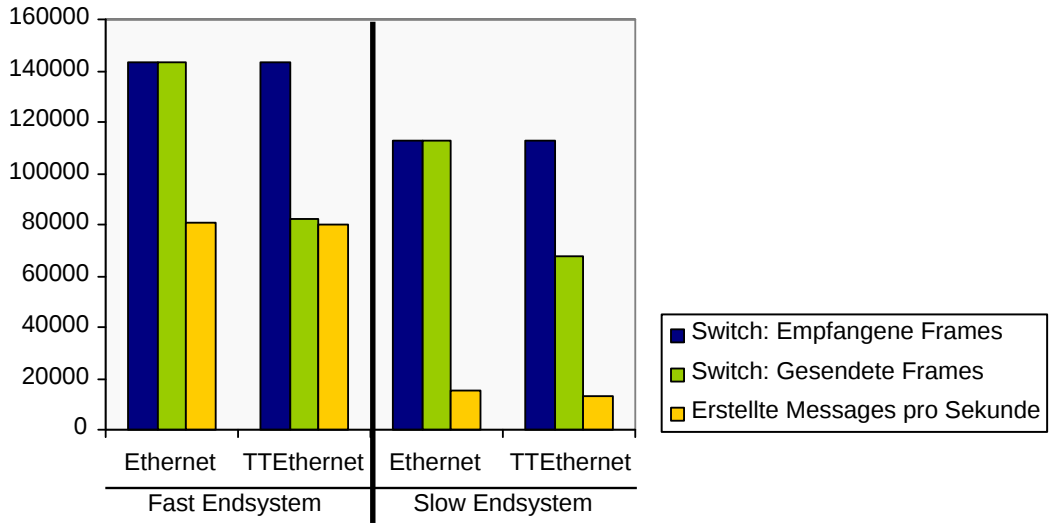

Abbildung 5.5: *Hier zeigen die empfangenen und versendeten Frames bei unterschiedlichen Switches die Performance von Time-Triggered-Ethernet. Die in gelb visualisierten, erstellten Messages pro Sekunde sind ein guter Indikator.*

mulation in Wirklichkeit gar nicht schneller läuft, obwohl das erste Diagramm 5.4 dieses vermuten ließ. Der Switch verbraucht nur mehr Ressourcen durch das Scheduling, wodurch weniger Nachrichten in gleicher Zeit erstellt werden können. Bei dem langsameren *Slow Endsystem* verschärft sich das Bild weiter. Hier können in der TTE-Simulation nur $\frac{1.465.838\ Messages}{\frac{10s}{0{,}089s}} = 13.045,9582$ Messages pro Sekunde erstellt werden, während es mit dem INET-Switch möglich ist $\frac{1755653\ Messages}{\frac{10s}{0{,}086s}} = 15.098,62$ Messages zu erzeugen.

5.3 Untersuchung der Latenz

Der letzte Abschnitt dieser Arbeit beschäftigt sich mit einer Latenzmessung in dem bereits verwendeten Netzwerk 5.3, das aus drei empfangenden und drei sendenden Endsystemen besteht. Dabei wird allerdings ein Best-Effort Stream an *Client 1* umgeleitet, wodurch dieser nun alle drei Trafficarten empfängt. Auszuwerten ist nun die Latenz mit der die Frames im Zielendsystem eintreffen. Der RC- und TT-VL kommen dabei vom *Server 1*, während die Best-Effort Nachrichten von *Server 2* verschickt werden. Der BAG-Value beträgt $1000\mu s$, was zu einer erhöhten Weiterleitung von BE-Nachrichten im Switch führt. Grafik 5.6 bildet einen Ausschnitt ab, der die Latenz mehrerer Pakete der drei unterschiedlichen Trafficarten, beim Eintreffen im Endsystem, zeigt. Dieser Ausschnitt wurde so gewählt, dass er eine Periode der gesamten Simulation zeigt.

