Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Projekt Bericht - SoSe 11

Hermand Dieumo Kenfack

TTEthernet Endsystem Modell für OMNeT++

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Dieser Bericht stellt die Arbeiten und Ergebnisse des zweiten Teils des zweisemestrigen Masterprojekts vor.

1.1 Allgemeine Zielsetzung

Das Gesamtziel des Masterprojekts über die beiden Semester ist es, ein Simulationsmodell für das TTEthernet-Protokoll zu entwickeln. Dieses soll ein Fundament für weitere Analysen und Evaluierungen der TTEthernet-Technologie darstellen. TTEthernet (vgl. Steiner, 2008) ist eine Echtzeit-Variante von Ethernet (vgl. Institute of Electrical and Electronics Engineers, 2005). Die Echtzeitfähigkeit wird gewährleistet durch die zeitliche Synchronisation aller Teilnehmer und das Aufstellen eines Zeitplans, welcher das simultane Senden mehrerer Echtzeitnachrichten auf dem selben Pfad ausschließt.

Das TTEthernet-Simulationsmodell soll auf der TTEthernet-Spezifikation von TTTech basieren (vgl. Steiner, 2008). Diese wurde zur Standardisierung bei der Society of Automotive Engineers (vgl. SAE - AS-2D Time Triggered Systems and Architecture Committee, 2009) eingereicht. Als Basisplattform zur Entwicklung des TTEthernet-Simulationsmodells wurde die OMNeT++-Simulationsumgebung (vgl. OMNeT++ Community, b) und das INET-Framework (vgl. OMNeT++ Community, a) festgelegt. Grund hierfür ist die Eignung von OMNeT++ für die Simulaion von Kommunikationsnetzwerken. Außerdem bietet das INET-Framework bereits ein Simulationsmodell des standard Ethernet-Protokolls, auf dem das TTEthernet-Simualtionsmodell basieren kann.

1.2 Rückblick Projekt 1

Im Rahmen des Projekts 1 wurde der TTEthernet-Protokollstack auf der Seite des Endsystems zum großen Teil entwickelt. Dafür wurde der INET-Standard-Ethernet-Protokollstack um den TTEthernet-Dienst erweitert. Dieser besteht aus einer Konfiguragtion, einem Scheduler und einem Klassifikationsmodul. Die Konfiguration wird offline durchgeführt und legt unter anderem fest, zu welchem Zeitpunkt Nachrichten gesendet/erwartet werden. Der Scheduler hat die Aufgabe das Versenden/-Empfangen der Nachrichten zu triggern. Dafür verwendet er die Nachrichtenplanung. Das Klassifikationsmodul ordnet die Nachrichten beim Eintreffen ein und leitet diese bei der Anordnung des Schedulers weiter. Diese Module stellen den wesentlichen Unterschied zum Standard-Ethernet dar. Weiterhin wurde eine Auswahl der TTEthernet-API-Funktionen (vgl. TTTech Computertechnik AG, 2008) implementiert. Dieses ermöglicht, echte Applikationen in der Simulationsumgebung vor ihren

Einsatz in echten Systemen zu entwickeln bzw. zu testen. Für eine detaillierte Beschreibung des Projekts 1 siehe Dieumo Kenfack (2010). Das TTEthernet-Endsystem-Modell konnte aufgrund seiner hohen Komplexität und der begrenzten Zeit des Projekts 1 nicht vollständig realisiert werden. Dieses wird im Rahmen des Projekts 2 sowie der Bachelorarbeit von Fabian Kempf (vgl. Kempf, 2011), welche das RC-Traffic-Modell nach der AFDX- und der TTEthernet-Spezifikation im Endsystem realisiert, weiter ausgebaut.

1.3 Projekt 2 Ziele

Im Rahmen des Projekts 2 soll das TTEthernet-Endsystem-Modell für OMNeT++ weiter ausgebaut werden mit dem Fokus auf folgenden Punkten:

- Komplettierung und Verbesserung des TTEthernet-Protokollstacks. Im Projekt 1 wurden lediglich Grundfunktionen (wie Senden und Empfangen von Nachrichten) der TTEthernet-API implementiert. Eine weitere wichtige API-Funktion ist die Callbackschnittstelle, welche dazu dient Funktionen der Applikationsschicht zu registrieren. Diese werden vom Protokoll-Layer aufgerufen wenn Nachrichten eintreffen. Im Projekt 1 wurden die Simulationsdaten (z. B. end-to-end delay) in der Applikationsschicht aufgenommen. Es wäre besser diese im Protokoll-Layer zu realisieren. So muss nicht jeder Applikationsentwickler dieses selber tun. Weiterhin sollen in TTEthernet-Protokollschicht Fehler (wie z. B. zur falschen Zeit ankommende Frames) erkannt und protokolliert werden.

- Integration des TTEthernet-Konfigurationsmodells in der Simulation. Das TTEthernet-Konfigurationsmodell ist ein von TTTech entwickeltes XML-Schema (vgl. TTTech Computertechnik AG) basierend auf dem Ecore-Format des Eclipse Modeling Framework (EMF) Projekts (vgl. Eclipse Foundation, a) zur Modellierung bzw. Beschreibung von TTEthernet-Netzwerken. Es ist vergleichbar mit dem FIBEX-XSD (Field Bus Exchange Format) (vgl. FIBEX Expert Group, 2009), welches zur Modellierung bzw. Beschreibung von LIN-, CAN- und FlexRay-Netzwerken dient. Die Integration des TTEthernet-Konfigurationsmodells soll es möglich machen, die für echte Systeme entworfene Konfigurationen in der Simulation einzusetzen und umgekehrt.

- Implementierung einer Video-Streaming-Applikation, womit echte Video-Daten durch die Simulation gesendet und empfangen werden können. Dadurch kann die Fähigkeit der Simulation für die Implementierung und das Testen von echten Applikationen gezeigt werden.

2 Beschreibung der Architekturkonzepte

In diesem Kapitel wird die Architektur des TTEthernet-Protokollmodells für die Simulation in OMNeT++ aus Sicht des Endsystems dargelegt. Details zum Entwurf des Switch-Modells können aus Steinbach (2010) und Steinbach u. a. (2011) entnommen werden.

2.1 TTEthernet Protokollstack

Der TTEthernet-Protokollstack entspricht dem Standard-Ethernet-Protokollstack mit Erweiterungen der Echtzeiteigenschaften des TTEthernet-Protokolls. Der TTEthernet-Protokollstack besteht aus drei Schichten: Die physikalische, die Data-Link- und die Applikationsschicht. Wobei die beiden Letzen sich in weiteren Subschichten unterteilen, und zwar die MAC- (Media Access Control), TTEthernet-Protokoll-, API- und Applikations-Subschicht (siehe Abbildung 2.1).

Die physikalische Schicht und die MAC-Subschicht korrespondieren mit denen des Standard-Ethernet-Protokollstacks von INET. Die TTEthernet-Protokollschicht beinhaltet die Echtzeiteigenschaften des TTEthernet-Protokolls. Diese wird im Abschnitt 2.2 dargelegt. Die TTEthernet-API ist eine von TTTech festgelegte Schnittstelle zum Initialisieren des Protokollstacks sowie Versenden/-Empfangen von Nachrichten auf der Applikationsebene. Die Berücksichtigung der API beim Entwurf des Endsystem-Modells soll es möglich machen Applikationen, die in echten Systemen laufen sollen, in der Simulation zu entwickeln und umgekehrt.

Die Architektur des TTEthernet-Protokollstacks hat sich vom alten Entwurf (aus dem Projekt 1) minimal geändert. In der alten Realisierung leitet die TTEthernet-Protokollschicht von der INET-LLC-Schicht (Logical Link Control) ab. Es wurde aber festgestellt, dass die Funktionalitäten der LLC-Schicht im TTEthernet-Protokoll nicht benutzt bzw. direkt in der TTEthernet-Protokollschicht realisiert wurden. Daher wurde aus der TTEthernet-Protokollschicht eine eigenständige Komponente gemacht. Dies verbessert die Verständlichkeit und die Wartung des Stacks.

