Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Florian Bartols

Leistungsmessung von Time-Triggered Ethernet Komponenten unter harten Echtzeitbedingungen mithilfe modifizierter Linux-Treiber

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Florian Bartols

Leistungsmessung von Time-Triggered Ethernet Komponenten unter harten Echtzeitbedingungen mithilfe modifizierter Linux-Treiber

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. Schmidt

Abgegeben am 14. Juli 2010

Thema dieser Bachelorarbeit

Leistungsmessung von Time-Triggered Ethernet Komponenten unter harten Echtzeitbedingungen mithilfe modifizierter Linux-Treiber

Stichworte

Echtzeit, Linux, Ethernet, Netzwerk, Treiber, zeitgesteuert, Automotiv, TTTech, TTEthernet, Switch

Kurzzusammenfassung

Diese Arbeit befasst sich mit dem Echtzeitübertragungsprotokoll Time-Triggered Ethernet. Es werden zuerst Grundlagen einer Echtzeiterweiterung im Linux-Kernel und dem Standard Ethernet behandelt. Eine anschließende genaue Betrachtung von Time-Triggered Ethernet folgt. Aufbauend auf den Grundlagen wird die Implementierung einer Messsoftware und die dazugehörige Modifizierung von Linux-Treibern erklärt und im Folgenden die Ergebnisse aus der Messung diskutiert.

Topic of this Bachelorsthesis

Performance analysis of Time-Triggered Ethernet components in harsh real-time conditions using modified Linux drivers

Keywords

Real-time, Linux, Ethernet, Networking, Drivers, Time-Triggered, Automotiv, TTTech, TTEthernet, Switch

Abstract

This thesis deals with the real-time transmission protocol Time-Triggered Ethernet. At first the fundamentals of a real-time Linux-Kernel-extension and the standard Ethernet are covered. A detailed analysis of Time-Triggered Ethernet follows. Building on the basic principles the implementation of a measurement software and the associated modification of Linux drivers are explained. In conclusion the results from the measurement are discussed.

Danksagung

Großen Dank gilt der Fujitsu Technologys Abteilung in Paderborn, insbesondere Andreas Büker, der mit seiner Unterstützung zur Verifikation der Messsoftware eine große Hilfe war. Der Zugriff auf spezielle Messhardware ermöglichte es, die Messsoftware zu verifizieren und zu analysieren.

Weiteren Dank gilt der RTEthernet-Gruppe der HAW, die mit ihren konstruktiven Kritiken immer wieder hilfreich zur Seite standen und durch Diskussionen die Lösung von Problemstellungen unterstützten.

Ich möchte mich außerdem bei meinen Eltern und Freunden für die Unterstützung während der Bearbeitung bedanken.

Inhaltsverzeichnis

Tabellenverzeichnis **7**

Abbildungsverzeichnis **8**

1 Einleitung **10**
1.1 Motivation . . . 10
1.2 Zielsetzung . . . 11
1.3 Inhaltlicher Aufbau der Arbeit . . . 12

2 Grundlagen **13**
2.1 Echtzeit . . . 13
2.1.1 Unterscheidung zwischen harter und weicher Echtzeit . . . 13
2.2 Real-Time und Linux . . . 14
2.2.1 Zwei Ansätze Linux echtzeitfähig zu machen . . . 16
2.3 Ethernet . . . 23
2.3.1 Eigenschaften . . . 24
2.3.2 Der Weg zu weichen Echtzeiteigenschaften . . . 25
2.3.3 Resümee weicher Echtzeiteigeschaften . . . 28

3 Time-Triggered-Ethernet **29**
3.1 Echtzeit und Ethernet . . . 29
3.1.1 Derivate echtzeitfähiger Ethernet Implementierungen . . . 30
3.2 Priorisierung, Nachrichtenklassen und -Erkennung im TTEthernet . . . 31
3.2.1 Nachrichtenklassen . . . 31
3.2.2 Nachrichtenerkennung . . . 32
3.2.3 Scheduling und Datenfluss im TTEthernet . . . 33
3.2.4 Integration der Frames im Systemschedule . . . 35
3.3 Synchronisation im TTEthernet . . . 35
3.3.1 Geräte und deren Rolle im Synchronisierungsprozess . . . 36
3.3.2 Aufbau der Synchronisationsframes . . . 39
3.3.3 Synchronisierungsfunktionen und deren Aufgabe . . . 39

4 Analyse und Implementation **42**

4.1 Beschreibung des Evaluationssystems . . . 42
4.1.1 Hardware . . . 42
4.1.2 Software . . . 43
4.1.3 Aufbau und Beschreibung der Software . . . 44
4.1.4 Aufbau der TT-Software und Treiber . . . 45
4.2 Zeitmessung . . . 48
4.2.1 Metriken der Netzwerk-Messtechnik . . . 48
4.2.2 Problematik der Zeitmessung . . . 48
4.3 Implementierung der Messsoftware . . . 51
4.3.1 Zeitstempel unter Linux . . . 51
4.3.2 Modifikation des Ethernetframes . . . 52
4.3.3 Auswertung der Timestamps in der Anwendung . . . 54

5 Zeitmessung, Verifikation und Auswertung 56
5.1 Zeitmessung . . . 56
5.2 Verifikation . . . 58
5.2.1 Oszilloskop . . . 58
5.2.2 Hardware-Paketgenerator/Sniffer . . . 59
5.3 Diskussion der Ergebnisse . . . 60
5.3.1 Abhängigkeit von Treiberoptionen . . . 61
5.3.2 Abhängigkeit zum Systemschedule . . . 62
5.3.3 Abhängigkeit vom Zeitpunkt des Zeitstempelns . . . 64
5.3.4 Zusammensetzung der gemessenen Zeit . . . 65
5.4 Bewertung der Ergebnisse . . . 66
5.4.1 Softwaremessung - Hardwaremessung . . . 67
5.4.2 Softwaremessung - formales Modell . . . 68

6 Zusammenfassung, Fazit Ausblick 70
6.1 Zusammenfassung und Fazit . . . 70
6.2 Ausblick . . . 72
6.2.1 Messung mit anderer Netzwerkhardware durchführen . . . 72
6.2.2 Weitere Arbeiten im Rahmen des RTEthernet-Projektes . . . 72
6.2.3 Möglicher Einsatz von TTEthernet im Automobil . . . 73

Literaturverzeichnis 75

Tabellenverzeichnis

3.1 Erkennung von zeitkritischem Verkehr 32

4.1 Übersicht der zu messenden Netzwerkmetriken 48

5.1 Übersicht der minimalen und maximalen Framelänge aller verwendeten Verfahren 69

Abbildungsverzeichnis

1.1 Bussystem eines modernen Fahrzeugs mit Vollausstattung 10

2.1 Interrupt Prioritätsklassen 15
2.2 Priotitäsinversion 17
2.3 Kooperativer Kernel 18
2.4 Interrupt Prioritäts-Inversion bei HardIRQs 20
2.5 Auswahlmöglichkeiten der Verdrängungsstrategien im Kernel 21
2.6 Ethernet Frame 24
2.7 Ethernet-Netztopolgien im Vergleich 26
2.8 Vergleich Ethernet-Hub und Switch 27
2.9 V-LAN-Frame 27

3.1 TT-Ethernet-Frame 32
3.2 Scheduling Strategie des TT-Ethernet 35
3.3 Integration der Nachrichten in den Systemschedule 36
3.4 Zwei-Wege-Synchronisation 37
3.5 TT-Netwerk mit mehreren Switches und Gerätekonfiguration 38

4.1 Aufbau des Evaluationssystems 43
4.2 Überblick Softwarestruktur 46
4.3 Mögliche Integrierung von Messmethodiken im System 49
4.4 Versuchsaufbau TU Dresden 50
4.5 Timestamps im Ethernetframe 55

5.1 Versuchsaufbau der Zeitmessung 56
5.2 Auswertung der Messung 57
5.3 Versuchsaufbau Verifikation mittels Hardwaresniffer 60
5.4 Messung mittels Hardwaresniffer. Synchronisierungsframe dient als Nullpunkt 61
5.5 Messergebnis mit veränderter Kopierschwelle im Vergleich zum unveränderten Treiber 62
5.6 Systemschedulekonfiguration 63
5.7 Softwaremessung abzüglich des Systemschedules 63
5.8 Gegenüberstellung Hardwaremessung Softwaremessung 64

5.9 Versuchsmessung des Verdoppelungsswitches 65
5.10 Bestimmung Netzwerkkarten-, Konfigurationsverzögerung 66
5.11 Gegenüberstellung Hardwaremessung - Softwaremessung 67

6.1 Möglicher Einsatz von TTEthernet als Backbone-Netz im Automobil 73

1 Einleitung

1.1 Motivation

Diese Arbeit ist aus dem Kontext des Realtime-Ethernet-Projekts der HAW entstanden, das sich mit dem Einsatz von Time-Triggered Ethernet (im weiteren Verlauf auch TTEthernet genannt) im Automobil beschäftigt.

Aktuell verfügbare Fahrzeuge besitzen eine Vielzahl von Sensoren und Reglern, die u.a. für Steuerung, Fahrsicherheit und -komfort während der Reise verantwortlich sind. Abbildung 1.1 stellt das Netzwerk eines aktuell verfügbaren Fahrzeugs mit Vollaustattung dar. Für verschiedene Aufgabenbereiche gibt es jeweils unterschiedliche Bussysteme, sodass eine inhomogene Gesamttopologie entsteht. FlexRay und CAN werden aktuell für zeitkritischen Datenverkehr genutzt, während MOST und LIN für eine Zustellung von weniger kritischem Verkehr genutzt werden.

Durch die Einführung von weiteren Assistenzsystemen, wie Spur- und Verkehrsschilderkennung wird die Bandbreite an zeitkritischem Verkehr weiter steigen. Aktuelle Bussysteme, wie FlexRay oder CAN stoßen mit ihren geringen Bandbreiten (FlexRay 10MBit/s, High Speed CAN 500 kbit/s Low Speed 125kbit/s (vgl. Zimmermann, 2008)) an die Grenzen ihrer Leistungsfähigkeit. Gerade Bildbearbeitung hat ein hohes Datenaufkommen und eine Kompression ist aufgrund des daraus resultierenden Informationsverlustes nicht erlaubt.

Darüber hinaus wird der Bedarf an Bandbreite auch im nicht zeitkritischen Verkehr weiter steigen. Vor allem die Einführung von weiteren Multimedia- und Navigationssystemen beeinflussen die Komplexität der Vernetzung. Zukünftige Systeme werden über einen Internetzugang verfügen, sodass die aktuellsten Staumeldungen in die Navigation einfließen werden.

Ethernet bietet mit seiner, im Vergleich zu den anderen Protokollen, eine hohe verfügbare Bandbreite und hat sich als Standard für lokale Netze etabliert. Durch die Verwendung von Switches lässt sich ein nahezu kollisionsfreies, hoch skalierbares Kommunikationsnetz aufbauen. TTEthernet fügt dem Standard Ethernet Echtzeitfähigkeit hinzu und kann mit seinen Eigenschaften als eine neue Alternative für zukünftige Anwendungen gesehen werden, die hohe Bandbreiten benötigen.

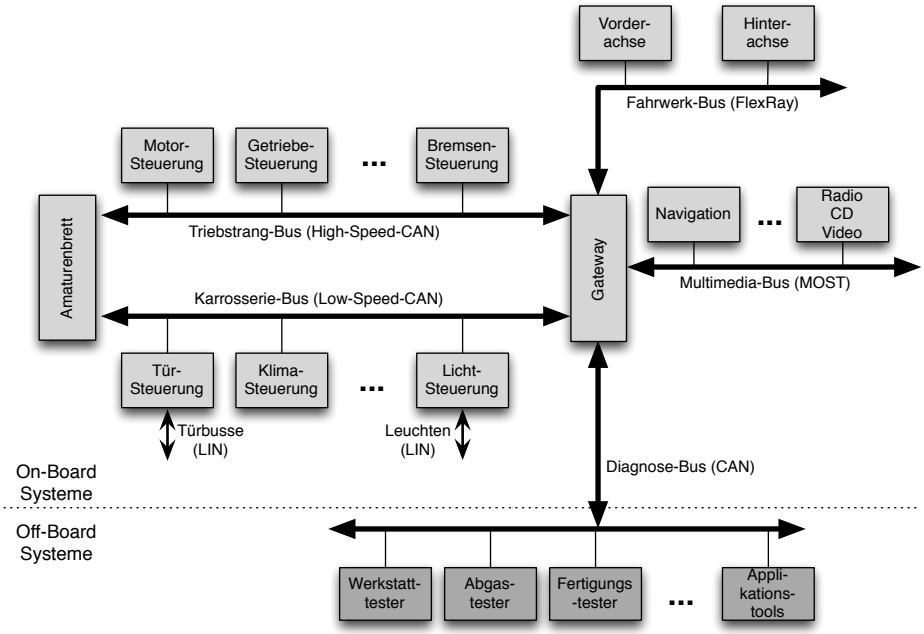

Abbildung 1.1: *Bussystem eines modernen Fahrzeugs mit Vollausstattung (vgl. Zimmermann, 2008; Gressl, 2010)*

1.2 Zielsetzung

Die Zielsetzung dieser Arbeit setzt sich aus zwei Teilaufgaben zusammen. Der erste Teil ist das Bring-Up des TTEthernet Evaluationssystems, der zweite Teil befasst sich mit einer Latenzmessung von TTEthernetkomponenten.

Das Bring-Up befasst sich mit der Analyse der TTEthernet-Spezifikation und einer anschließenden Inbetriebnahme des Evaluationssystems. Die mitgelieferten Hard- und Softwarekomponenten werden auf Funktion und Verhalten getestet, sodass ein klares Bild in der Handhabung des Evaluationssystems entsteht.

Aufbauend auf diesem Wissen soll ein Verfahren entwickelt werden, um Latenzzeiten von Echtzeitdaten in TTEthernetnetzwerken zu messen. Dabei kann nicht auf vorhandene Messsoftware zurückgegriffen werden, da Echtzeitdaten im TTEthernet einem Schedule unterliegen. Daten, die außerhalb dieses Schedules versendet werden, werden verworfen und eine Messung ist nicht möglich. Das Verfahren umfasst dabei die Konzeption der anschließenden Messungen und deren Versuchsaufbau und der Implementation einer geeigneten Software. Die im Linux-Kernel laufenden Treiber müssen dazu modifiziert und eine Anwendungssoftware entwickelt werden, die die Latenz und den Jitter anhand Zeitstempel berechnen, sodass eine anschließende Diskussion der Messergebnisse möglich ist.

1.3 Inhaltlicher Aufbau der Arbeit

In dieser Arbeit werden zuerst die Grundlagen besprochen, die für das Verständnis von Nöten sind. Zuerst wird dabei auf den Begriff Echtzeit eingegangen. Im Anschluss werden die Konzepte eines Echtzeitbetriebssystems erläutert und auf die Umsetzung innerhalb des Linux-Kernels erläutert. Die Grundlagen werden abgeschlossen mit einem Abschnitt, der sich mit Ethernet befasst und einen Weg zu weichen Echtzeiteigenschaften unter diesem Protokoll beschreibt.

Nachdem die Grundlagen besprochen sind, wird Time-Triggered Ethernet als ein echtzeitfähiges Übertragungsprotokoll besprochen. Ein kurzer Überblick auf das unterstützte Avionics Full Duplex Switched Ethernet-Protokoll leitet dieses Kapitel ein, gefolgt von der Beschreibung der verwendeten Nachrichtenklassen im TTEthernet. Der Zeit-Synchronisierungsdienst schließt dieses Kapitel ab.

Aufbauend auf dem Wissen, das durch die Grundlagen und TTEthernet bekannt ist, wird das ausgelieferte Evaluierungssystem beschrieben. Auf die mitgelieferte Soft- und Hardware wird ein Überblick gegeben und das Verhalten der Beispielapplikation erläutert. Anschließend wird auf die Implementierung der Messsoftware mittels Modifizierung der Netzwerktreiber eingegangen. Die erzielten Ergebnisse werden im anschließenden Kapitel diskutiert. Zudem wird auf den verwendeten Versuchsaufbau und eine Verifikation der Messung durch Hardware eingegangen.

Das letzte Kapitel fasst diese Arbeit zusammen, stellt mögliche Einsatzgebiete von TTEthernet vor und gibt Aufschluss auf anschließende Arbeiten im RTEthernet-Kontext.

2 Grundlagen

In diesem Kapitel werden die Grundlagen besprochen, die wichtig für das Verständnis der TTEthernet-Architektur und das analysierte Evaluations-System sind. Zunächst wird der Begriff „Echtzeit“ erläutert. Anschließend wird auf die Möglichkeit der Echtzeiterweiterung von Linux eingegangen und zum Schluss des Kapitels wird Ethernet aus dem Echtzeitkontext betrachtet.

2.1 Echtzeit

Unter Echtzeit versteht man in informationstechnischen Systemen, dass ein System auf ein Ereignis innerhalb einer bestimmten Zeit garantiert reagieren muss. Abhängig von den Systemanforderungen wird dann pro Anforderung unterschieden, wie schnell ein System auf ein Ereignis reagieren soll. Wenn das System nicht innerhalb des vordefinierten Zeitraums auf ein Ereignis reagiert, so ist die Reaktion nicht nur zu spät, sondern falsch.

Ein gutes Beispiel hierfür ist der Vergleich zwischen der Kraftstoffeinspritzsteuerung eines Kraftfahrzeugs und dessen Fahrwerkssteuerung. Beide Systeme unterliegen den Eigenschaften der Echtzeit, unterscheiden sich jedoch in ihren Reaktionszeiten signifikant.

Teile der Einspritzsteuerung müssen unterhalb $100\mu s$ auf ein auftretendes Ereignis reagieren, ansonsten kann es zu Fehlzündungen des Kraftstoffgemisches kommen, die mitunter große Schäden am Motor verursachen. In einer Fahrwerkssteuerung gibt es Teilanforderungen, die Reaktionszeiten unterhalb $10ms$ besitzen. Die Differenz der Zeiten resultiert aus den unterschiedlichen physikalischen Bedingungen der Systeme.

Echtzeit bedeutet also nicht so schnell wie möglich, sondern die garantierte Einhaltung von Zeitaussagen.

2.1.1 Unterscheidung zwischen harter und weicher Echtzeit

Im Allgemeinen besitzen Echtzeitsysteme eine Menge von Deadlines. Diese bestehen in der Regel aus einem verteilten System und bearbeiten verschiedene Aufgabe. Jedes Segment

besitzt eigene Zeitaussagen, die unter allen Umständen eingehalten werden müssen. Unterschieden wird weiterhin, wie das System auf eine Verletzung einer Deadline, also das Brechen einer Zeitaussage, reagiert.

Hart: Ein Echtzeitsystem hat harte Echtzeitanforderungen, wenn die Verletzung einer oder mehrerer Deadlines kritische Auswirkungen auf das System haben; zum Beispiel die angesprochene Motorsteuerung im Auto.

Weich: Ein Echtzeitsystem hat weiche Anforderungen, wenn die Verletzung keine kritischen Auswirkungen auf das System hat, dessen Einhaltung jedoch wünschenswert ist; zum Beispiel Voice-Over-IP. Im schlimmsten Fall werden zu spät eintreffender Frames verworfen und die Tonausgabe stockt.

(vgl. Tanenbaum, 2009, S. 208-209)

Harte und weiche Echtzeitanforderungen unterscheiden sich im Anwendungsgebiet. Sicherheitskritische Systeme haben häufig harte, Multimedia-Anwendungen weiche Anforderungen.

2.2 Real-Time und Linux

Entwickelt wurde Linux von Linus Torvalds. Er orientierte sich bei der Entwicklung sehr stark an Minix, das von Andrew S. Tanenbaum geschrieben wurde, um in der Lehre ein leicht zu verstehendes Betriebssystem einzusetzen. Jedoch entschied sich Torvalds dazu, den Micro-Kernel-Ansatz (Minimale Funktionalität im Kern) von Minix gegen einen monolithischen (alle Funktionen laufen im Kern) auszutauschen.

Die Vorteile eines Microkernels, Bestandteile einfach auszutauschen und fehlerhafte Komponenten haben keine kritischen Auswirkungen auf das System, wurden so genommen. Zusätzlich zugefügte Funktionalitäten (z.B. Hardware-Treiber, Echtzeitfähigkeit, etc . . .) laufen in einem Microkernel im Userspace. Linux umgeht dieses Problem, indem Kernelmodule während des Betriebes nachgeladen werden können. Jedoch kann ein fehlerhaft programmiertes Modul zum Totalabsturz des Betriebssystems führen (vgl. Tanenbaum, 2009, S. 101-104) .

1991 veröffentlichte Torvalds die erste Version von Linux, Version 0.0.1, und entwickelt es zusammen mit einer großen Gemeinschaft an Kernel-Programmierern stetig weiter. Mittlerweile (Stand Mai 2010) ist die stabile Kernelversion 2.6.33.4 veröffentlicht worden und es fließen Micro-Kernel-Konzepte, z.B. FUSE (FUSE - Filesystem in Userspace (vgl. Sourceforge.net, 2010)), ein.

Der große Vorteil von Linux ist die offene Verfügbarkeit des Quellcodes. Dadurch wird Linux auch immer mehr im Embedded-Systems-Bereich eingesetzt, da es bereits auf viele Hardware-Plattformen portiert worden ist. Als neuestes Beispiel kann die neue Android-Plattform angesehen werden, die auf einem Linux-Kernel basiert (vgl. www.android.com, 2010) . Dieses Betriebssystem wird vorwiegend für mobile Telefone und portable Geräte eingesetzt.

Linux ist ein General-Purpose-Betriebssystem, das auf jeder Systemkonfiguration performant ausgeführt werden soll. Es ist nicht echtzeitfähig. Der Scheduler von Linux trägt die Bezeichnung „completely fair“, da lang wartende Prozesse mit geringerer Priorität CPU-Zeit erhalten, auch wenn ein höher priorisierter Prozess Rechenzeit verlangt. In Echtzeitbetriebssystemen ist die Priorität das wichtigste Entscheidungskriterium des Schedulings. Diejenige Task, die momentan die höchste Priorität besitzt, bekommt CPU-Zeit zugewiesen.

Eine weitere Eigenschaft des Standard Linux-Kernels ist die Interruptbehandlung. Löst die Hardware einen Interrupt aus, wird dieser sofort bearbeitet. Eine momentan arbeitende Task wird unterbrochen, unabhängig ihrer Priorität. Die ISR (Interrupt Service Routine) sollte so kurz wie möglich sein und größere Aufgaben an anschließend arbeitende SoftIRQs, Tasklets oder Kernel-Threads übergeben. Nachdem diese fertig sind, kann die unterbrochene Task weiter ausgeführt werden, was zu langen Reaktionszeiten innerhalb des Prozesses führen kann.

Nachfolgende Abbildung 2.1 stellt das Prinzip der Interrupt-Behandlung, der Preemption-Ebenen und -Klassen und Prioritäten im standard Kernel dar. Die Preemption-Ebene gibt an, welche Priorität die Task innerhalb des Systems hat und von welchen anderen Prozessen sie unterbrochen werden kann.

Die höchste Priorität besitzen die interruptaffinen Prozesse. Gefolgt von den RT-Tasks (Realtime-Task), die für weiche Echtzeit-Aufgaben bestimmt sind. Diese Prozesse unterliegen einem extra Scheduling, welches über spezielle RT-Prioritäten gesteuert wird, die zwischen 1 und 99 liegen. Prozesse mit der RT-Priorität von 1 haben die geringste, von 99 die höchste Priorität und bekommen so öfter CPU-Zeit zugewiesen. Als Scheduling-Verfahren kann zwischen FIFO und Round-Robin gewählt werden. Alle anderen Tasks unterliegen dem fairen Scheduling und haben damit die geringste Priorität.

Ein solches Verhalten steht im Gegensatz zu dem, was ein Echtzeitbetriebssystem als Eigenschaften haben muss. Lange Latenzzeiten innerhalb eines Prozesses können die Einhaltung einer Deadline gefährden.

