Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Projekt 1
SS 2010

Hermand Dieumo Kenfack

TTEthernet Protokollstack für OMNeT++

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Hermand Dieumo Kenfack

Thema der Ausarbeitung

TTEthernet Protokollstack für OMNeT++

Betreuer

Prof. Dr. Korf, Franz; Prof. Dr. Schmidt, Thomas

Inhaltsverzeichnis

1 Einleitung

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

1.1 Zielsetzung

Das Gesamtziel des Masterprojekts über die beiden Semester ist, eine Simulationsumgebung für TTEthernet-Netzwerke zu schaffen. Diese soll ein Fundament für weitere Analyse und Evaluierung der TTEthernet-Technologie darstellen. TTEthernet (vgl. Steiner, TTEthernet specification, 2008) ist eine Echtzeit-Variante von Ethernet (IEEE 802.3). Die Echtzeitfähigkeit wird gewährleistet durch die Synchronisation aller Teilnehmer und das Aufstellen eines Zeitplans, welcher das simultane Senden zweier Echtzeitnachrichten ausschließt. Die Realisierung der TTEthernet-Technik in der Simulationsumgebung soll auf der TTEthernet-Spezifikation von TTTech basieren (vgl. Steiner, TTEthernet specification, 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 für die Entwicklung wurde die OMNeT++-Simulationsumgebung (vgl. OMNeT++ Community, 2010) und das INET-Framework (vgl. OMNeT++ Community2, 2010) festgelegt. Grund hierfür ist die Eignung von OMNeT++ für Ereignisbasierte- und Kommunikationsnetzwerke-Simulation. Das INET-Framework bietet bereits ein Modell des Standard Ethernet-Protokolls, worauf das TTEthernet-Protokoll-Modell basieren kann.

1.2 Aufgabenstellung

Der Schwerpunkt des Projekt 1 liegt darin, den TTEthernet-Protokollstack zu entwickeln (Design, Implementierung, rudimentärer Test, etc.). Dabei soll der Fokus auf folgende Punkte gelegt werden:

- Entwurf der Architektur mit Blick auf die Simulation
- Entwurf eines Konzepts für die Konfiguration des TTEthernet-Netzwerks
- Implementierung des Schedulers auf Basis des bestehenden Codes
- Implementierung der TTEthernet-Protokollschicht
- Implementierung von ausgewählten Funktionen der TTEthernet-API unter dem Fokus, dass Last im Rahmen der Simulation generiert wird und die Architektur-Konzepte evaluiert werden können.

2 OMNeT++ und INET-Framework Überblick

In diesem Kapitel wird ein Überblick über die zugrunde liegende Simulationsplattform, welche aus der OMNeT++-Simulationsumgebung und dem INET-Framework besteht, gegeben.

2.1 OMNeT++

OMNeT++ (vgl. OMNeT++ Community, 2010) ist eine C++-basierte, diskrete und ereignisorientierte Simulationsumgebung. Sie dient der Modellierung und Simulation von Kommunikationsnetzen, Multiprozessorsystemen und anderen verteilten oder parallelen Systemen. Sie stellt außerdem eine Bibliothek und ein Framework zur Verfügung, mit denen weitere Simulationsmodelle und Frameworks entwickelt werden können. Ein Beispiel hierfür ist das INET-Framework (siehe Abschnitt 2.2). OMNeT++ ist für akademische und nicht kommerzielle Zwecke frei nutzbar.

Ein OMNeT++-Simulationsmodell besteht aus hierarchisch geschachtelten Modulen. Man unterscheidet zwischen einfachen (simple) und zusammengesetzten (compound) Modulen. Zusammengesetzte Module bestehen aus mehreren einfachen oder zusammengesetzten Modulen. Das Verhalten des Modells wird in C++ in den einfachen Modulen implementiert. Diese werden durch Parameter konfiguriert und kommunizieren mittels Nachrichten (messages), welche über Ausgabe-Gates (output gate) gesendet und über Eingabe-Gates (input gate) empfangen werden. Die beiden Letzteren werden über Konnektoren (connection) miteinander verbunden. Der modulare und objektorientierte Ansatz von OMNeT++ ermöglicht es vorhandene Module wiederzuverwenden bzw. neue Module daraus abzuleiten.

Die Struktur eines Modells (Module und deren Verbindungen) wird mit der OMNeT++ Topologie-Beschreibungssprache NED (Network Description) definiert. OMNeT++ beinhaltet hierfür einen grafischen Editor. Die Parametrisierung der Module erfolgt entweder in der NED- (Modell-Topologie) oder in einer INI-Datei (Konfigurationsdatei). Es kann z. B. darin festgelegt werden, in welchen Abständen ein Traffic-Generator Nachrichten versenden soll.

2.2 INET-Framework

Das INET-Framework (vgl. OMNeT++ Community2, 2010) ist ein Simulations-Framework für die OMNeT++ -Simulationsumgebung, welches verschiedene Internet-Protokoll- und Applikationsmodelle enthält. Beispiele hierfür sind Ethernet, UDP, TCP, IP, IPv6, , etc. Das INET-Framework baut auf der OMNeT++-Simulationsumgebung auf und hat dieselben Eigenschaften (Objekt orientiert, modular, Wiederverwendbarkeit, etc.) und Konzepte (einfache und zusammengesetzte Module, Nachrichten, Gates, Konnektoren, etc.). Da TTEthernet auf Ethernet basiert, muss der TTEthernet-Protokollstack nicht komplett neu entwickelt werden. Viele Module des vorhandenen Ethernet-Stack-Modells (Kabel-, MAC-, LLC- und Applikations-Modul) können übernommen und nach bedarf auf TTEthernet erweitert werden.

3 TTEthernet Netzwerk Beschreibung

In diesem Kapitel wird TTEthernet allgemein, seine Bestandteile anhand eines Objektmodells und seine Traffic-Klassen beschrieben. Weiterhin wird das im Rahmen dieser Arbeit entworfene Konfigurations-Objektmodell für die Simulation dargelegt.

3.1 Allgemeines

TTEthernet ist eine Echtzeitvariante des Full-Duplex-Switched-Ethernets (Steiner, TTEthernet specification, 2008). Die Echtzeitfähigkeit beruht auf einem Verfahren, welches auf Zeitschlitzen aufbaut (TDMA -Time Division Multiple Access-). Dabei wird ein Zeitplan vordefiniert, nach dem alle Teilnehmer operieren müssen. Hierfür wird in jedem Zyklus ein/mehrere Sende-Zeitintervall(e) für jede (harte) Echtzeit-Nachricht festgelegt. Erwähnenswert ist, dass diese Zeitintervalle sich nicht überschneiden dürfen und gemeinsame Ressourcen (Switch-Verarbeitungseinheit) währenddessen für das Versenden bzw. die Weiterleitung der jeweiligen Nachricht frei sein müssen. Dies wird sichergestellt durch den Einsatz von speziellen Tools, womit die Zeitplanung erstellt und überprüft wird und Schedulern, welche das Versenden bzw. die Weiterleitung der Nachrichten Triggern. Bei einem TDMA-Verfahren ist es wichtig, dass alle Teilnehmer miteinander synchronisiert sind. Daher werden diese im TTEthernet über ein ausfallsicheres Synchronisationsprotokoll mit einer globalen Zeit synchronisiert.

3.2 Traffic-Klassen

TTEthernet ermöglicht nicht nur die Übertragung von Daten mit harten Echtzeitanforderungen, sondern auch mit weniger harten und ohne harten Echtzeitanforderungen. Dies wird anhand verschiedener Traffic-Klassen realisiert (vgl. Steiner, Bauer, Hall, Paulitsch, & Varadarajan, 2009).

