Diplomarbeit

Implementation von RTnet in ein Echtzeit Linux System sowie eine vergleichende Bewertung von RTnet und Profinet

Diplomarbeit angefertigt an der
Universität Lüneburg,
Fachbereich Automatisierungstechnik,
Studiengang Ingenieur-Informatik
zur Erlangung des akademischen Grades eines
Diplom-Informatikers

Vorgelegt von
Jan-Peter Schulz
Matr.Nr.: 153353,
am 01.09.2005

Erstprüfer: Prof. Dr. rer. nat. Dipl.-Inform. Helmut Faasch
Zweitprüfer: Prof. Dr.-Ing. Dipl.-Inform. Eckhard C. Bollow
Betrieblicher Betreuer: Dipl.-Math. Oliver Flick

31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik

Kurzfassung

Spezielle Verfahren, insbesondere in der Industrie, verlangen nach echtzeitfähigen Systemen und der (Echtzeit-) Vernetzung dieser Systeme.

Im Rahmen dieser Diplomarbeit wurde in ein von der Firma NPN entwickeltes Echtzeit Linux RTnet, ein Open Source Echtzeit-Ethernet, implementiert.
Die Strukturen und Verfahren des Netzwerks wurden aufgezeigt sowie Fehler und Problemstellen benannt.
Um die Leistungsfähigkeit von RTnet zu testen, wurde ein Benchmark Programm entwickelt, um einige exemplarische Zeiten zu ermitteln.

Des Weiteren wurde ein Überblick über einige ähnliche Produkte sowie über Verfahren, welche Echtzeit-Eigenschaften ermöglichen, erstellt.

Die vergleichende Bewertung von Profinet und RTnet ergab, dass beide Systeme Stärken und Schwächen haben und sich jeweils für andere Bereiche besser eignen.

Abstract

Special Processes, especially in the industry, require real-time systems, and the (real-time) network connection of these systems.

In this Diplomarbeit, an open source real-time Network was implemented in a real-time Linux, developed by the NPN.
The structures and the processes of the network were shown, errors and problems identified.
To test the performance of Rtnet, a benchmark program was developed, to get some characteristic values.

Furthermore, an overview about similar products, and about methods which realize real-time properties, was worked out.

The comparison showed that neither Rtnet or Profinet is the better real-time system. Both systems have their strengths and weaknesses, and are proper for different applications.

Inhaltsverzeichnis

1 Einleitung

Echtzeitbetriebssysteme sind ein wachsender Markt in der Automatisierungsbranche. Die Anwendungsgebiete reichen von kleinen Embedded Systems, eingesetzt z.B. in Haushaltsgeräten, bis hin zu leistungsstarken Rechnern zur Steuerung von Industrie-Anlagen. Die Systeme auf diesem Markt sind meist proprietär, und für den Anwender selber nicht skalierbar.

Aufgrund der steigenden Nachfrage, insbesondere dem konkreten Bedarf einer Roboterfirma, und der genannten Einschränkungen bestehender Systeme, hat sich der Firmenverbund NPN einer Linux basierten Open Source Lösung angenommen.
Diese RTAI/Fusion genannte Echtzeiterweiterung soll nun bis zur Marktreife in ein Linux System implementiert, konfiguriert und weiterentwickelt werden.
Das Hauptaugenmerk liegt hier nicht nur auf Reaktionszeiten und Leistung, sondern auch auf der einfachen Benutzung des Systems für Benutzer ohne Linux-Kenntnisse. Dieses Echtzeitbetriebssystem wird im Weiteren RT4Linux genannt.
Hindernisse auf dem Weg zu diesem System sind unter anderem die mangelhafte Dokumentation im Open Source Bereich, Versionsprobleme sowie Fehler in den Echtzeiterweiterungen RTAI/Fusion und RTnet, welche sich selbst noch in einem experimentellen Stadium befinden.

Mit wachsender Komplexität von automatisierten Anlagen ist eine zunehmende Vernetzung verteilter Steuerungen zu erwarten. Ein Echtzeitbetriebssystem für die Industrie verlangt deshalb nach einem Netzwerk, welches echtzeitfähig sein muss, jedoch auch die Übertragung von “nicht Echtzeit“ Daten (TCP/IP) ermöglicht.

Der Markt an Echtzeitnetzwerken ist nicht leicht überschaubar, viele Systeme sind proprietär und lassen sich nicht in die Karten schauen. Des Weiteren soll das RT4Linux kostenfrei und Open Source bleiben. Das einzige (*echte*) Open Source Projekt ist RTnet von der Universität Hannover. Die Quellen sind für jedermann sofort zugänglich und verwendbar.
Des Weiteren ist die Protokollspezifikation der PNO (Profibus Nutzer Organisation) Profinet zu untersuchen, welche aufgrund des Einflusses großer Automobilhersteller eine weite Verbreitung hat.

Der Übergang zwischen einem vollständigen Echtzeit Ethernet und einem klassischen Feldbus ist eher fließend, und ein standardisierter Leistungstest (Benchmark) ist aufgrund verschiedenster Anforderungen kaum möglich.

Ziel dieser Untersuchung ist die Implementierung von RTnet in das Echtzeit-Linux RT4LINUX, sowie eine detaillierte Dokumentation, um RTnet im RTOS Projekt weiter führen zu können.
Des Weiteren wird eine vergleichende Bewertung von RTnet und Profinet durchgeführt.

Kernpunkte dieser Bewertung sind:

- Komplexität (Einarbeitungszeit, Wartung, vorh. Dokumentation)
- Kompatibilität zu anderen Netzen,
- Investitionssicherung (in wieweit die vorhandene Infrastruktur genutzt werden kann),
- Hardware, die benötigt wird,
- inwieweit Debugmöglichkeiten zur Laufzeit möglich sind
- die Stabilität, die Sicherheit (Datenverlust) und
- die Geschwindigkeit (Benchmark)

2 Aufgabenstellung

2.1 Echtzeit in Datennetzen

Im Automatisierungsbereich auf der Sensor/Aktor Ebene werden Programme meist zyklisch mit definierten Zeiten abgearbeitet. In jedem Zyklus werden Eingänge gelesen, Algorithmen abgearbeitet und Ausgänge gesetzt. Die Dauer dieser Zyklen ist anwendungsabhängig, genauso wie die Menge der zu übertragenden Daten und der Anzahl der Teilnehmer. Einige Anwendungen benötigen möglichst synchrone Übertragung der Daten (geringer Jitter) bzw. die Vergabe eines möglichst genauen Zeitstempels, um z.B. Achsdrehzahlen synchron zu regeln. Andere wiederum brauchen extrem kurze Übertragungszeiten (Latenzzeiten), aber weniger synchrone Daten (Regelungen etc.).
Sind Sensoren und Aktoren, welche direkt miteinander interagieren, schon mit einem Kommunikationssystem (Profibus, Servolink, CANbus, etc) zu einem Modul verknüpft, so können diese wiederum mit einem Echtzeit Ethernet verbunden werden.
Dieses muss dann meist weniger kurze Zeiten einhalten und hat den Vorteil, dass die Controller (SPS, etc) dieser Module zentral über generische Protokolle wie http, etc. parametriert werden können (Parametrisierung über Webseiten). Daten können in einem solchen System direkt an die Steuerzentrale und die Archivierung der Sensordaten gesendet werden.