2.2 TTEthernet-Protokollschicht

Die TTEthernet Protokollschicht hat als Funktionalität die TTEthertnet-Echtzeiteigenschaften zu gewährleisten. Die Echtzeitfähigkeit von TTEthernet beruht auf einem Verfahren, welches auf Zeitschlitzen aufbaut (TDMA -Time Division Multiple Access-). Dabei wird ein Zeitplan offline vordefiniert, nach dem die Echtzeit-Nachrichten gesendet und empfangen werden. Alle Netzwerkteilnehmer müssen nach diesem Zeitplan operieren. Der Zeitplan ist Teil der **Konfiguration** des

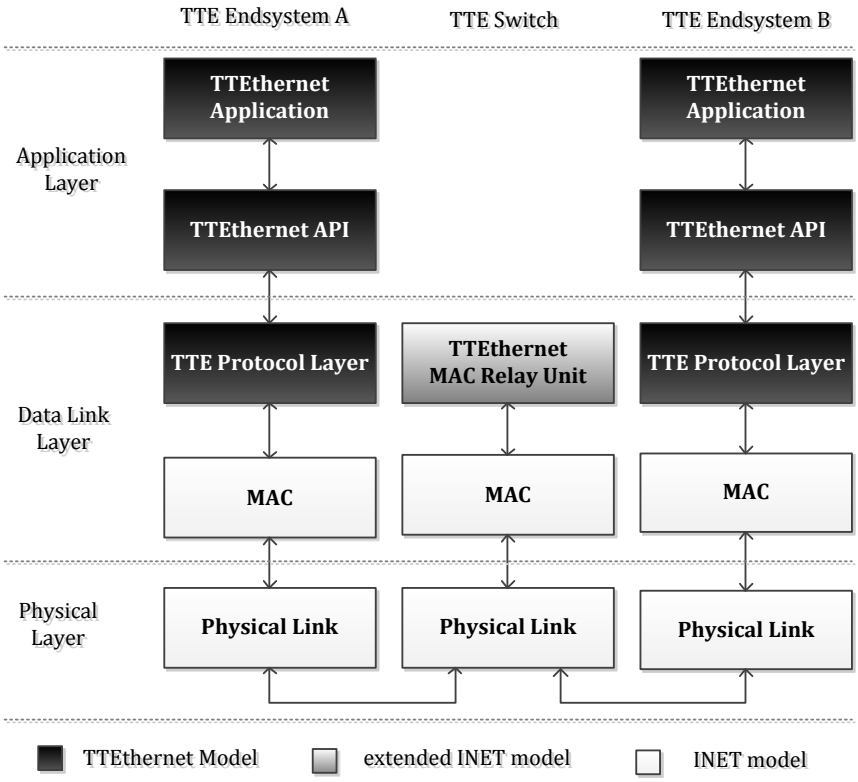

Abbildung 2.1: TTEthernet Protokollstack

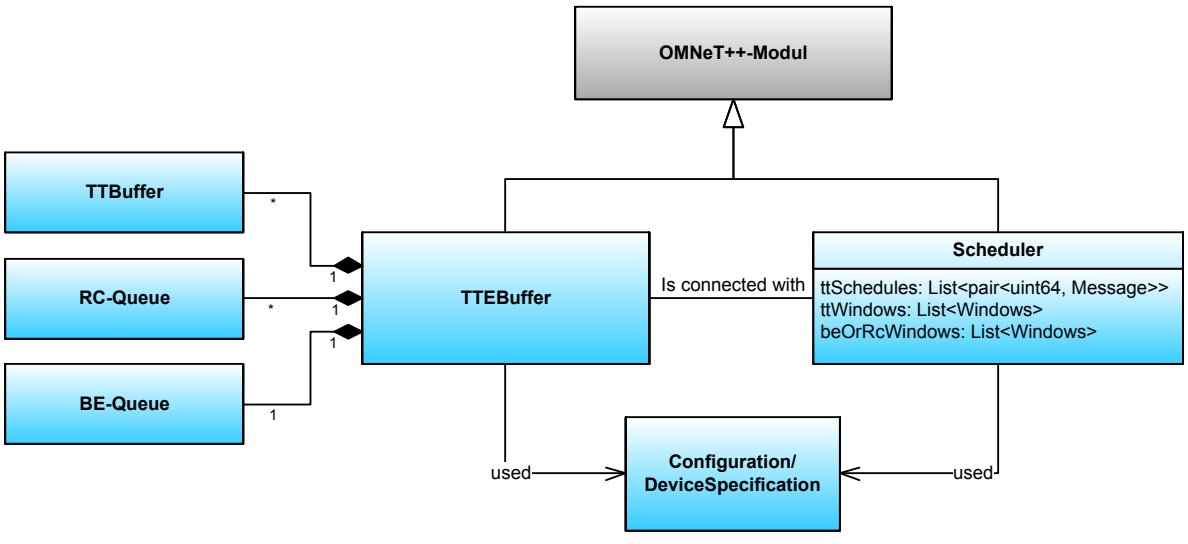

Abbildung 2.2: TTEthernet Protokollschicht Objektmodell

Netzwerks und befindet sich in der „DeviceSpecification“ jedes Devices (Endsysteme und Switsches). Der **Scheduler** triggert anhand des Zeitplans das Senden der Nachrichten aller Traffic-klassen. Der **TTEBuffer** besteht aus mehreren TTBuffers (einer pro TT-CT-ID), mehreren RC-Queuen (eine pro RC-CT-ID-Priorität) und einer BE-Queue. Der TTEBuffer klassifiziert die Nachrichten beim Eintreffen in den entsprechenden Puffern und leitet diese auf Befehl vom Scheduler weiter. Die Komponenten der TTEthernet-Protokollschicht und deren Beziehungen untereinander werden durch das Objekt-Modell in der Abbildung 2.2 dargestellt.

2.3 Komponenten des Konfigurationsprozesses

Die Konfiguration spielt eine sehr große Rolle in allen Entwicklungspahsen (vom Design bis zum Test) von TTEthernet Netzwerken. Beim Design werden durch die Konfiguration alle Teilnehmer eines Netzwerks mit ihrem physikalischen und logischen Verbindungen festgelegt. Ferner werden alle Nachrichten sowie deren Sender und Empfänger definiert. Zu jeder Echtzeit-Nachricht sind auch die zugehörigen Zeitanforderungen (maximale Übertragungszeit, maximaler Jitter, etc.) festzulegen. Wenn Fehler bei der Konfiguration auftreten, können die Funktionalitäten und die Anforderungen des Systems nicht eingehalten werden. Daher ist es unerlässlich, einen modellbasierten und automatisierten Konfigurationsprozess einzusetzen, da dadurch Fehler minimiert und Änderungen vereinfacht werden.

Ein möglichst modellbasieter und automatisieter Konfigurationsprozess für TTEthernet-Netzwerke wird in der Abbildung 2.3 dargestellt. Die Basis des Konfigurationsmodells ist das Protokoll-Metamodell. Dieses ist nichts anderes als ein UML-Modell der Protokoll-Spezifikation und legt fest, wie das Netzwerk konfiguriert werden soll. Es Definiert zum Beispiel, welche Typen von Teilnehmern im Netzwerk erlaubt sind (nur Endsysteme und Switches), wie die Teilnehmer miteinander verbunden werden sollen und welche Eigenschaften und Funktionalitäten diese haben können. Anhand eines UML-Tools werden aus dem Protokoll-Metamodell die Metamodellspezifikationsdatein (System- und Device-Specification.xsd) generiert. Diese werden von einem Code-Generator eingelesen, um den Code für die Konfigurationsobjekte und den XML-Modellparser zu generieren. Dieser wird entweder in echte Systeme oder in die Simulation eingebunden, jenachdem welche Plattform verwendet wird. Der Entwurf des Metamodells wird in der Regel „einmal“ durchgeführt, es sei den, die Spezifikation des Protokolls ändert sich. Nachdem das Metamodell feststeht, kann das Netzwerk designt werden. Dies wird in der Regel über einen Netzwerkdesigner (GUI-Tool) durchgeführt. Mithilfe des Netwerkdesigners kann z.B. grafisch festgelegt werden: alle Teilnehmer des Netzwerks, alle physikalische und logische Verbindungen, alle Nachrichten und deren Sende-Zeiten inklusive Synchronisations-Nachrichten und die zugehörigen Sender und Empfänger, Traffic-Profile und alle Routinginformationen. Nachdem das Netzwerk konfiguriert wurde, können die entsprechenden Konfigurationsdateien generiert werden (eine für das gesamte System und jeweils eine pro Teilnehmer). Alle Konfigurationsdatein zusammen stellen das Netwerkmodell dar. Anhand des Metamodells überprüft der Netzwerkdesigner, ob der Ingenieur/Benutzer das Netzwerk konform zum Metamodell designt hat, und gibt bei nicht Konformi-

tät entsprechende hinweise bzw. Fehler aus. Nachdem das Netzwerk konfiguriert wurde, kann dieses simuliert werden. Hierfür werden die Konfigurationsdateien in die Simulationsumgebung eingegeben. Diese parst daraufhin die Dateien anhand des Modellparsers und erzeugt die Objekte für die Simulation, z. B. Nachrichten-IDs, Puffer, Schedulingtabellen, Routingtabellen, etc. Bei der Analyse der Simulationsergebnisse werden die Netzwerkanforderungen in Betracht gezogen, um zu überprüfen, ob das Netzwerkmodell diese erfüllt. Wenn das nicht der Fall ist, kann das Netzwerkmodell über den Netzwerkdesigner angepasst werden.