- TT-Traffic (Time-Triggered Traffic): Dient zur Übertragung von Nachrichten mit harten Echtzeitanforderungen (TT-Nachrichten). Diese unterliegen wie bereits gesagt einer festen Planung und haben Vorrang vor allen anderen Nachrichten.
- RC-Traffic (Rate-Constrained Traffic): Dient zur Übertragung von Daten mit weniger harten Echtzeitanforderungen. Hierbei wird sichergestellt, dass für jede RC-Nachricht ausreichend Bandbreite zur Verfügung steht, indem sogenannte BAGs (Bandwidth Allocation Gap) festgelegt werden. Diese definieren die minimale Zeit zwischen zwei nachfolgende Nachrichten sowie die maximale Länge der Nachricht. Somit wird sichergestellt, dass Nachrichten mit einem begrenzten Jitter übermittelt werden. Dieses Protokoll entspricht dem AFDX (Avionics Full DupleX Switched Ethernet)- Protokoll (vgl. AIM, 2008) und wird mit der zweiten Priorität behandelt.
- BE-Traffic (Best-Effort Traffic): Dient zur Übertragung von Daten ohne harte Echtzeitanforderungen. Dies entspricht dem Standard-Ethernet-Traffic und wird mit niedrigster Priorität weitergeleitet. Eine zuverlässige Übertragung wird nicht gewährleistet.

Der Oberbegriff für TT- und RC-Traffic ist CT (Critical Traffic).

3.3 TTEthernet-Netzwerk Objektmodell

Ein TTEthernet-Netzwerk besteht aus mehreren Devices (Geräten), Links (physikalischen Verbindungen), Virtual-Links (logischen Verbindungen) und einer Netzwerkkonfiguration (siehe Abbildung 3-1).

Ein Device kann ein End-System oder ein Switch sein. Switches stellen Vermittlungskomponenten dar und verbinden End-Systeme miteinander. Ein Device hat:

- Mehrere Ports, welche seine externen Schnittstellen darstellen und als Verbindungen zu anderen Devices dienen.
- Eine Device-Spezifikation, welche seine Konfiguration beinhaltet (siehe Abschnitt 3.4).
- Ein Protokoll-Stack (siehe Kapitel 4), welcher aus mehreren miteinander in Bezug stehenden Protokollen besteht, die die für die Kommunikation mit anderen Devices notwendige Funktionalitäten bereitstellen.

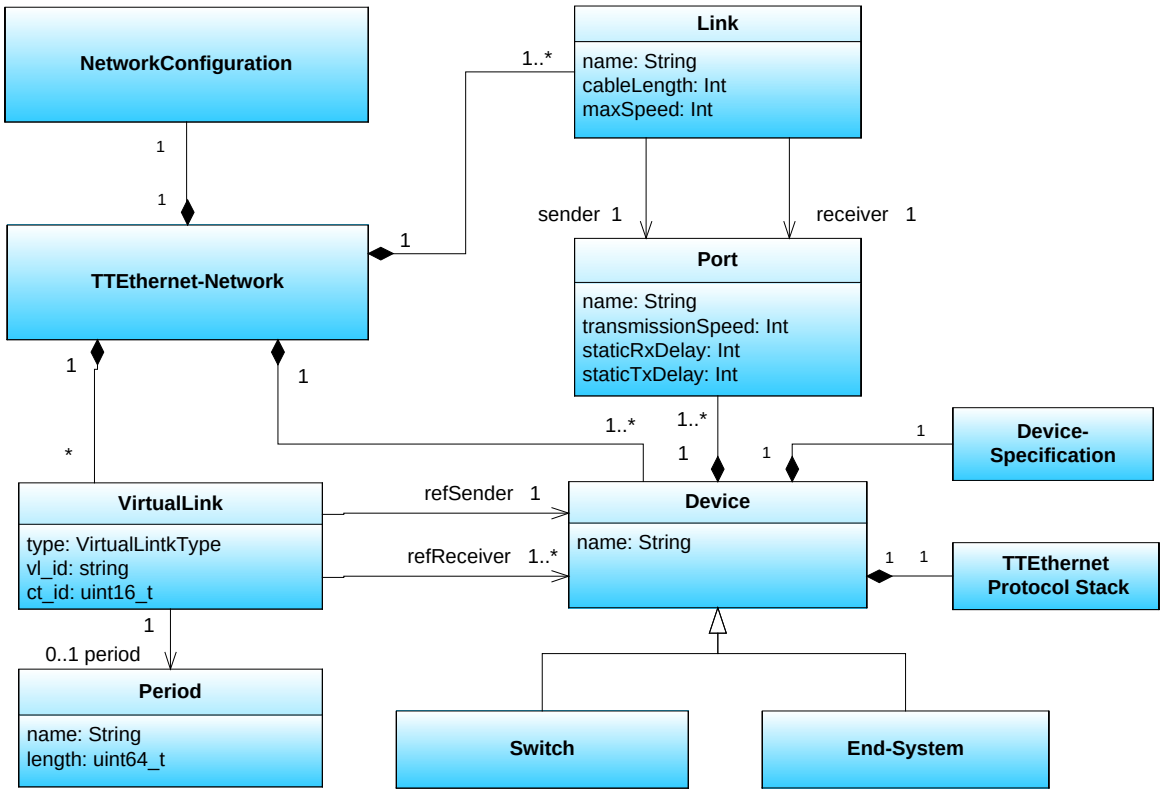

Abbildung 3-1: TTEthernet-Netzwerk Objektmodell (vgl. TTTech2, 2009)

Die Links stellen die physikalischen Verbindungen zwischen den Devices dar (Ports miteinander verbinden), die Virtuallinks dahingegen die logischen Verbindungen. Eine logische Verbindung ist ein Pfad zwischen einem Sender und einem oder mehreren Empfängern mit einer eindeutigen Kennung, der CT-ID (Critical Traffic Identification). Diese wird durch Time-Triggered- oder Rate-Constrained-Traffic über die physikalische Schicht realisiert (vgl. AIM, 2008).

Die Traffic-Klasse eines Virtual-Links wird mit der Eigenschaft VirtualLinkType (TT/RC-Virtual-Link) festgelegt. Seine Zeitanforderung wird mit der Periode definiert. Genauer gesagt mit der Eigenschaft „length“, welche im Fall von TT-Traffics die Sendezeit der Nachricht und bei RC-Traffics die BAGs (bandwidth allocation gap) beschreibt.

In der Netzwerk-Konfiguration werden netzwerkübergreifende Konfigurationen wie die Zyklusdauer, alle Virtual-Links und Perioden definiert. Außerdem beinhaltet sie eine Referenz jeder Device-Spezifikation.

Wie aus Abbildung 3-1 zu sehen ist, wurden die Synchronisierungskomponenten nicht modelliert, da die Devices in der Simulationsumgebung aufgrund der gleichen Zeitbasis (die Simulationszeit) als synchronisiert angenommen werden. Für die Modellierung und die Beschreibung der Synchronisation wird auf die TTEthernet-Netzwerkkonfiguration-Spezifikationen von TTTech (TTTech2, 2009) und die TTEthernet-Spezifikation (Steiner, TTEthernet specification, 2008) verwiesen.

3.4 TTEthernet Netzwerkkonfigurations-Objektmodell

Das Netzwerkkonfigurations-Objektmodell für die Simulation wurde in Anlehnung an den Netzwerkspezifikationsdateien (XSD-Datein) von TTTech realisiert (vgl. TTTech2, 2009). Die Konfiguration eines TTEthernet-Netzwerks wird in der Regel in XML-Dateien (systemspecification.xml, devicespecification.xml, etc.) durchführt und dann gegen die entsprechende Spezifikation (XSD-Datei) validiert. Die XML-Dateien werden schließlich in ein von den Devices verständliches Format (.c/.h-Dateien) über spezielle Tools umgewandelt. Diese werden für die Simulation in die Objekte überführen. Somit können Konfigurationen, die für echte Systeme vorgesehen sind, in der Simulation verwendet werden.