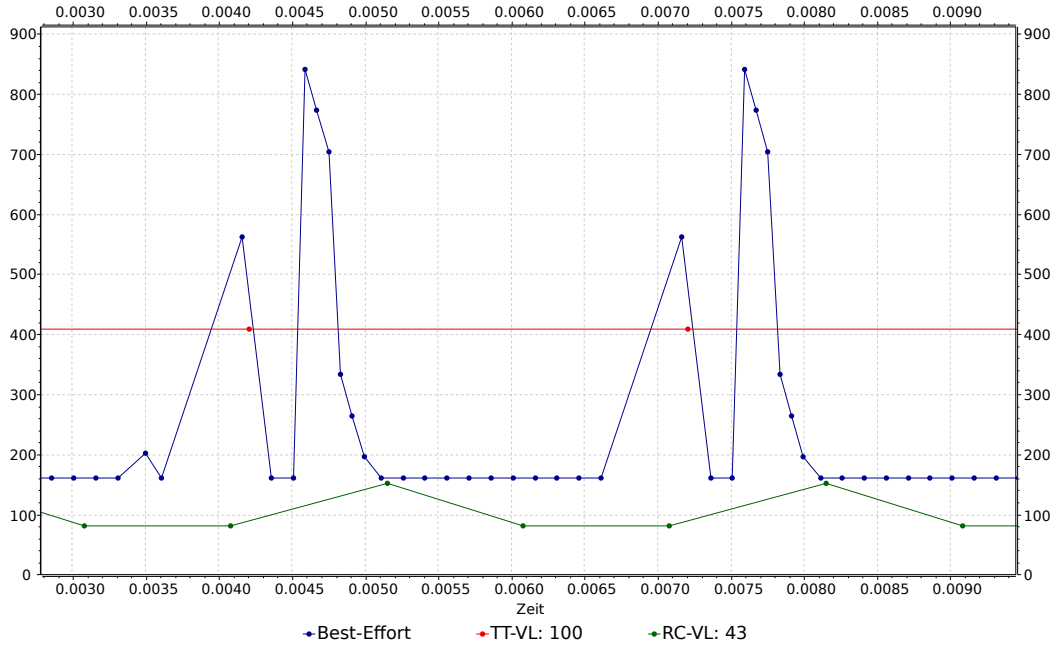

Abbildung 5.6: .

Die rote Linie beschreibt die Latenz der TT-Nachrichten mit der CT-ID 100. Durch die vordefinierten Empfangs- und Versendezeitpunkte von Time-Triggered Nachrichten sollte die Latenz hier durchgehend gleichbleiben, weil TT-Frames nicht durch andere Frames verzögert werden dürfen. Die beiden empfangenen abgebildeten Pakete zeigen die gleichbleibende Latenz von circa $400\mu s$. Diese wird hauptsächlich durch den Switch verursacht, der zwischen Empfangs- und Sendefenster die Nachrichten verzögert.

Interessanter ist die Latenz bei den RC und BE Nachrichten. Best-Effort Frames werden alle

$150\mu s$ im *Server 2* erstellt und in die Queue zum Versenden eingereiht. Somit kann man in der Latenz der Nachrichten den ansteigenden Füllstand der BE-Queue im Switch beobachten, wenn andere Frames eine höhere Priorität haben. Die Latenz beträgt bei den meisten BE-Frames um die $160\mu s$. Dies ist am blauen, geraden Strich zwischen $0,0052s$ und $0,0066s$ zu erkennen. Das heißt die Nachrichten werden ohne Verzögerung vom Switch weitergeleitet und kein anderes Frame blockiert den Ausgang. Das nächste BE-Frame erreicht *Client 1* allerdings mit einem deutlichen Abstand um $0.00725s$ herum und einer wesentlich größeren Latenz von $560\mu s$. Um dies zu erklären müssen wir ebenfalls den Schedule von *Server 2* beachten, der neben den Best-Effort Frames auch noch RC-Nachrichten verschickt. Der erste größere Peak kommt somit sowohl durch eine Verzögerung im Endsystem, als auch im Switch zu Stande, der kurz zuvor ein RC-Frame weitergeleitet hat (0.0071). Der zweite größere Peak bedeutet eine Ansammlung von BE-Frames in einer Queue, die nacheinander abgebaut wird. Dies wird auch durch das schnellere Eintreffen der Nachrichten deutlich. Hier werden in einem Abstand von $0.0007s$ nun sechs anstatt vier BE-Frames empfangen. Die Latenz der einzelnen Nachrichten nimmt dabei ab. Zu der Anstauung kommt es durch den Schedule eines TT-VLs, der über eine längere Zeit keinen Versand zulässt. Bei den Rate-Constrained Nachrichten lässt sich nur eine geringe Latenzveränderung ausmachen. Diese entsteht durch die leichte Verzögerung, wenn noch eine andere Nachricht auf dem Kanal übertragen wird.

6 Fazit

Die im Laufe dieser Bachelorarbeit erweiterte Simulation bietet dem Anwender eine Möglichkeit, die Konfiguration von AFDX-Netzwerken in einem transparenten Modell auszuführen. Das Modell bezieht sich dabei auf das von TTTech entwickelte Time-Triggered Ethernet, das eine Echtzeitlösung für ethernetbasierte Netzwerke darstellt. Zu diesem Zweck wurden zu Beginn die Merkmale des im ARINC-664 spezifizierten AFDX-Protokolls dargestellt. Dies war wichtig, da der zu implementierende, bandbreitenlimitierende Rate-Constrained Modus die Konzepte des “Avionics Full-Duplex Switched Ethernet Network” übernimmt. Dazu gehören in erster Linie die Prioritäten zwischen verschiedenen Virtual Links und die Bandwidth Allocation Gap, um die Steuerung der Bandbreite zu regulieren.