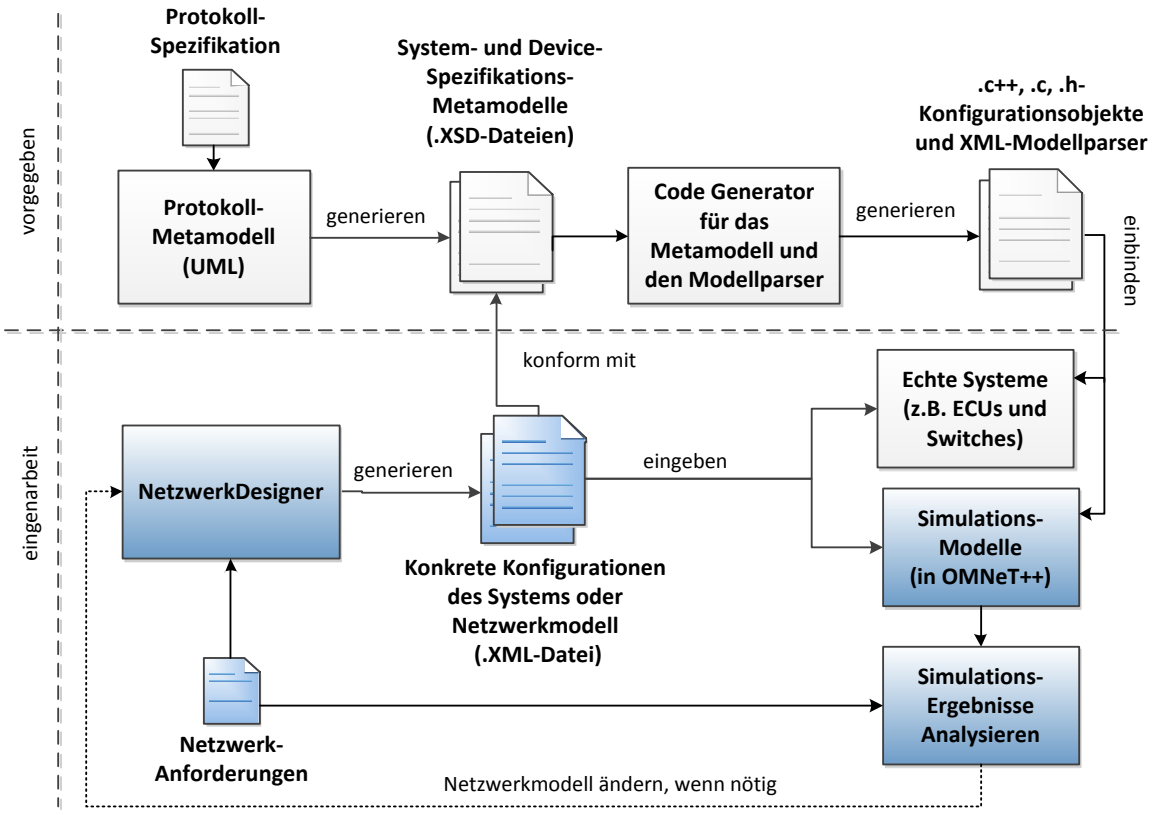

Abbildung 2.3: Konfigurationsprozess

3 Realisieren der Architekturkonzepte

In diesem Kapitel wird die Realisierung des Konfigurationsprozesses und Implementierungsdetails des Protokollstacks dargelegt.

3.1 Konfigurationsprozess

Bei der Realisierung des im Abschnitt 2.3 beschriebenen Konfigurationsprozesses (siehe Abbildung 3.1) wurde das von TTTech entwickeltes TTEthernet-Metamodell verwendet (vgl. TTTech Computertechnik AG). TTTech hat das Metamodell mit den EMF-UML2-Tools entwickelt. Das EMF (Eclipse Modeling Framework ist ein Open-Source-Java-Framework bzw. Eclipse-Plugin zur Modellierung von Applikationen und automatischen Generierung von Codes basierend auf strukturieten Modellen (vgl. Eclipse Foundation, a). Die von EMF-Tools generierten Spezifikationsdatein (.ecore) basieren nicht auf den Standard-XSD-Datentypen, sondern auf den Datentypen des Ecore-Metamodells (vgl. Eclipse Foundation, b).

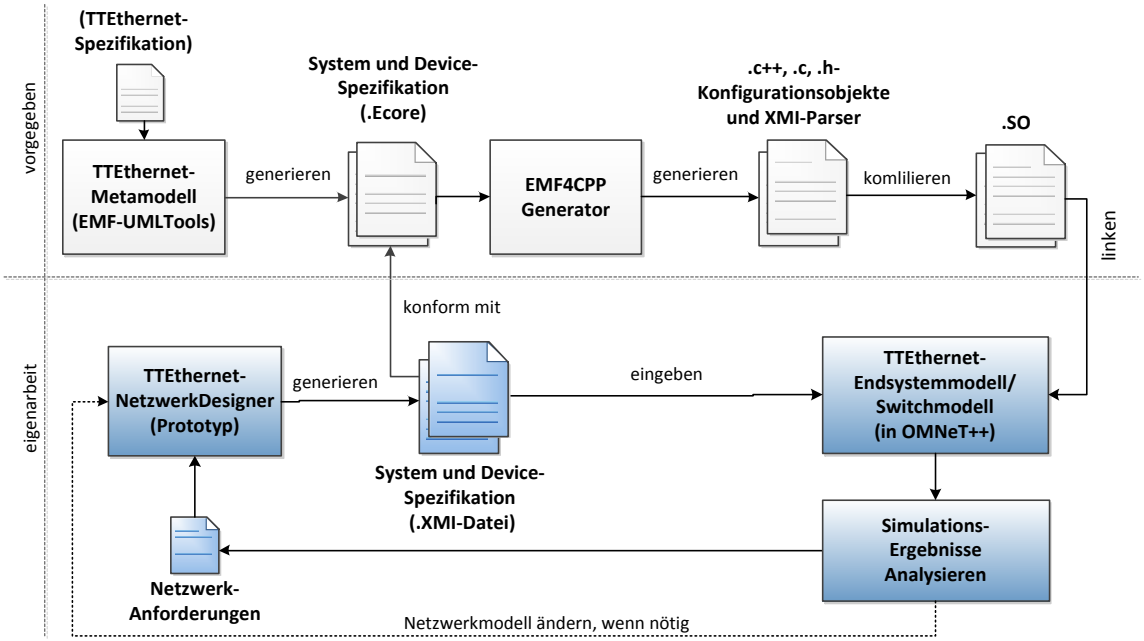

Abbildung 3.1: Realisierung des Konfigurationsprozesses

Das EMF bietet lediglich Framworks zur Generierung von Java Codes und Modelparser an. Das TTEthernet-Simulationsmodell ist jedoch in C++ implementiert. So bräuchte man eher einen C++ Code-Generator. Es konnte leider keiner der Standard XSD-Code-Genratoren verwendet werden, da diese keine Ecore-Datentypen verstehen. Diese Lücke schließt das Projekt EMF4CPP (vgl. Senac und Sevilla) der Universität Murcia. EMF4CPP besteht aus zwei Teilen. Einem Sourcecode-Generator, um aus Ecore-Meta-modellen C++-Code zu generieren, und den Bibliotheken für das Parsen und Serialisieren der Modelle im XMI-Format. Der generierte Code wird zu einer Bibliothek kompiliert. Die Simulationsmodelle werden gegen diese und die EMF4CPP-Libraries gelinkt. XMI-Dateien (Netzwerkmodell) die konform zum Metamodell sind, können nun von den Simulationsmodellen zur Laufzeit eingelesen werden (vgl. Senac u. a., 2010).

3.2 TTEthernet Protokollstack Verhalten

In diesem Abschnitt wird das Verhalten der Schichten des Protokollstacks und deren verhalten untereinander anhand von SDL-Diagrammen (Specification and Description Language) dargelegt.

3.2.1 Applikations-Verhalten

Abbildung 3.2 stellt das Verhalten der Applikationsschicht dar. Bei der Initialisierung muss jede Applikation über die API eine Callback-Methode für jede CT-ID und jeden BE-Channel registrieren, auf die sie Nachrichten empfangen soll. Eine Sammelmethode um mehrere CT-IDs bzw. BE-Channels zu registrieren, ist auch erlaubt. Diese soll jedoch mehrmals mit verschiedenen CT-IDs bzw. BE-Channels registriert werden. Anhand dieser Registrierung weißt die TTE-Protokollschicht welche Applikation welche Nachricht empfangen soll und ruft die entsprechende Methode auf, wenn die Nachricht eingetroffen ist. So kann die Applikation über die „Read-Funktion“ der API die Nachricht lesen. Die Applikation sendet ihre Nachrichten über die „Write-Funktion“ der API. Erwähnenswert ist, dass das Senden und Empfangen von TTEthernet-Nachrichten über die TTEthernet-API pufferbasiert ist. Für jede zu sendende TT-Nachricht wird ein Sendepuffer (TT_TX_Puffer) und für die zu empfangenen ein Empfangspuffer (TT_RX_Puffer) angelegt. Diese werden über die CT-ID der Nachricht eindeutig identifiziert. Für BE-Nachrichten dagegen wird für alle Channels eine Sende- und Empfangs-Queue angelegt. Um auf einen Puffer zuzugreifen, muss dieser gesperrt und danach wieder entsperrt werden (nach der Spezifikation der TTEthernet-API). Somit wird sichergestellt, dass die Applikation den Puffer nicht beschreibt, während dieser von der Protokollschicht gelesen wird und umgekehrt. Durch den direkten Zugriff der Applikationen auf die Puffer, ist es z.B. möglich, nur die nötigten Bytes der Nutzlast zu lesen. Somit wird die Echtzeit-Performance optimiert (vgl. TTTech Computertechnik AG, 2008)