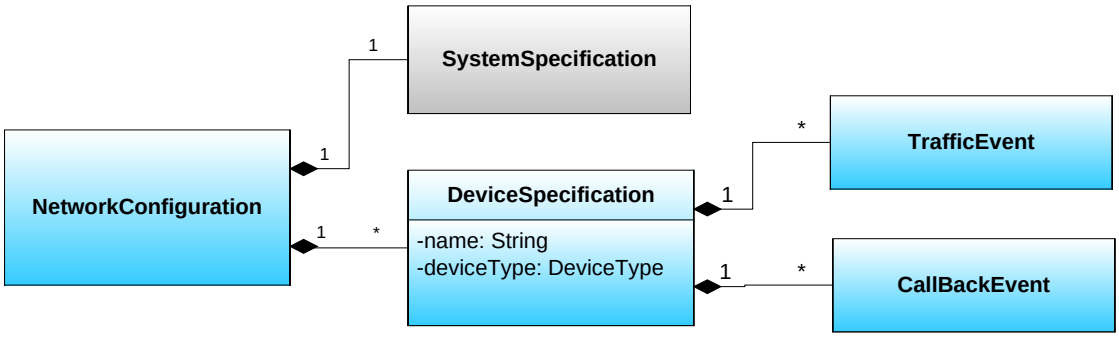

Abbildung 3-2: TTEthernet Netzwerkkonfigurations-Objektmodell

Eine TTEthernet-Netzwerkkonfiguration besteht aus einer Systemspezifikation und mehreren Device-Spezifikationen. Wobei für jedes Device eine Device-Spezifikation definiert werden muss. Die Systemspezifikation beschreibt die physikalische Topologie des Netzwerks. Sie definiert alle De-

vices (End-Systeme und Switches) und verbindet diese miteinander über Ports. Diese sind wiederum miteinander mit Links verbunden (siehe Abschnitt 3.3). Die Device-Spezifikation beschreibt die Konfiguration eines Devices. Sie definiert alle Traffic- und Callback-Events eines Devices.

Ein Traffic-Event (siehe Abbildung 3-4) definiert die zeitlichen Anforderungen („enventTime“, „windowStart-windowEnd“, etc.) und die Weiterleitungsinformationen („priority“, „ct_id“, etc.) einer Nachricht. Die Eigenschaft „eventTime“ eines Traffic-Events legt den Zeitpunkt fest, an dem die Nachricht gesendet oder erwartet wird (z. B. wird wie aus Abbildung 3-3-(a) zu sehen ist die Nachricht mit der CT-ID 1 jede 1000 µs gesendet). [windowStart- windowEnd] ist das Zeitintervall, in dem eine CT-Nachricht gesendet/empfangen werden muss (um gültig zu sein). Dieser stellt beim BE-Traffic ein Zeit-Fenster dar, indem BE-Nachrichten gesendet (siehe Abbildung 3-3-(c)) bzw. weitergeleitet (siehe Abbildung 3-3-(d)) werden. Ein Traffic-Event unterteilt sich in die Subtypen Outgoing- und Incoming-Traffic-Event. Mit Outgoing-Traffic-Events können CT-Nachrichten, die gesendet werden geplant und mit Incoming-Traffic-Events, die die erwartet werden. Bei der Modellierung von Incoming-Traffic-Events kann auch eine Liste von Outgoing-Events angelegt werden. Somit kann Multicast vorkonfiguriert werden. Weiterhin wird für alle Traffic-Klassen eigener In/Outgoing-Typ definiert. Erwähnenswert ist, dass TTIncoming-Traffic-Event eine Eigenschaft hat, das permanencePit (permanence point in time). Diese legt den Zeitpunkt fest, an dem die Nachricht bei allen Empfängern angekommen sein muss. Der RC-Traffic wurden hier modelliert, um ein vollständiges Traffic-Event-Objektmodell zu erhalten. Dieser wird jedoch im Rahmen dieser Arbeit nicht realisiert.

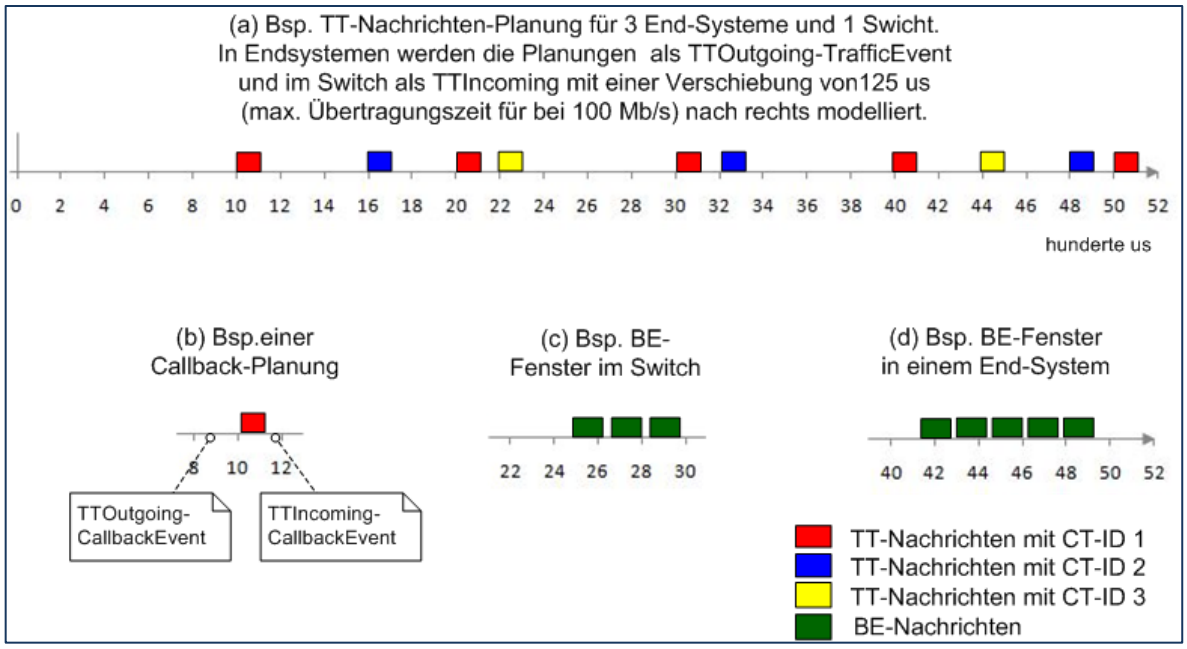

Abbildung 3-3: Beispiel einer TTEthernet-Nachrichten- und Callback-Planung

Traffic-Callback-Event (Abbildung 3-5): Hiermit kann ein Zeitpunkt festgelegt werden, in dem eine Funktion zum Senden/Empfangen einer Nachricht aufgerufen werden soll. Man kann z. B.

TToutgoing-CallBackEvents kurz vor (je nach der Dauer der Durchführung der Sende-Funktion) und TTIncoming kurz nach (je nach der Dauer der Übertragung der Nachricht) den jeweiligen In/Outgoing-Traffic-Events automatisch planen (siehe Abbildung 3-3-(c)). Somit können die Applikationen mit der Nachrichten-Plannung (dem Scheduling) synchronisiert werden.