Prioritätsinversionen (ein niederpriorer Task blockiert einen hochprioren Task) können zusätzlich für lange und unvorhersehbare Latenzzeiten sorgen. Eine Prioritätsinversion kann zum Beispiel bei drei konkurrierenden Prozessen auftreten (siehe Abbildung 2.2). Prozess A mit niedriger Priorität befindet sich in einem kritischen Bereich, wird aber von einem Prozess

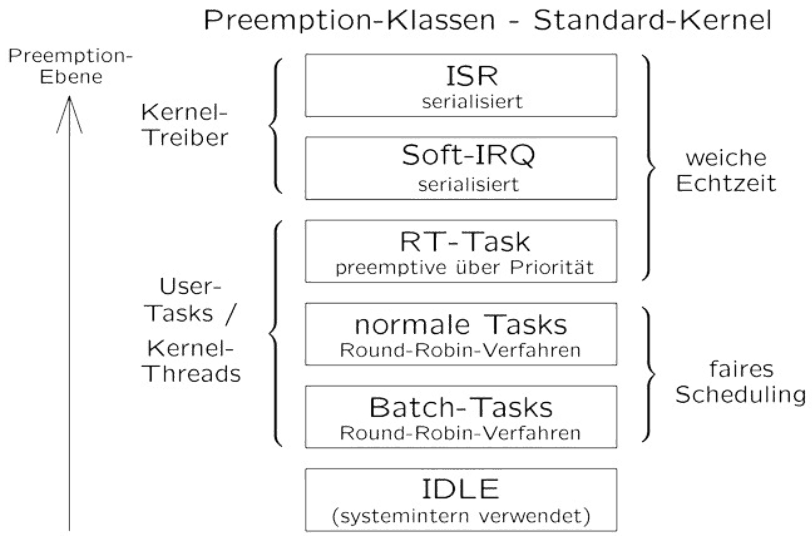

Abbildung 2.1: *Interrupt Prioritätsklassen (vgl. Klinger, 2008)*

B mit mittlerer Priorität verdrängt. Nun will ein Prozess C mit hoher Priorität in den kritischen Bereich, kann aber nicht, weil A die Ressource noch nicht abgegeben hat und von B blockiert wird. Der Prozess C muss warten bis B fertig gerechnet hat, damit A die Ressource abgeben kann.

Ein Mittel, dieses Dilemma zu verhindern, nennt sich Prioritätsvererbung. Dabei bekommt der Prozess A kurzzeitig die Priorität von C vererbt. Nun kann A B verdrängen und anschließend C die kritische Ressource übergeben. Leider ist dieses Mittel nicht immer erfolgreich um kritische Deadlines einzuhalten, da es auf Heuristiken aufbaut. Durch Analysen muss sichergestellt werden, dass der fehlerhafte Fall nicht eintrifft. Die Abbildung 2.2 stellt die Gesamtzeit von Prozess C mit und ohne Prioritätsvererbung dar (vgl. Yaghmour u. a., 2008, S. 399) .

Das wichtigste Verhalten, was ein Realtime Betriebssystem ausmacht, ist, dass es in einer vorhersagbaren Zeit auf ein Ereignis reagiert. Im Standard Kernel kann es passieren, dass Interrupts für eine unvorhersagbare Zeit deaktiviert werden wenn eine Kernel-Task einen kritischen Abschnitt betritt. Auftretende Interrupts werden unterdrückt, damit ein höherer Datendurchsatz erreicht wird. Die Task, die momentan CPU-Zeit bekommen hat, wird so nicht unterbrochen. Es kann somit nicht in vorhersehbarer Zeit auf das Ereignis reagiert werden (vgl. Yaghmour u. a., 2008, S. 357) .

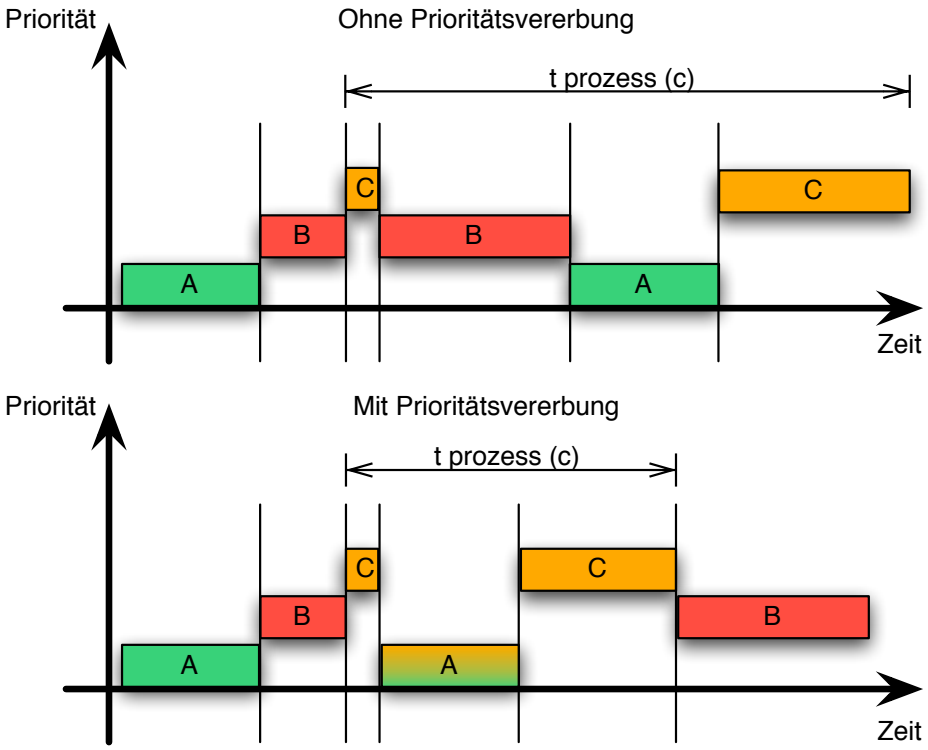

Abbildung 2.2: *Ausführungszeit von Prozess C bei Prioritätsinversion mit und ohne Prioritätsvererbung*

2.2.1 Zwei Ansätze Linux echtzeitfähig zu machen

Es gibt zwei verschiedene Ansätze Linux um Echtzeitfunktionalität zu erweitern. Die erste beschriebene Lösung ist, einen kooperativen Kernel, der hierarchisch gesehen zwischen der Hardware und dem Linux-Kernel liegt, zu verwenden. Die zweite basiert darauf, den gesamten Betriebssystem-Kern komplett unterbrechbar zu gestalten.

Kooperativer Kernel und Interrupt-Abstraktion

In dieser Arbeit wird diese Herangehensweise nur kurz erläutert, da das verwendete Betriebssystem auf dem Realtime-Kernel-Patch basiert, der anschließend genauer betrachtet wird.

Die Abbildung 2.3 zeigt die interne Struktur des Betriebssystems. Zwischen dem Linux-Kernel und den parallel anzusehenden RT-Tasks liegt der kooperative Realtime-Kernel. Dieser hat die Aufgaben, die Interrupts der Hardware zu überprüfen, ob sie einer RT-Anwendung

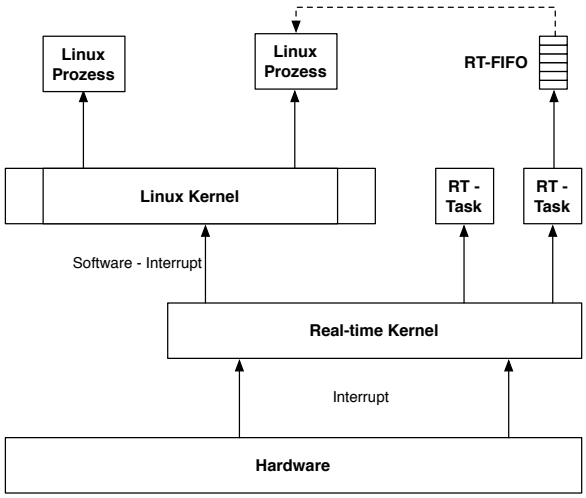

Abbildung 2.3: *Kooperativer Kernel (vgl. Abbot, 2006)*

oder zu einer Anwendung innerhalb des Linux-Kernels zugewiesen sind. Weiterhin übernimmt der RT-Kernel das Scheduling der RT-Anwendungen und des eigentlichen Linux-Kernels. Dieser hat die geringste Priorität und ist als Idle-Prozess anzusehen. Unter dem Linux-Kernel können weiterhin andere Tasks ausgeführt werden. Diese unterliegen zusätzlich dem Scheduler des Standard-Kernels.

Dieses Verfahren wird auch **Interrupt-Abstraktion** (vgl. Abbot, 2006) genannt, weil der Linux-Kernel keinen direkten Kontakt mehr zur Hardware hat. Teilen sich die RT-Task und der Linux-Prozess einen Interrupt, der Linux-Kernel betritt einen kritischen Bereich und schaltet die Interrupts ab, wird lediglich ein Software-Flag gesetzt. Der Linux-Kernel „denkt“, dass er die Interrupts ausgeschaltet hat, jedoch empfängt der Realtime-Kernel weiterhin Interrupts von der Hardware und leitet sie ggf. an RT-Tasks weiter.

Bei einer Interrupt-Flut kann es dazu führen, dass RT-Tasks keine Rechenzeit bekommen, da das Betriebssystem mit der Abarbeitung der IRQs beschäftigt ist.

Ziel dieser Lösung ist, dass nur der Teil des Betriebssystems modifiziert wird, der auch wirklich echtzeitfähig sein muss. Die Aufgaben, z.B. GUI-Darstellung, Logging-Funktionen, können weiterhin als normaler Linux-Task laufen. Problematisch ist, Standard Linux-Treiber und Software-Bibliotheken auf den Realtime-Kernel zu portieren, was u.U. zu einem höheren Entwicklungsaufwand führen kann.

Beispiele für diesen Ansatz sind:

- Xenomai (vgl. www.xenomai.org, 2010)
- RTAI (vgl. www.rtai.org, 2010)
- RTLinux (vgl. www.rtlinuxfree.com, 2010)

RT-Patch

Die andere Möglichkeit besteht darin, den gesamten Linux-Kernel echtzeitfähig zu gestalten. Der Vorteil ist, die gesamten mitgelieferten Treiber und Bibliotheken weiterhin zu benutzen, da die Kernel-Modifikation alle Bestandteile des Betriebssystems beeinflusst. Realtime-Anwendungen können auf einem normalen, x86-basierten Desktop-System entwickelt und getestet werden, um später auf ein Embedded-System, das auf einer anderen Hardware-Plattform, z.B. ARM aufgebaut ist, portiert zu werden.

Entwickelt wird diese Kernel-Erweiterung von einem Team um Ingo Molnar, einen ungarischen Kernel-Hacker. Molnar entschied sich gegen die Interrupt-Abstraktion, weil er die Vorteile einer RT-Erweiterung an die Benutzer des Standard Kernels weitergeben wollte. So können zum Beispiel der High-Resolution-Timer und die Prioritäten-Vererbung als solche Vorteile angesehen werden, die bereits ihren Weg in den Standard-Kernel genommen haben. In diesem Abschnitt wird der entwickelte RT-Patch genauer betrachtet, da auf ihm das mitgelieferte TTEthernet-Evaluierungssystem von TTTech basiert.

Eine Besonderheit von Linux ist, wie oben bereits erwähnt, dass nach der Ausführung einer ISR die weitere Interrupt-Behandlung mittels SoftIRQs, Tasklets oder Kernel-Threads erledigt wird. Da diese im Standard-Kernel der höchsten Priorität unterliegen, werden Userspace-Anwendungen und andere Kernel-Tasks von ihnen verdrängt. Ihrerseits können sie nur von einer anderen ISR verdrängt werden.

Der RT-Patch bereinigt ein signifikantes Problem dieser Interrupt-Behandlung. Unter besonderen Umständen kann es passieren, dass es zu einer Interrupt-Prioritäten-Inversion kommt. Ein aktuell arbeitender SoftIRQ mit einer hohen Priorität wird von einer ISR verdrängt, welche zu einem Prozess zählt, der mit geringerer Priorität läuft. Dieses Problem wird mittels einer speziellen HardIRQ-Thread-Funktion gelöst. Ein auftretender Interrupt startet einen neuen Kernel-Thread mit identischer Priorität, des zugehörigen Prozesses und wird anschließend dort bearbeitet.

Die Eigenschaft, dass eine ISR alle anderen Prozesse verdrängt, kann nicht verändert werden. Jedoch bringt dieser Ansatz deutlich kürzere Störungen mit sich, da nicht die gesamte Funktionalität der ISR ausgeführt werden muss. Lediglich das Starten eines neuen Threads stört die Ausführung des hochprioren Prozesses. Dieses Verhalten ist in der nachfolgenden

Abbildung 2.4 zu erkennen. Die Ausführungszeit von einem Prozess ist bei Aktivierung von HardIRQs kürzer, als ohne dieser Funktion und somit eine wichtige Eigenschaft im Echtzeitverhalten. Eine auftretende Interrupt-Flut kann mit dieser Funktion allerdings nicht gelöst werden.

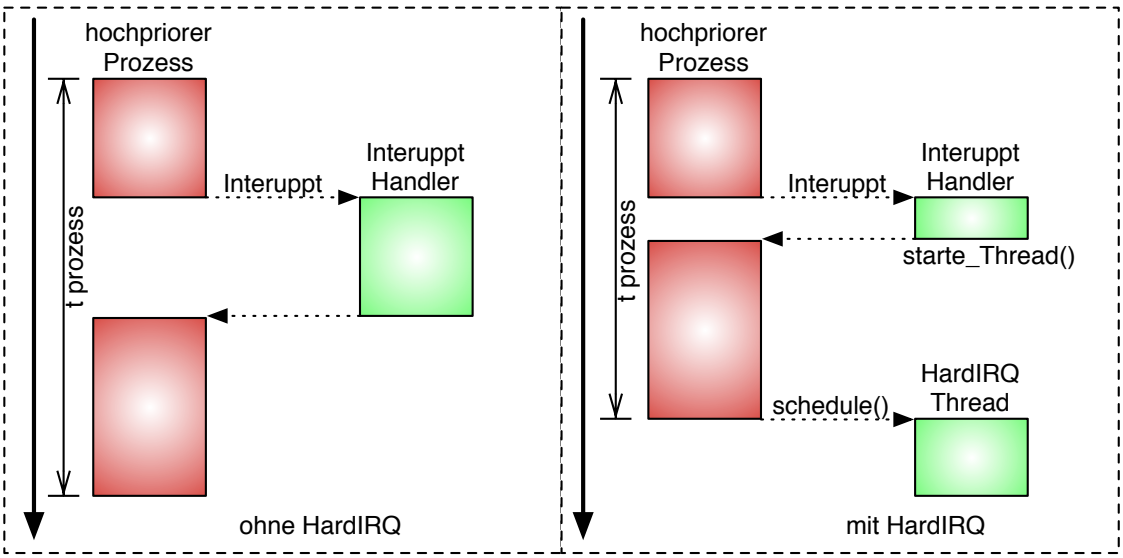

Abbildung 2.4: *Interrupt Prioritätsinversion bei HardIRQs (vgl. Yaghmour u. a., 2008)*

Threads, die durch die RT-Erweiterung gestartet wurden, laufen im Kernelspace ab und bekommen zusätzlich eine RT-Priorität. Dadurch ist es möglich, dass Userspace-Anwendungen, die dem RT-Schedule unterliegen, diese Kernel-Threads verdrängen können. Diese Threads unterliegen im Standard Kernel einem separaten Scheduler, der unterschiedliche Strategien verfolgt. Das Scheduling-Verfahren kann vor dem Übersetzen des Kernels ausgewählt werden und bietet folgende Möglichkeiten:

- No forced preemption
- Voluntary kernel preemption
- Preemptible kernel

Der RT-Patch erweitert diese Funktionalität um einen weiteren Punkt:

- Complete preemption

Die Funktionsweise der einzelnen Verdrängungsstrategien wird als nächstes erläutert.

No forced preemtion: Diese Einstellung im Kernel entspricht der Strategie des 2.4er-Kernels. Mit dieser Einstellung ist es nicht möglich, Kernel-Threads zu unterbrechen

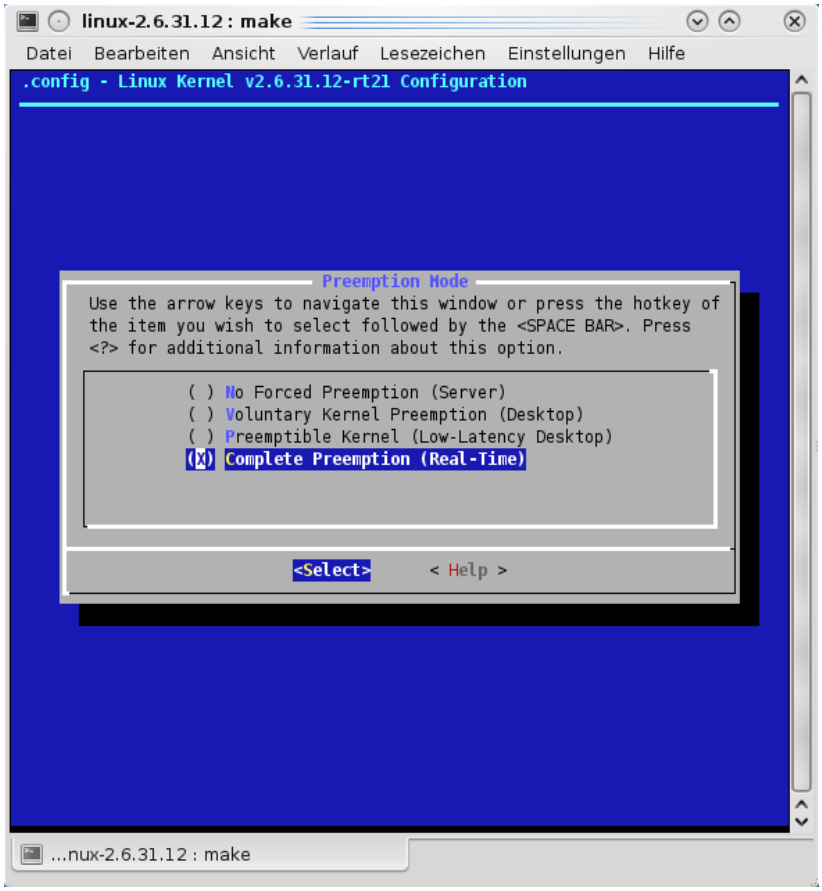

Abbildung 2.5: *Auswahlmöglichkeiten der Verdrängungsstrategien im Kernel*

und wird deshalb häufig für Server-Systeme verwendet, die große Batch-Aufgaben[1] bearbeiten müssen.

Voluntary kernel preemption: Mit der Einführung des 2.6er Kernel ist diese Einstellung hinzu gekommen. Sie ermöglicht eine Verdrängung an definierten[2] Punkten und ist damit abhängig von der Code-Implementierung. Diese Punkte im Kernel sind mit einer might_sleep()-Funktion erkenntlich gemacht. An dieser Stelle wird von den Kernel-Entwicklern sichergestellt, dass es zu keinem Deadlock kommen kann, wenn die CPU-Zeit einem anderen Prozess zugewiesen wird.

Ein hochpriorer Prozess kann nun einen niederprioren verdrängen, wenn dieser die stelle might_sleep() erreicht hat. Diese Einstellung ermöglicht es, im Vergleich zur “no forced preemption”, deutlich schnellere Latenzzeiten zu erhalten, da nicht gewartet werden muss, bis ein Prozess mit der Bearbeitung fertig ist.

[1] Serielle Ausführung von Programmen
[2] voluntary - absichtlich

Preemptible kernel: Die Verwendung der Prozessorentechnik SMP[3], die es ermöglicht, dass jeder CPU-Kern jeden Prozess im System bearbeiten kann, verlangt neue Sicherheitsaufgaben im Betriebssystem-Kern. Die Kerne verfügen über den gleichen Speicheradressraum und es ist sicherzustellen, kritische Bereiche so zu schützen, dass sie nur von einem der Kerne zur Zeit bearbeitet werden.

Dieses wird mittels Spinlocks (vgl. Yaghmour u. a., 2008) realisiert, die im Prinzip ein aktives Warten darstellen. In Mehr-Kernsystemen sind Spinlocks unabdingbar, da die Threads parallel auf mehreren Kernen ausgeführt werden können. Ist ein Bereich von einem Spinlock geschützt und ein CPU-Kern will diesen Bereich betreten, so muss er warten, bis dieser freigegeben wird. Die "preemptible kernel"-Einstellung erlaubt es, Prozesse in jedem Bereich zu unterbrechen, außer sie warten momentan in einem Spinlock. Spinlocks in Ein-Kernsystemen haben keine Auswirkungen, da es keine echte Parallelität gibt.

Diese Möglichkeit reduziert die Reaktionszeit des Systems erheblich, jedoch hat es noch immer keine harte Echtzeiteigenschaft, da Spinlocks sehr häufig im Kernel vorhanden sind, um konkurrierende Tasks in kritischen Abschnitten zu synchronisieren.

Complete preemption: Diese Möglichkeit ist erst auswählbar, nachdem der RT-Patch von Ingo Molnar verwendet wurde. Es ist die essentielle Eigenschaft, Linux harte Echtzeitfähigkeiten zu geben.

Aufbauend auf der Fähigkeit der "preemptible Kernel"-Option werden hier die Spinlocks in Mutexe[4] umgewandelt. Ein Prozess, der eine kritische Sektion betreten möchte, wartet nicht aktiv, sondern geht in den Sleep-Modus über. Dieses Verhalten ermöglicht es, den Kernel vollständig unterbrechbar zu gestalten und die Reaktionszeiten weiter zu verringern. Nun ist es möglich, wartende CPU-Kerne andere Aufgaben zu übergeben.

Dieses Verhalten kann allerdings nicht ohne Zugeständnisse an die Geschwindigkeit des Schedulings realisiert werden. Während kurze, kritische Bereiche, die mittels Spinlocks geschützt sind, schneller reagieren, sind die mit Mutexe langsamer, da hier die Task "aufgeweckt" werden muss.

Trotzdem ist die Verwendung der Mutexe sinnvoller, da in Echtzeitbetriebssystemen auf zeitlichen Determinismus mehr Wert gelegt wird, als auf hohen Datendurchsatz.

Der RT-Patch erweitert den Standard Linux-Kernel und geht in Sachen harter Echtzeiteigenschaften in die richtige Richtung. Allerdings muss von einem Einsatz in Bereichen, in denen Lebensgefahr droht, abgeraten werden. Durch die hohe Komplexität im Betriebssystemkern

[3] Symmetric multiprocessing
[4] Mutex - Mutable Exclusion - wechselseitiger Ausschluss

kann nicht hundertprozentig ausgeschlossen werden, dass es zu keiner Verletzung einer kritischen Deadline kommt. Für die später eingesetzte Mess- und Demonstrationssoftware ist diese Betriebssystemerweiterung verwendbar, da durch die Erweiterung genauere Messergebnisse zu erwarten sind.

Durch die mögliche Verwendung der Standard Treiber wird die Handhabung eines RT-Betriebssystems vereinfacht. Um sicherzustellen, dass keine Verzögerungen innerhalb der Hardwaretreiber auftreten, muss ein versierter Kernel-Programmierer den Treiber auf große Bereiche überprüfen, in denen Interrupts abgeschaltet werden. Ist es sichergestellt, kann dem Einsatz zugestimmt werden (vgl. Yaghmour u. a., 2008) .

Alternative Echtzeit-Betriebssysteme

Außer dem quelloffenen Linux gibt es eine Vielzahl anderer echtzeitfähiger Betriebssysteme. Die bekanntesten sind nachfolgend aufgelistet.

- QNX (vgl. www.qnx.com, 2010)
  Das Einsatzgebiet von QNX liegt in Echtzeit-Orientierten-Anwendungen im Multimedia Bereich. Der Einsatz in Bereichen in denen Menschen zu schaden kommen könnten ist zu vermeiden, da auch hier die hohe Komplexität des Betriebssystems dazu führen kann, das kritische Deadlines nicht eingehalten werden.
- Windows CE (vgl. www.microsoft.com, 2010)
  Hauptsächlich wird Windows CE im Embedded-System-Bereich eingesetzt. Vor allem mobile Geräte bauen auf dieser Plattform auf.
- OSEK-OS (vgl. www.osek-vdx.org, 2010)
  OSEK ist ein Betriebssystem, das von einem Konsortium aus Automobilherstellern standardisiert wird, ist speziell für Anwendungen im Automotiv-Bereich ausgerichtet und muss daher harte Echtzeiteigenschaften aufweisen. Es wird genau für den Einsatzbereich konfiguriert und zusammen mit der Anwendung kompiliert und gelinkt.

2.3 Ethernet

Zum Schluss des Kapitels wird auf das momentan am weitesten verbreitete Kommunikationsprotokoll im LAN[5] eingegangen. Ethernet in seiner Ursprungsform wurde 1976 von dem Amerikaner Bob Metclaf und David Blogg bei Xerox PARC entwickelt und basiert auf dem

[5] Local Area Network

ALOHANET-Protokoll, das auf Hawaii entwickelt wurde, um die verschiedenen Inseln miteinander zu vernetzen. Ethernet unterliegt seitdem einer ständigen Weiterentwicklung und hat mit dem ursprünglichen Ethernet nicht mehr viel gemeinsam (vgl. Tanenbaum, 2003, S.84 - 85) .

Zuerst werden die grundlegenden Eigenschaften von Ethernet besprochen und anschließend die Entwicklung von Ethernet mit weichen Echtzeiteigenschaften dargestellt.

2.3.1 Eigenschaften

Ethernet definiert die elektrische Übertragung von Signalen und dessen Paketformate und ist im OSI-Schichtenmodell der ersten (Bit-Übertragung) und der zweiten Schicht (Sicherung) zugeordnet. Die Abbildung 2.6 stellt den typischen Aufbau des Ethernet-Frames dar.