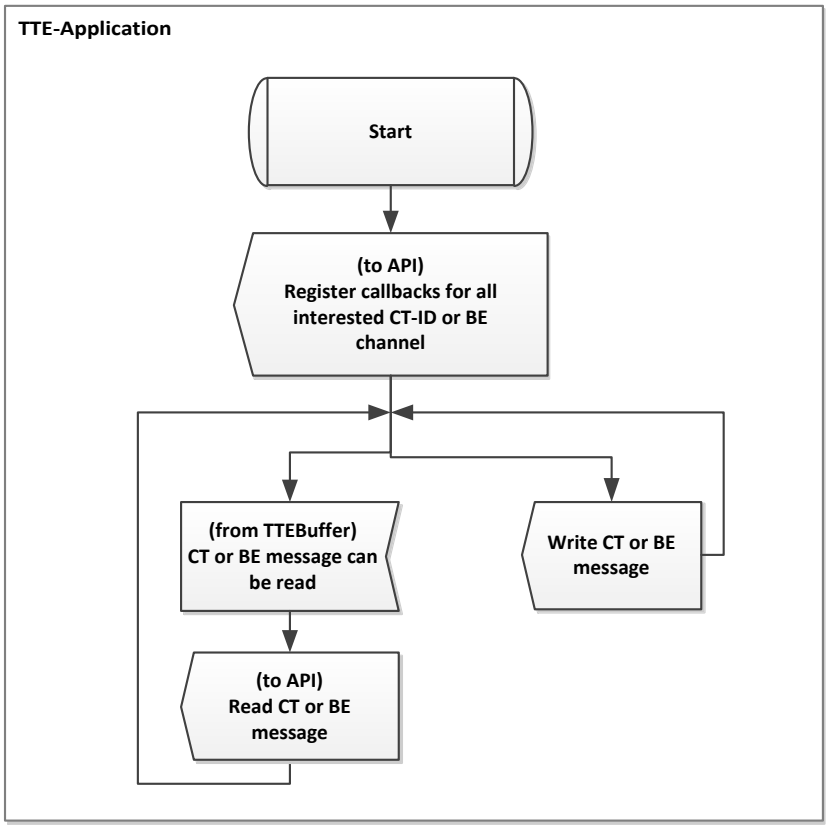

Abbildung 3.2: Applikation-Verhalten

3.2.2 API-Verhalten

Die TTEthernet-API (vgl. TTTech Computertechnik AG, 2008) trägt alle Callback-Registrierungen ein, die von der Applikation kommen. Außerdem konvertiert sie die von der Applikation kommenden TTEthernet-Frame in einen INET-Frame vor dem Speichern in dem entsprechenden Puffer. Durch die Konvertierung des TTE-Frames in INET-Frame, wird die Erzeugungszeit des Frames von der Simulation gespeichert. Diese entspricht auch der Sendezeit, die für die Berechnung der Verzögerungszeit der Nachricht beim Empfangen verwendet wird. Wenn die Applikation eine Nachricht empfangen will, ruft sie die entspreche Lese-Funktion der API auf und übergibt dabei die CT-ID bzw. Channel-ID und die Traffic-Klasse. Somit weiß die API auf welchen Puffer zugegriffen werden soll. Die API ließt dann den INET-Frame aus dem entsprechenden Puffer und konvertiert diesen in einen TTE-Frame zurück, welcher in den Applikations-Puffer gespeichert wird bzw. dem Zeiger darauf der Applikation bekannt gegeben wird. Anschließend merkt sich die API die Empfangszeit der Nachricht, ließt die Sendezeit und rechnet die Ende-zu-Ende-Verzögerung aus.

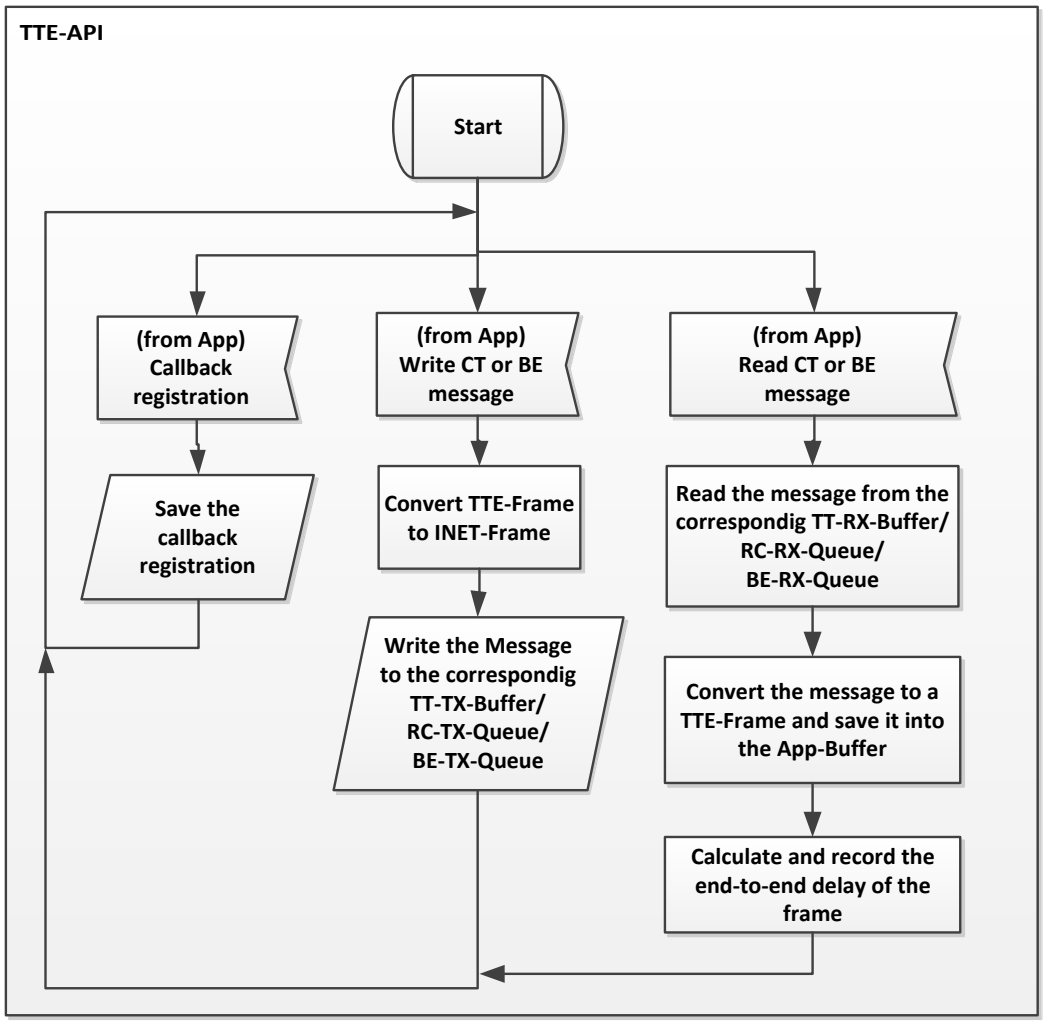

Abbildung 3.3: Verhalten der TTEthernet-API

3.2.3 Scheduler-Verhalten

Das Verhalten des Schedulers wird in der Abbildung 3.4 dargestellt. Der Scheduler hat als Aufgabe das Senden von „critical messages“ zu triggern und die Berechnung der Lücken zum Senden von „best effort messagesänhand des zur Design-Zeit erstellten Zeitplans. Dafür muss er die in der Simulation eingebundenen DeviceSpecification des Devices einlesen und die Schedule-Tabelle konstruieren. In jedem Zyklus plant er dann eigene interne Nachrichten (oder auch „self message“ nach OMNeT++) nach dem Schedule. Wenn die Interne Nachrichten eintreffen (z. B. Send-TT), leitet er diese Nachricht an dem TTEBuffer weiter. Somit kann der TTEBuffer die entspechende Nachricht lesen und diese der MAC-Schicht weiterleiten.