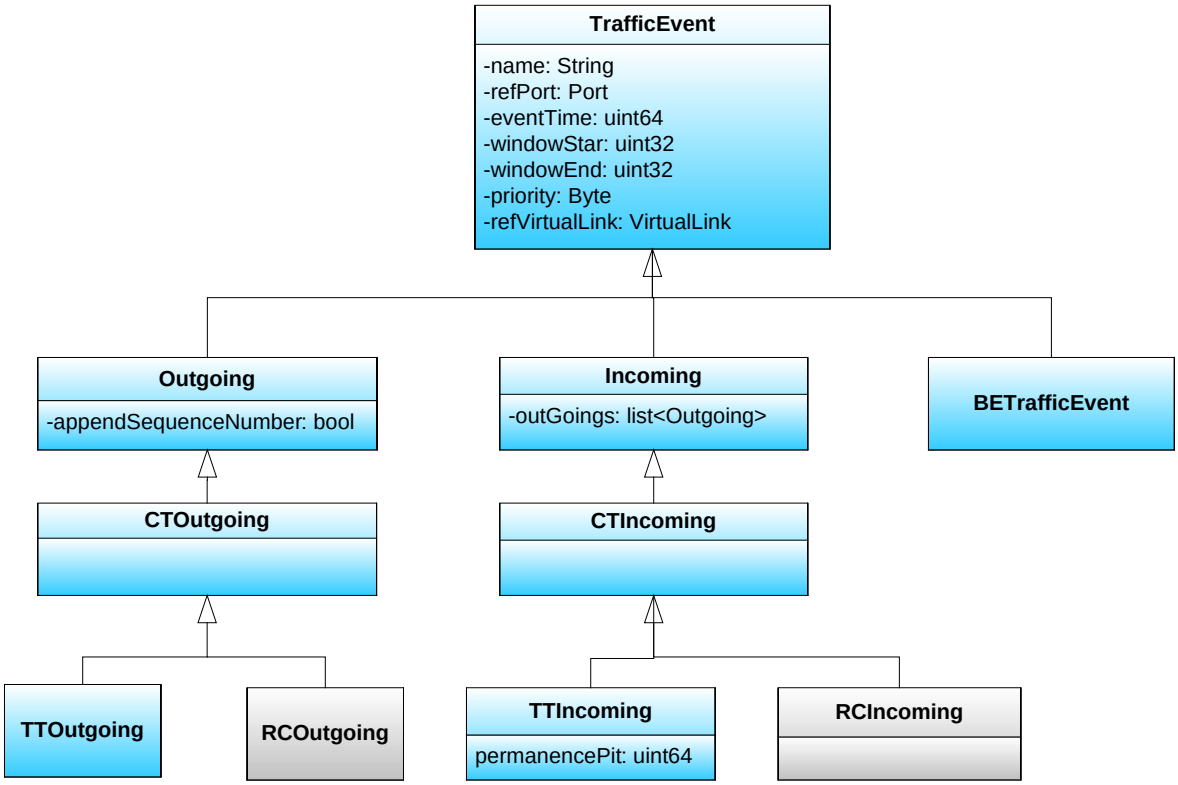

Abbildung 3-4: Traffic-Event-Objektmodell

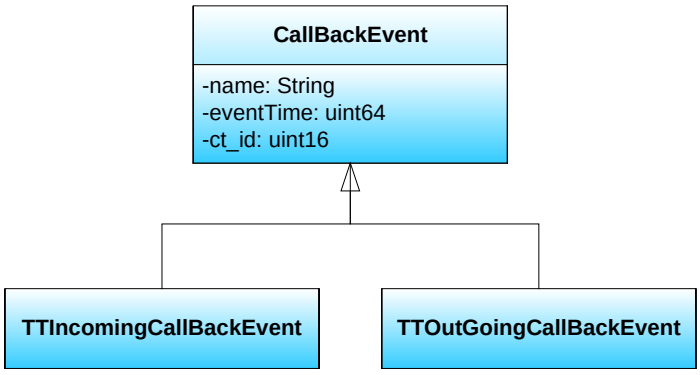

Abbildung 3-5: Callback-Event Objektmodell

4 TTEthernet-Netzwerk Protokollstack

Der TTEthernet Protokoll-Stack entspricht dem Standard-Ethernet-Protokoll-Stack mit Erweiterungen auf die TTEthernet-Technik. Der TTEthernet-Protokollstack besteht aus drei Schichten, die physikalische, die Data-Link- und die Applikation-Schicht. Wobei die beiden Letzen sich in weitere Subschichten unterteilen (siehe Abbildung 4-1). Dieses Kapitel gibt eine grobe Beschreibung der Schichten und deren Subschichten sowie die an diesen vorgenommene Änderungen bzw. Erweiterungen, um TTEthernet in der Simulation zu realisieren.

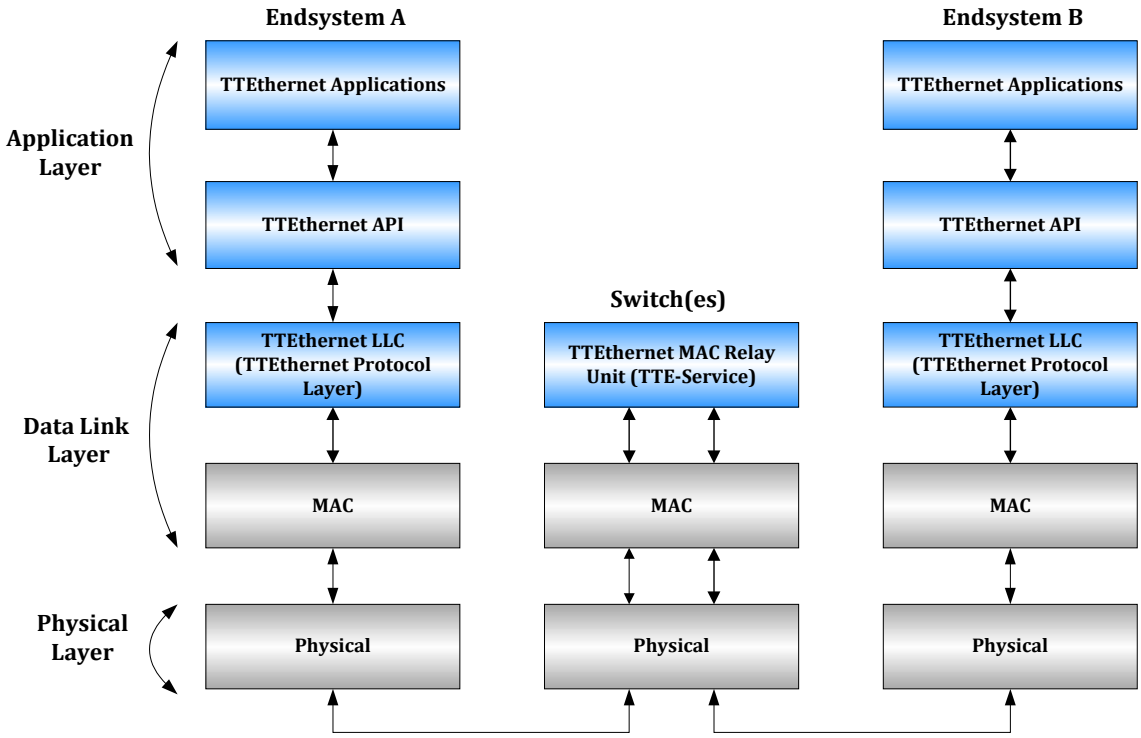

Abbildung 4-1: TTEthernet-Netzwerk Protokollstack

4.1 Applikationsschicht

In der INET-Ethernet-Implementierung, ist die Applikationsschicht unmittelbar mit der LLC(Logical Link Control)-Subschicht verbunden. Die LLC-Subschicht realisiert den Datenaustausch zwischen zwei Endsystemen. Dafür stellt sie eine gut definierte Dienstschnittstelle für die Applikationsschicht bereits (siehe Abschnitt 4.2.2). Die Dienste der Applikationsschicht werden über sogenannte DSAP (Destination Service Access Point) und die SSAP (Source Service Access Point) angeboten. Die Applikationsschicht ist jedoch bei der TTEthernet Implementierung mit der LLC-Subschicht über die TTEthernet-API (von TTTech definiert) verbunden. Sie bietet ihre Dienste über die CT-IDs für CT-Applikationen und Channel-IDs für BE-Applikationen, die ursprünglichen DSAP und SSAP sind also für TTEthernet-Applikationen irrelevant.

Das Senden und Empfangen von Nachrichten über die TTEthernet-API ist pufferbasiert. 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 schreibt, während dieser von der LLC-Subschicht gelesen wird und umgekehrt. Durch den (in)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 dieser optimiert. Die Sende- und Empfangsfunktionen der TTEthernet-API wurden im Rahmen diese Arbeit implementiert. Weitere Funktionen sind für zukünftige Arbeiten geplant. Für eine detaillierte Beschreibung der TTEthernet-API siehe (TTTech3, 2010).

Die Motivation die TTEthernet-API beim Designen des Protokoll-Stacks zu berücksichtigen, ist das Ermöglichen der Entwicklung bzw. des Testens von echten Applikationen in der Simulationsumgebung vor dem Einsatz im echten System.

4.2 Sicherungsschicht

Die Sicherungsschicht (Data Link Layer) garantiert eine sichere Übertragung zwischen zwei benachbarten Stationen. Im LAN wird sie in zwei weitere Subschichten unterteil, die MAC- und die LLC-Subschicht, welche in diesem Abschnitt beschieben werden.