2.2 Echtzeit Ethernet gegenüber feldbusbasierter Vernetzung

In den 70er Jahren waren analoge Regler der Stand der Technik, und es war kaum vorstellbar, dass sich digitale mikroprozessorgesteuerte Regler bis in die unteren Preissegmente durchsetzen würden. Während sich in den 90er Jahren Feldbussysteme immer stärker durchgesetzt haben, wird der Ruf nach einem industrietauglichen Ethernet immer stärker.

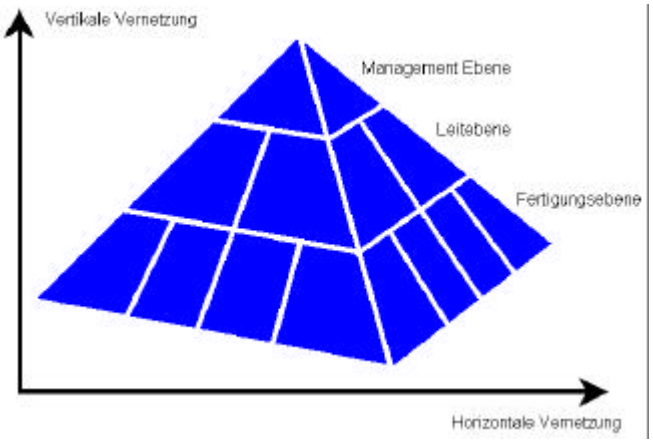


Abbildung 3 Firma ohne Vernetzung

Es sind nicht mehr nur Sensoren, Regler und Aktoren im klassischen Sinne zu vernetzen, sondern auch Kameras und Computer zur Bilderkennung und Überwachung, graphische Benutzeroberflächen auf der Leitebene, Datenbanken und Fernwartungssysteme.
Ein industrietaugliches Ethernet kann eine Firma von der Fertigungsebene (Sensor Aktor Ebene) bis in die Managementebene mit einem einheitlichen System vernetzen (Abbildung 3 und 4) und ermöglicht über das Internet die Fernwartung von Maschinen und Anlagen über jede Entfernung.

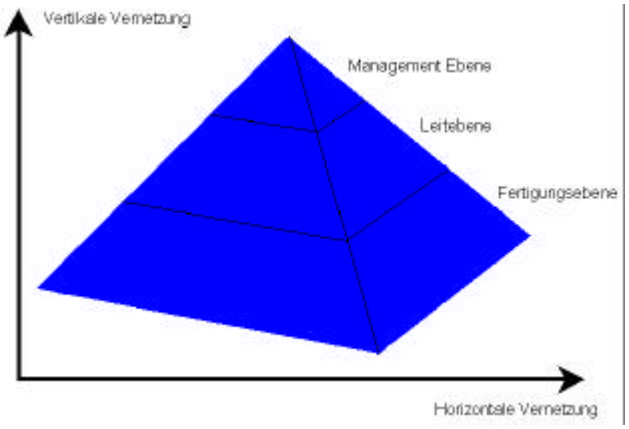


Abbildung 4 Firma mit Vernetzung

Durch die einheitliche Ethernet-Technologie ist man nicht mehr an einen Anbieter gebunden. Die nötige Hardware (Netzwerkkarten, Hubs, Switches, Kabel) ist weltweit und kostengünstig erhältlich.
Echtzeitdaten und normale Daten können über das gleiche Medium gesendet werden.

Den hohen Datenaufkommen in einer solchen Vernetzung kann Ethernet mit hohen Übertragungsraten von bis zu 10 Gigabit /s begegnen, Interbus, im Vergleich, dagegen nur 500 kBit/s.

Allerdings sollte man die Hürden bei der Einführung eines Echtzeit Ethernets nicht außer Acht lassen.
Es hat sich immer noch kein System als Standard durchgesetzt, was auch daran liegt, dass viele Systeme auf spezielle Anwendungen ausgerichtet sind und so ein Vergleich aller Systeme schwer fällt. JetSync z.B. wirbt mit besonders kleinem Jitter, während Profinet IRT mit möglichst kleinen Latenzzeiten aufwartet und Ethercat viele Teilnehmer in kurzer Zeit bedienen kann.
Da die Einführung eines Echtzeit Ethernets Kosten für die Installation und Ausbildung des bedienenden Personals erfordert und noch unklar ist, welches System bzw. welche Systeme sich durchsetzen, halten sich noch viele Firmen zurück.

3 Stand der Technik

3.1 Echtzeit Betriebssystem

3.1.1 Echtzeit

Definition [Wikipedia 05 - Echtzeit]: „Echtzeit bedeutet, dass das Ergebnis einer Berechnung innerhalb eines gewissen Zeitraumes garantiert vorliegt, das heißt bevor eine bestimmte Zeitschranke erreicht ist."

Echtzeit bedeutet nicht, wie oft angenommen wird, dass etwas ohne Zeitverlust bzw., da das nicht möglich ist, mit einem nicht merkbaren Zeitverlust geschieht. Es bedeutet nur, dass etwas zu einem garantierten Zeitpunkt fertig ist. Ein Echtzeitsystem muss immer vorhersagbar, also deterministisch auf externe Ereignisse reagieren.
Die Zeit die dafür gebraucht wird, im Weiteren Latenzzeit genannt, kann je nach Anwendung unterschiedlich lang sein.
Die Latenzzeit in Systemen mit kurzen Reaktionszeiten, wie z.B. dem ABS System eines Autos, kann im Mikrosekundenbereich liegen, während bei trägen Systemen wie Temperatur regelungen einige Sekunden ausreichen.
Des Weiteren muss zwischen harter und weicher Echtzeit unterschieden werden.
Harte Echtzeit bedeutet, dass es eine bestimmte Zeitgrenze gibt, bis zu der eine Reaktion eingetreten sein muss. Tritt diese nicht ein, so ist das Ergebnis nutzlos oder es treten Schäden auf. Ein Beispiel wäre die Computer gestützte Steuerung eines Jets(Fly-by-wire): Wird die Echtzeit verletzt, stürzt der Jet ab.
Weiche Echtzeit bedeutet, dass sich ab einer Zeitgrenze die Qualität eines Ergebnisses verschlechtert, beispielsweise die Wiedergabe von Filmen auf einem Computer: bei Verletzung der Echtzeit würde der Film "ruckeln", aber nicht zerstört sein.