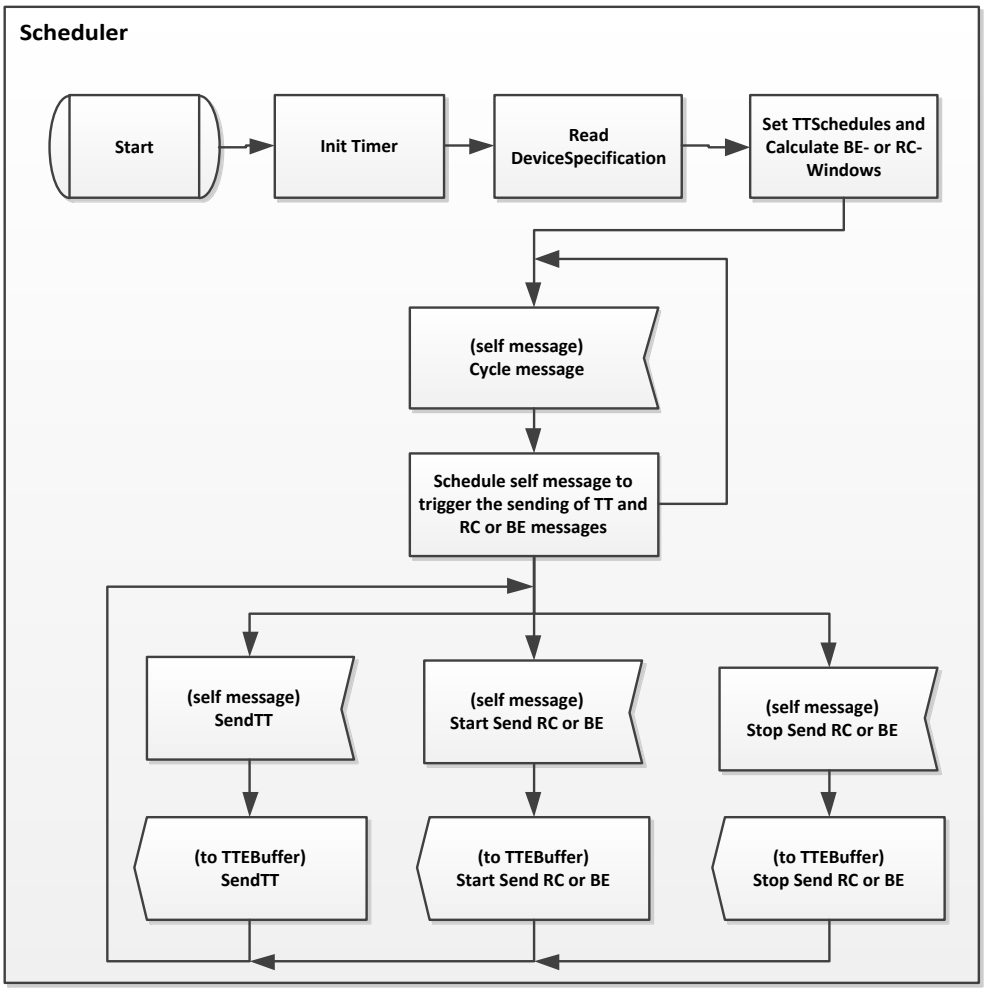

Abbildung 3.4: Scheduler-Verhalten

3.2.4 TTEBuffer-Verhalten

Das Verhalten des TTEBuffers wird durch Abbildung 3.5 illustriert. Der TTEBuffer kann zwei „Befehl-Nachrichten“ von Scheduler bekommen. Einerseits, damit er BE- bzw RC-Nachrichten sendet bzw. nicht mehr sendet andererseits, damit er eine TT-Nachricht sendet. Beim Letzteren ließt er die TT-Nachricht aus dem entsprechenden TT-TX-Buffer (TT-Tranceiver-Buffer) und leitet diese der MAC-Schicht weiter. Wenn er den Befehl „Start send BE“ empfängt, leitet er so lange BE-Nachrichten der MAC-Schicht weiter, sofern die Queue nicht leer ist, bis er den Befehl „Stop send BE“ Empfängt. Darüber hinaus kann der TTEBuffer eine Nachricht von der MAC-Sicht empfangen. Hierauf überprüft er, ob sich eine Applikation für diese eingetroffene Nachricht registriert hat. Wenn dies der Fall ist, wird die Empfangszeit im Falle einer TT-Nachricht überprüft und die Nachricht verworfen, wenn die Zeit nicht eingehalten wurde. Danach wird die Nachricht in dem entsprechenden RX-Buffer/Queue

(Receive-Buffer) gespeichert und der Applikation über die registriete Callback-Methode bekannt gegeben, dass die Nachricht gelesen werden kann. Wie das genaue Triggern und Weiterleiten von RC-Nachrichten gehandhabt wird ist nicht Bestandteil dieser Arbeit. Siehe hierfür die Bachelor-Arbeit von Fabian Kempf (vgl. Kempf, 2011).

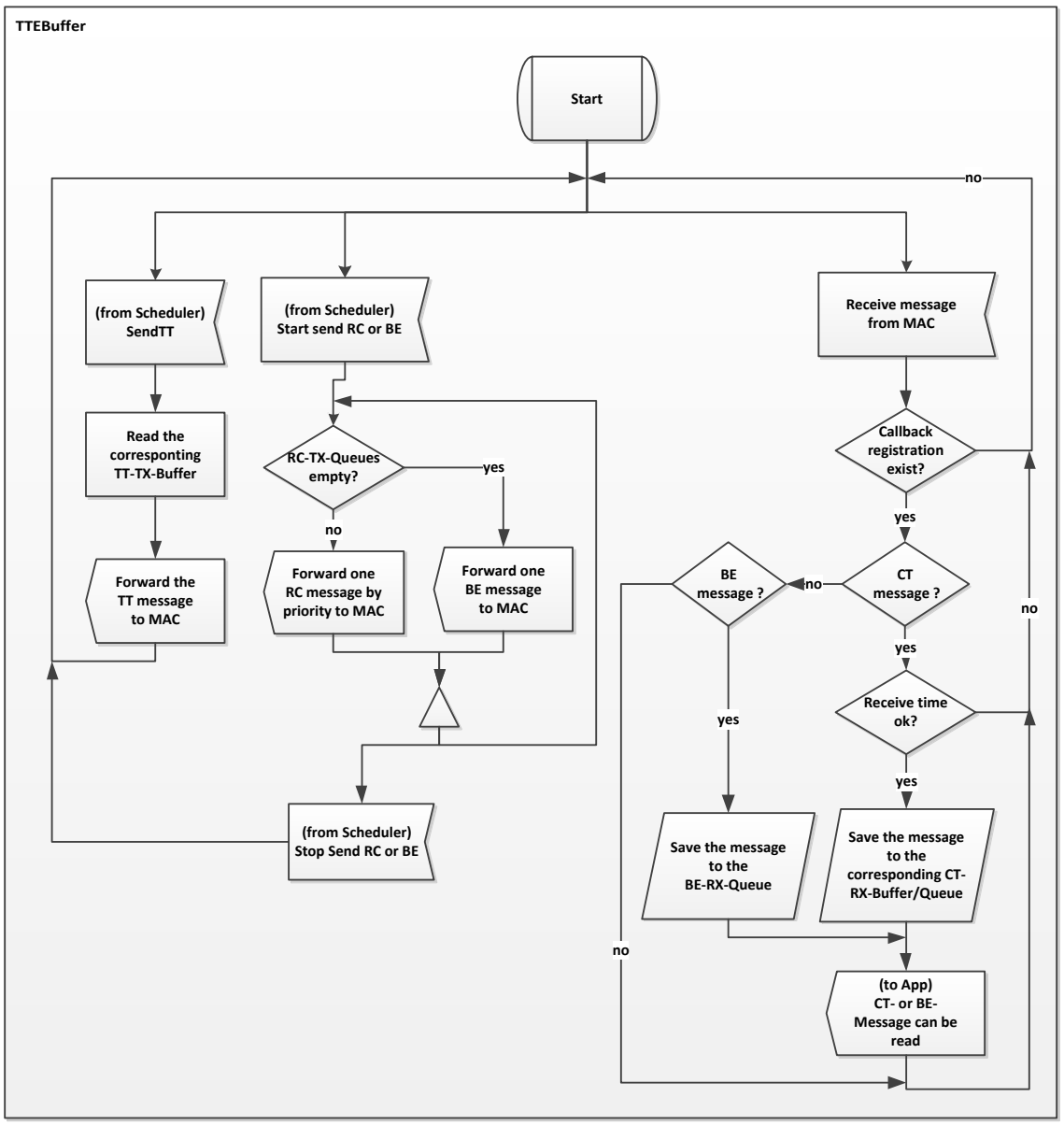

Abbildung 3.5: TTEBuffer-Verhalten

4 Design eines exemplarischen Netzwerks

In diesem Kapitel wird das zum Testen des Protokollstacks und des Konfigurationsprozesses erstelle exemplarische Netzwerk beschrieben.