4.2.1 Media Access Control

Zu den Aufgaben der MAC-Subschicht gehört, die physikalische Adressierung (MAC Adressierung), die Aufteilung von Bitströmen aus der Bitübertragungsschicht in Frames, die Fehlerkontrolle (z. B. durch die Berechnung der CRC-Summe (Cyclic Redundancy Check)) und die Regelung des Zugriffs auf das Übertragungsmedium (vgl.Held, 2003, S. 52). Ethernet benutzt für den Letzteren den CSMA/CD-Mechanismus (Carrier Sense Multiple Access/Collision Detection), wenn sich mehrere Endsysteme in einer Kollisionsdomäne befinden. Dies wird jedoch im Switched-Ethernet nicht benötigt, da sich jedes Endsystem in einer eigenen Kollisionsdomäne befindet. Aus diesem Grund wurde das INET-MAC-Modell ohne CSMA/CD für den TTEthernet-Protokollstack übernommen.

4.2.2 Logical Link Control

Die LLC-Subschicht realisiert den Datenaustausch zwischen zwei Endsystemen. Dafür stellt sie eine gut definierte Dienstschnittstelle für die darüber liegende Schicht bereit. Zu diesen Diensten gehört z. B. das Multiplexen und Demultiplexen von Protokollen. Sie kann außerdem nach Bedarf eine Fluss-Kontrolle, Bestätigung und Fehlerbenachrichtigung bieten. Sie unterstützt sowohl einen verbindungsorientierten als auch einen verbindungslosen Dienst. Der Letzterer ist für Echtzeitdatenverkehr besser geeignet und entspricht auch der INET-Ethernet-LLC-Implementierung.

Das LLC-Protokoll verwendet zur eindeutigen Darstellung der einzelnen Felder die PDUs (Protocol Data Units). Die PDUs bestehen aus einer Ziel- und Quell-Adresse, einem Kontroll-Feld und den Daten der höheren Protokolle (Abbildung 4-2). Der Zugang zu den höheren Schichten (in diesem Fall die Applikationsschicht) erfolgt über den DSAP (Destination Service Access Point) und den SSAP (Source Service Access Point) (vgl. Hein, 1998, S. 280).

Diese Schicht wurde in der Simulationsumgebung auf die für TTEthernet-Netzwerke benötigte Funktionalitäten erweitert. Das heißt, dass der TTEthernet Protokoll Layer Teil dieser Schicht ist (siehe Abbildung 4-1 und nächster Abschnitt).

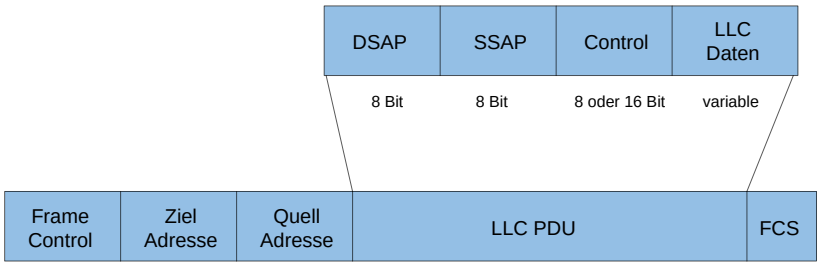

Abbildung 4-2: Format der LLC PDUs (vgl. Hein, 1998, S. 280)

4.2.3 TTEthernet Logical Link Control

Die TTEthernet-LLC-Subschicht erweitert die Standard Ethernet-LLC-Subschicht des INET-Frameworks auf den TTEthernet-Dienst (siehe Abbildung 4-3). Sie ist mit einem Scheduler verbunden und reagiert auf seine Kommandos. Diese sind nachrichtenbasiert und legen fest, welche Nachrichten von der TTEthernet-LLC-Subschicht zum Versenden ausgelesen oder welche Callback-Funktion aufgerufen werden sollen. Ist zum Beispiel der Zeitpunkt zum Versenden einer TT-Nachricht eingetreten, sendet der Scheduler ein TTOutgoing-Traffic-Event-Kommando mit der entsprechenden CT-ID an die TTEthernet-LLC-Subschicht. Diese ließt daraufhin die Nachricht aus dem TT_TX_Puffer, verpackt diese in einen INET-Ethernet-Frame und leitet sie schließlich an die MAC-Subschicht weiter. Darüber hinaus ist die TTEthernet-LLC-Subschicht mit der MAC-Subschicht verbunden und nimmt ihre Dienste entgegen. Die Sendefunktion der TTEthernet-LLC-Subschicht wird durch Abbildung 4-4 und seine Empfangsfunktion durch Abbildung 4-5 illustriert.

Der Scheduler erstellt seine Zeitplanung anhand der in der Devicespezifikation vorgenommenen Konfiguration des Devices (siehe Abschnitt 3.4). Der in dieser Arbeit verwendete Scheduler basiert auf dem von Till Steinbach entwickelten in Rahmen seiner Masterprojekt2-Arbeit (vgl. Steinbach, 2009, S. 9). Dieser wurde so angepasst, dass er die hier modellierte Devicespezifikation verwenden und Traffic- bzw. Callback-Event-Nachrichten an die TTEthernet-LLC-Subschicht senden kann.

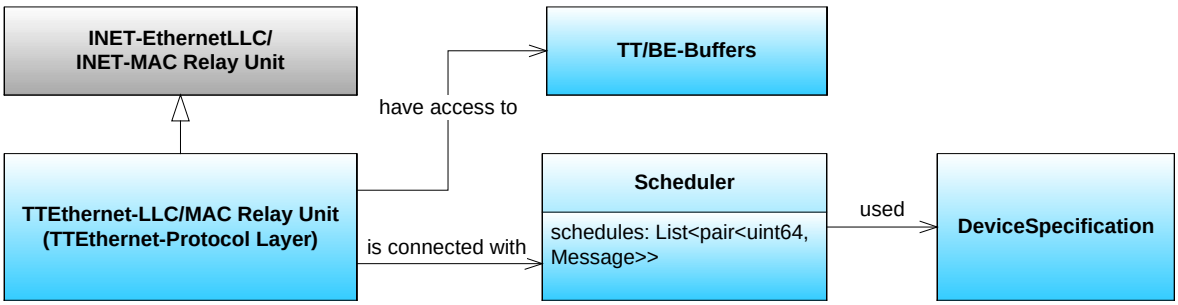

Abbildung 4-3: TTE Protokoll-Schicht/Dienst Objektmodell

4.3 Physikalische Schicht

Die physikalische Schicht (Physical Layer), auch Bitübertragungsschicht, definiert die mechanischen und elektrischen Eigenschaften sowie die Funktionen und Abläufe, bei der Bitübertragung. Die physikalischen Verbindungen werden in OMNeT++ durch sogenannte „channels“ modelliert. Ein Channel kann auf einem DatarateChannel erweitert werden, sodass wichtige Eigenschaften eines physikalischen Mediums gesetzt werden können. Beispiele hiefür sind die Datenrate zur Modellierung der Übertragungsdauer eines Pakets und die Bit-Fehlerrate bzw. Paket-Fehlerrate zur Modellierung von Übertragungsfehlern.

4.4 TTEthernet MAC Relay Unit

Die TTEthernet-MAC-Relay-Unit stellt die Hauptkomponente eines TTEthernet-Switches in der Simulation dar. Sie klassifiziert den eingehenden Traffic und leitet diesen erst nach dem Zuschlagen des Schedulers weiter. Weiterhin hat sie zur Aufgabe zu überprüfen, ob die Ankunftszeit einer TT-Nachricht in Ordnung ist und die vorgesehene RC-Bandbreite eingehalten wird. Ein Prototyp des TTEthernet-Switches wurde von Till Steinbach entwickelt (vgl. Steinbach, 2009, S. 8).