Die Latenzzeiten im Echtzeit Ethernet liegen bei:
100 Millisekunden: Standard Kommunikation - zeitunkritische Daten
10 Millisekunden: Fabrik Automatisierung - Prozessdaten
>1 Millisekunden: Motion Control -Taktsynchroner Datenaustausch
[Profinet Broschüre 05]

31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik

3.1.2 GNU/Linux

GNU/Linux ist ein freies Betriebssystem auf Grundlage des von Linus Torvald geschriebenen Kernels (Linux) und des GNU-Systems.
Der Linux Kernel ist ein, ursprünglich für x86 Architekturen geschriebener, Unix (Linus + Unix = Linux) ähnlicher Betriebssystemkernel, welcher unter der GNU/GPL-Lizenz steht und somit frei verfügbar und einsetzbar ist [Kofler 2000].
Das GNU System wurde von Richard Stallmann zur Schaffung eines freien Betriebssystems entwickelt. Da es jedoch Schwierigkeiten mit der Entwicklung eines eigenen Kernels gab, wurde übergangsweise auf den Linux Kernel zurückgegriffen – welcher sich bis heute bewährt hat.

3.1.3 RTAI/Fusion

Im Weiteren wird zwischen Userspace und Kernelspace unterschieden.
In einem Betriebssystem gibt es mehrere Ausführungsebenen mit verschiedenen Rechten und Eigenschaften.

- Der Kernelspace bildet die unterste Ebene, mit den meisten Rechten, der Interruptkontrolle und dem Scheduling.
- Der Userspace bildet die oberste Ebene, in der die Benutzerprozesse arbeiten. Dieser Bereich hat weniger Rechte als der Kernelspace. Allerdings können Prozesse im Userspace weniger Schaden anrichten und leichter manipuliert werden [Maurer 04].

RTAI/Fusion verfolgt das Ziel, ein Linux System durch Erweiterungen echtzeitfähig zu machen und, im Gegensatz zum reinen RTAI, die Ausführung von Echtzeitprogrammen im Userspace zu ermöglichen.

Das Prinzip der Erweiterung ist nicht die Modifikation des Kernels, sondern die Implementation eines Adeos genannten Micro-Kernels, welcher, dem eigentlichen Betriebssystem vorgelagert, die Verteilung der Ressourcen regelt und den (nicht Echtzeit) Linux Kernel als Idle-Task verwaltet.
Echtzeitfähigkeit kann nur im Kernelspace erreicht werden, da nur dort die nötigen Rechte wie Interruptkontrolle, Scheduling, etc. freigegeben sind.
Im nativen RTAI können Echtzeitanwendungen nur als Adeos-Kernelmodule gestartet werden, was aber einige Einschränkungen mit sich bringt. Kernelmodule lassen sich nur schwer debuggen, es ist kein Zugriff auf alle Module möglich, die Gefahr den ganzen Computer in einen nicht definierten Zustand zu bringen ist relativ hoch.

Fusion ermöglicht es deshalb, Echtzeitanwendungen im Userspace als “normale“ Programme zu starten. Dort ist es möglich über Gerätedateien die Module direkt anzusprechen und die Libc, die Standard C-Bibliothek des GNU Projekts, zu nutzen. Zusätzlich sind die Ressourcen der Prozesse im Userspace voneinander getrennt, so dass sie sich nicht gegenseitig stören können. Fusion startet den Prozess im Kernelspace, kann ihn aber bei Bedarf (z.B.: Zugriff auf Bibliotheken) zwischen Kernelspace und Userspace transportieren (Abbildung 5).

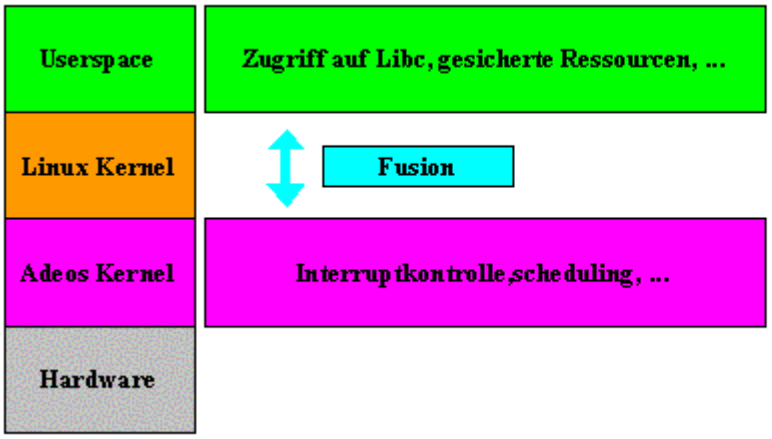


Abbildung 5 - RTAI/Fusion [Kizka 05]

3.1.4 Betriebssystem RT4Linux

Das RT4Linux Betriebsystem ist ein natives Linux mit der RTAI/Fusion Echtzeit Erweiterung.
Im Laufe dieser Diplomarbeit wurde von der Kernel Version 0.2.6 auf den Kernel 2.6.11.10 umgestiegen. Des Weiteren wurde das den Kernel umgebende System (GNU System) ständig weiterentwickelt: So wurden Bibliotheken ausgetauscht, Scripte modifiziert und von Fusion 0.6.9 auf Fusion 0.7.4 gewechselt.

3.2 Echtzeit Ethernet

3.2.1 Ethernet

Ethernet ist eine rahmenbasierte Vernetzungstechnologie für Computernetze mit definierten Kabeltypen, Signalisierungen für die Bitübertragungsschicht, Paketformaten und Protokollen. Teilnehmer eines Ethernets sind über die Ethernet- oder MAC-(Media Access Control) Adresse definiert. Diese ist die Hardware Adresse des Netzwerkgerätes, weltweit einmalig, und wird, z.B. durch das ARP (Address Resolution Protocol), für die Vermittlungsschicht in eine IP-Adresse aufgelöst.
Die Maximale Paketgröße (MTU - Maximum Transmission Unit) für Ethernet beträgt 1500 Byte Nutzdaten plus 18 Byte Header, also 1518 Byte (bei VLAN – Virtual-Lan 22Byte Header – 1522Byte MTU). Wird die Größe der Nutzdaten überschritten, so muss das Paket fragmentiert werden, d.h., es wird in mehrere Pakete zerlegt, welche die MTU nicht überschreiten.
[wikipedia – Ethernet]