4.1 Designen mit dem prototypischen TTEthernetDesigner

Ein Tool für das Designen von TTEthernet-Netwerken und die automatische Generierung der XML-Datein, welche das Netzwerk beschreiben, ist unabdingbar, da die manuelle Erstellung der XML-Datein sehr zeitaufwendig und fehleranfällig ist. Dieses sollte mit dem Tool TTEPlan von TTTech durchgeführt werden. Da dieses während dieser Arbeit noch nicht zur Verfügung stand, wurde entschieden einen prototypischen TTEthernet-Netzwerkdesigner zu entwickeln. Dieser wurde anhand des Microsoft Visio Automation (vgl. Microsoft) verwirklicht. Durch die Visio Automation ist es möglich, MS-Visio (Microsof-Visio) Plugins oder eigene Applikationen basierend auf dem MS-Visio Zeichenblatt zu entwickeln. MS-Visio stellt ein SDK (Software Development Kit) zur Verfügung, womit alle Shapes eines Zeichenblattes und eines Shapes gescannt werden können. Außerdem werden auch Funtionen zur Verfügung gestellt, womit die Shapes-Eigenschaften abgefagt und gesetzt werden können. Beispiel Shapes-Eigenschaften sind: Text, Name, Höhe, Breite, Verbindungen zu den andereren Shapes, etc. Über die MS-Visio Schablone ist es auch möglich, eigene Shapes zu erstellen und diese beim Zeichnen zu verwenden. So wurden spezielle Shapes, welche die Entitäten (Endsystem, Switches, physikalische und logische Links, etc.) von TTEthernet-Netzwerken darstellen, erstellt. Eine mit dem prototypischen TTEthenet-Netzwerkdesigner erstellte phisikalische und logische Topologie mit zwei Switches, zwei TT-Sender (videoServer1 und 2) und zwei TT-Empfänger (videoClient1 und 2) wird durch Abbildung 4.1 illustriert. Diese Konfiguration ist auch die Basis für das exemplarische Netzwerk. Neben dem grafischen Entwurf von TTEthernet-Komponenten, wurde auch das TTEthernet-Objekt-Modell (vgl. TTTech Computertechnik AG) in c# im TTEtnernet-Netzwerkdesiger implementiert. Das heißt, dass es für jede TTEthernet-Shape-Entität ein korespondierendes c#-Objekt gibt. Dies soll es möglich machen, nur durch Objekt-Instanzierung das Netzwerk zu gestalten. So ist man nicht auf eine maximale Anzahlt von Knoten bzw. Nachrichten wie bei der graphischen Modellierung limitiert.

Im prototypischen TTEthernet-Netzwerkdesigner wurde lediglich ein statisches Scheduling für maximale Frames implementiert. Grund hierfür ist die Komplexität des dynamischen Schedulings, welches von vielen Faktoren wie der Topologie, der Anzahl der Switches und der Frame-Länge abhängt. Das gesamte statische Scheduling basiert auf der „start_time“. Diese ist der Zeitpunkt im dem

der TTEBuffer den Befehl des Schedulers zum Weiterleiten der Nachricht an die MAC-Schicht empfangen soll. Um z. B. den Sendezeitpunkt der Applikation festzulegen, wird die Zeit, die die Applikation benötigt um die Nachricht dem Protokoll-Layer zu senden (APP_TO_PROTOCOL_LAYER_TIME) von der „start_time“ substrahiert. Analog hierzu werden alle anderen Zeitpunkte (incoming-start/end, outgoing-start/end) festgelegt siehe Abbildung 4.2.

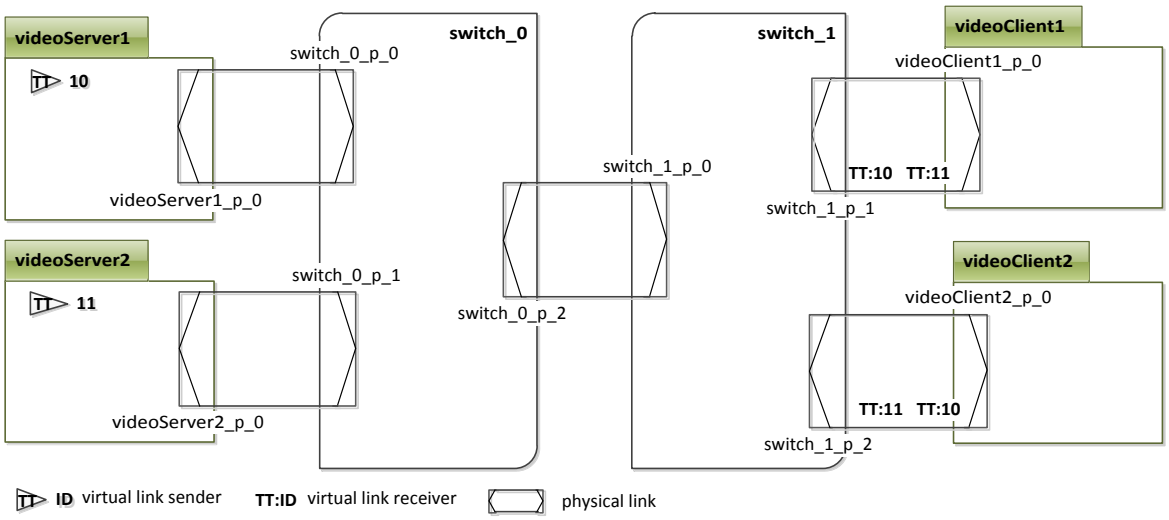

Abbildung 4.1: Topologie des exemplarischen Netzwerks in TTEthernet-Netzwerkdesigner

4.2 Kommunikationszyklus des examplarischen Netzwerks

Der Kommunikationszyklus des examplarischen Netzwerks wird in der Abbildung 4.3 angezeigt. Man kann daraus sehen, dass die Time-Triggered-Nachricht mit der ID 10 jede Millisekunde und die mit der ID 11 alle zwei Millisekunden gesendet werden. Best-Effort-Nachrichten werden in den Lücken weitergeleitet, das heißt, wenn TT-Nachrichten nicht gesendet werden. Im dem Zyklus aus Abbildung 4.3 können grob geschätzt 4 BE-Nachrichten mit maximalen Frame-Längen bei einer Übertragungsrate von 100 Mbit/s weitergeleitet werden ohne, dass es zu zu großen Verzögerungen kommt. Erwähnenswert ist, dass sich alle BE-Teilnehmer diese Lücken teilen müssen. Die Weiterleitung der BE-Nachrichten erfolgt dann nach dem FIFO (First In First Out) Prinzip, wenn diese gepuffert werden müssen.

4.3 Topologie im OMNeT++

Wie aus der Abbildung 4.4 zu entnehmen ist, sind zu den Endsystemen in der Abbildung 4.1 weitere drei hinzugefügt worden. Diese sind lediglich Best-Effort-Endsysteme. Wobei der „genAppServer“ und