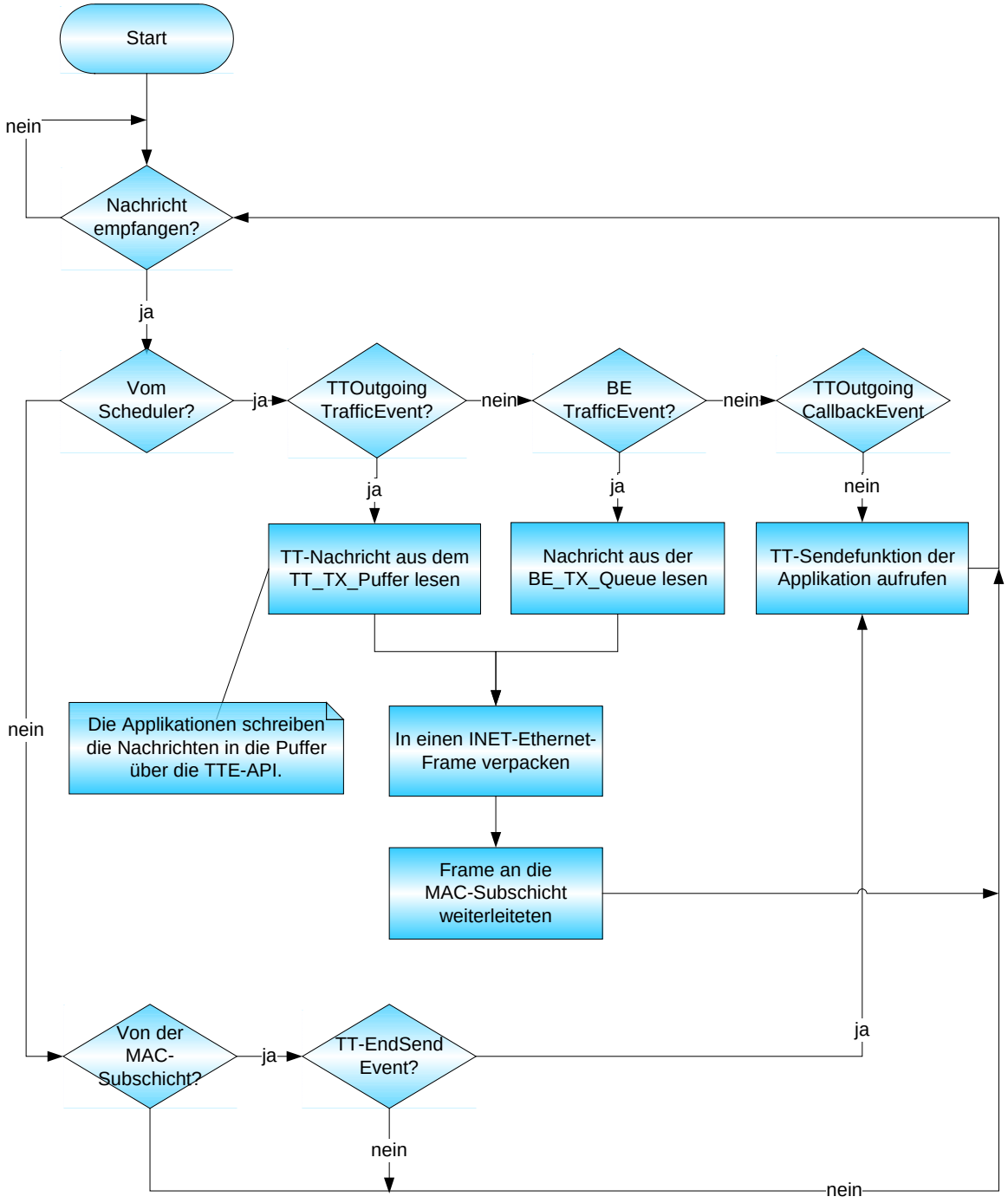

Abbildung 4-4: TTEthernet LLC Sendefunktion

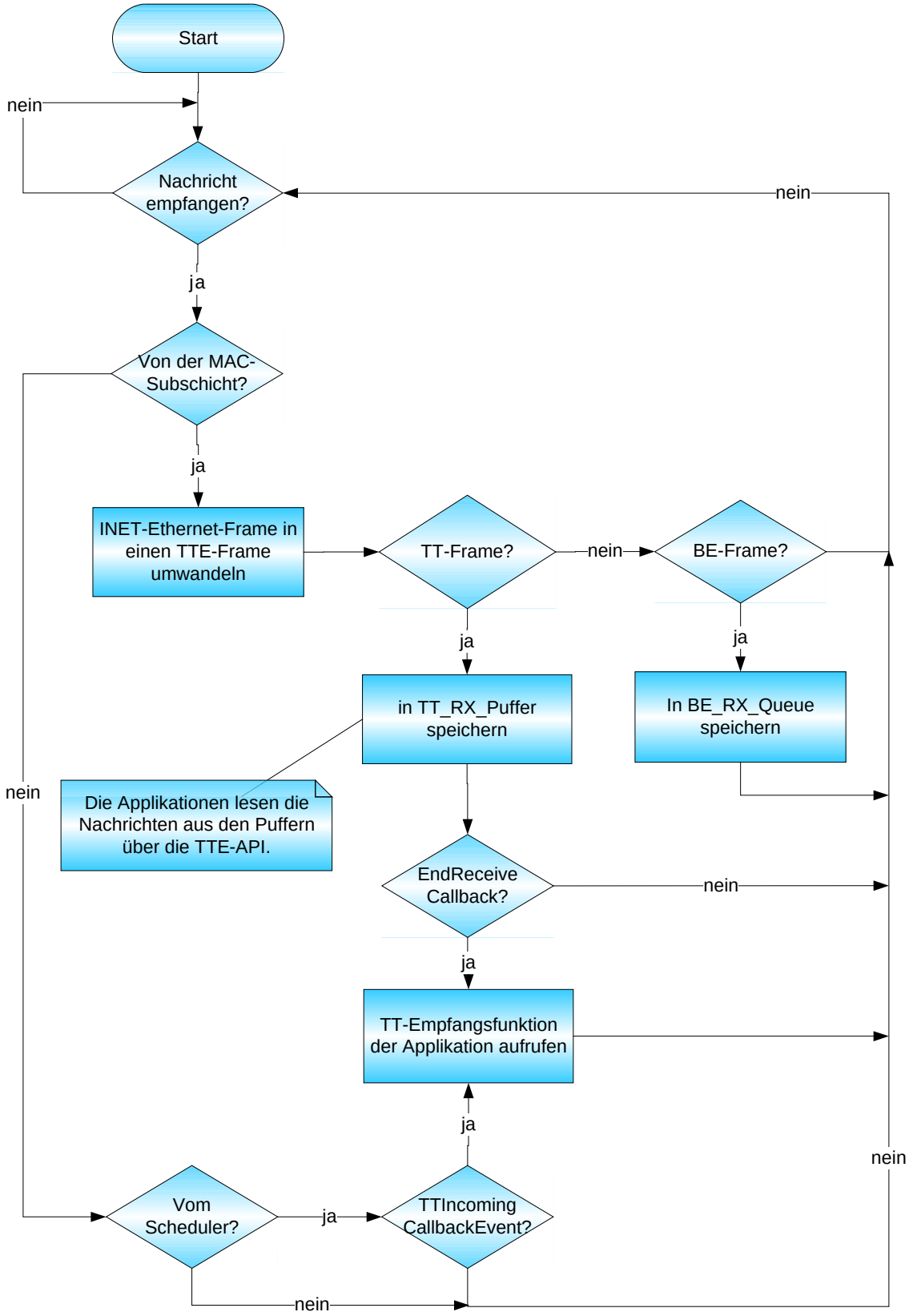

Abbildung 4-5: TTEthernet LLC Empfangsfunktion

5 Beispiel-Netzwerk und Test-Applikationen

In diesem Kapitel wird das aufgestellte Kleinnetzwerk zum rudimentären Testen des TTEthernet-Protokollstacks und der Implementierung des Konfigurationsmodells, die Testapplikationen und die Simulationsergebnisse dargelegt. Das Ziel des Tests ist sicherzustellen, dass TT- und BE-Nachrichen über die TTEthernet-API gesendet und empfangen werden können. Weiterhin sollen auch die Nachrichten-Verlustraten und die „end-to-end delays“ in betrag gezogen werden.