Ethernet Pakete beinhalten eine CRC Prüfsumme, beim CRC (**C**yclic **R**edundancy **C**heck), der zyklischen Blockprüfung, wird wie bei der Kreuzsicherung das Prüfwort über den gesamten Block gebildet. Hierbei wird aber der Datenblock als langes zusammenhängendes Wort betrachtet, das durch ein genormtes Polynom dividiert wird. Der Rest der Division wird als CRC-Prüfwort an den zu übertragenden Nutzdatenblock angehängt.
[Könen 93]

3.2.2 Buszugriffsverfahren

Um auf das Ethernet zuzugreifen, gibt es verschiedene Möglichkeiten, wobei sich im privaten und Büro-Bereich das CSMA/CD Verfahren durchgesetzt hat. Aufgrund seiner Eigenschaft, nicht deterministisch zu sein, kann es in der Industrie für harte Echtzeit Anwendungen mit kurzen Reaktionszeiten zwar nicht genutzt werden, wird aber der Vollständigkeit halber mit aufgeführt.

CSMA/CD:

Das „Carrier-Sense-Multiple-Acess-/ -Collison-Detection“ Verfahren wird weltweit am meisten in Computernetzen eingesetzt.
Bei dem CSMA/CD Verfahren kann jeder Computer im Netz zu jeder Zeit senden, vorrausgesetzt kein anderer Computer sendet. Senden mehrere Computer gleichzeitig, wird dies bemerkt, das Senden eingestellt und ein Störsignal gesendet, damit alle anderen die Kollision auch bemerken. Alle Teilnehmer stellen das Senden ein und warten eine zufällige Zeit. Der Computer mit der kürzesten Zeit sendet erneut, die Anderen senden, wenn das Netz wieder frei ist. Ist die zufällige Zeit mehrerer Computer gleich, so warten sie wieder eine zufällige Zeit ab (Abbildung 6). Dadurch ist das CSMA/CD Verfahren nicht deterministisch, da sich die maximale Wartezeit nicht voraus sagen lässt.

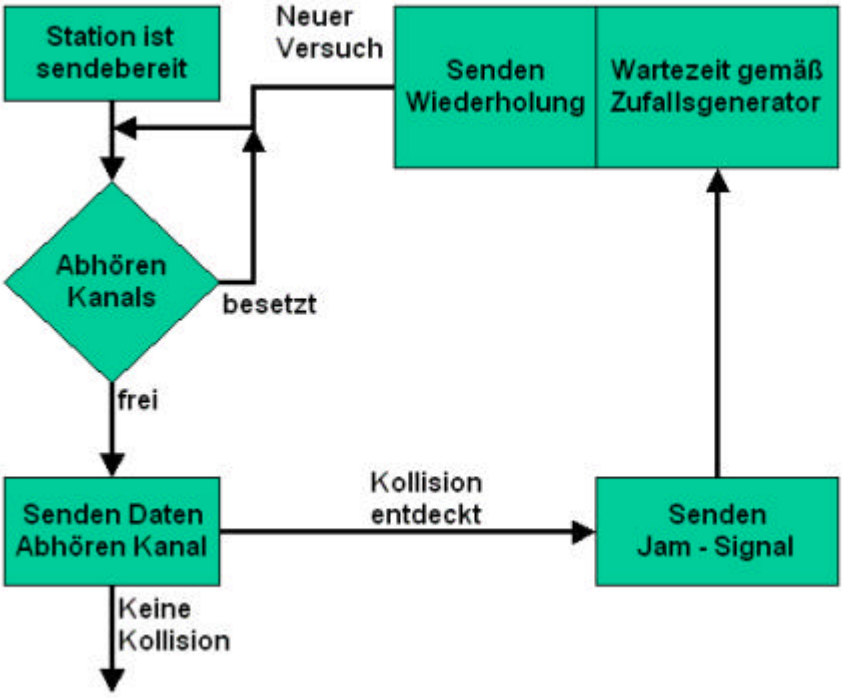


Abbildung 6 - CSMA/CD Verfahren [Könen 93]

31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik

TDMA - Synchrones Verfahren

Beim synchronen TDMA (Time Division Multiple Access) Verfahren wird jedem Teilnehmer ein fester Zeitabschnitt (Slot) in einem Zyklus zugewiesen.
Vorteile dieses Verfahrens sind seine Eigenschaft, deterministisch zu sein, da jedem Teilnehmer eine konstante Übertragungsrate zugesichert werden kann, und dass die Teilnehmer durch den Zeitabschnitt identifiziert werden können.
Nachteilig ist, dass nicht genutzte Slots Bandbreite verbrauchen.
Das TDMA Verfahren ist ein Zeitmultiplex Verfahren, d.h. dass die verschiedenen Teilnehmer sich das Medium Netz über die Zeit aufteilen.
Die verfügbare Übertragungszeit wird in Zyklen aufgeteilt, welche wiederum in Zeitschlitze (Slots) aufgeteilt sind (Abbildung 7).
Jeder dieser Zeitschlitze ist einem Teilnehmer des Netzwerkes zugeteilt, d.h. in seinem zugewiesenen Slot darf der jeweilige Teilnehmer Daten senden.
Somit kann jeder “Besitzer“ eines Slots einmal pro Zyklus senden.
Allerdings müssen die Teilnehmer, um nicht miteinander zu kollidieren, ihre Uhren miteinander synchronisieren, da nur zu Beginn des Zyklus eine Synchronisierungsnachricht vom Master gesendet wird. Die einzelnen Slaves müssen den Startpunkt ihres Slots selber berechnen.

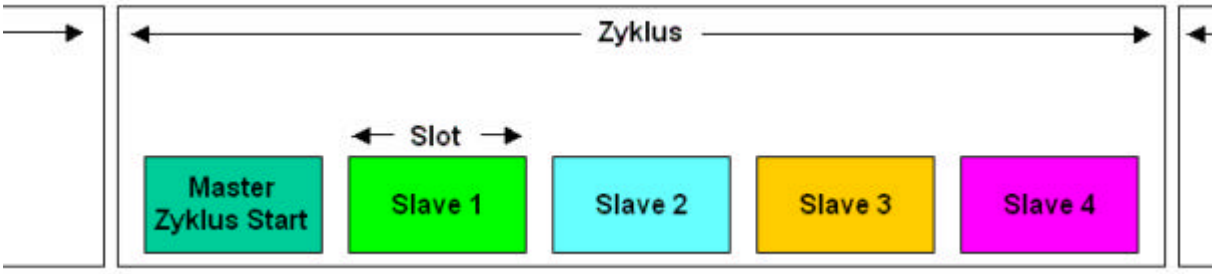


Abbildung 7 TDMA Verfahren

Die Länge des Zyklus ist von der Anzahl der Slaves abhängig, der Abstand der Slots (Offset) von der Genauigkeit der Synchronisation.
Aufgrund der Kollisionsfreiheit des Systems, können verschiedenste Netzwerktopologien mittels Hubs realisiert werden.
Hubs sollten Switches vorgezogen werden, da Switches die Pakete analysieren ,was bei dem TDMA Verfahren überflüssig ist und zu höheren Latenzzeiten führt.