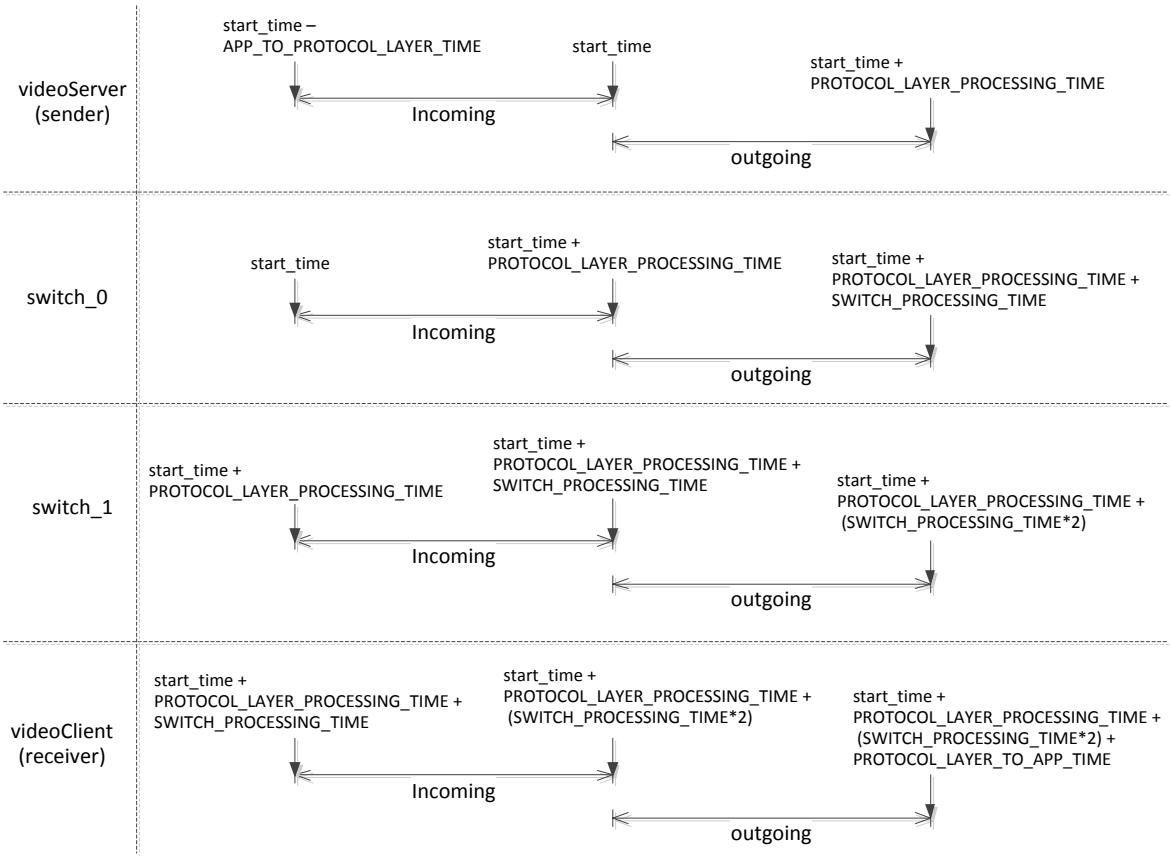

Abbildung 4.2: Statisches Scheduling des examplarischen Netzwerks

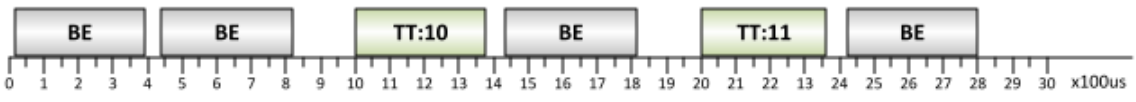

Abbildung 4.3: Kommunikationszyklus des examplarischen Netzwerks

der „genAppServer2“ Sender sind und der „genAppClient“ Empfänger ist. Erwähnenswert ist, dass alle Komponenten des exemplarischen Netzwerks in der Simulation mit echten Daten kommunizieren. Die videoServer1 und 2 lesen Video-Bilder ein, bilden daraus mehrere TT-Frames und senden diese an die viedeoClients. Die videoClients bilden aus den empfangenen TT-Frames die Video-Bilder und speichert diese. Die von den viedoClients gespeicherten Bilder werden dann von einer externen echten Video-Applikation gelesen und angezeigt. Die genAppServers sind Störquellen, welche jeweils 2 bis 3 BE-Frames mit maximaler Frame-Länge willkürlich innerhalb des Zyklus senden.

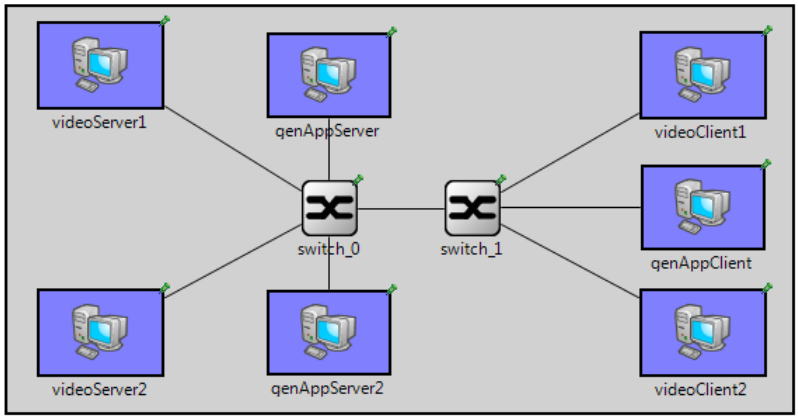

Abbildung 4.4: Topologie des exemplarischen Netzwerks in OMNeT++

4.4 Simulationsergebnisse und Auswertung

Es wurden die Ende-zu-Ende-Verzögerungszeiten für Best-Effort-Frames und für Time-Triggrered-Frames im examplarischen Netzwerk berechnet. Die Berechnung erfolgte nur bei den Empfängern. Die TT-Ende-Zu-Ende-Verzögerungen des ViedeoClient1 werden in der Abbildung 4.5 gezeigt, während die BE-Ende-zu-Ende-Verzögerungen des genAppClients in der Abbildung 4.6 dargestellt werden. Man kann bei dem TT-Traffic eine konstante Verzögerung von 370μs für maximale Frames (1500 Bytes) erkennen. Bei dem Best-Effort-Traffic dagegen sind nicht nur größere Verzögerungen zu erkennen, sondern diese steigen auch mit der Zeit immer weiter. Grund hierfür ist, dass BE-Frames mit niedriegsten Priorität weitergeleitet werden und somit gepuffert werden müssen. Bei Best-Effort sind Verzögerungen von 361 bis 1198μs zu erkennen. Somit ergibt sich ein maximaler BE-Jitter von 837μs. Aus diesen Ergebnisse kann man schließen, dass der TT-Traffic ein deterministisches Zeitverhalten aufweist, der BE-Traffic dagegen nicht. Die hier ermittelte maximale TT-Verzögerung ist vergleichbar mit den Ergebnissen aus den Arbeiten Steinbach u. a. (2010), Steinbach u. a. (2011) und Bartols (2010). In diesen Arbeiten wurde außerdem Verzögerungen für minimale TT-Frames sowie weitere Metriken wie der Jitter, Verzögerungs-Verteilung, etc. ermittelt.

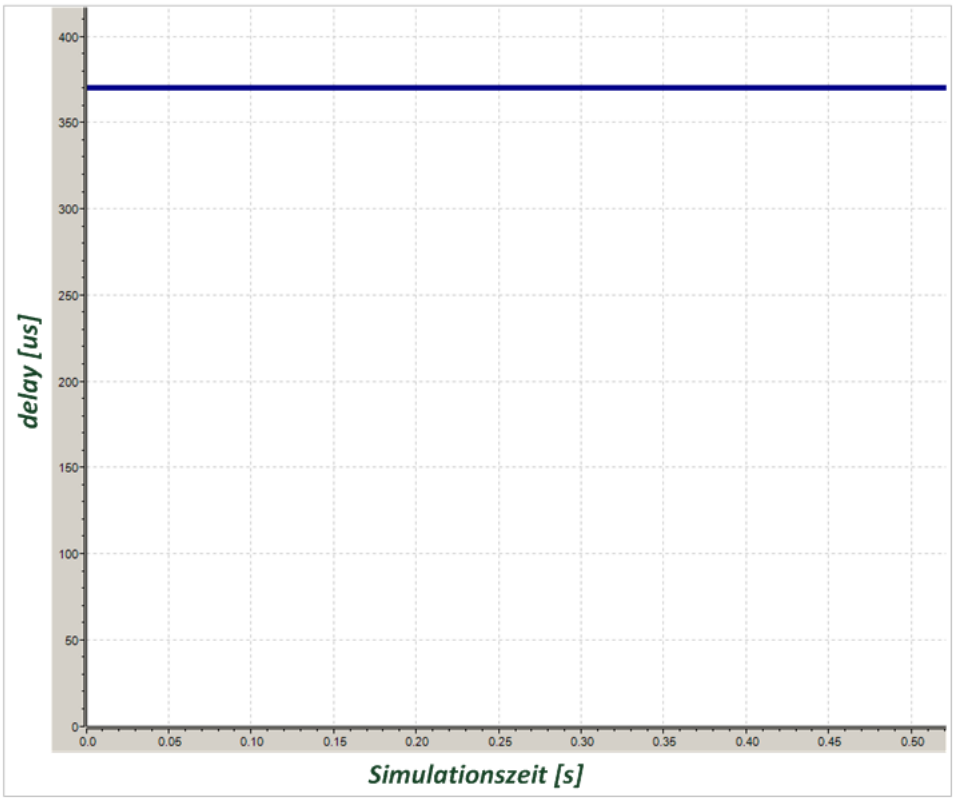

Abbildung 4.5: TT Ende-zu-Ende Verzögerung des videoClient1 (OMNeT++-Plot)

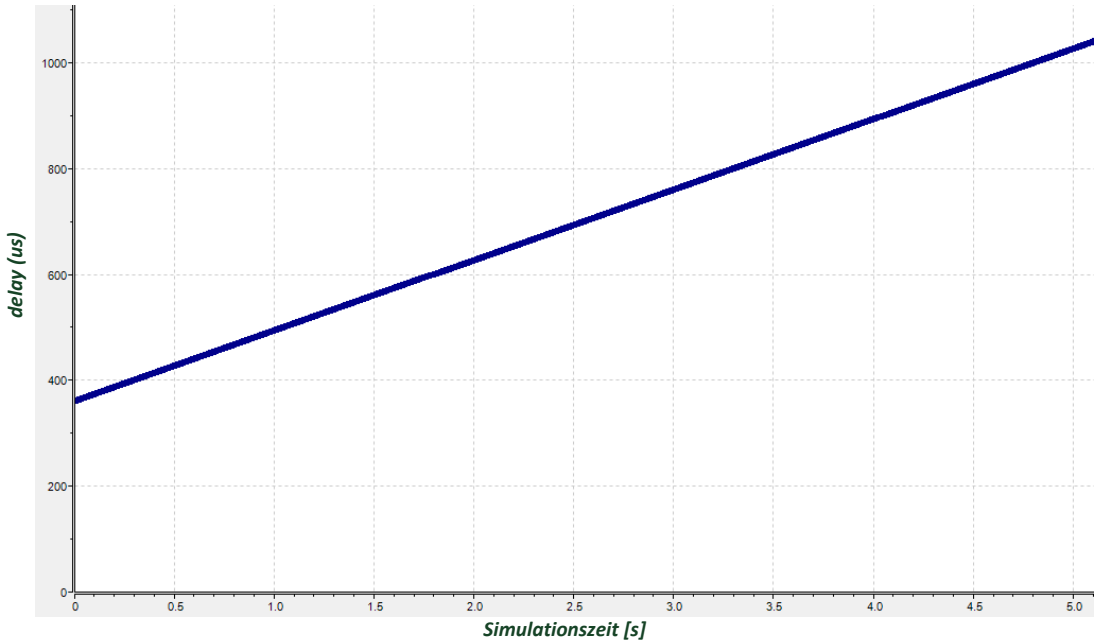

Abbildung 4.6: BE Ende-zu-Ende Verzögerung des genAppClient (OMNeT++-Plot)

5 Zusammenfassung und Ausblick

Dieses Kapitel fast die Arbeit zusammen und gibt einen Ausblick.