Ein Frame beginnt mit einer acht Byte langen Preamble, die dafür sorgt, dass sich die Netzwerkgeräte auf den ankommenden Frame synchronisieren, da keine separate Taktleitung vorhanden ist. Dieses wird mittels einer sieben Byte langen alternierenden Bitfolge von „10101010“ realisiert. Das letzte Byte der Preamble signalisiert der Netzwerkkarte den Start eines Frames und wird SOF (Start-Of-Frame) genannt. Die Bitfolge des SOF unterscheidet sich zur Preamble in den letzten beiden Bits und sieht folgendermaßen aus: „10101011“.

Nachfolgend im Frame stehen die Ziel- und Quell-Adresse, die jeweils eine Länge von sechs Byte besitzen. Zusammen mit dem zwei Byte großen Typ-Feld, oder nach IEEE 802.3 dem Längen-Feld, bilden sie den Ethernet-Header.

Anschließend beginnt das bis zu 1500 Byte große Datenfeld, auch Payload genannt, das die eigentlichen Daten enthält. Dieses Feld ist nie kleiner als 46 Byte und wird ggf. aufgefüllt. Der Payload wird von höheren Protokollen, wie IP genutzt um ihre Daten zu transportieren.

Der Frame wird mit einer vier Byte langen CRC-Prüfsumme abgeschlossen, die dafür sorgt, dass Fehler bei der Übertragung (z.B. wenn ein Bit kippt) erkannt werden.

Somit hat ein Ethernet-Frame eine Länge zwischen 64 und 1518 Byte, da die Preamble nicht zum Frame zugezählt wird.

Ethernet fungiert in dieser Hinsicht wie ein Briefumschlag, der in lokalen Netzen dafür sorgt, dass Nachrichten an die richtigen Empfänger gesendet werden.

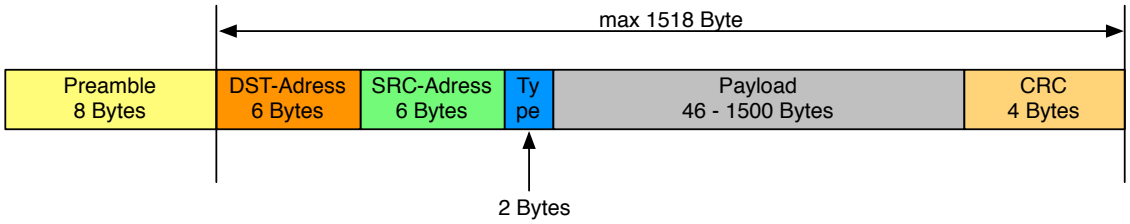

Abbildung 2.6: *Ethernet-Frame (vgl. Kurose und Ross, 2008)*

2.3.2 Der Weg zu weichen Echtzeiteigenschaften

ALOHANET

Wie bereits erwähnt, basiert Ethernet auf dem ALOHANET-Protokoll. Es besitzt einen Mechanismus, um wiederholte Kollisionen zu vermeiden. Das Protokoll arbeitet folgendermaßen:

1. wenn eine Nachricht vorhanden, so sende sie und
2. wenn eine Kollision aufgetreten ist, so wiederhole die Übertragung später.

Dieses Verfahren ermöglicht eine Datenkommunikation ohne zentralen Koordinator, hat allerdings keinen deterministischen Medienzugriff, da die verwendete Strategie zur Bestimmung des Zeitpunktes zum erneuten Senden zufällig ausgewählt wird. Außerdem treten Kollisionen sehr häufig auf, da kein Überblick vorhanden ist, ob ein anderer Teilnehmer das Medium belegt.

Shared-Ethernet / Half-Duplex

Mit der Einführung von Ethernet 1976 wurde dieses geändert. CSMA/CD[6] (vgl. Kurose und Ross, 2008) hat die Eigenschaft, dass, bevor eine Übertragung gestartet wird, überprüft wird, ob der Medienzugriff erfolgen darf. Ein Zugriff ist nur erlaubt, wenn das Medium nicht von einem anderen Teilnehmer belegt ist.

Datenkollisionen können mit dieser Technik allerdings nicht unterdrückt werden, da durch Signallaufzeiten mehrere Sender gleichzeitig ein freies Medium vorfinden können.

[6] Carrier Sense Multiple Access / Collision Detection

Der Begriff „Shared-Ethernet“ leitet sich von dem gemeinsam genutzten Medium ab, an dem jeder Teilnehmer angeschlossen ist. Zu Beginn der Bürokommunikation wurde eine Bustopologie als Vernetzung verwendet. Die Abbildung 2.7 stellt die verschiedenen Ethernet-Netztopologien dar.

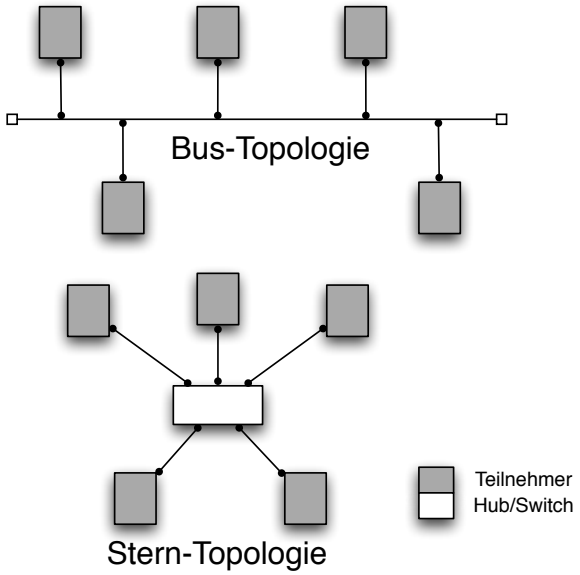

Abbildung 2.7: *Ethernet-Netztopolgien im Vergleich*

Die Verlegetechnik als Bus hat allerdings einen großen Nachteil, da eine Beschädigung des Kabels das gesamte Netz beeinflussen kann. Eine entscheidene Verbesserung wurde zunächst mit der Verwendung eines Ethernet-Hubs erreicht. Die Struktur wurde von Bus auf Stern umgestellt, sodass es einen zentralen Knotenpunkt gibt, an dem die Teilnehmer angeschlossen sind.

Da jedoch ein Ethernet-Hub nichts anderes macht als Signale zu verstärken und auf alle angeschlossenen Ports zu leiten, konnte das Kollisionsproblem nicht behoben werden. Ein Hub ist intern ein logischer Bus und somit gilt weiterhin ein Zugriff auf ein gemeinsames Medium, das keinen deterministischen Medienzugriff erlaubt.

Switched Ethernet / Full Duplex

Eine weitere Verbesserung erfuhr das Ethernet durch den Einsatz von Switches. Die vom Ethernet-Hub bekannte sternförmige Netzstruktur bleibt erhalten und wird durch die Eigenschaften eines Switches weiter verbessert.

Switches arbeiten auf Layer zwei im ISO/OSI-Referenz-Modell und können daher Ethernet-Frames anhand der Adresse analysieren. Signale werden nicht mehr nur aufbereitet, sondern die Pakete an einem bestimmten Port geleitet. Jeder Port hat eine separate Kollisionsdomäne und Zusammenstöße treten nicht mehr auf. Switched-Ethernet ermöglicht die „Full-Duplex“ - Übertragung, da gleichzeitig über die Puffer gesendet und empfangen werden kann. Jeder Port verfügt über einen eigenen Empfangs- und Sende-Puffer, der mit dem Speicher des Switches verbunden ist, um Nachrichten mittels der Forwarding-Tabelle auf andere Ports zu kopieren. Die Abbildung 2.8 stellt den Unterschied zwischen einem Ethernet-Hub und einem Switch dar.

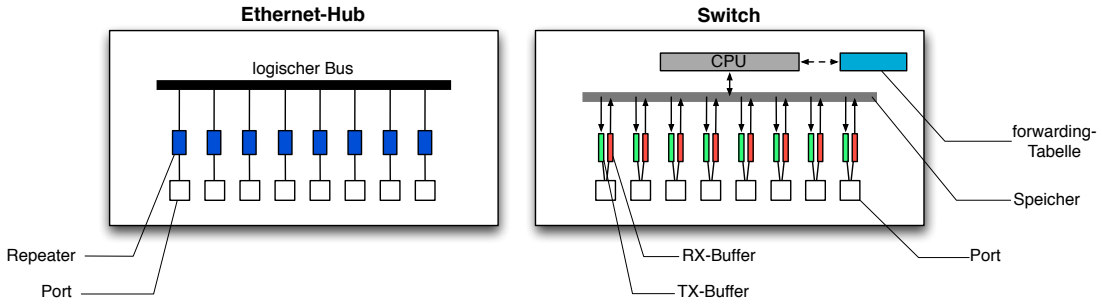

Abbildung 2.8: *Vergleich Ethernet-Hub und Switch (vgl. GE Fanuc Embedded Systems, 2007)*

Im Switched-Ethernet wird jede Nachricht bei der Übertragung gleich behandelt. Man spricht daher von einer Best-Effort-Übertragung. Für weiche Echtzeit ist also auch dieses Verhalten nicht ausreichend, da nicht zwischen Nachrichtenarten mit verschiedenen Protokollen unterschieden werden kann.

Switched-Ethernet mit Priorität

Im IEEE-802.1Q-Standard gibt es einen Ansatz, eine Priorisierung der Nachrichten zu ermöglichen (vgl. Jeffree, 2006) . Der 802.1Q-Standard dient zur Erstellung von Virtual Local Area Networks (VLANs). Oft ist es aus administrativer Sicht sinnvoll, große LANs in kleinere VLANs aufzuteilen, sodass Teilnehmer aus einem VLAN unerreichbar für Teilnehmer aus einem anderen VLAN sind.

Der Standard Ethernet-Frame wird mit einem zusätzlichen zwei Byte großen, VLAN-Tag-Feld erweitert. Es dient zu Identifizierung von VLANs und ggf. auch zur Priorisierung solcher. Die nachfolgende Abbildung 2.9 gibt einen Überblick eines V-LAN-Frames. Es ist ersichtlich, dass sich die Gesamtlänge um vier Byte verlängert hat. Moderne Netzwerkkarten sind in

der Lage, beide Arten von Frames zu verarbeiten. Erkannt werden diese Frames anhand der statischen VLAN-Kennung (0x8100), die ansonsten das Typ- oder Längen-Feld darstellen.

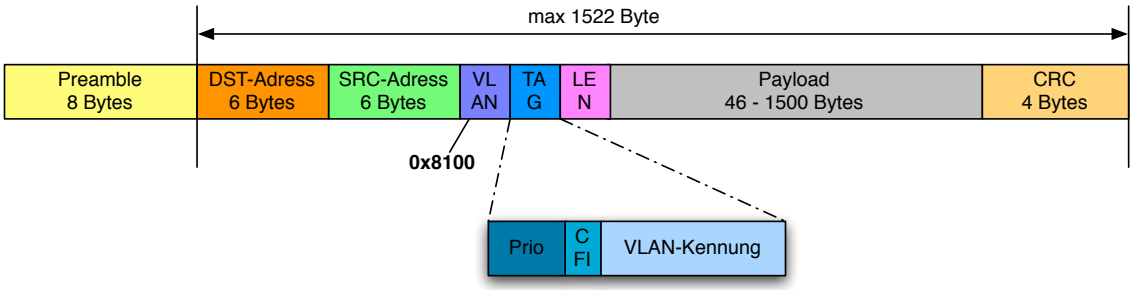

Abbildung 2.9: *V-LAN-Frame (vgl. Tanenbaum, 2003)*

2.3.3 Resümee weicher Echtzeiteigeschaften

Einen Vergleich der Roundtrip-Zeit und des Jitters zwischen Shared-Ethernet, Switched-Ethernet und Switched-Ethernet mit der 802.1Q-Erweiterung findet man in einer Konferenzausarbeitung der Universität Rio Grande de Norte (vgl. de M. Valentim u. a., 2008) . Aus dieser Arbeit ist deutlich ersichtlich, dass eine Priorisierung der Daten eine erhebliche Auswirkung auf den Datentransfer hat. Es wurde erreicht, dass der Roundtrip und Jitter des hochprioren Datenverkehrs bei steigender Daten-Belastung konstant bleibt. Für zeitkritische Systeme ein wichtiges Merkmal, da hier der Jitter gering und der Roundtrip konstant gehalten werden muss.

Ein deterministischer Medienzugriff ist auch mit dieser Ethernet-Variante nicht möglich, da Nachrichten mit gleicher Priorität gleich behandelt werden und so eine Best-Effort-Übertragung vorliegt.

Im nächsten Kapitel wird Time-Triggered Ethernet näher erläutert, das zusätzlich zur Priorisierung von Nachrichten auch einen deterministischen Medienzugriff zulässt.

3 Time-Triggered-Ethernet

Die Grundlagen von Echtzeit und Ethernet sind im letzten Kapitel besprochen worden. Es werden zuerst alternative echtzeit Protokolle auf Ethernetbasis vorgestellt. Anschließend wird TTEthernet (vgl. Steiner, 2008) im einzelnen vorgestellt und die für die Arbeit relevanten Eigenschaften erläutert.

3.1 Echtzeit und Ethernet

Ethernet ist in seiner Grundform durch die in den Grundlagen beschrieben Eigenschaften nicht echtzeitfähig. Die Übertragungsform wird als Event-basierter Verkehr realisiert, sodass kein deterministischer Medienzugriff vorliegt.

TTEthernet wurde entwickelt, um den steigenden Bandbreitenbedarf in eingebetteten Systemen zu kompensieren, indem auf etablierte Konzepte zurückgegriffen wurde. Die Verwendung von Fahrerassistenzsystemen wie Abstands-, Spur- und Schilderkennung und die in Zukunft verfügbare Technologie X-by-Wire (Beschleunigen/Bremsen/Steuern über Kabel) führen zu einem steigenden Datentransferaufkommen, welches mit den momentan verfügbaren Kommunikationsprotokollen nur unter großen Aufwand zu bewältigen ist. Es werden eine Vielzahl von Systemen eingesetzt, die unterschiedliche Protokolle verwenden und für die jeweiligen Aufgaben konzipiert sind. Es gibt Echtzeitbussysteme und Systeme für nicht zeitkritischen Verkehr, wodurch eine heterogene Gesamttopologie entsteht.

Etablierte Konzepte bedeutet in diesem Fall, dass mit Ethernet das meist verbreiteste Protokoll für lokale Netze als Basis verwendet und mit einer Systemzykluszeit á la FlexRay erweitert wird. Beide Protokolle entsprechen dem aktuellen Stand der Technik, sodass auf ein großes Spektrum an Know-How zugegriffen werden kann.

TTEthernet vereint die Möglichkeit, zeitkritischen und Best-Effort-Verkehr mit einer hohen Bandbreite über das gleiche Medium zu übertragen.

3.1.1 Derivate echtzeitfähiger Ethernet Implementierungen

TTEthernet ist nicht die einzige Echtzeiterweiterung für Ethernet. Der Vollständigkeit halber wird in diesem Abschnitt kurz auf das AFDX-Protokoll eingegangen und eine Liste von weiteren echtzeitfähigen Ethernet-Protokollen gezeigt.

Avionics Full Duplex Switched Ethernet

Avionics Full Duplex Switched Ethernet (AFDX) ist ein von Airbus vorgestelltes Protokoll, dass eingesetzt wird, um eine höhere Bandbreite innerhalb der Flugzeugelektronik zu erhalten. Es basiert auf den Spezifikationen von ARINC 664 (Aeronautical Radio Incorporated) und IEEE 802.3.

Durch den Einsatz von Fly-By-Wire (vgl. Kluxmann und Malik, 2007) im Airbus A320 seit 1987 erhöhte sich der Datentransfer im Passagierflugbereich erheblich. Die Steuersignale werden dabei komplett elektronisch, ohne mechanische Verbindung vom Steuerknüppel zu den Rudern geleitet. Diese Technik erlaubt es, die vom Piloten ausgeführten Steuerbefehle auf Fehler zu überprüfen, sodass z.B. ein „Überziehen“ (das Flugzeug erleidet durch Geschwindigkeitsmangel einen Abriss der Luftströmung an den Tragflächen und verlässt den sicheren Zustand) nicht mehr möglich ist und damit ein Plus an Sicherheit erreicht wird. Zusätzlich lässt sich die Verwendung von fehleranfälliger und schwerer Mechanik vermeiden.

Die Datenübertragung bei ADFX wird, anders als bei anderen echtzeitfähigen Übertragungsprotokollen, über eine Limitierung der Bandbreite erreicht. Ein Netzwerkknoten kann mehrere virtuelle Links besitzen, die über einen gemeinsamen Ethernetanschluss mit anderen Systemen kommunizieren. Die einzelnen virtuellen Links teilen sich die gesamte Bandbreite des Ethernetanschlusses und so muss sicher gestellt werden, das jeder Link eine garantierte Bandbreite zur Verfügung hat, die zur Designzeit bekannt sein muss.

Um die Limitierung zu erhalten, werden so genannte BAGs (Bandwidth Allocation Gap) (vgl. GE Fanuc Embedded Systems, 2007) verwendet, die eine Lücke im Medienzugriff darstellen und somit die virtuellen Links von der Datenübertragung trennen.

AFDX wird bereits erfolgreich im neuen A380 von Airbus eingesetzt. Andere Modelle wie A400m oder die Boeing 787 Dreamliner befinden sich in der Testphase und verwenden ebenfalls dieses System (vgl. AIM GmbH) .

Weitere Protokolle

Einen Überblick derzeitig verfügbarer echtzeit Ethernetprotokolle kann auf der Homepage der FH-Reutlingen (vgl. Hochschule Reutlingen) eingesehen werden. Viele der bereits eingesetzten Varianten finden in der Prozess- und Automatisierungstechnik Verwendung. Hier einige Auszüge:

- PROFInet IO (vgl. PROFIBUS & PROFINET International)
- Powerlink (vgl. Ethernet POWERLINK Standardization Group)
- EtherCAT (vgl. Ethercat Technology Group)

3.2 Priorisierung, Nachrichtenklassen und -Erkennung im TTEthernet

Eine Priorisierung von Nachrichten ist im Standard Ethernet ohne 802.1Q Erweiterung nicht vorgesehen. Jede Nachricht wird auf ihrem Weg vom Sender zum Empfänger gleich behandelt, sodass es vorkommen kann, dass wichtige Nachrichten u.U. verzögert werden. Im Switched-Ethernet entstehen solche Verzögerungen im Switch, wenn ein Port zur Übertragungszeit mit einer anderen Nachricht belegt ist, da dieser nach dem FIFO-Prinzip arbeitet.

3.2.1 Nachrichtenklassen

TTEthernet definiert daher drei Nachrichtenklassen, die unterschiedliche Prioritäten aufweisen und somit das Problem der Best-Effort-Übertragung beseitigen.

Time-Triggered-Traffic (TT) wird ausschließlich für zeitkritischen Datenverkehr verwendet, der hohe Anforderungen in kurzen Latenzzeiten, niedrigen Jitter und deterministischen Medienzugriff hat. TT-Frames haben die höchste Priorität in diesem System und können nicht verdrängt werden.

Rate-Constrained-Traffic (RC) ist ein eventbasierter Verkehr, der eine definierte Bandbreite garantiert hat. Nachrichten können im Switch verzögert werden, da sie nicht abhänging von der globalen Systemzeit sind. Diese Nachrichten basieren im Wesentlichen auf dem AFDX-Protokoll-Standard (vgl. GE Fanuc Embedded Systems, 2007) .

Best-Effort-Traffic (BE) entspricht dem Standard Ethernet Verkehr. Es kann nicht sichergestellt werden, wann und ob eine Nachricht übertragen wird. Nachrichten dieses Typs nutzen die restliche Bandbreite im System und besitzen die niedrigste Priorität.

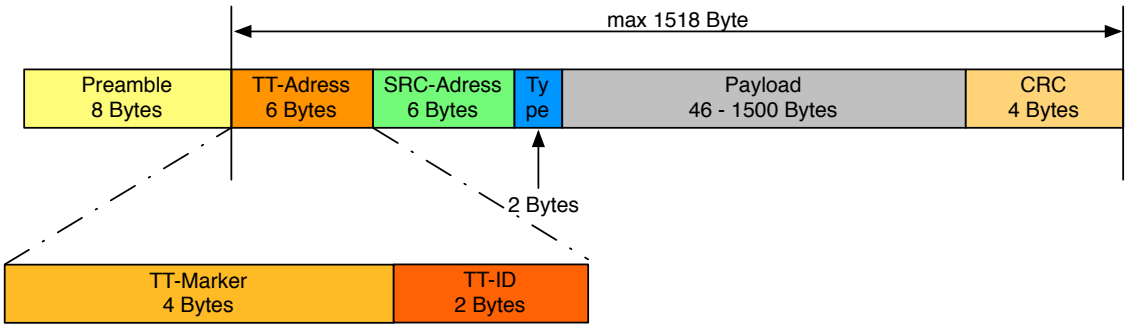

Abbildung 3.1: *TT-Ethernet-Frame*

3.2.2 Nachrichtenerkennung

TTEthernet basiert auf Standard-Ethernet ohne 802.1Q-Erweiterung im Ethernetframe. Die Priorität einer Nachricht kann daher nicht an einem gesonderten Feld erkannt werden. Um zwischen hoch und niederprioren Nachrichten zu unterscheiden, wird der Standard-Ethernetframe im Zieladress-Feld anders interpretiert (siehe Abbildung 3.1), welches normalerweise den Empfänger repräsentiert.

Das Zieladress-Feld hat einen 32-Bit großen TT-Marker, mit dem zeitkritische Nachrichten gekennzeichnet werden und ein 12-Bit großes TT-ID-Feld, welches die Nachricht an sich identifiziert und definiert, ob es sich um einen TT- oder RC-Frame handelt. So ist es möglich, innerhalb eines TT-System bis zu 4096 verschiedene zeitkritische Nachrichten zu verwenden. Die Markierung muss auf allen im System befindlichen Teilnehmern gleich sein.

Bei einer ankommenden Nachricht im TT-Switch wird das Zielfeld mit der Bitmaske 0xFFFFFFFF0000 maskiert (siehe Tabelle 3.1). Entspricht das Ergebnis dem konfigurierten TT-Marker, handelt es sich um einen zeitkritischen Frame. Anhand der TT-ID wird dann im Switch entschieden, ob es sich um einen TT-Frame, der im Schedule gesendet werden muss, oder um einen RC-Frame handelt.

Weiterhin unterscheiden sich Standard-Ethernetframes von zeitkritischen Frames im Typ-Feld des Frames. IEEE hat für zeitkritischen Traffic ein spezielles Bitmuster 0x88D7 reserviert (vgl. IEEE, 2010) .

	TT-Marker				TT-ID	
Ziel	0x03	0x04	0x05	0x06	0x00	0x64
Maske	0xFF	0xFF	0xFF	0xFF	0x00	0x00
Ergebnis	0x03	0x04	0x05	0x06	0x00	0x00

Tabelle 3.1: *Erkennung von zeitkritischem Verkehr*

Der Unterschied zwischen TT- und RC-Nachrichten findet sich in der Konfiguration im TT-Switch. Dieser scheduled TT-Frames in feste Zeitschlitze, während RC-Nachrichten über BAGs (siehe AFDX) gesteuert werden. Weiterhin bietet TT-Ethernet die Möglichkeit, RC-Nachrichten bis zu sieben unterschiedliche Prioritäten zu vergeben, sodass eine zusätzliche Priorisierung der Nachrichten vorgenommen wird.

BE-Nachrichten nutzen den normalen Ethernet-Frame ohne Modifikation.

Zeitkritische Ethernetframes werden im TTEthernet nicht an bekannte Teilnehmer (Adressen) im Netz gesendet, sondern werden als Nachrichtenart im Netz verteilt. Die Zustellung erfolgt anhand statischer Routen im Switch, sodass dieser anhand des Typus und der TT-ID entscheidet, über welche Ports die Nachricht geschickt wird. TTEthernet erlaubt es weiterhin, feste Routen auch für BE-Traffic zu verwenden, wenn im voraus klar ist, wie der Verkehr stattfindet.

Nicht bekannte Nachrichtentypen werden immer als BE-Traffic interpretiert und haben damit die geringste Priorität. Der TT-Switch arbeitet dann wie ein normaler Switch und erlernt die Routen.

3.2.3 Scheduling und Datenfluss im TTEthernet

TT-Nachrichten sind die einzigen Nachrichten, die einem festen Schedule unterliegen. Alle anderen Nachrichten werden gesendet, sobald genügend Bandbreite auf dem Medium vorhanden ist.

RC-Nachrichten mit gleicher Priorität werden nach dem FIFO-Prinzip behandelt. Der Frame, der zu erst ankommt wird zu erst bearbeitet.

Vom Systemarchitekten muss zur Designzeit allerdings sichergestellt werden, dass nie zwei TT-Nachrichten zu gleichen Zeit gesendet werden.

Preemption

Ein weiteres Merkmal von TTEthernet ist die Möglichkeit, niederpriore Nachrichten zu verdrängt (zu sehen in Abbildung 3.2). Befindet sich eine RC- oder BE-Nachricht im Switch beim Senden und es wird eine TT-Nachricht erwartet, so wird das Senden der RC oder BE-Nachricht sofort abgebrochen und die TT-Nachricht kann gesendet werden. Somit wird sichergestellt, dass der zeitkritische Datenverkehr ohne Verzögerungen übermittelt wird.