TDMA -Polling

Das Polling Verfahren arbeitet ähnlich wie das TDMA Verfahren, allerdings sendet der Master vor jedem Slot dem jeweiligen Slave eine Aufforderung zum Senden. Die Slaves dürfen also nur nach Aufforderung senden, dadurch sind für jeden Sendevorgang zwei Nachrichten notwendig, die Aufforderung (Request) und die tatsächliche Nachricht (Response) (Abbildung 8).

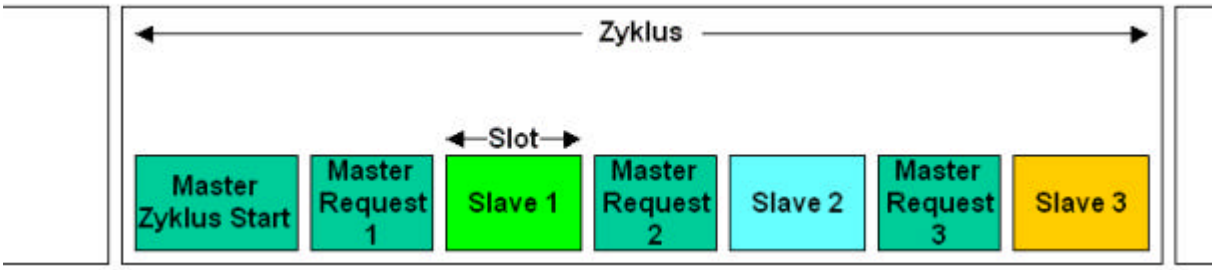


Abbildung 8 Polling Verfahren

Token passing

Im Token passing Verfahren [Göhring, Kauffels] wird unter den Teilnehmern ringförmig ein Token weitergegeben (Abbildung 9). Nur der Teilnehmer, welcher im Besitz des Tokens ist, darf Pakete senden bzw. an das Token anhängen, welcher dann, als “besetzt“ markiert, weitergereicht wird. Erhält der nächste Teilnehmer das besetzte Token, prüft er, ob er der Empfänger ist. Ist er nicht als Empfänger eingetragen, sendet er das Token weiter. Kommt das Token am Empfänger an, so kopiert dieser die Daten, markiert das Token als “empfangen“ und schickt es weiter. Es bleibt dann solange “besetzt“, bis es wieder am Sender ankommt. Dieser prüft, ob die Daten mit den gesendeten Daten übereinstimmen und empfangen wurden. Dann setzt er das Token auf “frei“. Stimmen die Daten nicht überein, so werden die Daten bis zu drei mal erneut gesendet, danach wird von einem Fehler ausgegangen.

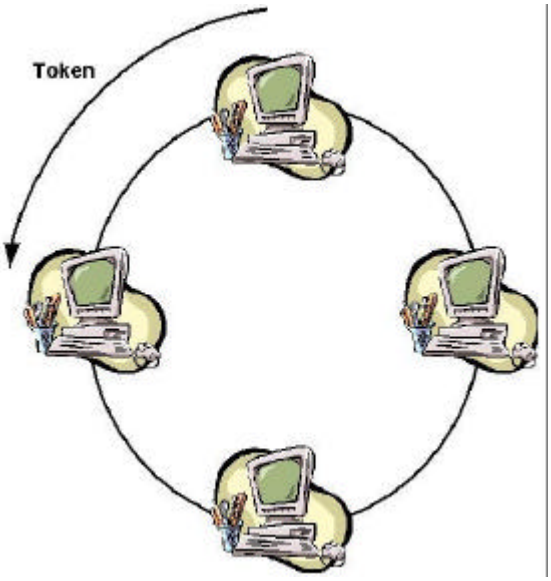


Abbildung 9 Token passing

Das Token passing Verfahren ist deterministisch aber auch sehr fehleranfällig. Es müssen daher Vorkehrungen gegen folgende Fehler getroffen werden:

- Verlust des Tokens
- Doppeltes Token
- Ausfall eines Teilnehmers
- Endlos kreisendes Paket (bei Verlust des Empfängers)

Mit steigender Anzahl der Teilnehmer des Netzes steigen allerdings auch stark die Latenzzeiten, da die Laufzeit des Tokens von der Anzahl der Teilnehmer abhängt.

Synchronisierungsverfahren - Time Stamp IEEE 1588

„Eine präzise Zeitbasis ist insbesondere für verteilte Systeme in der Automation wichtig. Das in IEEE 1588 spezifizierte Precision Time Protocol (PTP) synchronisiert Uhren genauer als eine Mikrosekunde. Es ist zugeschnitten auf lokale Netzwerke wie Ethernet, die Multicast Nachrichten erlauben. Die Ansprüche an die lokalen Uhren und an die Netz- und Rechenkapazität sind vergleichsweise bescheiden.“[Weibel 04]

Das Verfahren Time Stamp nach IEEE 1588 synchronisiert die Systemuhren in einem Netzwerk.
Zu Beginn wird die am besten verfügbare Uhr als Master Clock (Referenz) ausgewählt, Diese sendet an alle anderen Uhren (Slave Clocks) Synchronisationsmeldungen.
Da während der Übertragung der Meldungen im Netzwerk Zeit vergeht, wird diese gemessen und bei der Synchronisation berechnet.