5.1 Physikalische Topologie

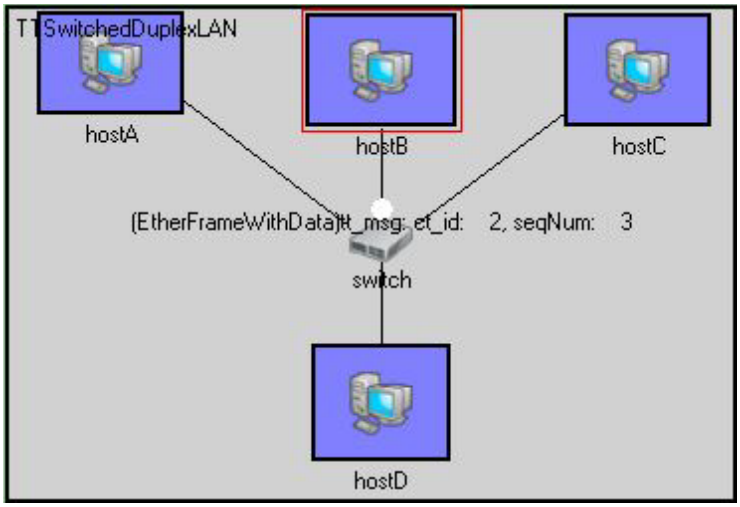

Abbildung 5-1: TTEthernet Bespiel-Netzwerk physikalische Topologie

Das Test-Netzwerk ist ein Full-Duplex-Switched-Ethernet-Netzwerk mit einem 100 Mb/s Kabelmodell. Es besteht aus vier End-Systemen (HostA bis HostD) und einem Switch (siehe Abbildung 5-1). Auf jedem End-System läuft eine Client- oder Server-Applikation, die über die TTEthernet-API TT/BE-Nachrichten sendet bzw. empfängt. Bei diesem Aufbau senden die Clients von HostA bis HostC während der Server von HostD empfängt.

Der Nachrichteninhalt einer Applikation (Nutzlast) besteht aus:

- Einem Nachrichten-Namen (50 Bytes), welcher von OMNeT++ für die Animation der Simulation verwendet wird.
- Einer Sequenznummer (4 Bytes), die von einem Client beim Versenden einer Nachricht fortlaufend erhöht wird. So kann der Server überprüfen, ob eine Nachricht verloren gegangen ist.
- Eine Sende- und Empfangszeit (16 Bytes) der Nachricht. Der Client setzt beim Versenden der Nachricht die Sendezeit und der Server beim Empfangen die Empfangszeit und rechnet anschließen die Verzögerung (delay) der Nachricht aus.
- Füllbytes (1500 Bytes maximale Nutzlast - 70 Bytes echte Nutzlast)

5.2 Logische Verbindungen und Nachrichten Planung

Die Nachrichten- und Callback-Planung des Test-Netzwerks entsprechen denen aus Abbildung 3-3.

- Die Nachrichten werden im Zyklus von 5000 µs geplant.
- Der HostA-Client sendet TT-Nachrichten mit der CT-ID 1 und mit einer Periode von 1000 µs, der HostB-Client mit CT-ID 2 und einer Periode von 1600 µs und der HostC-Client mit CT-ID 3 und einer Periode von 2200 µs.
- Die Clients von HostB und C senden neben TT- auch BE-Nachrichten
- Der HostD-Server empfängt alle TT- und BE-Nachrichten.
- Die Nachrichten mit den CT-ID 1 und 2 werden über automatisch geplante Callback-Funktionen gesendet und empfangen während die mit der CT-ID 3 in zufälligen Intervallen gesendet (1500 µs - 2000 µs) und empfangen (500 µs - 1000 µs) werden.
- BE-Nachrichten werden in zufälligen Intervallen von 500 µs bis 1000 µs gesendet und von 100 µs bis 200 µs empfangen.

5.3 Simulationsergebnis und Interpretation

Die Simulationsdurchführung dauerte 10000 µs (zwei Zyklen). Das Ergebnis wird durch Abbildung 5-2 dargestellt. Dabei konnten folgendes festgestellt werden.

- Die TT-Nachrichten mit den CT-ID 1 und CT-ID 2 wurden alle mit einer konstanten Verzögerung von 270 µs empfangen. Dieses erfolgte aufgrund der Synchronisierung der Applikationen mit dem Scheduling über Callback-Funktionen. Die 270 µs konstante Verzögerung entsteht aus 20 µs + 125 µs + 125 µs. Wobei 20 µs eine grobe Schätzung der Zeit ist, die von der Callback-Funktion zur Vorbereitung und zum Schreiben der Nachricht benötigt wird. Die 125 µs ist die maximale Übertragungszeit eines Frames bei 100Mb/s (vgl. Schudel, 2010).
- Bei den TT-Nachrichten mit CT-ID 3 kann man größere (> 1000 µs) Verzögerungen feststellen, da die Sende/Empfangs-Applikationen mit dem Scheduling nicht synchronisiert wurden. Nachrichten können aus demselben Grund überschrieben werden, wenn der Empfänger diese nicht schnell genug lesen kann (wie z. B. die Nachricht mit der Sequenznummer 3). So kann man konkludieren, dass für ein verlustfreies Empfangen und minimale Verzögerungen von TT-Nachrichten eine Synchronisierung der TT-Applikationen mit dem Scheduling unerlässlich ist.
- Der HostB-Client hat 12 BE-Nachrichten gesendet und 11 davon sind angekommen. Der HostC-Client hat 12 BE-Nachrichten gesendet und 9 davon sind angekommen. Man kann aber nicht daraus schließen, dass die nicht angekommenen Nachrichten verloren gegangen sind, da diese noch hätten ankommen können, wenn die Simulation weiter laufen würde. Also kann man schlussfolgern, dass mit einer durchdachten Nachrichten-Planung genug BE-Nachrichten weitergeleitet werden können.

source (ct_id/hostname)	Sequence number	send time(µs)	receive time(µs)	delay(µs)
be_msg: hostB:	1	822.947057	1898.075762	1075.128705
be_msg: hostB:	2	1718.809573	2360.630303	641.82073
be_msg: hostB:	3	2387.507653	3637.7278	1250.220147
be_msg: hostB:	4	2957.683041	4338.321497	1380.638456
be_msg: hostB:	5	3857.262322	6973.210284	3115.947962
be_msg: hostB:	6	4428.938964	7335.505193	2906.566229
be_msg: hostB:	7	4981.892767	7452.596151	2470.703384
be_msg: hostB:	8	5709.967934	7573.634407	1863.666473
be_msg: hostB:	9	6321.128626	7686.527036	1365.39841
be_msg: hostB:	10	7000.882577	8768.671483	1767.788906
be_msg: hostB:	11	7834.265932	8932.17737	1097.911438
be_msg: hostC:	1	718.793604	2538.445978	1819.652374
be_msg: hostC:	2	1410.514364	3358.012405	1947.498041
be_msg: hostC:	3	2106.90676	3525.900358	1418.993598
be_msg: hostC:	4	2791.027528	7861.595649	5070.568121
be_msg: hostC:	5	3780.336699	8022.378715	4242.042016
be_msg: hostC:	6	4600.297207	8158.749792	3558.452585
be_msg: hostC:	7	5337.097418	8315.769469	2978.672051
be_msg: hostC:	8	5999.16792	8459.62962	2460.4617
be_msg: hostC:	9	6950.467155	8658.467003	1707.999848
tt_msg: ct_id: 1	1	980	1250	270
tt_msg: ct_id: 1	2	1980	2250	270
tt_msg: ct_id: 1	3	2980	3250	270
tt_msg: ct_id: 1	4	3980	4250	270
tt_msg: ct_id: 1	5	4980	5250	270
tt_msg: ct_id: 1	6	5980	6250	270
tt_msg: ct_id: 1	7	6980	7250	270
tt_msg: ct_id: 1	8	7980	8250	270
tt_msg: ct_id: 1	9	8980	9250	270
tt_msg: ct_id: 2	1	1580	1850	270
tt_msg: ct_id: 2	2	3180	3450	270
tt_msg: ct_id: 2	3	4780	5050	270
tt_msg: ct_id: 2	4	6580	6850	270
tt_msg: ct_id: 2	5	8180	8450	270
tt_msg: ct_id: 3	1	1692.190854	3048.487747	1356.296893
tt_msg: ct_id: 3	2	3227.708883	4717.629737	1489.920854
tt_msg: ct_id: 3	4	6902.190444	8067.270209	1165.079765

Abbildung 5-2: Ausgabe des HostD-Servers

6 Zusammenfassung und Ausblick

In diesem Kapitel wird eine Zusammenfassung und einen Ausblick dieser Arbeit gegeben.