Der Hauptteil beschäftigt sich mit dem konkreten Simulationsmodell. Anhand von Aktivitätsdiagrammen der einzelnen Komponenten im Endsystem wird deren Funktionsweise erläutert und dokumentiert. Die Konzepte des TTE-Switches konnten größtenteils übernommen werden, und es war nur nötig, die im Endsystem erklärten und implementierten Funktionen auch hier einzuarbeiten. Anschließend findet eine Validation des erweiterten Simulationsmodells statt, um die korrekte Implementierung zu testen. Hierbei werden Blackbox-Tests, die sowohl im Endsystem als auch im Switch die wichtigsten Funktionen testen, durchgeführt. Im abschließenden Kapitel geht es um die Simulation kleinerer Netzwerke, um Unterschiede zwischen einem Standard-Ethernet Switch und dem implementierten Time-Triggered-Ethernet Switch zu analysieren. Dazu werden die beiden erwähnten Switche einem Performancetest unterzogen und getestet, wie sich der TTE-Switch unter Last mit vielen Virtual Links verhält.

Ziel dieser Arbeit war es, Grundlagenwissen über das AFDX-Protokoll zu geben, um ein Verständnis über den Rate-Constrained Traffic Modus zu schaffen und entsprechend in die Simulation einzuarbeiten. Beides wurde erfolgreich erzielt, sodass es mit der entstandenen Simulation nun möglich ist, in OMNeT++ die Konfigurationsdateien von Time-Triggered Ethernet einzulesen und Netzwerke bezüglich ihrer Last zu simulieren. Somit können Aussagen über das Zeitverhalten auf echter Hardware gemacht werden, bevor entsprechende Komponenten wie Switche und Endsysteme angeschafft würden. Zusammenfassend wurde erreicht, dass die implementierten Funktionen gemäß ihrer Spezifikation im AFDX-Protokoll funktionieren, Best-Effort Nachrichten mit geringerer Priorität übertragen werden und der be-

reits vorher integrierte Time-Triggered Traffic Modus in seiner Funktionalität nicht behindert wird. Somit können komplexe TT-Ethernet-Netzstrukturen kostengünstig simuliert werden.

Obwohl diese Arbeit mit großer Sorgfalt erstellt und die Funktionen implementiert wurden, kann im Rahmen der vorliegenden Arbeit kein vollständig integriertes AFDX-Protokoll aufgrund seines Umfanges realisiert werden. In zukünftigen Erweiterungen können Funktionen wie Redundanzmanagement, verschiedene Arten von Ports und eine vollständige Unterstützung des TTE-Modells implementiert werden. Zudem muss eine Zusammenführung der verschiedenen Codeteile von Endsystem und Switch stattfinden, damit diese auf ähnliche Weise funktionieren und keine Redundanz im Code besteht. Wurde dies durchgeführt, gilt es umfangreichere Endsysteme zu entwerfen, um sie in die Simulation einfließen zu lassen und realitätsnahe Bedingungen zu erzeugen.

Abschließend ist zu sagen, dass mit der Simulation von Time-Triggered-Ethernet eine Grundlage geschaffen wurde, die aktuelle Standards wie das AFDX-Protokoll unterstützen und somit die Simulation von bandbreitenlimitierten Ethernet mit zeitbasierten Verfahren verbindet.

Abkürzungsverzeichnis

AFDX	Avionics Full-Duplex Switched Ethernet
API	Application Programming Interface
ARINC	Aeronautical Radio, Incorporated
BAG	Bandwidth Allocation Gap
BE	Best-Effort
CT	Critical-Traffic
GPL	GNU General Public License
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
LSB	Least Significant Bit
MAC	Media Access Control
Mbit	Megabit
NED	Network Description
OMNeT++	Objective Modular Network Testbed in C++
OSI	Open Systems Interconnection
PCF	Protocol Control Frame
TCP	Transmission Control Protocol
TT	Time-triggered
TTE	Time-triggered Ethernet
UDP	User Datagram Protocol
VL	Virtual Link
VLID	Virtual Link Identifier
VLS	Virtual Link Scheduling
XML	Extensible Markup Language