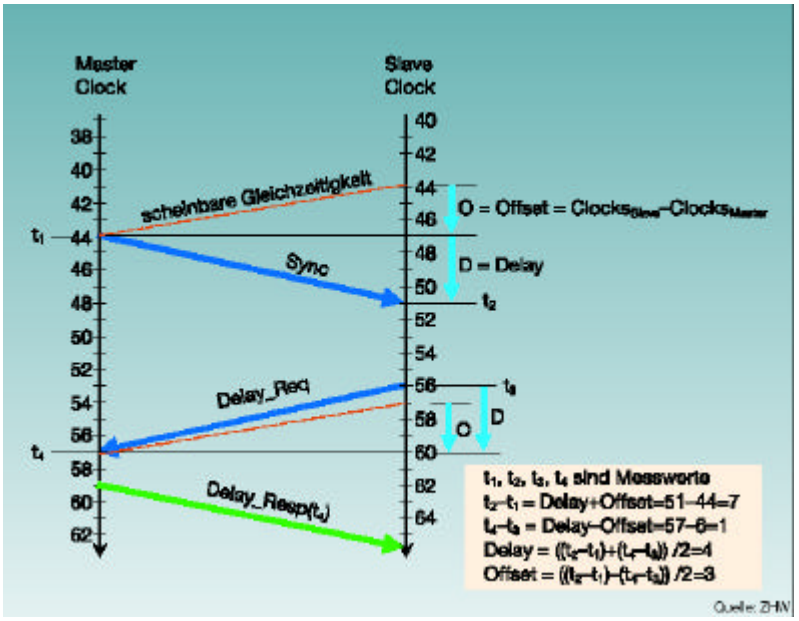


Abbildung 10 Synchronisation IEEE1588

Abbildung 10 zeigt das Abgleichen einer Slave Clock mit der Master Clock.
Der Master sendet eine Sync-Nachricht (t1) an den Slave, welche, angekommen beim Slave, veraltet ist (t2).

Der Slave sendet seine eigene Zeit (t3) als Delay-Req-Nachricht an den Master zurück, dieser misst die Zeit des Eingangs der Nachricht (t4) und sendet sie an den Slave.
Aus den vier Zeiten berechnet der Slave die Übertragungszeit (Delay), wonach er seine Uhr stellt, und die Abweichung (Offset), um die Geschwindigkeit der Uhr auf die Geschwindigkeit der Master-Uhr einzustellen.
Der tatsächliche Algorithmus ist ein wenig komplexer, da schon vom Messen der Zeit bis zum Absenden ein Delay entsteht, deshalb wird die Zeit dort ein zweites Mal gemessen und als "follow-up" Nachricht nachgeschickt. [Weibel 04]

Hardware Verfahren - Kollisionsfreies Ethernet über Switches

Im Gegensatz zu Hubs, welche ankommende Datenpakete an alle Ports weiterleiten (Broadcast), schicken Switches die Datenpakete nur an den Empfänger. Dazu wird im Switch eine Tabelle angelegt, an welchem Port welche Teilnehmer liegen. Senden zwei Teilnehmer an den gleichen Empfänger, so hat der Switch die Möglichkeit Datenpakete zu puffern. Im Voll-Dulex-Modus kann über den gleichen Port sowohl gesendet als auch empfangen werden. Dadurch ist die Gefahr von Kollisionen ausgeschlossen.
Der Nachteil von Switches ist, dass die Latenzzeit höher ist als die Zeiten der Hubs. Switches müssen erst die MAC Nummer des Datenpakets analysieren, um es dann an den richtigen Port weiterzuleiten. Wenn Pakete mehrerer Teilnehmer das gleiche Ziel haben, müssen sie diese Pakete puffern.
Um deterministische Eigenschaften garantieren zu können, müssen allerdings Algorithmen implementiert werden, welche die Abarbeitung der Datenpakete im Switch regeln.

3.3 Echtzeit Ethernet Produkte

Aufgrund der Vorteile und dem Bestreben einflussreicher Firmen (Siemens, VW, etc) hat sich eine Nachfrage nach Echtzeit Ethernet Produkten auf dem Markt gebildet.
Von Open Source Lösungen, die rein auf herkömmlicher Hardware basieren, bis zu proprietären Produkten, welche extrem geringe Zeiten mittels spezieller Hardware erreichen, ist alles vertreten.
Die folgende Beschreibung verschiedener Lösungen ist beispielhaft und hat nicht den Anspruch auf Vollständigkeit.
RTnet sowie Profinet werden in den Kapiteln 3.4 und 3.5 genauer beschrieben.

Die folgende Tabelle soll einen kleinen Überblick über einige Produkte verschaffen.

	Ethercat	JetSync	Ethernet Powerlink	Profitnet RT	Profinet IRT	RTnet
Hardware	speziell	Standard	Standard	Standard	speziell	Standard
Open Source	nein	-	Mitglieder	PNO Mitglieder	PNO Mit-glieder	Ja
Linienstruktur möglich	Ja	Nein	Ja	Nein	Ja	Ja
Verfahren	siehe Text	Zeit - Stempel	Polling	*ASICS*	Switches	Zeitscheibe

[Quelle: Pfeifer 03]

31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik

3.3.1 EtherCat

Das Funktionsprinzip von EtherCat ähnelt eher dem Prinzip eines Feldbusses als dem eines Ethernets. Auf Seiten des PCs werden zwar nur Standard Ethernetkarten eingesetzt, die niedrigen Latenzzeiten werden aber dadurch erreicht, dass die I/O Geräte mittels spezieller Hardware auf niedrigster Ebene angesprochen werden. Ein Ethernet Telegramm durchläuft so mehrere Slaves (EtherCat - Klemmen auf FPGA Basis), welche sich während des Durchlaufs die Ihnen zugehörigen Daten aus dem Telegramm entnehmen bzw. hinzufügen (Abbildung 11) [www.beckhoff.com].

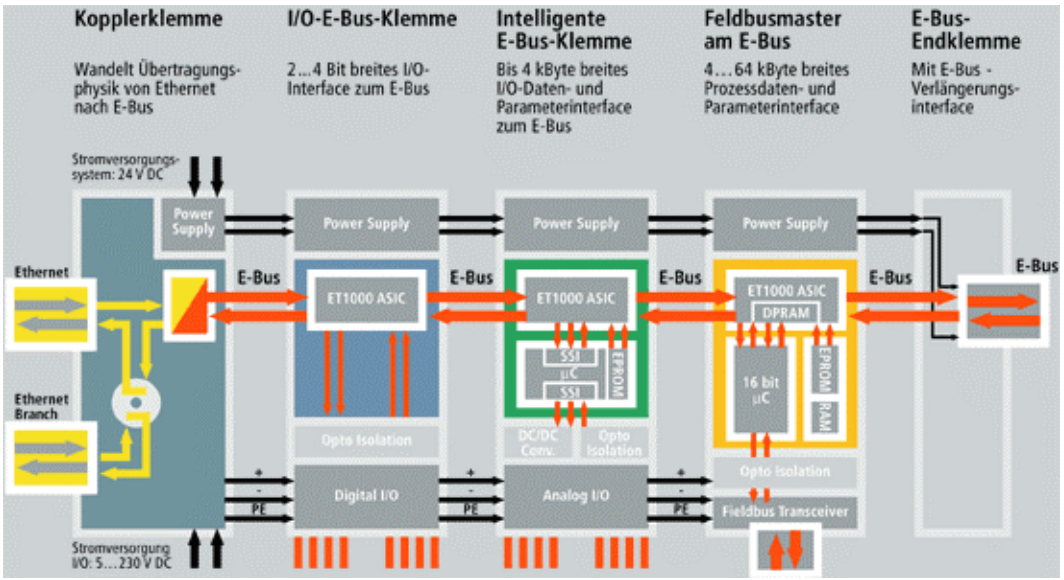


Abbildung 11 - Ethercat Klemme [www.beckhoff.com]