Dieses Vorgehen hat jedoch den Nachteil einer schlechteren Medienauslastung. Informationen einer Nachricht werden u.U. doppelt gesendet. Außerdem muss sichergestellt werden, dass diese Nachricht beim Sender als nicht komplett angesehen, bzw. als unbrauchbar gekennzeichnet werden.

Um dieses Problem zu lösen, gibt es eine Idee der TU-Wien (vgl. Mikolasek u. a., 2008) . Diese Arbeit befasst sich mit der Segmentierung von Standard Ethernet-Nachrichten innerhalb eines TT-Systems. Verdrängte Nachrichten im Switch werden mit einem falschen CRC (Cyclic Redundancy Check) versehen, sodass diese als nicht vollständig beim Empfänger gekennzeichnet sind. Nachdem der Versand einer TT-Nachricht vollendet ist, kann der zweite Teil der verdrängten Nachricht gesendet werden, welche den korrekten CRC der gesamten Nachricht enthält.

Im Empfänger werden beide Teile zusammengesetzt und mit dem CRC verglichen. In Simulationen stellte sich heraus, dass die Netzwerklast erheblich sinkt, wenn dieses Verfahren eingesetzt wird.

Das preemptive Verhalten wird momentan jedoch nur als Beta-Variante unterstützt und ist in dieser Form nicht in dieser Bachelorarbeit behandelt.

Zeitliche Blockierung

Ein Weiterer Ansatz zeitkritischen Datenverkehr den Vorrang in der Übertragung zu ermöglichen, ist die zeitliche Blockierung einer niederprioren Nachricht und ist in dieser Form im TT-Switch der Bachelorarbeit unterstützt.

Der Switch „weiß“, wann als nächstes eine TT-Nachricht gesendet werden soll und erkennt an der Länge des RC-/BE-Frames, dass diese nicht mehr gesendet werden kann, ohne den Schedule einer TT-Nachricht zu stören. Diese Nachricht wird solange blockiert, bis genug Zeit vorhanden ist, sie ohne Probleme zu senden.

Ein Nachteil dieser Variante ist, dass Lücken im Schedule entstehen, und somit eventuell Bandbreite verschenkt wird. Weiterhin kann es passieren, dass der Puffer für BE-Nachrichten zu klein ist und somit Nachrichten verworfen werden.

Einen Überblick beider Verfahren ist in der Abbildung 3.2 zu sehen.

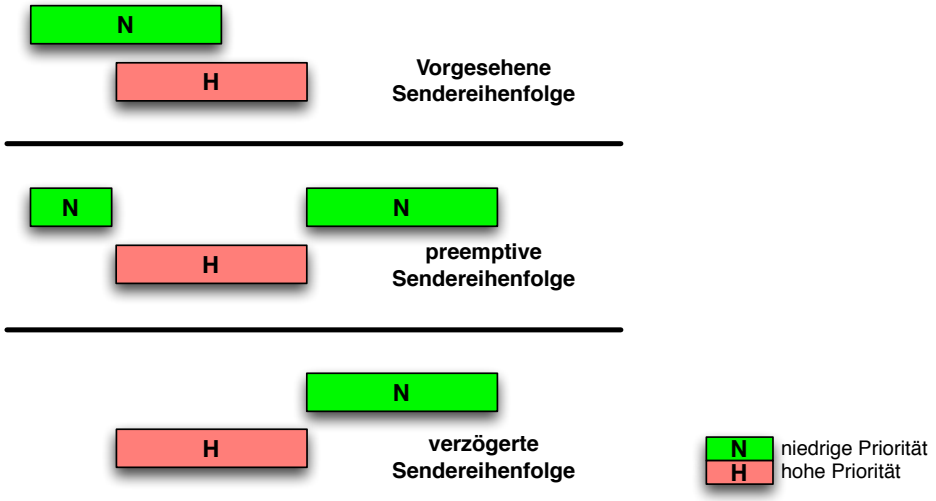

Abbildung 3.2: *Schedulingstrategie des TT-Ethernet (vgl. Steiner u. a., 2009)*

3.2.4 Integration der Frames im Systemschedule

In der Abbildung 3.3 wird gezeigt, wie der Fluss der Nachrichten von zwei Sendern auf einen Empfänger integriert wird. Sender1 sendet im 3ms Takt TT-Messages an den Empfänger, Sender2 sendet im 2ms Takt. Damit es zu keiner gleichzeitigen Übertragung von TT-Nachrichten kommt, ist der 2ms Zyklus von Sender2 zur Zykluszeit von Sender1 verschoben. Es wird zur Designzeit sichergestellt, dass keine gleichzeitige Übertragung von TT-Nachrichten vorliegt. Zusätzlich zu den TT- werden noch RC- und BE-Nachrichten gesendet. Man kann erkennen, dass RC- durch TT-Nachrichten verdrängt werden, sie aber dennoch beim Empfänger ankommen. Die BE-Nachrichten werden mit niedrigster Priorität von allen anderen Nachrichten verdrängt. Ein zuverlässiger Nachrichtenfluss ist nicht erkennbar.

Dieses Verfahren benötigt einen Schedule, der im gesamten System identisch ist. Daraus folgt, dass ein Synchronisationsmechanismus im System vorhanden sein muss. Im nächsten Abschnitt wird die Synchronisation näher erläutert.

3.3 Synchronisation im TTEthernet

Bevor die Synchronisierungsfunktion und deren Ablauf besprochen wird, muss zunächst erklärt werden, warum Synchronisation in verteilten Echtzeit-Systemen überhaupt nötig ist.

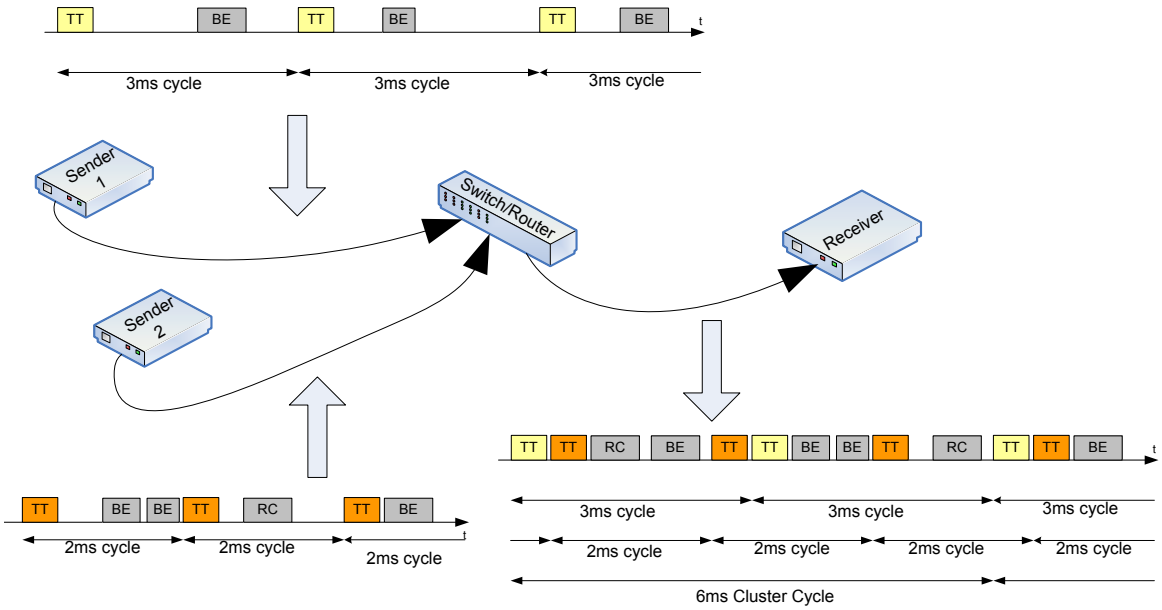

Abbildung 3.3: *Integration der Nachrichten in den Systemschedule (vgl. Steiner, 2008)*

Betrachtet wird der Systemschedule. Jeder Teilnehmer im Netz muss eine globale Sicht der Zeit haben, damit die verschiedenen Aufgaben in der zeitlich korrekten Reihenfolge abgearbeitet werden können. Die eingebauten Taktgeber arbeiten allerdings nie hundertprozentig synchron, da z.B. kleinste Temperaturänderungen das Schwingverhalten eines Quarzes beeinträchtigen kann. Daraus folgt, dass über eine bestimmte Zeitspanne Differenzen in den lokalen Uhren auftreten.

Ein weiterer Grund, warum die Synchronisation nötig ist, beruht auf der Zielsystemimplementierung. In verteilten Systemen ist es so, dass Embedded-Systeme ohne Betriebssystem und gleichzeitig IT-Systeme mit Betriebssystem eingesetzt werden. Die verschiedenen Auflösungen der internen Uhren kann auch eine Verschiebung der lokalen Zeit zur Systemzeit begünstigen.

Die Synchronisation im TTEthernet ist unerlässlich, da durch sie das gesamte System initialisiert wird. Eine Kommunikation auf Echtzeitebene wäre sonst nicht möglich, da sich die lokalen Zeiten beim Systemstart unterscheiden.

3.3.1 Geräte und deren Rolle im Synchronisierungsprozess

Synchronisation bedeutet auch, dass es eine Arbeitsteilung im verteilten System geben muss, damit ein klarer Ablauf stattfinden kann. Eine klare Hierarchie im Prozess wird in der Regel durch eine Master-/Slave-Architektur erreicht.

TTEthernet bietet in dieser Hinsicht eine fehlertolerante Zwei-Wege-Synchronisation, in der die Teilnehmer (Endsysteme) im Netz folgende Rollen übernehmen:

Synchronisation-Master: (Im weiteren Verlauf SM genannt), leiten die Synchronisation ein, in dem sie Synchronisations Frames (Protocol-Control-Frame - PCF siehe Abschnitt 3.3.2) senden.

Compression-Master: (CM), sammeln Synchronisations-Frames, errechnen eine neue Zeit und senden diese an alle Teilnehmer.

Synchronisation-Client: (SC), synchronisieren ihre aktuelle Zeit mit Hilfe der Frames, und leiten diese ggf. weiter.

Der SM hat im Sychronisierungsprozess einen Sonderstatus, da er sowohl Master als auch Client ist. Das heißt, sobald dieser die Synchronisation mittels Senden eines Sychronisations-Frames eingeleitet hat, verhält er sich wie ein Client, indem er ankommende Frames gegebenenfalls weiterleitet und seine eigene Uhr synchronisiert.

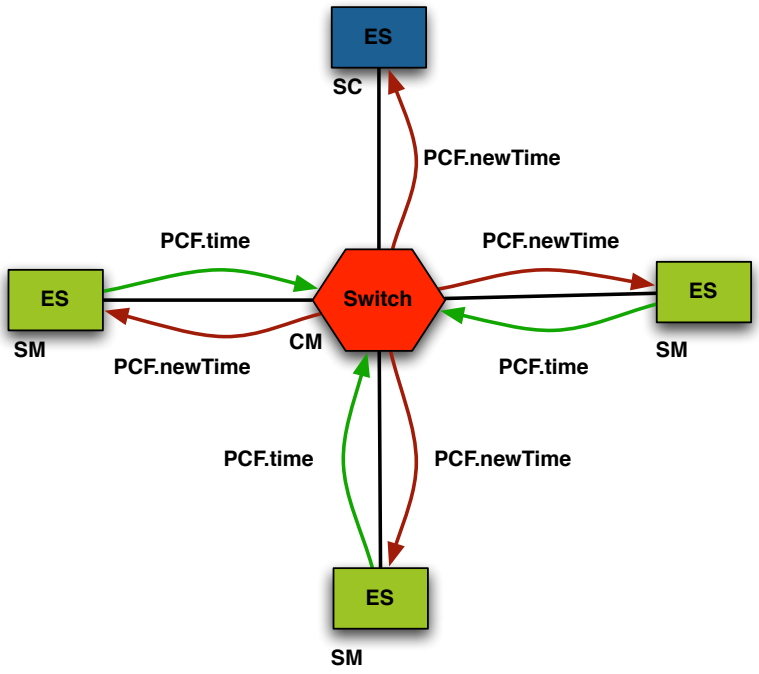

Abbildung 3.4: *Zwei-Wege-Synchronisation*

Im Beispielnetz (siehe Abbildung 3.4) ist erkenntlich, wie die Zwei-Wege-Synchronisation abläuft. Im ersten Schritt senden alle SM ihre Synchronisationsnachrichten zum CM (ersichtlich an den grünen Pfeilen), der daraufhin eine neue Zeit berechnet und als zweiten Schritt die neue Zeit an alle Teilnehmer im Netz sendet (rote Pfeile).

Aufgrund der Fehlertolerenz kann es in einem Time-Triggered Ethernet Netzwerk mehrere SM geben, damit im Fehlerfall eines Gerätes eine Synchronisation weiterhin möglich ist (vgl. Steiner, 2008) .

Das Verhalten eines Endsystems hängt nicht von der Netzstruktur ab, sondern wird zur Modellierungszeit festgelegt. Ein CM muss nicht zwingend im Switch implementiert sein, ist aber oft zu bevorzugen, da er in der Mitte einer Stern-Struktur liegt und somit zu allen anderen Teilnehmern direkt verbunden ist und dadurch nur Signallaufzeiten als Verzögerungen auftreten können.

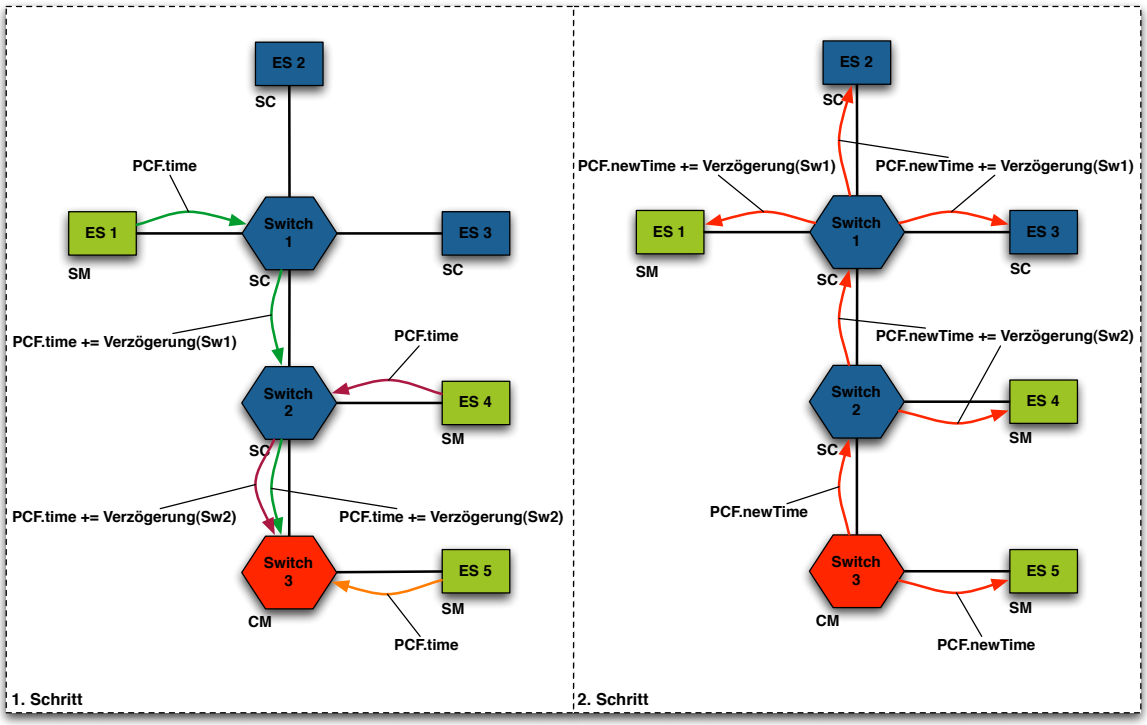

Abbildung 3.5: *TT-Netwerk mit mehreren Switches und Gerätekonfiguration*

In größeren TT-Netzwerken (siehe Abbildung 3.5) mit mehreren Switches kann eine Konfiguration gewählt werden, in der Switch1 und Switch2 als SC fungieren. Dieser Aufbau hat den Vorteil, dass sie selber keine Synchronisations-Frames produzieren, sondern nur weiterleiten, bzw. konsumieren. Durch die Weiterleitung im SC/SM treten Verzögerungen auf, die dem Frame hinzugefügt werden, damit die Zeit weiterhin konsistent zur Gesamtzeit bleibt.

3.3.2 Aufbau der Synchronisationsframes

Für den Synchronisierungsdienst im TTEthernet werden Ethernetframes minimaler Framegröße verwendet, die das Typenfeld auf 0x891d gesetzt haben und werden Protocol Control Frames (PCF) genannt. Sie sind in mehrere Felder unterteilt, die die Synchronisationsdaten enthalten.

Die wichtigsten Felder sind Typ und Zeit. Das Zeitfeld enthält die aktuelle Zeit eines SM oder die neue berechnete Zeit eines CM. Das Typfeld spezifiziert zudem den Inhalt des PCF, da sie auch dazu verwendet werden das gesamte Netzwerk zu initialisieren.

PCFs basieren auf RC-Nachrichten und haben die höchste Priorität dieser Nachrichtenklasse. Somit werden sie nur von TT-Nachrichten verdrängt, deren Zustellung zwingend erforderlich ist um keine kritische Deadline zu verletzen. Diese Verdrängungsverzögerungen werden dem PCF im Zeitfeld hinzugefügt.

3.3.3 Synchronisierungsfunktionen und deren Aufgabe

Nachdem der Ablauf einer Zwei-Wege-Synchronisation dargestellt wurde, muss danach auf zwei essentielle Funktionen der Synchronisation eingegangen werden.

Zum einen dient die Message-Permanence Funktion dazu, die dynamischen Verzögerungen, die bei der Weiterleitung entstehen zu kompensieren. Es folgt daraus, dass die Reihenfolge, in der die Frames beim Empfänger ankommen, korrigiert wird.

Die Message-Compression ist das eigentliche Herzstück der Synchronisation eines Netzes mit mehreren SM. Sie berechnet aus allen ankommenden Frames einen neue Zeit, die die Synchronisierung der Teilnehmer sorgt.

Message-Permanence

TTEthernet definiert einen bestimmten Zeitpunkt im Synchronisationsprozess. Dieser hat die Aufgabe, die Sendereihenfolge der PCFs beim Empfänger zu korrigieren.

In einem TTEthernet-Netzwerk mit mehreren SM, einem CM und dazwischen liegenden SC (siehe Abbildung 3.5) kann es vorkommen, dass die Empfangsreihenfolge von PCF nicht mit der zeitlichen Sendereihenfolge übereinstimmen. Synchronisationsnachrichten basieren auf RC-Nachrichten, die verdrängt werden können.

Die Korrektur benötigt daher bestimmte Zeiten. Jedes Gerät, welches sich im Netz befindet und dynamische Verzögerungen der Nachrichten hervorruft, fügt diese im Zeitfeld in der Synchronisationsnachricht ein. Nachdem ein PCF empfangen wurde, wird von der vorher

berechneten Worst-Case-Verzögerung die dynamische Verzögerung subtrahiert. Die ankommende Nachricht wird um genau diese berechnete Zeit verzögert, bis sie verarbeitet wird. Somit wird die ursprüngliche Sendereihenfolge wieder hergestellt.

In einem TTEthernet-System gibt es zwei verschiedene Arten von Verzögerungen. Dynamische Verzögerungen untergliedern sich dabei in dynamische Sendeverzögerung, dynamische Weiterleitungsverzögerung und dynamische Empfangsverzögerung.

- Eine dynamische Sendeverzögerung tritt im Synchronisations- oder im Compression-Master bei der Generierung von PCF auf, wenn sich ein anderes Datenpaket im Übertragungsprozess befindet und so den Kommunikationsweg blockiert.
- Eine dynamische Weiterleitungsverzögerung tritt in allen beteiligten Synchronisationsgeräten auf, wenn zwischen dem Master und dem Client weitere Geräte liegen. Diese können eine Weiterleitungsverzögerung hervorrufen, indem ein anderes Datenpaket den Kommunikationsweg blockiert. Die Weiterleitung des Protokoll-Control-Frames verzögert sich.
- Eine dynamische Empfangsverzögerung tritt in allen beteiligten Geräten auf, wenn interne Aufgaben den Empfang von Protokoll-Control-Frames verzögern. Es ist die zeitliche Distanz zwischen dem Empfangszeitpunkt des PCF und dem Start der Message-Permanence-Funktion.

Statische Verzögerungen unterteilen sich in äquivalente Bereiche. Diese sind jedoch nicht von anderen Aktionen abhängig und verzögern Pakete immer um eine gerätspezifische Zeitspanne, die zur Designzeit bekannt ist und werden immer dem Frame hinzugefügt.

Message-Compression

Im vorherigen Abschnitt wurde die Message-Permanence-Funktion besprochen. Sie dient dazu, die Reihenfolge der abgeschickten Frames zu korrigieren. In diesem Abschnitt wird die Message-Compression erläutert, die im Synchronisationsprozess eine Durchschnittszeit aus allen im Netz befindlichen SMs errechnet.

Die Message-Compression-Funktion befindet sich im Compression-Master und wird asynchron zur lokalen Zeit ausgeführt. Gestartet wird sie, wenn PCF empfangen werden und nicht, wenn ein bestimmter Zeitpunkt im Zyklus erreicht wird. Sie empfängt und sammelt PCF von allen Synchronisations-Masters im System.

Auch hier ist es möglich, dass Pakete auf ihrem Weg verzögert werden und so zeitliche Differenzen vom Sendezeitpunkt zum Empfangszeitpunkt auftreten. Die Message-Permanence-Funktion wird zu Beginn der MC-Funktion ausgeführt und stellt so sicher, dass die dyna-

mischen Verzögerungen kompensiert werden. Danach kann die eigentliche Ausführung der Message-Compression erfolgen.

Von allen empfangenen PCF, wird die zeitliche Differenz zum ersten empfangen Frame benutzt, um anschließend eine neue Durchschnittszeit zu errechnen.

Synchronisierung der Zeit im Gerät

Nachdem besprochen wurde, wie die neue Zeit mittels Message-Compression-Funktion berechnet wird, muss nun geklärt werden, wie die Zeit auf den jeweiligen Endsystemen synchronisiert wird. Unterschieden wird dabei, ob es sich um den Compression-Master oder um Synchronisation-Master/-Client handelt.

Die Zeit kann als ein lokaler Zähler angesehen werden, der zyklisch von Null bis zum Systemzykluszeit zählt. Gestartet wird die Synchronisation im SM, wenn der Zähler den Wert Null hat. An diesem Zeitpunkt wird ein PCF gesendet.

Die Synchronisation im CM findet unmittelbar nach der Message-Compression statt. Sobald diese ein gültiges Ergebnis produziert hat, wird die Zeit aktualisiert und ein PCF gesendet, der die Master und Clients im System synchronisiert. Wenn ein Master/Client einen Aktualisierungsframe im System empfangen hat, wird zunächst mittels Message-Permance-Funktion die Verzögerung korrigiert und anschließend synchronisiert.

Die Synchronisation ist somit abgeschlossen und es kann sichergestellt werden, dass synchrone Aufgaben garantiert bei jedem Teilnehmer zur richtigen Zeit ausgeführt werden.

Die Funktionsweise von Time-Triggered Ethernet, insbesondere der Synchronisierungsmechnanismus, ist in diesem Kapitel behandelt worden. Aufbauend wird auf das von TTTech ausgelieferte Evaluationssystem eingegangen.

4 Analyse und Implementation

Nachdem die Grundlagen und TTEthernet besprochen wurden, wird in diesem Kapitel auf die Inbetriebnahme der Zeitmessung eingegangen. Zuerst wird das Evaluationssystem im Auslieferungszustand beschrieben und anschließend das Konzept der Zeitmessung erläutert.

4.1 Beschreibung des Evaluationssystems

TTTech liefert an interessierte TTEthernet-Kunden Evaluationssysteme aus, um im Vorfeld Möglichkeiten des Einsatzgebietes von TTEthernet zu recherchieren und Anwendungen zu entwickeln. Angeboten wird momentan ein System mit 100Mbit/s und ein 1Gbit/s-System. Die Systeme beinhalten jeweils einen Switch, mehrere Endsysteme mit vorinstallierter Software und Analyse- und Konfigurations-Werkzeuge.

In dieser Arbeit wird auf das 100Mbit/s-System eingegangen, da hierauf alle Tests und Messungen durchgeführt wurden.

4.1.1 Hardware

Die ausgelieferte Hardware (siehe Abbildung 4.1) beinhaltet neben dem TT-Switch zwei Endsysteme, die jeweils auf einer Intel Atom-Architektur aufbauen und deren CPUs eine Taktfrequenz von 1.6GHz besitzen. Eine Besonderheit ist beim Embedded-PC zu finden, der über zwei Netzwerkinterfaces verfügt.