Abbildungsverzeichnis

Literaturverzeichnis

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

[Bob Pickles 2006] BOB PICKLES: Avionics Full Duplex Switched Ethernet (AFDX) / SBS Technologies. URL `http://www.sierrasales.com/pdfs/AFDX_Overview.pdf`. – Zugriffsdatum: 2011-05-09, 2006. – Forschungsbericht

[Boger 1999] BOGER, Marko: *Java in verteilten Systemen - Nebenläufigkeit, Verteilung, Persistenz*. dpunkt, 1999. – I–XVI, 1–352 S. – ISBN 978-3-932588-32-7

[Condor Engineering Incorporated 2005] CONDOR ENGINEERING INCORPORATED: AFDX / ARINC 664 Tutorial (1500-049) / Condor Engineering. URL `http://www.acalmicrosystems.co.uk/whitepapers/sbs8.pdf`. – Zugriffsdatum: 2010-10-09, 2005. – Forschungsbericht

[Dieumo Kenfack 2010] DIEUMO KENFACK, Hermand: *TTEthernet Protokollstack für OMNeT++*. November 2010. – Projekt 1 Bericht

[Fluehr 2010] FLUEHR, Holger: Avionik und Flugsicherungstechnik : Einfuehrung in Kommunikationstechnik, Navigation, Surveillance. In: *Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on*, 2010, S. 347. – ISBN 978-3-642-01612-7

[GE Fanuc Intelligent Platforms] GE FANUC INTELLIGENT PLATFORMS: *TTEthernet - A Powerful Network Solution for Advanced Integrated Systems*. GE Fanuc Intelligent Platforms. – URL `http://www.gefanuc.com`

[Institute of Electrical and Electronics Engineers 2005] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS: IEEE 802.3: LAN/MAN CSMA/CD Access Method / IEEE. 2005 (IEEE 802.3-2005). – Standard

[Jasperneite u. a. 2004] JASPERNEITE, Juergen ; WATSON, Kym ; THEIS, Michael: Echtzeit-Kommunikation mit Switched-Ethernet: Ein Verfahren zur Bestimmung von oberen Zeitschranken. In: *A u. D Kompendium* (2004), S. 158–160. – URL

http://www.aud24.net/pi/media/pdf/aud/audk2004/ad4b0304.pdf. – Zugriffsdatum: 2011-06-08

[Mikolasek u. a. 2008] MIKOLASEK, V. ; ADEMAJ, A. ; RACEK, S.: Segmentation of standard Ethernet messages in the Time-Triggered Ethernet. In: *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, September 2008, S. 392–399

[OMNeT++ Community a] OMNET++ COMMUNITY: *INET Framework for OMNeT++ 4.1*. – URL http://inet.omnetpp.org/

[OMNeT++ Community b] OMNET++ COMMUNITY: *OMNeT++ 4.1*. – URL http://www.omnetpp.org

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

[Rech 2007] RECH, Joerg: *10 Gigabit pro Sekunde ueber Kupfer*. 2007. – URL http://www.heise.de/netze/artikel/10-Gigabit-pro-Sekunde-ueber-Kupfer – Zugriffsdatum: 2011-07-28

[Saad 2003] SAAD, Alexandre: *Das Automobil als Anwendungsgebiet der Informatik - ein Auto ohne Informatik geht das?* 2003. – URL http://subs.emis.de/LNI/Proceedings/Proceedings32/GI-Proceedings.32-4 – Zugriffsdatum: 2011-08-01

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

[Steiner u. a. 2009] STEINER, W. ; BAUER, G. ; HALL, B. ; PAULITSCH, M. ; VARADARAJAN, S.: TTEthernet Dataflow Concept. In: *Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on*, Juli 2009, S. 319–322

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

[Tanenbaum und van Steen 2007] TANENBAUM, Andrew S. ; STEEN, Maarten van: *Verteilte Systeme*. 2., Aufl. PEARSON STUDIUM, 2007. – URL http://www.amazon.de/gp/redirect.html%3FASIN=3827372933%26tag=ws%26lc – ISBN 3827372933

[Varga 2001] VARGA, András: The OMNET++ discrete event simulation system. In: *Proceedings of the European Simulation Multiconference*. Prague : SCS – European Publishing House, Juni 2001, S. 319–324

Inhalt der beigelegten DVD

Digitale Version dieser Arbeit Portable Document Format (PDF)

- Fabian_Kempf-Simulation_von_AFDX-Netzwerken.pdf

INET Framework

Sourcecode des erweiterten TTEthernet-Modells

Validierungsnetzwerke der unterschiedlichen funktionalen Tests

Analysenetzwerke der Performance- und Lasttests

Versicherung über Selbständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 23. August 2010	
Ort, Datum	Unterschrift