Je nach Anwendung können so mit einem Telegramm bis zu 1500 digitale I/O Geräte angesprochen werden (bei einem Byte pro Gerät, da maximale Paketlänge bei Ethernet 1500Byte + Header).
EtherCat scheint somit für die Kommunikation zwischen mehreren Echtzeit PCs nicht sehr geeignet.
“EtherCat ist ein Feldbussystem auf Ethernet Basis, welcher mittels einer herkömmlichen Ethernet Karte angesprochen werden kann. Mögliche Einsatzgebiete sind die Steuerung einer Anlage mittels eines Computers oder die Messdatenerfassung“ [Presseinformation EtherCat Technologiegroup 04]. Gleiches lässt sich aber auch mit PCI Feldbuskarten erreichen, da Ethercat Feldbusdaten in ein Ethernetframe verpackt, die Slaves aber auf die Daten genauso wie auf einen Feldbus zugreifen. Ethercat ist eher ein Feldbus, welcher sich mit einer Ethernetkarte steuern lässt.

3.3.2 JetSync

Die genaue Synchronisation der Teilnehmer des Echtzeitnetzwerkes ist das Prinzip von JetSync. Der Master sendet Synchronisierungstelegramme an die Slaves, welche mittels eines nicht näher öffentlich spezifizierten “Statistik Algorithmus“ ihre lokalen Uhren synchronisieren.
Zeitkritische Daten bekommen einen Zeitstempel und werden über TCP/IP verschickt, da dieses Verfahren keine Übertragungszeiten garantiert, müssen die Daten anhand des Zeitstempels synchronisiert werden.
Da TCP/IP eingesetzt wird und somit keine Übertragungszeiten garantiert werden können, kann man JetSync kaum als Echtzeit Ethernet betrachten, sondern eher als synchronisiertes Ethernet.

3.3.3 Ethernet Powerlink Protected Mode

Ethernet Powerlink garantiert Echtzeitfähigkeit durch ein erweitertes Polling Verfahren. Zu Beginn des Frames werden während der “Start Phase“ die Uhren synchronisiert. Dann folgt die “Isochrounous Phase“, in der die Echtzeitdaten mittels des Polling Verfahrens ausgetauscht werden.
Wenn alle Stationen gesendet haben, beginnt die “Asynchrounous Phase“, in der Nicht-Echtzeitdaten wie Parametrierungen oder Konfigurationen ausgetauscht werden können. Ein Slave, welcher in dieser Phase senden möchte, muss sich dafür während seines Echtzeitslots beim Master anmelden. Er erhält dann eine weitere Zeitscheibe zugeteilt. (Invite).
Am Ende der “Asynchrounous Phase“ folgt meist noch eine “Idle Phase“, um die Zyklen gleich lang zu halten, und somit einen geringen Jitter aufzuweisen (Abbildung 12).
Um bei mehreren Slaves die Zykluszeiten gering zu halten, besteht die Möglichkeit, Teilnehmer, welche nicht mit jedem Zyklus Zeitkritische Daten senden müssen, nur jedem n-ten Zyklus einem Slot zuzuordnen. So können sich mehrere Stationen einen Slot teilen. [Pfeifer 03]

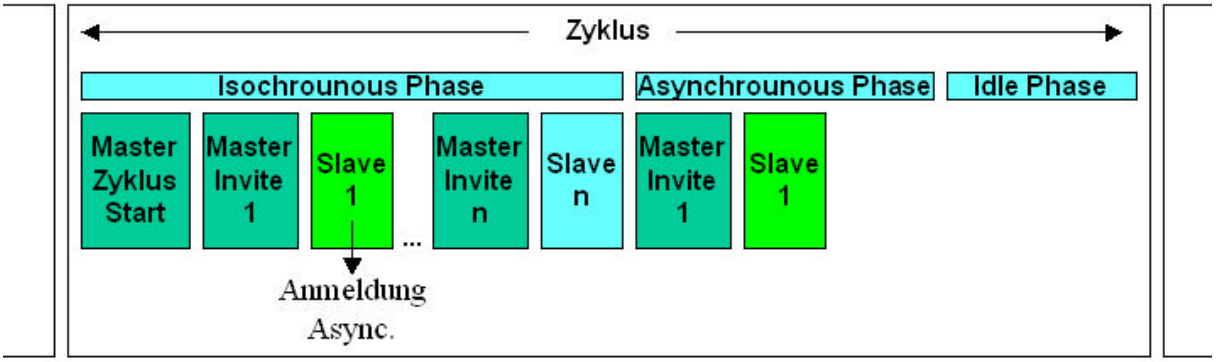


Abbildung 12 - Powerlink Zugriffsverfahren

31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik

3.4 RTNET

3.4.1 Konzept

RTnet folgt dem Konzept, die Lücke zwischen von spezieller Hardware gestützten Lösungen für extrem kurze Übertragungszeiten und abgespeckten Lösungen, welche nur in wohl dimensionierten und kontrollierten Lastsituationen maximale Übertragungszeiten mit akzeptabler Wahrscheinlichkeit versprechen können, zu schließen. RTnet ist herstellerunabhängig und unterliegt der GNU/GPL. Es wird von der Universität Hannover und einigen weiteren Entwicklern weltweit weiterentwickelt.
Es bietet ein Gerüst aus Protokollen und Werkzeugen für die Dimensionierung, Betrieb, Wartung und Protokollierung eines echtzeitfähigen Netzwerks.
Die Computer können mittels Hubs zu den gewünschten Topologien vernetzt werden (Abbildung 13). Aufgrund des verwendeten TDMA Verfahrens muss das RTnet von anderen Netzten getrennt werden, es ist jedoch möglich zeitunkritische Daten durch das Netz zu tunneln und über ein Gateway von anderen Rechnern darauf zuzugreifen.