Der 100MBit/s TT-Switch besitzt acht Ethernet-Ports und unterstützt nur Full-Duplex. Angeschlossene Geräte, die diese Fähigkeit nicht unterstützen (z.B. Ethernet-Hubs), werden nicht erkannt und es kann keine Kommunikation erfolgen. Diese Einschränkung ist unter anderem wichtig bei der Verifikation der Zeitmessung.

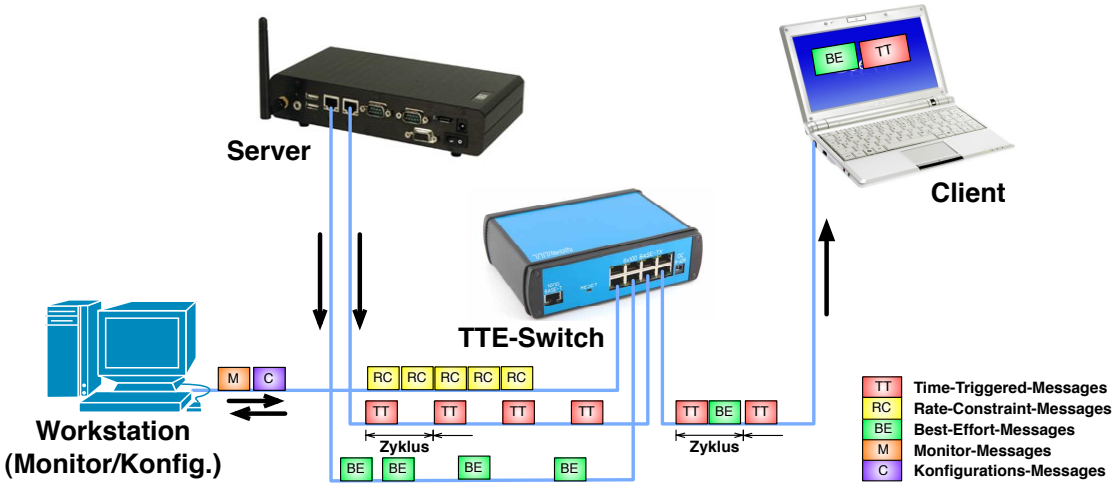

Abbildung 4.1: *Aufbau des Evaluationssystems*

4.1.2 Software

Zu Demonstrationszwecken wird eine Video-Streaming-Anwendung mitgeliefert, die das Verhalten von TTEthernet anschaulich darstellt. Der Embedded-PC fungiert in dieser Applikation als Server, der Laptop dient als Client und stellt das Video dar. Am Ende dieses Abschnittes wird das Verhalten der Anwendung genauer erklärt, da zuerst auf die mitgelieferte Software eingegangen wird.

Die Endsysteme basieren auf einem Ubuntu-Linux und dem Kernel 2.6.24.12 mit RT-Patch. Die Eigenschaften dieses Betriebssystems wurden in den Grundlagen (siehe Abschnitt 2.2.1) erläutert. Auf dem Server ist kein Window-Manager installiert, es steht lediglich eine Konsole zur Konfiguration zur Verfügung.

Auf dem Client sind zusätzlich zu einer grafischen Oberfläche sämtliche Entwicklertools (z.B. der GCC[1], make, etc.) installiert. TTTech liefert zudem den Quellcode für die TTEthernettreiber, der Streaming-Applikation und des Timers mit, sodass eigene Anwendungen, aufbauend auf der Beispiel-Applikation, geschrieben werden können. Der TTEthernet-Protokoll-Stack wird nur als vorkompiliertes Modul mitgeliefert.

Für Überwachungszwecke gibt es ein Werkzeug der Workstation, das Monitordaten (z.B. aktuelle Anzahl verworfener Best-Effort-Frames, allgemeine Fehler oder den Zustand des Switches) vom TTEthernet-Switch auswertet und darstellt. Außerdem ist in diesem Tool ein Paketgenerator integriert. Ein Paketgenerator dient in einem Netzwerk dazu, physikalisch Last auf einem Medium zu erzeugen, unabhängig von einer Anwendung, die auf dem Medium kommuniziert.

[1] GNU C Compiler

Weiterhin wird ein Paket-Sniffer mitgeliefert, um den aktuellen Datenverkehr zu analysieren. Hierbei handelt es sich um keine Eigenentwicklung, sondern um ein bekanntes Werkzeug, Wireshark (vgl. www.wireshark.org, 2010) , welches in der Netzadministration erfolgreich eingesetzt wird. Wireshark unterstützt in der ausgelieferten stabilen Version 1.2.4 das TTEthernet Protokoll, ohne zusätzlicher Installation eines Plug-Ins.

Um eine neue Konfiguration auf den Switch zu laden, gibt es ein eigenständiges Tool. Dieses lädt ein vorher kompiliertes HEX-File auf den Switch und überprüft die Konfiguration auf Fehler. Eine Beispielkonfiguration, angepasst an die Streaming-Anwendung, liefert TTTech mit. Die Konfigurationsdatei ist als Python v.2.5-Skript geschrieben und kann so leicht der eigenen Konfiguration angepasst werden.

Verhalten der Beispielanwendung

Die Beispielanwendung (siehe Abbildung 4.1) ist als Videostreaming-Applikation implementiert. Dabei sendet der Server das gleiche Video jeweils als Best-Effort- (BE) und Time-Triggered-(TT) Nachricht. Der TT-Switch vereinigt beide Datenströme auf eine Verbindung zum Client, dieser stellt die zwei identischen Videos dar.

Wird der mitgelieferte Paketgenerator so konfiguriert, dass dieser Rate-Constraint- (RC) Nachrichten erzeugt, so werden die BE-Nachrichten aufgrund der Store-and-Forward-Architektur im Switch verworfen, wenn nicht genügend Speicher vorhanden ist. Die Folge ist, dass das BE-Video ruckelt oder stoppt, je nach Belastung von RC-Nachrichten.

Dieses Verhalten demonstriert sehr anschaulich, welche Auswirkungen die verschiedenen Nachrichtenprioritäten auf ein TTEthernet-Netzwerk haben. Geht man davon aus, dass statt des Videos Steuerbefehle versendet werden, so ist erkenntlich, welche Nachrichtenart zu bevorzugen ist.

4.1.3 Aufbau und Beschreibung der Software

Die Streaming-Anwendung ist komplett in C geschrieben, deren Quelldateien von TTTech mitgeliefert werden. Aufbauend auf diesem Code ist es dem Entwickler möglich, eigene Anwendungen zu schreiben. Einführend wird das Programm beschrieben und anschließend auf die interne Struktur eingegangen. Die interne Struktur ist insofern wichtig zu verstehen, da alle Anwendungen, die TT-Nachrichten versenden, auf diese Struktur zurück greifen.

Die Streaming-Anwendung

Die Demonstrationssoftware von TTTech ist modular aufgebaut, sodass jede Aufgabe von einer Funktion (siehe Abbildung 4.2) abgearbeitet wird.

Unterschieden wird zwischen Serveranwendung und Clientanwendung, die jedoch die gleichen grundlegenden Kommunikations- und Programmfunktionen besitzen. Die Anwendungen bekommen vom System eine Echtzeit-Priorität zugewiesen und werden vom Scheduler FIFO (First in First Out) behandelt. Beide Anwendungen arbeiten mit der RT-Priorität von 80, d.H. sie besitzen eine höhere Priorität als beispielsweise Hard-IRQ-Threads, die eine Priorität von 50 aufweisen. Eine Überprüfung mittels des Befehls:

```
1 root@demo1:~#ps ax -o -e pid,rtprio,comm
```

gibt Aufschluss darüber, wie die Prioritäten im System verteilt sind.

Der Server enkodiert ein Bild und sendet es anschließend über die zwei Netzwerkkarten als BE- und TT-Nachricht an den Client. Die Kommunikation wird mittels Raw-Sockets realisiert, die Ethernet-Frames senden, ohne ein höheres Protokoll (z.B. IP/TCP/UDP) zu verwenden. Linux erlaubt die Verwendung von Raw-Sockets nur als Superuser, sodass die Anwendungen mit root-Rechten ausgeführt werden müssen.

Im Client werden die Bilder empfangen, dekodiert und anschließend dargestellt. Da der Client nur eine physikalische Netzwerkkarte hat, ermöglicht der TT-Protokoll-Layer, zwei logische Netzwerkinterfaces zu betreiben. Auf die Implementierung wird später genauer eingegangen (siehe nächsten Abschnitt 4.1.4). Das Programm wird in zwei Instanzen ausgeführt, um eine Darstellung der zwei Videos zu ermöglichen. Durch Parameterübergabe wird entschieden, welches Interface an den Socket gebunden wird.

4.1.4 Aufbau der TT-Software und Treiber

Einen Überblick über den Aufbau der Software von TTTech liefert die Abbildung 4.2. Die Implementierung des TTE-Protokoll-Layers ermöglicht es, Userspace-Anwendungen über das Linux-Socket-API kommunizieren zu lassen. Das hat den Vorteil, dass Anwendungen einfach zu entwickeln und auch für nicht TTEthernet-Spezialisten relativ simpel zu verstehen sind.

Das Protokoll-Layer-Modul erzeugt ein oder mehrere Linux-Netzwerkinterfaces, abhängig von der Architektur und Anzahl der physikalischen Netzwerkkarten. Diese können wie gewohnt über Linux konfiguriert werden, sodass es möglich ist, IP und höhere Protokolle über

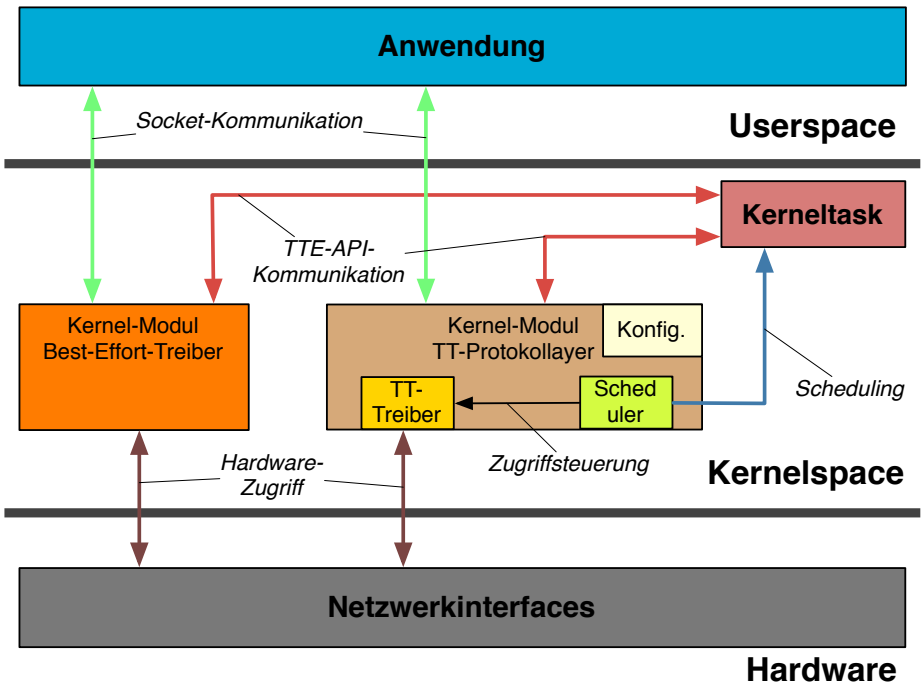

Abbildung 4.2: *Überblick der Software-Struktur*

ein TT-Interface zu verwenden. Das Modul übernimmt zusätzlich den Zugriff auf die Hardware über einen angepassten Treiber, da jede Netzwerkkarte herstellerspezifische Einstellmöglichkeiten besitzt und sich zusätzlich in der Architektur unterscheiden kann.

Bei dem Best-Effort-Treiber-Modul handelt es sich um das Standard Linux-Treiber-Modul für die verwendete Netzwerkkarte und übernimmt damit die weiterhin funktionierende Interrupt gesteuerte Übertragung. Der Unterschied zwischen dem TT-Treiber und dem Standard Linux-Treiber besteht darin, dass die Interrupts für das Senden und Empfangen ausgeschaltet sind, sodass der Scheduler des Protokolllayers diese Funktion übernimmt.

Userspace-Anwendungen können nicht mit dem Scheduler des Protokoll-Layers synchronisiert werden. Die send()-Funktion ist, wie die receive()-Funktion, als ein blockierender Aufruf implementiert worden. Erst nachdem der Frame gesendet, bzw. empfangen wurde, arbeitet das Programm die nächsten Anweisungen ab. Im Worst-Case-Fall kann es passieren, dass das Programm einen Zyklus verpasst und somit die Deadline verletzt.

Synchronisierte Anwendungen zum TTEthernet-Schedule müssen daher im Kernel-Space ablaufen, da nur hier die Schnittstelle zum Protokoll-Layer-Scheduler benutzbar ist.

TTTech stellt für Kernel-Space Anwendungen ein eigenes Kommunikations-API zur Verfügung. Das TTE-API unterscheidet sich in der grundsätzlichen Handhabung nicht von der

Socket-API. Der Programmierer muss jedoch zusätzlich darauf achten, dass er keine Speicherzugriffsverletzung ermöglicht. Ein fehlerhafter Zugriff führt zum Systemabsturz, da keine Speicherschutzsmechanismen im Kernel-Space vorhanden sind.

Die folgenden Pseudocodeausschnitte stellen die unterschiedliche Handhabung beim Senden und Empfangen eines TT-Frames dar.

Listing 4.1: *Senden und Empfangen über Socket-API*

```
//prepare frame
ethernetframe_t send_frame, received_frame;
send_frame.dst = 0x030405060064;
send_frame.src = 0x020202020202;
send_frame.typ = 0x88E7;
send_frame.data = "Hello World";

//open a raw-socket, bind the filedescriptor, send a message
socket_t socket = socket(interface_name, raw);
bind(socket);
send(socket, send_frame);

//receive a message
recv(socket, received_frame);
```

Listing 4.2: *Senden und Empfangen über TTE-API*

```
tt_buffer_t buffer_send, buffer_received;
//prepare frame
ethernetframe_t send_frame, received_frame;
send_frame.dst = 0x030405060064;
send_frame.src = 0x020202020202;
send_frame.typ = 0x88E7;
send_frame.data = "Hello World";

//get a tt_output_buffer and send message
get_outputbuf(buffer_send);
write_outputbuf(send_frame, buffer_interface);

//get a tt_input_buffer and receive message
get_inputbuf(buffer_received);
read_inputbuf(received_frame, buffer_received);
```

Die Konfiguration des TT-Protokoll-Layers wird in einem separaten File vorgenommen (siehe Abbildung 4.2), dass beim Kompilieren eingebunden wird. Zusätzlich zum Schedule werden

hier die Nachrichten-Typen definiert, die gesendet und empfangen werden. Zu den Parametern zählen außerdem Kernel-Tasks, die dem Scheduler als Funktionspointer übergeben werden. Weiterhin kann die Anzahl der verschiedenen Buffer eingestellt werden, da, abhängig der Anzahl der zusendenden Nachrichten, mehr oder weniger Buffer gebraucht werden. Eine fehlerhafte Konfiguration kann das Protokoll-Layer-Modul zum Absturz bringen und somit das System unbrauchbar machen.

4.2 Zeitmessung

Aufbauend auf dem Wissen des Evaluationssystems wird auf die Zeitmessung eingegangen. Zuerst werden wichtige Metriken der Netzwerk-Messtechnik erläutert, anschließend auf verschiedene Messkonzepte eingegangen und zum Schluss die ausgewählte Methode beschrieben.

4.2.1 Metriken der Netzwerk-Messtechnik

In einem Computernetzwerk gibt es verschiedene Metriken, die Informationen über die Netzbeschaffenheit und -Qualität liefern. In der Regel liefern Betriebssysteme Werkzeuge für die Messung aus. Eine Übersicht der wichtigsten Metriken ist in (Held, 2008, S. 176) zu finden.

In dieser Arbeit wird die Latenz und der Jitter eines TTEthernet-Switches gemessen. Die folgende Tabelle 4.1 beschreibt die beiden Metriken.

Metrik	Beschreibung	Maßeinheit
Latenz (Laufzeit)	Unter Latenz versteht man die Zeit, die ein Frame benötigt, um von einem Sender zum Empfänger zu gelangen	μs
Jitter (Varianz)	Entspricht der maximalen Varianz mehrerer Latenzmessungen	μs

Tabelle 4.1: *Übersicht der zu messenden Netzwerkmetriken*

4.2.2 Problematik der Zeitmessung

Die Betriebssystem-Werkzeuge (z.B. ping) erfüllen Ihren Zweck für Best-Effort-Übertragung im Millisekundenbereich. Diese Programme setzen für ihre Messung auf höhere Protokolle, wie IP und ICMP. Für Echtzeit-Übertragungssysteme und Messungen mit einer Genauigkeit

von einer Mikrosekunde erfüllen diese Tools allerdings nicht die Voraussetzungen, da sie dem Betriebssystem unterliegen. Durch Scheduling, Hardwarezugriff und Speichermanagement gibt es bei dieser Messmethode immer einen Overhead (Aufschlag auf die eigentliche Messung), der nicht vorhersagbar ist. Daraus ergeben sich ein erhöhter Jitter und ungenaue Messergebnisse. Für eine Messung im Millisekundenbereich, die bei einer Best-Effort-Übertragung akzeptabel ist, reicht diese Genauigkeit aus.

Eine Möglichkeit dieses Problem zu „umschiffen“ ist die Verlagerung der Messmethodik so weit wie möglich in Richtung Hardware (siehe Abbildung 4.3), sodass Betriebssystemaufrufe kleine oder wenn möglich, keine Auswirkungen auf die Messung haben.

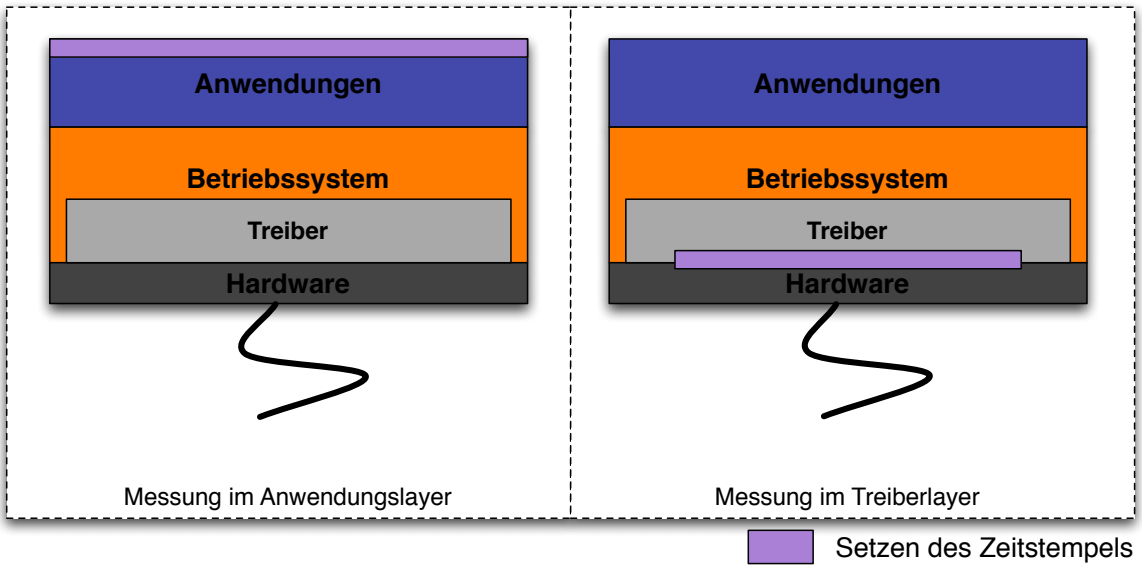

Abbildung 4.3: *Mögliche Integrierung von Messmethodiken im System*

In einem Computersystem mit Betriebssystem heißt das, die Zeitmessung in die Treiber zu verlegen. Wird ein Frame gesendet, so wird die aktuelle Zeit in den Frame geschrieben und anschließend auf die Hardware kopiert. Beim Empfangen des Frames wird sofort nach dem Aufruf der ISR die Zeit gespeichert und anschließend in den Frame kopiert.

Ein weiteres Problem einer Zeitmessung ist die Uhrensynchronisierung. Sender und Empfänger müssen exakt die gleiche Zeit besitzen damit es keine Abweichung in den Messergebnissen gibt. Die TU Dresden (vgl. Loeser und Haertig, 2004) hat in einer Arbeit beschrieben, wie harte Echtzeit-Kommunikation über Switched Ethernet ermöglicht werden kann und dieses mittels Messungen der Latenz erwiesen.

In der Messung sind mehrere Knoten an einem Switch angeschlossen (siehe Abbildung 4.4). Zwei dieser Knoten sind zusätzlich über ein weitere Verbindung („black cable“) direkt

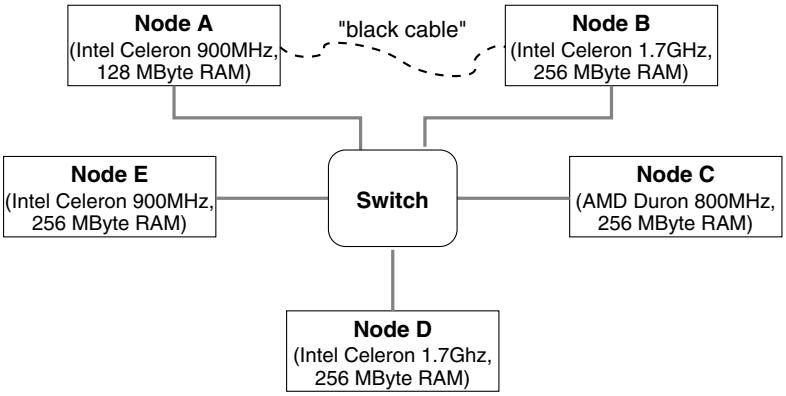

Abbildung 4.4: *Versuchsaufbau TU Dresden mit „black cable“ (vgl. Loeser und Haertig, 2004)*

angeschlossen, damit eine Zeitsynchronisierung dieser Knoten stattfindet. Somit kann eine exakte Zeitmessung erreicht werden.

Dieser Aufbau bereitet allerdings zusätzlichen Implementierungsaufwand. Eigens dafür muss zusätzlich ein Dienst zur Verfügung stehen, der die Zeit synchronisiert. Außerdem kann es zu weiterem Overhead kommen, wenn dieser Dienst nicht zuverlässig arbeitet. Daher wird in dieser Arbeit ein anderer Messaufbau verwendet. Gesendet und empfangen wird in diesem Fall auf dem selben Gerät. Durch diesen Aufbau wird kein extra Synchronisierungssdienst nötig und trotzdem eine maximale Genauigkeit erreicht.

Ein anderes Hindernis bei der Zeitmessung ist die Möglichkeit des Betriebssystems ein CPU-Throttling durchzuführen, in Abhängigkeit zur Systemauslastung. Dabei wird die CPU-Taktfrequenz angepasst um Energie zu sparen. Da die Messmethode (siehe nächsten Abschnitt 4.3) auf der CPU-Frequenz, basiert muss dieses Feature unbedingt abgeschaltet sein, da andernfalls die gemessenen Zeiten nicht korrekt sind. Eine Überprüfung über das /proc-Verzeichnis

```
demo@demo1:~$ cat /proc/acpi/processor/P001/info
processor id:            0
acpi id:                 1
bus mastering control:   yes
power management:        yes
throttling control:      no
limit interface:         no
```

gibt Aufschluss darüber, dass die CPU-Throttling-Funktion ausgeschaltet ist. Das /proc-Verzeichnis wird in Linux verwendet um Systeminformationen anzuzeigen und zu verändern.

4.3 Implementierung der Messsoftware

Die Implementierung der Messsoftware erfolgt unter Berücksichtigung der o.g. Probleme. Die Zeitmessung findet sowohl im Linux-Treiber, als auch in der Anwendung statt. Im Treiber werden die Zeitstempel gesetzt, in die Frames kopiert und in der Anwendung werden die Zeitstempel ausgewertet. Die Berechnung wird mittels Fließkommaoperationen realisiert. Diese wäre nur unter großem Aufwand im Kernel durchführbar (vgl. Love, 2005) .

4.3.1 Zeitstempel unter Linux

Die Zeitstempel können unter Linux in zwei unterschiedlichen Arten gesetzt werden. Diese unterscheiden sich dabei sowohl im Aufruf, als auch in der dahinter stehenden Logik und der Auflösung der Zeit.

do_gettimeofday() Dieser Aufruf liefert die aktuelle Zeit in einer speziellen Datenstruktur zurück, die als Daten die aktuellen Sekunden und Mikrosekunden beinhalten. „In den Quellen steht, dass do_gettimeofday() eine Auflösung „im Bereich von Mikrosekunden“ auf vielen Architekturen hat. Die Präzision variiert aber von Architektur zu Architektur und kann in älteren Kerneln kleiner sein“ (Corbet u. a. (2005)).

get_cycles() Abhängig der verwendeten Prozessorarchitektur liefert dieser Aufruf den aktuellen CPU-Zyklus. Dabei wird auf ein 64-Bit Zählregister der CPU zugegriffen, das pro Zyklus inkrementiert wird. Auf einem 1GHz Rechner läuft der Wert alle 4,2 Sekunden über. Sollte die Architektur dieses Register nicht zur Verfügung stellen, so liefert dieser Aufruf immer den Wert 0 zurück (vgl. Corbet u. a., 2005) .