6.1 Zusammenfassung

In dieser Arbeit wurde der TTEthernet-Protokollstack für die Simulation in OMNeT++ entwickelt. Es wurde zuerst ein Überblick über die Simulationsplattform, welche aus der OMNeT++-Simulationsumgebung und dem INET-Framework besteht, gegeben. Aufgrund des modularen und objektorientierten Einsatzes von OMNeT++ konnte man Teile des vorhandenen Ethernet-Modells wiederverwenden bzw. erweitern. Die physikalische Schicht und die MAC-Subschicht wurden ohne Weiteres übernommen. Die LLC-Subschicht wurde auf den TTEthernet-Dienst erweitert. Weiterhin wurde in dieser Arbeit eine Auswahl der TTEthernet-API-Funktionen implementiert. Dies ermöglicht es echte Applikationen in der Simulationsumgebung vor ihren Einsatz in echten Systemen zu entwicklen bzw. testen. Darüber hinaus wurde ein Konfigurationsobjektmodell basierend auf der TTEthernet-Netzwerkkonfigurationsspezifikation von TTTech entworfen und implementiert. Der von Till Steinbach entwickelte Scheduler wurde so angepasst, dass er das Konfigurationsmodell verwenden kann und mit der LLC-Subschicht kommunizieren kann. Zum rudimentären Testen des TTEthernet-Protokollstacks und der Implementierung des Konfigurationsmodells wurde ein Kleinnetzwerk mit vier Endsystemen und einem Switch eingerichtet. Das Ergebnis zeigte dass, um ein verlustfreies Empfangen und eine minimale Verzögerung von TT-Nachrichten zu erreichen, die Applikationen mit dem Scheduling bzw. der LLC-Subschicht synchronisiert werden sollen. Weiterhin konnte man feststellen, dass mit einer durchdachten Nachrichten-Planung genug BE-Nachrichten weitergeleitet werden können.

6.2 Ausblick

Diese Arbeit gehört zu den Ersten im Rahmen der Entwicklung einer Simulationsumgebung für TTEthernet-Netzwerke, demzufolge gibt es in diesem Zusammenhang noch viele Herausforderungen zu meistern. Was die Konfiguration anbelangt wurde z. B. die Systemspezifikation nicht modelliert. Die physikalische Verbindung der Komponenten erfolgte manuell über den NED-Editor. Dies kann bei der Erstellung von großen Netzwerken unflexibel und zeitintensiv sein. Außerdem wurde das Senden/Weiterleiten von RC-Nachrichten nicht realisiert. Dies soll komplettiert werden.

Wie bereit angesprochen, wurden nur ausgewählte Funktionen der TTEthernet-API implementiert. Diese soll allmählich vollständig implementiert werden. Als Nächstes könnten z. B. Callback-Funktionen implementiert werden, die nach dem vollständigen Senden einer Nachricht bzw. beim Empfangen einer Nachricht aufgerufen werden. Dadurch können minimale Verzögerungen von TT-Nachrichten erreicht werden.

Die Speicherverwaltung ist auch ein interessantes Thema, vor allem beim Simulieren von großen Netzwerken, in denen „echte Nutzlast“ gesendet wird, wie im Fall von TTEthernet-Netzwerken. In den Standard Implementierungen in OMNeT++ bzw. INET wird hingegen nur eine Pseudo-Nutzlast verwendet, die nur durch die Nutzlast-Länge simuliert wird.

Bis jetzt wurden zum Testen der realisierten Funktionalitäten lediglich einfache kleine Netzwerke aufgebaut. In der Realität besteht ein Netzwerk aus mehreren Komponenten (Endsystemen, Switches), wobei einige davon redundant vorhanden sein können. Die Modellierung solche Netzwerke soll in der TTEthernet-Simulationsumgebung möglich sein.

Für das Testen, die Analyse und die Bewertung von TTEthernet-Netzwerken in der Simulation unter realitätsnahen Bedingungen sollen realistische Topologien (z. B. typische Fahrzeug-Netzwerk-Topologien) und Traffic-Profile erstellt werden. Beim Letzteren soll die Integration aller drei Traffic-Klassen (TT, RC und BE) in Betracht gezogen werden. Weiterhin sollen für die Analyse der Test-Ergebnisse systematische Verfahren verwendet werden. Man kann z. B. Skripte erstellen, womit die Ergebnisse automatisch analysiert werden können oder die Analyse-Werkzeuge von OMNeT++ verwenden.

Für das Projekt 2 sind folgende Themen von Interesse: die Komplettierung des Konfigurations-Objektmodells und der TTEthernet-API-Implementierung, das Designen von großen TTEthernet-Netzwerken, der Entwurf einer Traffic-Profile-Schnittstelle für Applikationen und eine systematische Auswertung der Simulationsergebnisse auf der Applikationsebene.

Abkürzungsverzeichnis

AFDX	Avionics Full DupleX Switched Ethernet
API	Application Programming Interface
BE	Best-Effort
CRC	Cyclic Redundancy Check
CSMA/CD	Carrier Sense Multiple Access/Collision Detection
CT	Critical Traffic
CT-ID	Critical Traffic Identification
DSAP	Destination Service Access Point
IP	Internet Protocol
Kfz	Kraftfahrzeug
LAN	Local Area Network
LLC	Logical Link Control
MAC	Media Access Control
NED	Network Description
OMNeT++	Objective Modular Network Testbed in C++
PDU	Protocol Data Unit
RC	Rate-Constrainted
SSAP	Source Service Access Point
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TT	Time-Triggered
TTE	Time-Triggered Ethernet
TTEthernet	Time-Triggered Ethernet
UDP	User Datagram Protocol
WWW	World Wide Web
XML	Extensible Markup Language
XSD	XML Schema

Abbildungsverzeichnis

Literaturverzeichnis

AIM. (2008). *AFDX*. Abgerufen am 08 2010 von http://www.afdx.com/afdx.html

Hein, M. (1998). *Ethernet: Fast Ethernet, Gigabit Ethernet* (2., aktualisierte und erweitete Auflage Ausg.). Bonn: ITP-Verlag.

Held, G. (2003). *Ethernet Networks: Design, Implementation, Operation, Management.* (Four Edition Ausg.). West Sussex: John Wiley & Sons, Ltd.

OMNeT++ Community. (2010). *OMNeT++*. Abgerufen am 10 2010 von http://www.omnetpp.org

OMNeT++ Community2. (2010). *INET Framework for OMNeT++*. Abgerufen am 10 2010 von URL-http://inet.omnetpp.org

SAE - AS-2D Time Triggered Systems and Architecture Committee. (2009). *Time-Triggered Ethernet (AS 6802).* Abgerufen am 08 2010 von URL http://www.sae.org

Schudel, G. (2010). *Bandwidth, Packets Per Second, and Other Network Performance Metrics*. Abgerufen am 10 2010 von sisco: http://www.cisco.com/web/about/security/intelligence/network_performance_metrics.html

Steinbach, T. (2009). *Eine Plattform für die eventbasierte Simulation von time-triggered Ethernet Netzwerken.* Hamburg: HAW Hamburg.

Steiner, W. (November 2008). TTEthernet specification. TTTech Computertechnik AG.

Steiner, W., Günther, B., Brendan, H., Michael, P., & Srivatsan, V. (2009). *TTEthernet Dataflow Concept.*

Tanenbaum, A. S. (2003). *Computernetzwerke* (4., überarbeitete Auflage Ausg.). München: Pearson Studium.

TTTech. (2010). *TTEthernet*. Abgerufen am 10 2010 von URL http://www.tttech.com

TTTech2. (2009). *Network Configuration Documentation*. Abgerufen am 10 2010 von URL http://www.tttech.com

TTTech3. (2010). *TTEthernet-API*. Abgerufen am 10 2010 von http://www.tttech.com