5.1 Zusammenfassung

In dieser Arbeit wurde das TTEthernet-Endsystem-Modell für OMNeT++ realisiert. Dafür wurde das im Rahmen des Projekts 1 angefangene TTEthernet-Protokollstack für OMNeT++ weiter ausgebaut und das von TTTech entwickelte TTEthernet-Konfigurationsmodell im Endsystem-Modell integriert. Bei der Erweiterung des Protokollstacks lag der Fokus besonders auf der Implementierung der Callback-Schnittstellen der API, der Aufnahme der Simulationsdaten in der Protokollschicht, der Fehler-Erkennung und Protokollierung (wie z.B. zur falschen Zeit ankommenden Frames), etc. Es wurde außerdem das Verhalten der Protokollschichten untereinander ausführlich beschrieben.

Da die Konfiguration eine sehr große Rolle in allen Entwicklungsphasen von TTEthernet-Netzwerken, insbesondere beim Entwurf, spielt, wurde ein möglichst modellbasierter und automatisierter Konfigurationsprozess entwickelt. Dadurch werden Fehler vermeiden, die bei manuellen Konfigurationen auftreten können. Außerdem werden Änderungen vereinfacht und vor allem wird viel Zeit gespart. Zum Konfigurationsprozess gehört das Designen von Netzwerken mit einem Netzwerkdesigner (GUI-Tool). Es gab leider während dieser Arbeit noch kein TTEthernet-Netzwerkdesigner, sodass eine prototypische Version davon anhand von Visio Automation entwickelt wurde.

Zum Testen des Konfigurationsprozesses und des Protokollstacks wurde ein examplarisches Netzwerk erstellt und eine echte Viedeosteaming-Applikation implementiert. Dadurch konnte man zeigen, dass Applikationen die für echte Systeme bestimmt sind in der Simulation entwickelt bzw. getestet werden können und umgekehrt. Bei der Auswertung der Simulationsdaten des exemplarischen Netzwerks konnte das erwartete Verhalten beim TT-Traffic feststellen werden, welches ein deterministisches Zeitverhalten ist. Beim TT-Traffic konnte man eine fast konstante Verzögerung unabhängig vom aufkommenden BE-Traffic feststellen. Beim BE-Traffic hängt jedoch das Zeitverhalten vom Aufkommen des Traffics ab (je mehr Traffic desto höher die Verzögerungen).

5.2 Ausblick

Es wurde im Nachhinein festgestellt das die Architektur des Endsystem-Modells und die des Switch-Modells sehr unterschiedlich sind. Der Switch hat z. B. der DelegatorIn und -Out Prinzip zum Klassifizieren und Weiterleiten der Nachrichten (vgl. Steinbach u. a., 2011, S. 4-5). Diese Funktion

wird jedoch im Endsystem vom TTEBuffer-Modul realisiert. Außerdem arbeiten der Switch- und der Endsystem-Scheduler völlig unterschiedlich. Besser wäre eine gemeinsame Architektur wenn möglich auch Implementierung für gemeinsame Funktionen wie das Schedling und die Klassiefizierung der Nachrichten zu haben. Diese würde den Code deutlich reduzieren und die Wartung erheblich vereinfachen.

Der im Rahmen dieser Arbeit entwickelte TTEthernet-Netzwerk-Designer ist lediglich ein Prototyp und noch nicht vollständig. Man sollte auf Grund der Relevanz und Komplexität des Themas auf jedem Fall eine Bachelor- oder Master-Arbeit machen. Es wäre dann wichtig dass der TTEtherdesiger mit den EMF-Tools entwickelt wird. So hätte man nicht nur ein Plattform unabhägiges Tool sondern auch das TTEthernet-Objekt-Modell schon in Java vorhanden. Außerdem soll unbedingt ein dynamisches Scheduling basierend z.B. auf die in Steiner (2010) vorgestellten Verfahren, implementiert werden.

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

[Bartols 2010] Bartols, Florian: *Leistungsmessung von Time-Triggered Ethernet Komponenten unter harten Echtzeitbedingungen mithilfe modifizierter Linux-Treiber*. Hamburg, HAW Hamburg, Bachelorthesis, Juli 2010. – Bachelorthesis

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

[Eclipse Foundation a] Eclipse Foundation: *Eclipse Modeling Framework*. – URL `http://www.eclipse.org/modeling/emf/`. – Zugriffsdatum: 2011-02

[Eclipse Foundation b] Eclipse Foundation: *Eclipse Modeling Framework*. – URL `http://www.eclipse.org/modeling/emf/`. – Zugriffsdatum: 2011-01-29

[FIBEX Expert Group 2009] FIBEX Expert Group: Data Model for ECU Network Systems (Field Bus Data Exchange Format) / ASAM. Januar 2009 (3.1). – Specification

[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

[Kempf 2011] Kempf, Fabian: *Simulation von AFDX-Netzwerken basierend auf Rate-Constrained Traffic für Time-Triggered Ethernet in OMNeT++*. Juli 2011. – Bachelorthesis

[Microsoft] Microsoft: *Visio 2010 Automation*. Microsoft. – URL `http://msdn.microsoft.com/en-us/library/ee861526.aspx`. – Zugriffsdatum: 2011-07

[OMNeT++ Community a] OMNeT++ Community: *INET Framework for OMNeT++ 4.0*. – URL `http://inet.omnetpp.org/`

[OMNeT++ Community b] OMNeT++ Community: *OMNeT++ 4.0*. – URL `http://www.omnetpp.org`

[SAE - AS-2D Time Triggered Systems and Architecture Committee 2009] SAE - AS-2D Time Triggered Systems and Architecture Committee: *Time-Triggered Ethernet (AS 6802).* 2009. – URL `http://www.sae.org`. – Zugriffsdatum: 2010-12-11

[Senac und Sevilla] Senac, Andrés ; Sevilla, Diego: *EMF4CPP.* – URL `http://www.catedrasaes.org/trac/wiki/EMF4CPP`. – Zugriffsdatum: 2011-01-29

[Senac u. a. 2010] Senac, Andrés ; Sevilla, Diego ; Martínez, Gregorio: EMF4CPP: a C++ Ecore Implementation. In: Avila-García, Orlando (Hrsg.) ; Cabot, Jordi (Hrsg.) ; noz, Javier M. (Hrsg.) ; Romero, Jose R. (Hrsg.) ; Vallecillo, Antonio (Hrsg.): *Actas del VII Taller sobre Desarrollo de Software Dirigido por Modelos* Bd. 4. San Sebastián : SISTEDES, September 2010, S. 98–106

[Steinbach 2010] Steinbach, Till: *Eine Plattform für die eventbasierte Simulation von time-triggered Ethernet Netzwerken.* Juli 2010. – URL `http://papers.till-steinbach.de/s-peste-10.pdf`. – Bericht

[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

[Steinbach u. a. 2011] Steinbach, Till ; Dieumo Kenfack, Hermand ; Korf, Franz ; Schmidt, Thomas C.: An Extension of the OMNeT++ INET Framework for Simulating Real-time Ethernet with High Accuracy. In: *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, 2011. – to appear

[Steinbach u. a. 2010] Steinbach, Till ; Korf, Franz ; Schmidt, Thomas C.: Comparing Time-Triggered Ethernet with FlexRay: An Evaluation of Competing Approaches to Real-time for In-Vehicle Networks. In: *8th IEEE Intern. Workshop on Factory Communication Systems.* Piscataway, New Jersey : IEEE Press, Mai 2010, S. 199–202

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

[Steiner 2010] Steiner, Wilfried: An Evaluation of SMT-based Schedule Synthesis For Time-Triggered Multi-Hop Networks. In: *31st IEEE Real-Time Systems Symposium*, Dezember 2010

[TTTech Computertechnik AG] TTTech Computertechnik AG: *TTEthernet Network Configuration Documentation.* TTTech Computertechnik AG. – URL `http://www.tttech.com`

[TTTech Computertechnik AG 2008] TTTech Computertechnik AG: *TTEthernet Application Programming Interface.* TTTech Computertechnik AG. Dezember 2008. – URL `http://www.tttech.com`