Für die Messung ist die get_cycles-Lösung die bessere Wahl, da hier die größtmögliche Auflösung vorhanden ist. Mögliche Verzögerungen, die der do_gettimeofday()-Funktionsaufruf liefert, treten nicht auf, da direkt von einem CPU-Register gelesen wird. Die Frequenz der CPU vom Zielsystem beträgt 1,6 GHz, der damit auftretende Überlauf des Registers beeinträchtigt die Messung nicht, da maximal auftretende Latenzen unterhalb $3ms$ (abhängig vom TTEthernet Zyklus) zu erwarten sind.

Außerdem ist man nicht auf die Begrenzung der Zeiten in Mikrosekunden angewiesen, wie es die Datenstruktur vorgibt. Theoretisch könnten Differenzen im Nanosekunden-Bereich gemessen werden.

4.3.2 Modifikation des Ethernetframes

Um später die Latenzzeit zu ermitteln, müssen zwei Zeitstempel vorhanden sein. Eine Möglichkeit, dieses zu erreichen ist, die Werte auf dem Rechner zu speichern, sobald ein Frame gesendet und empfangen wird. Die Schwierigkeit darin besteht, die Zeitstempel einem Frame zuzuordnen, sodass auch die richtige Stopzeit zur richtigen Startzeit gezählt wird. Dieses bedeutet einen hohen Implementierungsaufwand und kann zusätzlich zu Schwierigkeiten führen, da Kernelspace-Anwendungen schlecht zu debuggen sind.

Die andere Möglichkeit besteht darin, die Stempel nicht zu speichern, sondern den zu versendenden Frame zu modifizieren. Die Nutzdaten im Frame werden bei dieser Herangehensweise zerstört. D.h. werden Messframes gesendet, kann keine Datenkommunikation stattfinden. Dafür wird Wissen über die interne Linux-Treiberarchitektur benötigt, da hier Speicherbereiche geändert werden, die aus dem Userspace nicht zugreifbar sind.

Die zweite Herangehensweise ist trotzdem die sinnvollere, da diese für zukünftige Messungen schnell abgeändert werden kann, wenn es darum geht, Zeitmessungen über mehrere Knoten zu betreiben. Zudem lassen sich so die Zeitstempel den einzelnen Frames sehr leicht zuordnen.

Modifikation des TT-Treibers

Damit die Frames modifiziert werden, muss im Treiber auf den Speicherbereich zugegriffen werden, die die Frames beschreiben.

TTTech liefert den TTEthernet-Treiber mit zugehöriger Dokumentation für die Hardware-Architektur mit. Durch Code-Analyse muss die Stelle ausfindig gemacht werden, an der gesendet wird. Der Funktionsaufruf eth_tx_buf() kennzeichnet diese Stelle.

TTTech verwendet für das Senden und Empfangen einen eigenen Datentyp, auf den entweder direkt (roh), oder über spezielle Datentypen (TT-Frame, BE-Frame) zugegriffen werden kann, sodass die Handhabung unkompliziert ist. Auf die Bereiche von Ziel- und Quelladresse, Typ und Daten kann einfach zugegriffen werden.

Folgender Code-Ausschnitt 4.3 stellt das Modifizieren innerhalb des TT-Treibers beim Senden dar. Damit nicht jeder Frame modifiziert wird, ist vorher zu überprüfen, ob es sich um einen TT-Frame handelt.

Listing 4.3: *Modifizieren des Frames beim Senden*

```
1 eth_tx_buf()
  {
3 ...
    cycles_t timestamp;
```

```
5
    //make sure we only modify tt-messages
7   if (tx_buff->ct_frame->type == TTMSG)
    {
9    //get current status of cpu-cycles
     timestamp = get_cycles();
11   memcpy(&tx_buff->ct_frame->data[0], &timestamp, sizeof(cycles_t));
    }
13 ...
}
```

Modifikation des Realtek-Treibers

Die Frames werden über den BE-Treiber empfangen, der dem Realtek-Linux-Treiber entspricht, für den jedoch kein Quellcode installiert ist. Die Quelldateien vom Kernel können jedoch unter Ubuntu leicht mit Hilfe des Paketmanagers nachinstalliert werden. Anschließend ist zu klären, welcher Treiber verwendet wird. Mit dem Befehl

```
root@demo1:~# lsmod
```

werden alle momentan verwendeten Kernelmodule aufgelistet, sodass anschließend der richtige Treiber in den Quelldateien gefunden werden kann.

Die Modifikation ist ähnlich der des TT-Treibers. Zuerst ist eine Code-Analyse durchzuführen, um herauszufinden, an welcher Stelle Ethernetframes empfangen werden. Die Funktion rtl8169_rx_interrupt() wird aufgerufen, wenn die Netzwerkkarte einen Interrupt auslöst und es sich um einen empfangenen Frame handelt. In der ISR wird der Zeitstempel „timestamp_old“ direkt zu Beginn gesetzt.

Da hier der Standard Linux-Code verwendet wird, wird mittels Socket-Buffer auf den Ethernetframe zugegriffen. Diese Datenstruktur wird im Linux-Kernel verwendet, um den gesamten Netzwerkdatenverkehr zu bewältigen, sodass auch hier einfach auf Speicherbereiche zugegriffen werden kann, die den Frame beschreiben (vgl. Corbet u. a., 2005) . Der Codeausschnitt 4.4 stellt das Prinzip der Modifikation dar.

Beim Empfangen ist besonders darauf zu achten, dass nur der zeitkritische TT-Traffic modifiziert wird. Wird der gesamte Datenverkehr mit einem Timestamp versehen, so ist eine Datenkommunikation auf IP-Ebene (z.B. ssh, ftp, etc. . .) nicht mehr möglich, da der Ethernetpayload verändert wird und so höhere Protokolle unbenutzbar macht.

Listing 4.4: *Modifizieren des Frames nachdem ein Frame empfangen wurde*

```
1 cycles_t timestamp_stop;
```

```
//ISR
rtl8169_interrupt()
{
...
    //get current status of cpu-cycles
    timestamp_stop = get_cycles();
...
    rtl_rx_interrupt();
...
}

// handle received frames
rtl8169_rx_interrupt()
{
...
    //make sure we only modify TTFrames
    //all other frames will be handled as standard Ethernettraffic
    if(skb->protocol == TTFRAME)
    {
        memcpy(&skb->data[4], &timestamp_stop, sizeof(cycles_t));
    }
...
}
```

Die benötigten Zeitstempel für die Zeitmessung sind jetzt vorhanden und können jetzt ausgewertet werden.

4.3.3 Auswertung der Timestamps in der Anwendung

Als Basis der Auswertungssoftware dient die von TTTech mitgelieferte Demo-Applikation (siehe Abschnitt 4.1.3 Beschreibung der Software). Die Auswertung wird als nachstehend erläutert.

Die Anwendung wird zusätzlich mit einem Thread erweitert, der nur für den Empfang von Ethernetframes verantwortlich ist. Somit braucht kein eigener Prozess gestartet zu werden und das Senden und Empfangen wird getrennt behandelt. Dafür wird der BE-Socket, der in der Demo-Applikation für den Versand vom BE-Video verantwortlich ist, verwendet. Der Empfangsthread wartet auf eintreffende Frames, überprüft deren Typ auf TT-Nachrichten und

wertet anschließend die Dauer der Übertragung aus. Die Dauer der Übertragung wird nach Formel 4.2 berechnet.

$$\Delta cycles_{CPU} = cycles_{stop} - cycles_{start} \tag{4.1}$$

$$t_{Messung} = \frac{1}{f_{CPU}[GHz]} * \Delta cycles_{CPU} \tag{4.2}$$

Statt der Videodaten werden Frames mit variabler Größe des Payloads gesendet, der mit 0xAA initialisiert ist. Die Größe und Anzahl zu empfangener Frames wird über Startparameter übergeben und das Programm endet, sobald die Anzahl erreicht ist.

Die Abbildung 4.5 zeigt den Inhalt eines modifizierten Ethernetframes. Die ersten Bytes repräsentieren den Startzeitpunkt, gefolgt vom Empfangszeitpunkt. Zusätzlich wurde in der Anwendung ein Paketzähler implementiert, sodass überprüft werden kann, ob Frames verloren gegangen sind. Dieser Zähler ist weiterhin hilfreich, wenn mittels Wireshark Ethernetpakete analysiert werden, da man so den Paketfluss von TT-Messframes überblicken kann.

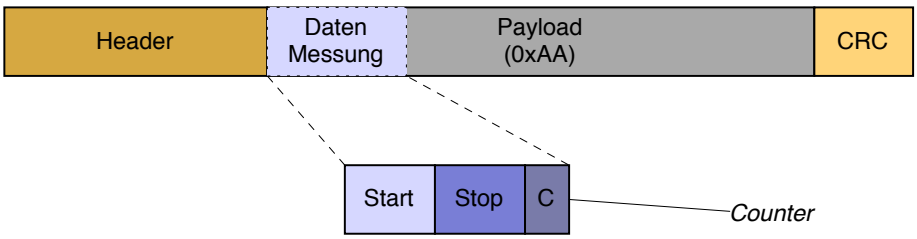

Abbildung 4.5: *Timestamps im Ethernetframe*

Zusätzlich wurde ein Pythonskript geschrieben, das für die Berechnung des Jitters und für eine tabellarische Darstellung der Ergebnisse über mehrere Messungen zuständig ist. Die Ausgaben der Messapplikation wurden in eine Datei umgeleitet. Das Skript liest die geschrieben Zeiten aus allen Ausgabedatein einer Messung und berechnet anschließend die Minimal-, Maximal-, Durchschnittswerte mit Standardabweichung und den resultierenden Jitter. Die Werte werden danach in einer Tabelle im CSV-Format abgespeichert.

Die CSV-Datei kann unter einem beliebigen Office-Programm verwendet werden, um grafische Auswertungen mit Diagrammen zu generieren.

Nachdem die Software und die Hardware des Evaluationssystems erklärt worden sind, wird im nächsten Kapitel die Zeitmessung zusammen mit den Messergebnissen aufbauend auf der Implementierungsbeschreibung der modifizierten Treiber erläutert.

5 Zeitmessung, Verifikation und Auswertung

Dieses Kapitel beschreibt die Latenzmessung des TTEthernet-Switches, den dazugehörigen Versuchsaufbau und stellt die gemessenen Zeiten dar. Die gemessenen Ergebnisse werden in einer nachfolgenden Diskussion erläutert. Eine Verifikation der Messmethodik mittels spezial Hardware wird im Anschluss betrachtet. Abschließend wird das Ergebnis der eigenen Messmethode mit dem der Verifikation und dem formalen Modell der Arbeit (vgl. Steinbach u. a., 2010) von Till Steinbach, Franz Korf und Thomas C. Schmidt verglichen.

5.1 Zeitmessung

Abbildung 5.1 zeigt den verwendeten Versuchsaufbau der Zeitmessung. Der Server sendet jeweils in Abständen von $3ms$ TT-Frames und empfängt sie über das BE-Netzwerkinterface, sobald sie vom Switch weitergeleitet wurden. Mit diesem Versuchsaufbau ist es möglich, die Latenz des Switches und dessen Jitter zu Messen.

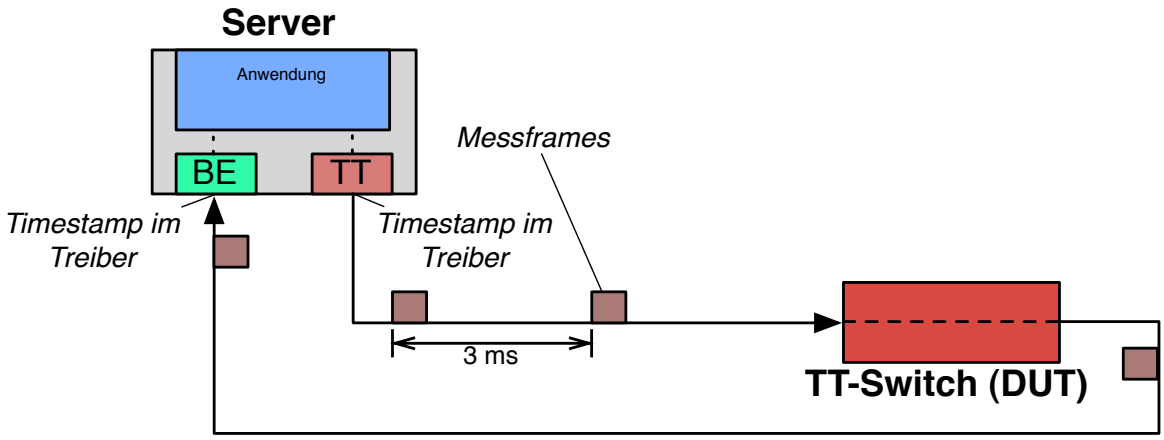

Abbildung 5.1: *Versuchsaufbau der Zeitmessung*

Der Aufbau wird bei allen Zeitmessungen verwendet und nur bei der Verifikation durch einen Switch und spezial Hardware erweitert.

Erste Zeitmessungen liefern folgendes Ergebnis (siehe Abbildung 5.2). Ersichtlich ist hierbei ein plötzlicher Anstieg der Latenz um $90\mu s$ bei einer Framegröße von 128 Byte, der für Irritation sorgt im Bezug auf die Korrektheit der Messmethode und ist bedingt durch den Hardwaretreiber der Netzwerkkarte. Im Verlauf dieses Kapitels (siehe Diskussion der Ergebnisse 5.3) wird noch weiter darauf eingegangen.

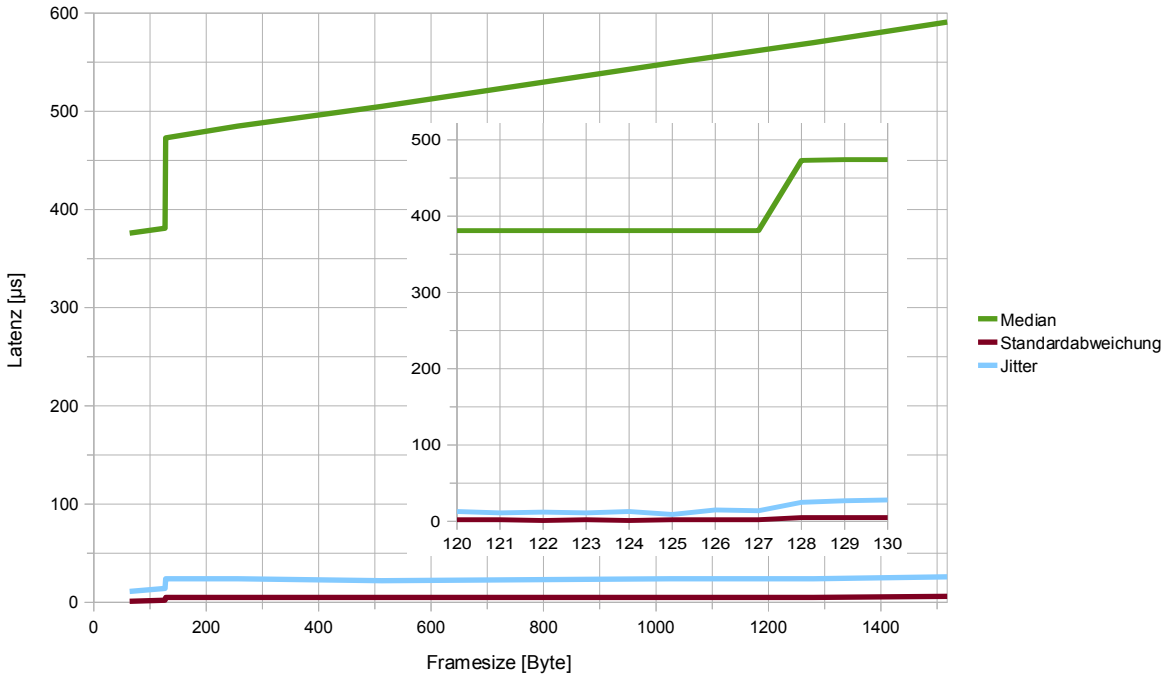

Abbildung 5.2: *Auswertung der Messung*

Alle Messungen werden mit einer einheitlichen Konfiguration durchgeführt, die durch den $3ms$ Zyklus gekennzeichnet sind. Die Messungen halten sich an die durch das IETF-RFC 2544 (vgl. RFC2544, 1999) vorgeschlagenen Framegrößen für Ethernet (64, 128, 256, 512, 1024, 1280, 1518 Byte). Dieses RFC befasst sich mit Leistungsanalysen von Netzwerkkomponenten und schlägt Messaufbau, -Methoden und Parameter für Netzwerktopologien und -Protokolle vor. Für jede Framegröße wurden 100 Frames ausgemessen und die Latenz und der Jitter bestimmt.

5.2 Verifikation

Die erzielten Messergebnisse liefern einen ersten Überblick des Verhaltens des TTEthernet Switches und der verwendeten Hardware, insbesondere der Netzwerkinterfaces. Damit keine falschen Ergebnisse erzielt werden, muss die Softwaremessmethode überprüft und dieses mittels Versuch verifiziert werden.

5.2.1 Oszilloskop

Der Einsatz eines Oszilloskops zur Verifikation der Messsoftware ist in der Theorie die ideale Möglichkeit, da mit diesen Geräten Zeitunterschiede im Nanosekundenbereich gemessen werden können.

Bei dieser Art Messung muss ein geeignetes Oszilloskop vorhanden sein, das 100Mbit/s-TX-Ethernet analysieren kann. Während der Bearbeitung dieser Arbeit war kein passendes Gerät verfügbar.

Ethernet überträgt keine logischen Pegel für ein Bit (logische 1 - pos. Spannung; logische 0 - keine Spannung) auf dem physikalischen Medium, sondern ist mit einer speziellen Codierung versehen, die es ermöglicht, das Medium nahezu gleichspannungsfrei zu halten. MLT-3 (Multilevel Transmission Encoding - 3 level) (vgl. Spurgeon, 2000) wandelt die zwei logischen Pegel in drei Spannungspegel (pos. Spannung, keine Spannung, negative Spannung) um.

MLT-3 reicht nicht aus, um die Spannungsfreiheit zu ermöglichen. Zusätzlich muss eine weitere Kodierung dafür sorgen, dass gleiche Bitfolgen unterschiedlich dargestellt werden. Im 100MBit/s-Ethernet wird deswegen das 4B/5B-Kodierungsmuster verwendet, welches aus vier Bits fünf Bits zur Übertragung wandelt (vgl. Spurgeon, 2000) .

Moderne Netzwerk-Interfaces sind zudem in der Lage, die Übertragungsgeschwindigkeit und das Duplex-Verfahren automatisch festzustellen. Dieses Autonegotiation-Signal (vgl. Spurgeon, 2000) wird mittels Puls-Nachrichten erzeugt, welche alle 16ms verschickt werden. Anhand dieses Signals erkennt die Netzwerkkarte weiterhin, ob eine aktive Verbindung besteht.

Problematisch dabei ist, dass ein korrektes Triggern mit dem nicht ethernetfähigen Oszilloskop auf das TX-RX-Signal so sehr schwierig ist. Zudem lässt sich so nur sehr schwer erkennen, wann ein Paket gesendet, bzw. Empfangen wird, da immer ein Pegelwechsel auf dem Medium vorhanden ist. Versuche mit dem vorhandene Oszilloskop scheiterten daher.

5.2.2 Hardware-Paketgenerator/Sniffer

Eine andere Art der Verifikation ist von Nöten, da der Versuch mittels Oszilloskops fehlgeschlagen ist und ein geeignetes Oszilloskop fehlt. Zur Überprüfung muss ein Gerät in der Lage sein, Ethernet zu analysieren. Die Entscheidung ist dabei auf einen Hardware Paketgenerator gefallen, der von der Fujitsu-Technologies-Serverentwicklungsabteilung zur Verfügung gestellt wird, da an der HAW kein geeignetes Messwerkzeug vorhanden ist.

Hierbei handelt es sich um ein Gerät der Firma Ixia (vgl. www.ixiacom.com, 2010) , die hauptsächlich Hardware zum Testen von Netzwerkkomponenten herstellen. Der Paketgenerator und -Analyser verfügt über mehrere Ethernetports und ist mit einer synchronisierten Uhr versehen, sodass eine exakte Zeitmessung mit einer Genauigkeit von $10ns$ über mehrere Ports möglich ist.

Dieses Gerät ist normalerweise so konfiguriert, dass ein Port Netzverkehr erzeugt und der anderer Port auf ankommende Frames horcht. Diese Konfiguration ist allerdings nicht verwendbar bei dem TT-Switch, da TT-Frames, die außerhalb des Schedules liegen, verworfen werden. Der Paketgenerator kann nicht zum Systemschedule synchronisiert werden. Allerdings kann das Ixia-Gerät so konfiguriert werden, das auf beiden Ports gehorcht werden kann und damit die Differenz eines versendeten Frames gemessen werden kann.

Nachfolgende Abbildung 5.3 beschreibt den verwendeten Versuchsaufbau. Der Server erzeugt weiterhin Ethernetframes, die im Schedule versendet werden. Zusätzlich hinzugekommen ist ein Switch, der dafür sorgt, dass Frames verdoppelt werden. Die Verdoppelung ist zwingend erforderlich, damit die Differenz des TT-Switches gemessen werden kann und wird dadurch erreicht, dass die versendeten Frames einer Multicast-Ethernetadresse entsprechen. Der Sniffer empfängt mit diesem Aufbau auf Port1 und Port2 jeweils denselben versendeten Frame, sodass die Verzögerung berechnet werden kann.

Multicast-Ethernetadressen unterscheiden sich von Unicast-Adressen im Zieladressenfeld im ersten Byte. Ist das niederwertigste Bit gesetzt, so wird der Frame als Multicast-Frame behandelt und an alle anderen angeschlossenen Teilnehmer gesendet. Werden keine höheren Protokolle, die eine Gruppenkommunikation zulassen (z.B. IP, IGMP), verwendet, entspricht Multicast einem Broadcast.

Idealerweise sollte der verwendete Switch zur Verdoppelung keine Verzögerungen aufweisen und auf den Ports balanciert sein, dass die Frames gleichzeitig versendet werden. Somit werden Messungenauigkeiten vermieden, dass auf einem Port der Frame früher gesendet wird. Versuche, den Switch mit einem Ethernet-Hub zu tauschen, scheiterten aufgrund der fehlenden Halbduplexunterstützung (siehe Beschreibung der Hardware im Vorherigen Kapitel 4.1.1) des TT-Switches. Ein Hub hat keine Verzögerung beim Weiterleiten, da er Frames nicht analysiert, sondern die Signale auf alle Ports schaltet.

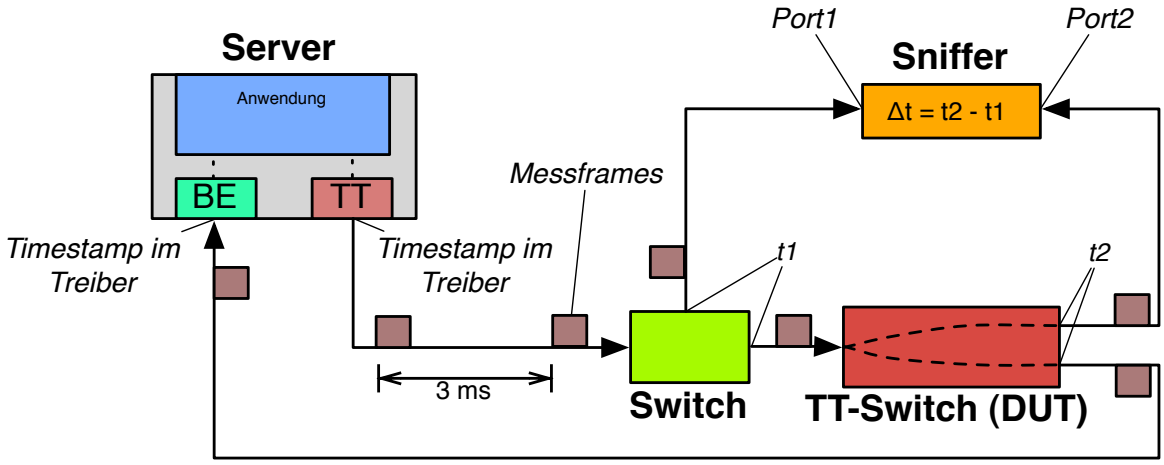

Abbildung 5.3: *Versuchsaufbau Verifikation mittels Hardwaresniffer*

Das Ergebnis der Messung ist der Abbildung 5.4 zu entnehmen. Ersichtlich ist eine abnehmende Differenz von Port1 zu Port2 in Abhängigkeit der Framegröße, die im nächsten Abschnitt erklärt wird. Die Zeiten repräsentieren die Ankunftszeitpunkte der TT-Frames an Port1 und Port2 abhängig der Synchronisierungsframes, die den Anfang eines Systemschedules kennzeichnen (siehe nächsten Abschnitt Abhängigkeit zum Systemschedule 5.3.2) und damit dem Null-Punkt entsprechen.

5.3 Diskussion der Ergebnisse

Nachdem die Messergebnisse aus den beiden Messungen bekannt sind, müssen diese nun diskutiert werden, da auf den ersten Blick nicht abgelesen werden kann, wie die Ergebnisse zu interpretieren sind. Es liegen eine Menge von Phänomenen vor, die bei der Messung (siehe Abbildung 5.2) auftreten. Vor allem der Sprung der Latenz von $90\mu s$ sorgt für Verwirrung, da ein lineares Ergebnis zu erwarten gewesen wäre. Die Diskussion serialisiert deshalb die einzelnen Besonderheiten, damit am Ende ein klares Bild entsteht.

Zuerst soll deshalb auch das beobachtete Phänomen des Sprungs erklärt werden. Anschließend wird die gesamte Latenz im Zusammenhang mit dem Systemschedule genauer betrachtet, sodass ersichtlich wird, aus welchen Teilen die Zeit zusammenhängt. Eine entwickelte mathematische Funktion zur Berechnung der Latenz schließt diesen Abschnitt ab.

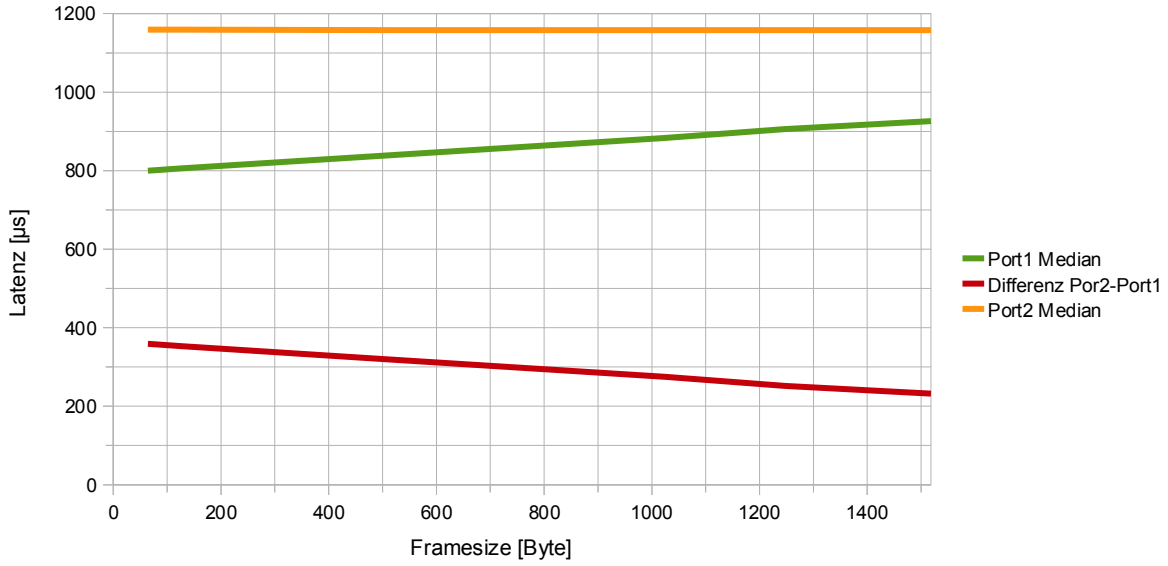

Abbildung 5.4: *Messung mittels Hardwaresniffer. Synchronisierungsframe dient als Nullpunkt*

5.3.1 Abhängigkeit von Treiberoptionen

Betrachtet wird zunächst das Ergebnis der Messung aus Abbildung 5.2. Ersichtlich ist, dass die Latenz eine lineare Steigung von $0,08\frac{\mu s}{Byte}$ $(0,01\frac{\mu s}{Bit})$ aufweist, was der Übertragungszeit von $100Mbit/s$-Ethernet entspricht. Diese Steigung ist sowohl vor dem Sprung, als auch hinterher vorhanden.

Durch Änderungen der Empfangskonfiguration (siehe Abbildung 5.5) im BE-Netzwerktreiber konnte eine lineare Steigung ohne Sprung erreicht werden. Standardmäßig ist die Hardware der Netzwerkkarte so konfiguriert, dass es keinen Schwellwert (threshold (vgl. Realtek Semiconductor Corp., 2006)) gibt, an denen Daten vom Speicher der Netzwerkkarte in den Speicher des Host-Rechners kopiert werden. Frames werden komplett empfangen und anschließend zum Host kopiert.

Eine Veränderung des Thresholds auf 64 Byte erzielt das folgende (siehe Abbildung 5.5) Ergebnis. Es wird spekuliert, dass die Hardware zwei verschieden schnelle Speicher zu Verfügung stellt. Zwischen 64 und 127 Byte Framegröße wird der schnelle Speicher verwendet, ab 128 Byte der langsame. Allerdings kann dieses nicht belegt werden, da keine Dokumentation zur Speicherbelegung der Hardware vorhanden ist und ein Kontakt zu Realtek nicht zur Verfügung steht.

Anhand dieses Versuches kann davon ausgegangen werden, dass es eine treiber- und hardwarespezifische Verzögerung gibt, die im Nachhinein abgezogen werden kann. Dazu wird

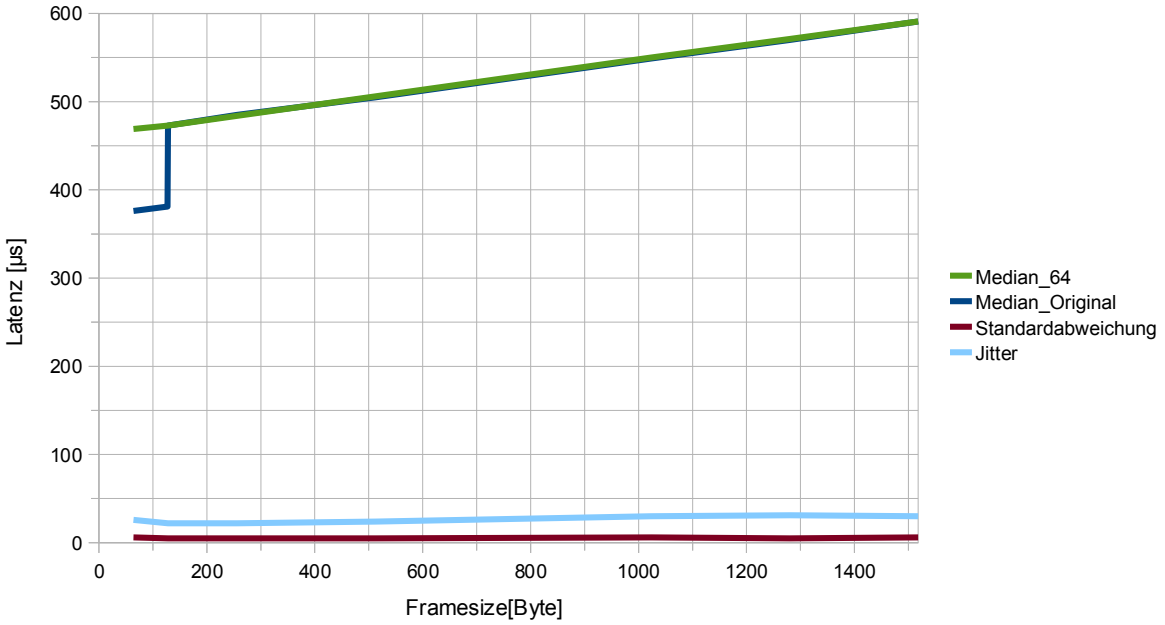

Abbildung 5.5: *Messergebnis mit veränderter Kopierschwelle im Vergleich zum unveränderten Treiber*

von der gemessenen Zeit der Systemzyklus und die Laufzeit eines Frames abgezogen (siehe Abschnitt 5.3.4). Man erhält als Verzögerung auf dem Intervall bis 127 Byte Framegröße $20\mu s$ ab 128 Byte $116\mu s$. Für andere Netzwerkhardware muss sie mit der gleichen Rechnung neu bestimmt werden.

5.3.2 Abhängigkeit zum Systemschedule

Nachdem ersichtlich ist, dass eine Abhängigkeit zur Treiberkonfiguration besteht, muss die Latenz zusammen mit dem Systemschedule untersucht werden.

Die Konfiguration des Systemschedules kann der Abbildung 5.6 entnommen werden. Synchronisierungsframes werden am Anfang jedes Zyklus gesendet. Zum Zeitpunkt $800\mu s$ werden TT-Frames versendet und vom TT-Switch empfangen. Im TT-Switch werden die Frames bis zum Zeitpunkt $1150\mu s$ verzögert und anschließend gesendet. Frames die minimal außerhalb des Schedules liegen werden so toleriert. Anhand des SOF der Preamble (siehe Kapitel Grundlagen Ethernet 2.3) im Frame überprüft der TT-Switch die Ankunftszeit, sodass die Framelänge keinen Einfluss auf den Empfangszeitpunkt hat. Gleiches gilt beim Versendezeitpunkt.

Die Differenz von $350\mu s$ ($1150\mu s - 800\mu s = 350\mu s$) im TT-Switch kann als statische

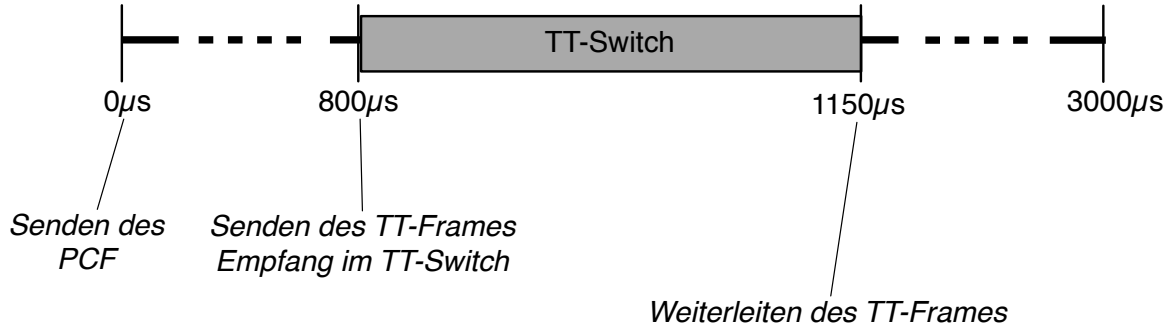

Abbildung 5.6: *Systemschedulekonfiguration*

Verzögerung angesehen und als Konstante der Messung abgezogen werden, wodurch das Ergebnis (Abbildung 5.7) erzielt wird. Die RT-Erweiterung im Linux-Kernel sorgt dafür, dass die Anwendung eine höhere Priorität besitzt als andere Interrupts. Es kann sicher gegangen werden, dass Nachrichten pünktlich um $800\mu s$ gesendet werden.

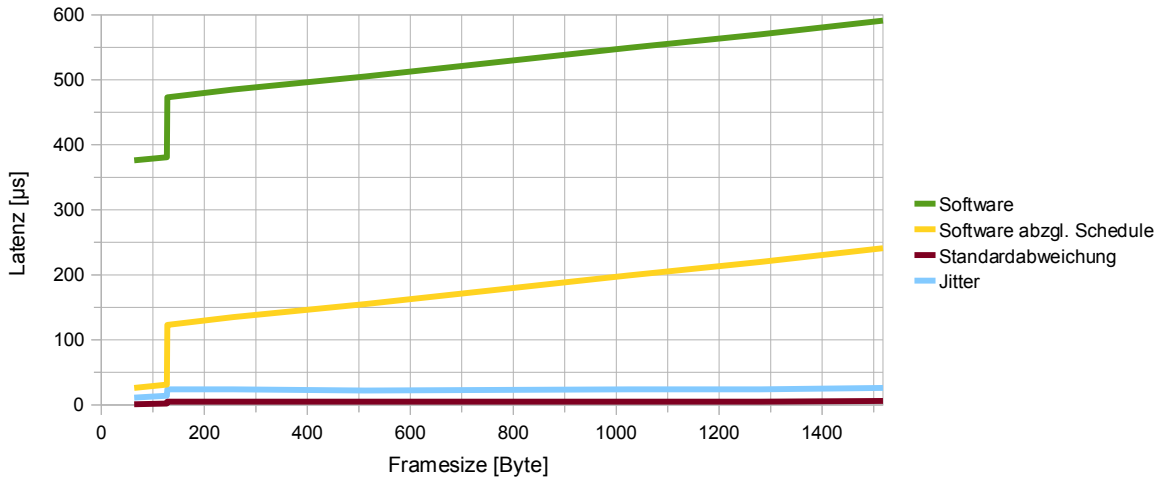

Abbildung 5.7: *Softwaremessung abzüglich des Systemschedules*

Des Weiteren erlaubt die Konfiguration den Empfang von verzögerten TT-Frames, sodass die durch den Verdoppelungsswitch verzögerten Frames trotzdem als im Schedule angesehen werden.

Der konstante Zeitpunkt beim Weiterleiten im TT-Switch führt dazu, dass bei der Softwaremessung die Paketlaufzeit nur beim Empfangen zugezählt und bei der Messung mittels Hardwaresniffer eine konstante Messung an Port2 erzielt wird.

5.3.3 Abhängigkeit vom Zeitpunkt des Zeitstempelns

Sowohl Treiberkonfiguration und Systemschedule beeinflussen das Ergebnis der Messung. Zusätzlich muss der Zeitpunkt des Zeitstempelns im Ixia-Sniffer und der Softwaremessung untersucht werden. Auffällig ist hierbei die Latenz, wenn mittels Hardwaresniffer gemessen wird (zu sehen in Abbildung 5.4). Dieses Phänomen widerspricht ersten Überlegungen, da hier die Latenz im Gegensatz zur Softwaremessung abzunehmen scheint (siehe Abbildung 5.8).

Die Erklärung hierfür ist am Ixia-Hardwaresniffer zu finden, der die Timestamps setzt, sobald ein Frame im Netzwerkinterface eintrifft. Die Paketlaufzeit spielt damit keine Rolle. Im Gegensatz dazu wird der Timestamp mittels eigener Softwaremessung erst gesetzt, wenn ein Frame komplett empfangen und auf den Host kopiert wurde.

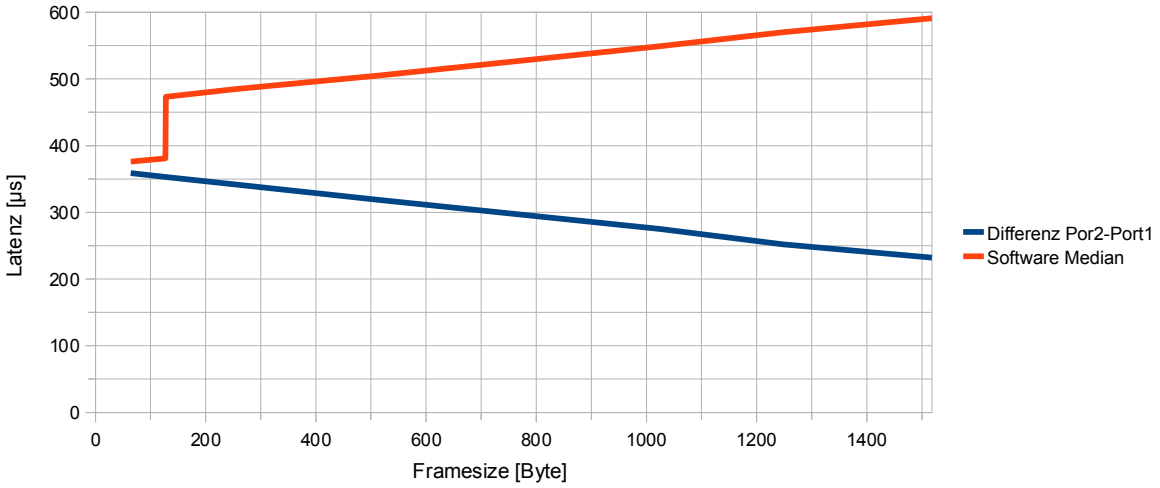

Abbildung 5.8: *Gegenüberstellung Hardwaremessung Softwaremessung*

Durch die andere Timestampbehandlung (setzen des Stempels sobald Frame eintrifft) des Ixia-Hardwaresniffers folgt, dass die Zeit auf Port2 konstant ist (siehe Abbildung 5.4) und sich mit dem Versendezeitpunkt (siehe Abbildung 5.6) im Switch deckt. Die lineare Steigung auf Port1 kann auf den verwendeten Switch zur Verdoppelung zurückgeführt werden. Die Store-and-Forward-Architektur hat keine konstante Verzögerung, da hier die Frames zwischengespeichert werden und erst nach komplettem Empfang weitergeleitet werden.

Dieses konnte mittels Versuch belegt werden, in dem nur der Verdoppelungsswitch mit der Softwarevariante ausgemessen wurde (siehe Abbildung 5.9). Ersichtlich ist hier eine doppelte Steigung der Latenz ($2 * 0,08 \frac{\mu s}{Byte}$). Die Laufzeit setzt sich somit aus der Verzögerung beim Weiterleiten im Switch und beim Empfangen des Frames in der Netzwerkkarte zusammen.

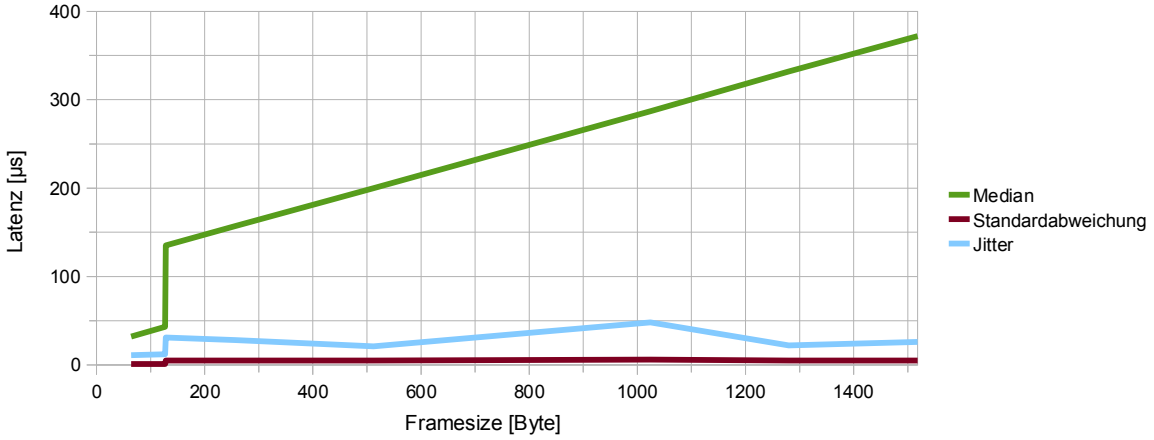

Abbildung 5.9: *Versuchsmessung des Verdoppelungsswitches*

Bei der Messung mit dem IXIA-Sniffer (siehe Abbildung 5.3) hat dieses zur Folge, das Frames vom Switch verzögert werden und somit später im TT-Switch eintreffen. Der Zeitstempel wird gesetzt, wenn ein Frame im Ixia-Sniffer eintrifft. Es entfällt die Steigung beim Empfangen an Port2. Dementsprechend nimmt die Differenz zwischen Port1 und Port2 in Abhängigkeit der Framegröße mit einfacher Steigung ($0,08\frac{\mu s}{Byte}$) ab (zu sehen in Abbildung 5.8), da die Ankunftszeit an Port1 steigt und an Port2 konstant ist (siehe Abbildung 5.4).

5.3.4 Zusammensetzung der gemessenen Zeit

Zum Schluss dieses Abschnittes wird die Zusammensetzung der gemessenen Zeit untersucht, sodass man eine mathematische Funktion erhält, die in Abhängigkeit zur Framegröße die Latenz bestimmt.

Bekannt ist, dass sich die gemessene Zeit aus mehreren Teilen zusammensetzt. Der Systemschedule von $350\mu s$ ist ein Teil. Weiterhin spielt dabei die Laufzeit eines Frames eine Rolle, die bei einem 100Mbit/s Ethernet Netzwerk $0,08\frac{\mu s}{Byte}$ entspricht.

Werden diese bekannten Zeiten von der Messung abgezogen, so erhält man die Verzögerung, die von der Netzwerkkarte und dessen Konfiguration ausgehen. Abbildung 5.10 stellt die Verzögerung dar.

Die ausgehende Verzögerung von der Netzwerkkarte ist auf beiden Intervallen nahezu konstant und beträgt auf dem Intervall bis $127Byte$ Framegröße $20\mu s$; ab $128Byte$ $116\mu s$. Die Funktion 5.1 beschreibt die Messung. Der Systemschedule und die Treiberverzögerung ($t_{Treiber64}$ und $t_{Treiber128}$) werden als konstant betrachtet. Die Laufzeit ist abhängig von der Framegröße.

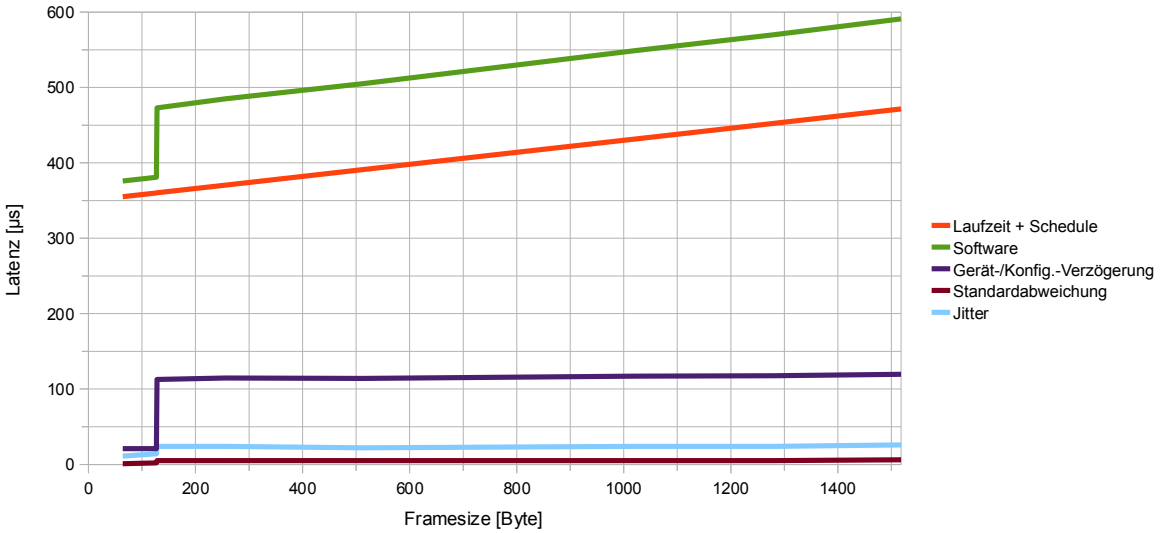

Abbildung 5.10: *Bestimmung Netzwerkkarten-, Konfigurationsverzögerung*

$$
\begin{aligned}
x &:= \text{Framesize[Byte]}, x \in \{64 \ldots 1518\} \\
t_{Laufzeit}(x) &:= 0,08 \frac{\mu s}{Byte} * x \\
t_{Messung}(x) &:= \begin{cases} t_{Laufzeit}(x) + t_{Systemschedule} + t_{Treiber64} & \text{falls } x < 128 \\ t_{Laufzeit}(x) + t_{Systemschedule} + t_{Treiber128} & \text{sonst} \end{cases}
\end{aligned} \tag{5.1}
$$

Mit dieser Funktion ist es möglich die Latenz und Hardwareverzögerung zu berechnen, sodass ein Vergleich mit anderen Netzwerkkarten ermöglicht wird.

5.4 Bewertung der Ergebnisse

Die Zusammensetzung der Latenz mittels eigener Messmethodik ist geklärt worden und die Phänomene beschrieben, sodass ein klares Bild der Messung entstanden ist. In diesem Abschnitt wird die Softwaremessung mit der Hardwaremessung und anschließend mit dem formalen Modell (vgl. Steinbach u. a., 2010) aus dem Papier von Till Steinbach, Franz Korf und Thomas C. Schmidt verglichen.

5.4.1 Softwaremessung - Hardwaremessung

Damit eine faire Gegenüberstellung beider Messmethoden möglich ist, muss bei der Hardwaremessung die Paketlaufzeit pro Byte zugerechnet werden, da der Sniffer die Pakete stempelt, sobald sie eintreffen. Zusätzlich muss der Schedule-Overhead abgezogen werden, da der PCF als Null-Punkt verwendet wird (siehe Abbildung 5.4). Die Differenz zwischen PCF und TT-Frame beträgt $800\mu s$ und kann damit abgezogen werden. Durch diese Änderungen erhält man die Zeit, die die Verzögerung im Switch und die Laufzeit zum Empfänger einbezieht und damit vergleichbar mit der Softwaremessung ist. Zusätzlich muss hier die Treiber-/Hardware-Verzögerung abgezogen werden.

Das Ergebnis der Gegenüberstellung ist der folgenden Abbildung 5.11 zu entnehmen.

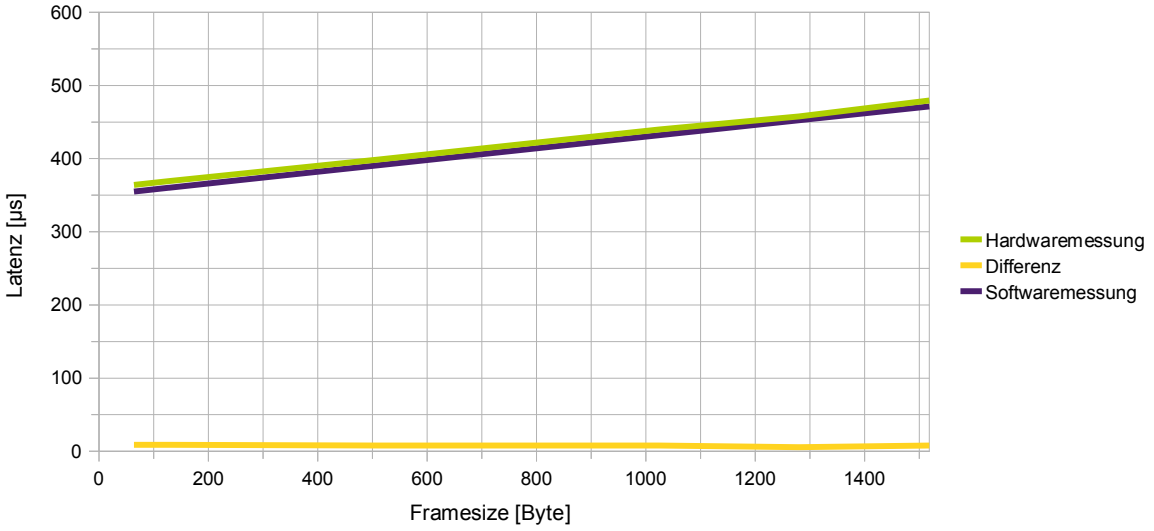

Abbildung 5.11: *Gegenüberstellung Hardwaremessung - Softwaremessung*

Die Differenz beider Ergebnisse liegt bei $8\mu s$, die aus dem unterschiedlichen Versuchsaufbau resultieren. Die Verwendung des normalen Switches zur Verdopplung kann diesen Unterschied hervorrufen.

Durch die Verzögerung aller Frames wird auch die Synchronisation zwischen Server und TT-Switch beeinträchtigt, da den Synchronisationsframes keine Verzögerung hinzugefügt wird, sodass sich beide Schedules minimal unterscheiden. Das heißt, dass beim Weiterleiten im Switch weitere Verzögerungen hinzukommen können.

5.4.2 Softwaremessung - formales Modell

Damit das Messergebnis vergleichbar mit dem formalen Modell ist, muss auch hier die Verzögerung, die von dem Treiber ausgeht, heraus gerechnet werden. Zum Vergleich wird dann die Formel 5.2 aus Steinbach u. a. (2010) verwendet, die die maximale Latenz eines TTEthernet-Netzwerks berechnet.

$$t_L = t_{WD} * l_W + (n_s + 1) * l_F * t_b + \sum_{i=1}^{n_s} t_{SD_i} \tag{5.2}$$

t_{WD} steht für den Signallaufzeit auf dem Physikalischen Medium, wobei l_W die Kabellänge ist. Die Anzahl verwendeter Switches n_s und die Länge eines Frames l_F mit der Laufzeit pro Bit t_b sind weitere Parameter des Modells. Die Summe aller durch TT-Switches hervorgerufenen Verzögerungen wird durch $\sum_{i=1}^{n_s} t_{SD_i}$ repräsentiert. Dabei ist darauf zu achten, dass diese Zeit vom Absenden eines Frames bis zum Eintreffen im nächsten Switch einbezieht.

In diesem Versuchsaufbau gibt es nur einen Switch, daher setzt sich die Summe (siehe Rechnung 5.3) aller Switchverzögerungen aus der Schedulingverzögerung (t_{SchedD}) abzüglich der Übertragungszeit zusammen.

$$\begin{aligned} \sum_{i=1}^{1} t_{SD_i} &= t_{SchedD} - l_F * t_b \\ t_L &= t_{WD} * l_W + (n_s + 1) * l_F * t_b + t_{SchedD} - l_F * t_b \end{aligned} \tag{5.3}$$

Wendet man diese Formel auf den verwendeten Versuchsaufbau an, so kommt man auf folgendes Ergebnis (siehe Rechnung 5.4) für minimale und für maximale Framelänge (siehe Rechnung 5.5). Dabei wird von einer Signallaufzeit von $10\frac{ns}{m}$, einer Kabellänge von $0,5m$ ausgegangen und $350\mu s$ als Schedulingverzögerung des Switches.

$$\begin{aligned} t_L &= 10\,\frac{ns}{m} * 0,5m + (1+1) * 64Byte * 8 * 0,01\frac{\mu s}{bit} \\ &\quad + (350 - 64Byte * 8 * 0,01\frac{\mu s}{bit}\mu s) \\ t_L &= 5ns + 2 * 5,12\mu s + (350\mu s - 5,12\mu s) \\ t_L &= 355,125\mu s \end{aligned} \tag{5.4}$$

$$\begin{aligned} t_L &= 10\,\frac{ns}{m} * 0,5m + (1+1) * 1518Byte * 8 * 0,01\frac{\mu s}{bit} \\ &\quad + (350 - 1518Byte * 8 * 0,01\frac{\mu s}{bit}\mu s) \\ t_L &= 471,445\mu s \end{aligned} \tag{5.5}$$

Die nachfolgende Tabelle stellt die Ergebnisse aus den verschiedenen Verfahren dar. Die verwendete Sofwarelatenzmessung stimmt mit dem formalen Modell überein.

	Software-Methode	**Hardware-Verifikation**	**formales Modell**
minimale Framelänge	$355\mu s$	$364\mu s$	$355,125\mu s$
maximale Framelänge	$471\mu s$	$479\mu s$	$471,445\mu s$

Tabelle 5.1: *Übersicht der minimalen und maximalen Framelänge aller verwendeten Verfahren*

6 Zusammenfassung, Fazit Ausblick

Dieses abschließende Kapitel umfasst ein Resümee der Arbeit und gibt einen Überblick über künftige Arbeiten.

6.1 Zusammenfassung und Fazit

Ziel dieser Arbeit war es, ein Bring-Up des TTEthernet Evaluationssystems durchzuführen und eine anschließende Latenzmessung des TTSwitches zu ermöglichen. Beide Ziele wurden erfolgreich erreicht, sodass eine große Wissensbasis über TTEtherent entstanden ist.

Zusammenfassung

Zuerst wurden die Grundlagen von Echtzeit und der -Linuxerweiterung besprochen. Wenn von Echtzeit gesprochen wird, heißt es nicht, dass so schnell wie möglich auf ein Ereignis reagiert wird, sondern in einer garantierten Zeitspanne. Die RT-Erweiterung lässt dieses zu, indem der Kernel an jeder Stelle unterbrechbar gestaltet wird, sodass User-Space-Anwendungen Kernel-Tasks verdrängen können. Eine Modifizierung der Interruptbehandlung findet zusätzlich statt, indem eine ISR keine Funktionalität besitzt, sondern eine Kernel-Task startet, die die Funktion übernimmt.
Ethernet besitzt in seiner Grundform keine Echtzeitfähigkeiten und kann durch die Einführung von Switched-Ethernet und VLAN-Tagging mit der 802.1Q-Erweiterung auf weiche Echtzeit ohne Vorhersage in der Übertragung verbessert werden.

TTEthernet basiert auf Switched-Ethernet, sodass die Framestruktur, das Übertragungsprotokoll und die Netztopologie (Stern) unverändert bleiben. Zusätzlich wird es um die Fähigkeit einer zeit- und prioritätengesteuerten Übertragung erweitert. TTEthernert definiert drei Nachrichtenklassen, die höchste Priorität besitzen TT- (Zeitgesteuert), gefolgt von RC- (Limitierung der Bandbreite) und BE-Nachrichten (keine garantierte Übertragung). Damit eine zeitgesteuerte Übertragung möglich ist, wird im TTEthernet eine Zwei-Wege-Zeitsynchronisierung verwendet. Teilnehmer in diesem Prozess lassen sich als Synchronisationsmaster (SM), -Client (SC) und Compressiomaster (CM) einteilen, wobei im ersten

Schritt die Synchronisation im SM durch das Versenden eines PCF gestartet und im zweiten die neu berechnete Zeit an alle Teilnehmer gesendet wird. Die Synchronisation wird mittels Ethernetframes minimaler Framelänge realisiert und basiert auf RC Nachrichten, sodass eine Zustellung garantiert ist und weiterhin eine sichere Übertragung ermöglicht wird.

Das Evaluationssystem besteht aus einem TT-Switch und zwei Endsystemen, die auf einer Intel Atom-Archtiktur basieren. Zusätzlich werden Konfigurations- und Überwachungswerkzeuge von TTTech, eine Beispielapplikation und angepasste Netzwerkkarten-Treiber mit Quellcode mitgeliefert. Der TT-Protokollstack wird als vorkompiliertes Modul mitgeliefert.
Aufbauend auf den Quelldateien wurde eine Messsoftware implementiert, die es ermöglicht, kostengünstig TT-Netzwerke zu vermessen, ohne Spezialhardware zu verwenden. Dazu wurden die Treiber für TT-Netzwerkinterfaces abgeändert, dass diese einen Zeitstempel in die Frames vor dem Versenden und direkt nach Empfang kopieren. Die Auswertung der Zeiten wurde durch Veränderung der Beispielapplikation erreicht. Die Zeitstempel basieren, um eine möglichst hohe Genauigkeit zu erreichen, auf dem CPU-Zählregister, das pro CPU-Zyklus inkrementiert wird.

Die gemessenen Ergebnisse wurden mit einem alternativen Versuchsaufbau und Spezialhardware verifiziert. Eine anschließende Diskussion offenbarte, dass die gemessenen Zeiten von der verwendeten Netzwerkhardware und dessen Treiberkonfiguration auf Best-Effort-Seite abhängig ist. Eine weitere Abhängigkeit besteht zum konfigurierten TTEthernet-Zyklus, der Ungenauigkeiten der lokalen Uhren toleriert. Weiterhin ist erkenntlich geworden, dass der Zeitpunkt des Zeitstempelns eine große Rolle spielt und die Laufzeit eines Ethernetframes der Latenz zugerechnet werden muss. Mit dem Ende der Diskussion wurde erreicht, dass man die Latenz mit einer mathematischen Funktion berechnen und diese auch auf andere Messhardware adaptieren kann. Ein Vergleich mit anderer Hardware kann so gezogen werden.
Zum Schluss wurden die gemessenen Zeiten aus der eigenen Softwaremethode und der Verifikation miteinander verglichen. Dabei stellte sich heraus, dass die gemessene Latenz mittels Hardware ca. $8\mu s$ langsamer ist. Weiterhin wurde eine Gegenüberstellung des eigenen Messverfahrens mit dem formalen Modell von Till Steinbach durchgeführt, mit dem Fazit, dass beide Ergebnisse identisch sind.

Fazit

Das Ziel einer genauen Zeitmessung auf Basis günstiger Standard Hardware zu ermöglichen ist damit erreicht worden. Es können Zeitmessungen durchgeführt werden, die auf der Basis von Millisekunden liegen. Das Prinzip dieses Messverfahrens kann auch auf ein Best-Effort-Netz angewendet werden. Dazu muss lediglich der TT-Socket gegen einen BE-Socket beim Senden getauscht werden.

Während der Bearbeitung mussten viele Problemstellungen gelöst werden. Eine Einarbeitung in die Linux-Treiber-Entwicklung und Kernel-Programmierung hat das Verständnis von Betriebssystemen und Netzwerken erhöht.

6.2 Ausblick

Zum Schluss dieser Arbeit wird ein Ausblick auf weitere Arbeiten im RTEthernet-Projekt und auf einen möglichen Einsatz von TTEthernet im Automobil gegeben.

6.2.1 Messung mit anderer Netzwerkhardware durchführen

Im Kapitel Zeitmessung, Verifikation und Auswertung wurde ersichtlich, dass die gemessene Zeit abhängig von der verwendeten Netzwerkkarte und dessen Treiberkonfiguration ist. Anhand der entwickelten Funktion zur Berechnung der Latenz sollte der Versuch mit einer anderen Netzwerkkarte wiederholt werden. Idealerweise würde sich eine Netzwerkkarte mit guter Dokumentation anbieten, damit von vornherein eindeutig ist, wie schnell der Zugriff auf die Hardware ist. Durch diesen Versuch wird erhofft, die Messung ohne anschließende Berechnung genauer zu halten und eine zusätzliche Verifikation der Messmethode zu erhalten.

Die entwickelte Funktion müsste an die neu verwendete Hardware angepasst werden, sodass auch hier eine Berechnung in Abhängigkeit zur Framegröße ermöglicht wird.

6.2.2 Weitere Arbeiten im Rahmen des RTEthernet-Projektes

Im Rahmen des RTEthernet-Projektes werden momentan folgenden Arbeiten ausgeführt und oder sind geplant.

Simulation von TTEthernet Till Steinbach und Hermand Dieumo Kenfack arbeiten derzeit an einer Simulation von TTEthernet Netzwerken, um zu ermöglichen, komplexe TT-Ethernet-Netzstrukturen kostengünstig zu untersuchen.

Cliententwicklung Kai Müller entwickelt für seine Bachelorarbeit einen TTEthernet-Client, der ohne komplexen Betriebssystem auf einem ARM-9 Board implementiert wird.

charakteristische Netzwerklastgenerierung Hermand Dieumo Kenfack befasst sich im Master-Projekt mit einer charakteristischen Netzlastgenerierung, die in die Simulation einfließen wird.

Analayse Protokollimplementierung Mehmet Bulut wird in seiner Bachelorarbeit voraussichtlich eine Analyse des TTEthernet-Protokolls durchführen. Dabei wird ein eigener TTEthernet-Protokoll-Stack entwickelt, der als ein Linux-Kernelmodul implementiert wird.

Demonstrator Es soll ein Demonstrator entwickelt werden, der das Verhalten eines TTEthernet-Netzwerkes anhand einer praxisnahen Anwendung demonstriert. Im Gespräch ist derzeit eine Steer-by-Wire-Anwendung mittels Force-Feedback-Lenkrad.

6.2.3 Möglicher Einsatz von TTEthernet im Automobil

In der Einleitung wurde ein Überblick über die derzeitige Netztopologie im Automobil gegeben (siehe Abbildung 1.1). In diesem letzten Abschnitt soll kurz auf zwei mögliche Einsatzszenarios von TTEthernet im Automobil eingegangen werden (vgl. Gressl, 2010) .

Das erste mögliche Einsatzgebiet von TTEthernet ist eine Adaption als weiteres Kommunikationssystem im Automobil (vgl. Gressl, 2010) und kann für neue Anwendungsgebiete im X-By-Wire-, Fahrassistenz und Infotainment-Bereich verwendet werden.

Die zweite Möglichkeit besteht darin, TTEthernet als Backbone-Netzwerk einzusetzen (vgl. Gressl, 2010) , um beispielsweise eine einfache Front-Heck-Kommunikation zu erlauben, in der sämtliche Arten von Daten über ein gemeinsames Medium gesendet werden (siehe Abbildung 6.2.3).

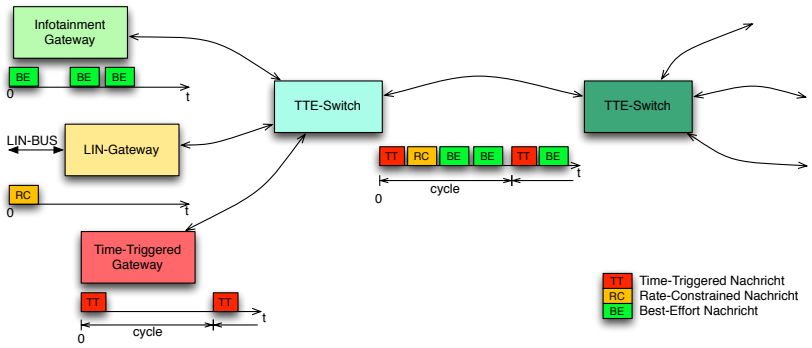

Abbildung 6.1: *Möglicher Einsatz von TTEthernet als Backbone-Netz im Automobil (vgl. Gressl, 2010)*

Durch diese Einsatzweise kann die Verwendung von Kabeln deutlich reduziert werden, da nicht für jedes System ein separate Verbindung verlegt werden muss und auf Standard Kabel zurückgegriffen werden kann. Eine Einsparung von Kosten und Gewicht wird somit erreicht.

Abschließend ist zu sagen, dass TTEthernet eine neue und aussichtsreiche Technologie ist, die je nach Einsatzgebiet Vorteile gegenüber jetzigen Bussystemen aufweist. Vor allem in Bezug auf nutzbare Bandbreite ist dieses System den jetzigen deutlich überlegen und mit Ethernet als Basis sind Bandbreiten von bis zu 10Gbit/s denkbar.

Literaturverzeichnis

[Abbot 2006] ABBOT, Doug: *Linux for embedded and real-time applications - 2nd Edition*. Butterworth Heinemann, Mai 2006. – ISBN 0-7506-7932-8

[AIM GmbH] AIM GMBH: *Avionics Databus Solutions*. – URL http://www.afdx.com/. – Zugriffsdatum: 24.02.2010

[Corbet u. a. 2005] CORBET, Jonathan ; RUBINI, Alessandro ; KROAH-HARTMAN, Greg: *Linux Device Drivers, Third Edition*. O'Reilly Media, Inc., 2005. – ISBN 978-0-596-00590-0

[Ethercat Technology Group] ETHERCAT TECHNOLOGY GROUP: *Ethercat*. – URL http://www.ethercat.org

[Ethernet POWERLINK Standardization Group] ETHERNET POWERLINK STANDARDIZATION GROUP: *Powerlink*. – URL http://www.ethernet-powerlink.org/

[GE Fanuc Embedded Systems 2007] GE FANUC EMBEDDED SYSTEMS: *AFDX/ARINC 664 Protocol Tutorial*. April 2007

[Gressl 2010] GRESSL, Bernhard: *Platforms and Architectures: Automotive Use Cases*. Presentation. July 2010

[Held 2008] HELD, Gilbert: *Carrier Ethernet: Providing the Need for Speed*. Auerbach Pubn, März 2008. – ISBN 978-1420060393

[Hochschule Reutlingen] HOCHSCHULE REUTLINGEN: *Informationsportal für Echtzeit-Ethernet in der Industrieautomation*. – URL http://www.pdv.reutlingen-university.de. – Zugriffsdatum: 24.02.2010

[IEEE 2010] IEEE: *IEEE EtherType Registration Authority*. URL: http://standards.ieee.org/regauth/ethertype/eth.txt. APRIL 2010

[Jeffree 2006] JEFFREE, Tony: *Virtual Bridged Local Area Networks*. May 2006

[Klinger 2008] KLINGER, Andreas: *Echtzeit unter Linux mit dem RT-Preemption-Patch*. Elektronik Praxis. Oktober 2008. – URL http://www.elektronikpraxis.vogel.de. – Zugriffsdatum: 28.01.2010

[Kluxmann und Malik 2007] KLUXMANN, Niels ; MALIK, Arnim: *Lexikon der Luftfahrt, 2. akt. Auflage*. Springer, 2007. – ISBN 978-3540-49095-1

[Kurose und Ross 2008] KUROSE, James F. ; ROSS, Keith W.: *Computer Networking - A Top-Down Approach*. Pearson Education, Inc., 2008. – ISBN 978-0-321-51325-0

[Loeser und Haertig 2004] LOESER, J. ; HAERTIG, H.: Low-latency hard real-time communication over switched Ethernet. In: *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*, june-2 july 2004, S. 13 – 22. – ISSN 1068-3070

[Love 2005] LOVE, Robert: *Linux-Kernel-Hanbuch - Leitfaden zu Design und Implementierung von Kernel 2.6*. Addison Wesley, 2005. – ISBN 3-8273-2204-9

[de M. Valentim u. a. 2008] M. VALENTIM, R.A. de ; MORAIS, A.H.F. ; BRANDAO, G.B. ; GUERREIRO, A.M.G.: A performance analysis of the Ethernet nets for applications in real-time: IEEE 802.3 and 802.3 1 Q. In: *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, July 2008, S. 956–961. – ISSN 1935-4576

[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*, Sept. 2008, S. 392–399

[PROFIBUS & PROFINET International] PROFIBUS & PROFINET INTERNATIONAL: *Profinet*. – URL http://www.profibus.com/pn/

[Realtek Semiconductor Corp. 2006] REALTEK SEMICONDUCTOR CORP.: *Integrated Gigabit Ethernet Controller For PCI-Express Applications - Registers Datasheet*. April 2006

[RFC2544 1999] BRADNER, S.: *RFC2544 - Benchmarking Methodology for Network Interconnect Devices*. URL: http://www.ietf.org/rfc/rfc2544.txt. March 1999

[Sourceforge.net 2010] SOURCEFORGE.NET: *FUSE - Filesystem in Userspace*. 2010. – URL http://fuse.sourceforge.net. – Zugriffsdatum: 24.02.2010

[Spurgeon 2000] SPURGEON, Charles E.: *Ethernet - The Definitive Guide*. O'Reilly Media, Februar 2000. – ISBN 978-1-56592-660-8

[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 (WFCS 2010)*. Piscataway, NJ, USA : IEEE Press, May 2010, S. 199–202

[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*, July 2009, S. 319–322

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

[Tanenbaum 2003] TANENBAUM, Andrew S.: *Computer-Netzwerke - 4., aktualisierte Auflage*. Pearson Studium, 2003. – ISBN 3-8273-7046-9

[Tanenbaum 2009] TANENBAUM, Andrew S.: *Moderne Betriebssysteme - 3., aktualisierte Auflage*. Pearson Studium, Oktober 2009. – ISBN 978-3-8273-7342-7

[www.android.com 2010] WWW.ANDROID.COM: *What is Android?* 2010. – URL http://www.android.com. – Zugriffsdatum: 24.02.2010

[www.ixiacom.com 2010] WWW.IXIACOM.COM: *Ixia - Leader in Converged IP Testing*. 2010. – URL http://www.ixiacom.com. – Zugriffsdatum: 30.05.2010

[www.microsoft.com 2010] WWW.MICROSOFT.COM: *Microsoft Windows Embedded CE*. 2010. – URL http://www.microsoft.com/windowsembedded/en-us/products/windowsce/default.mspx. – Zugriffsdatum: 10.03.2010

[www.osek-vdx.org 2010] WWW.OSEK-VDX.ORG: *OSEK VDX Portal*. 2010. – URL http://www.osek-vdx.org. – Zugriffsdatum: 10.03.2010

[www.qnx.com 2010] WWW.QNX.COM: *QNX Neutrino RTOS*. 2010. – URL http://www.qnx.com. – Zugriffsdatum: 10.03.2010

[www.rtai.org 2010] WWW.RTAI.ORG: *RTAI - the RealTime Application Interface for Linux from DIAPM*. 2010. – URL http://www.rtai.org. – Zugriffsdatum: 24.02.2010

[www.rtlinuxfree.com 2010] WWW.RTLINUXFREE.COM: *RTLinux*. 2010. – URL http://www.rtlinuxfree.com. – Zugriffsdatum: 24.02.2010

[www.wireshark.org 2010] WWW.WIRESHARK.ORG: *Wireshark*. 2010. – URL http://www.wireshark.org. – Zugriffsdatum: 24.02.2010

[www.xenomai.org 2010] WWW.XENOMAI.ORG: *Xenomai: Real-Time Framework for Linux*. 2010. – URL http://www.xenomai.org. – Zugriffsdatum: 24.02.2010

[Yaghmour u. a. 2008] YAGHMOUR, Karim ; MASTERS, Jon ; BEN-YOSEFF, Gilad ; GERUM, Phillipe: *Building Embedded Linux Systems - 2nd Edition*. O'Reilly Media, Inc., August 2008. – ISBN 978-0-596-52968-0

[Zimmermann 2008] ZIMMERMANN, Ralf: *Bussystem in der Fahrzeugtechnik - 3. Auflage*. Vieweg + Teubner, September 2008. – ISBN 978-3-8348-0447-1

Inhalt der beiliegenden CD

/measurement/tte_es/ Treibermodifikationen

/measurement/video_app/server/ Software zum Zeitmessen

/measurement/skript/ Python v2.5-Skript zur Auswertung

/pdf/ Die Arbeit als PDF-Dokument

Versicherung über Selbständigkeit

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

Hamburg, 14. Juli 2010	
Ort, Datum	Unterschrift