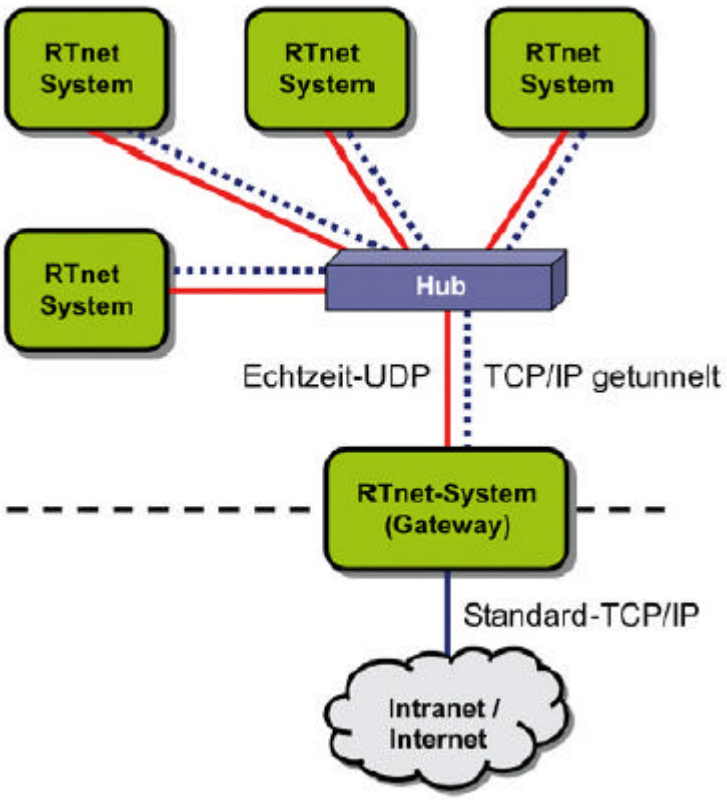


Abbildung 13 RTnet Topologie [Kizka 05]

3.4.2 Funktionsweise

Hardware

RTnet unterstützt bisher zehn verbreitete Netzwerkkarten Chipsätze.
Weitere Treiber können selber entwickelt werden.

Paketmanagement

Pakete, die gesendet werden sollten, werden von der Sende-Task durch den Stack geschleust, wohingegen eingehende Pakete vom Netzwerk-Treiber an einen RTnet Paket-Manager übergeben werden, welcher die Pakete nach ihrem Protokolltyp an die zuständige Funktion übergibt. Die Priorität des Handlers muss somit immer höher sein als andere RTnet nutzende Applikationen.
Der Stack und der Treiber nutzen eine spezielle Datenstruktur (rtskb) für den Puffer.
Alle rtskb Strukturen haben eine feste Größe (MTU) und werden während des Setups alokiert.
RTnet Anwendungen müssen einen eigenen rtskb Pool erzeugen.

UDP-Stack

RTnet basiert auf einem eigenen UDP Stack, welcher auf Echtzeit Eigenschaften getrimmt wurde.
Das dynamische ARP (Adress Resolution Protocol) wird nur noch statisch während des Setups ausgeführt, der Routing Prozess wurde vereinfacht, die Routing-Tabellen für die limitierte Anzahl der RTnet Einträge optimiert.
Die Kommunikationsdaten der Echtzeit Anwendung werden nach Durchlaufen des Stacks dem Rtmac (**R**eal **T**ime **M**edia **AC**cess) Modul übergeben, der mit dem Treiber der Netzwerkkarte kommuniziert (Abbildung 14).
Nicht-Echtzeit Anwendungen kommunizieren weiterhin mit dem Linux Stack, welcher über einen virtuellen Ethernettreiber (vnic) auf das Rtmac Modul umgelenkt wird. Dieser virtuellen Schnittstelle kann mittels ifconfig auch eine eigene IP zugewiesen werden.

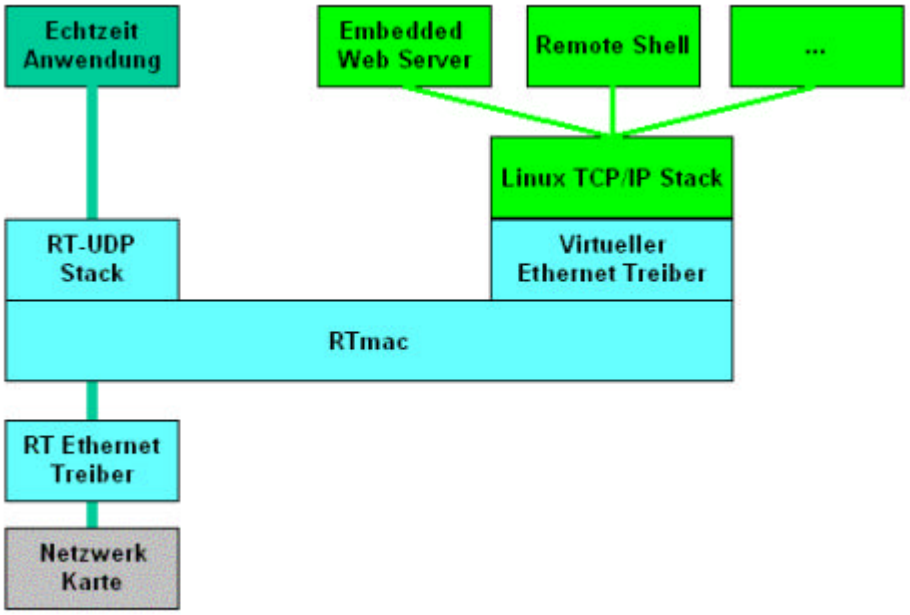


Abbildung 14 RTnet UDP-Stack

RTMAC

Das Modul Rtmac ist zuständig für den Determinismus des Medienzugriffs (Media Acess control (MAC)) von RTnet. Es fängt die Pakete ab, bevor sie den Kartentreiber erreichen, und übergibt sie dem Zugriffsverfahren.
Das MAC Zugriffverfahren ist in RTnet als TDMA Verfahren implementiert, durch das modulare Design können allerdings auch andere MAC Verfahren implementiert werden.
Wenn Rtmac geladen ist, hat es die exklusive Kontrolle über die Übertragungen des Netzwerk Treibers.
Der Rtmac Paket Header:
Rtmac Pakete sind als Ethernet Typ 0x9021 identifizierbar.
Das Version Feld enthält 0x02 zur Identifikation der Rtmac Version
Die Flags signalisieren, ob es sich um Nicht Echtzeitdaten (0) oder Echtzeitdaten handelt (1-7).

Typ 2 Byte	Version: 0x02 1 Byte	Flags 1Byte

Abbildung 15 RTmac Header

31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik

3.4.3 TDMA

Das Time Division Multiple Access Verfahren ist als Master / Slave Methode implementiert.
Der Master sendet periodisch Synchronisierungspakete (SOF Start of Paket), die dem Netzwerk den Beginn eines TDMA – Rahmens signalisieren. Der Master kann mittels eines oder mehrerer Backup Master redundant ausgelegt werden, diese springen ein, falls der Master ausfällt.
Alle Teilnehmer des Netzwerkes haben exklusive Zeitfenster, relativ zum SOF in welchen sie Daten senden dürfen.

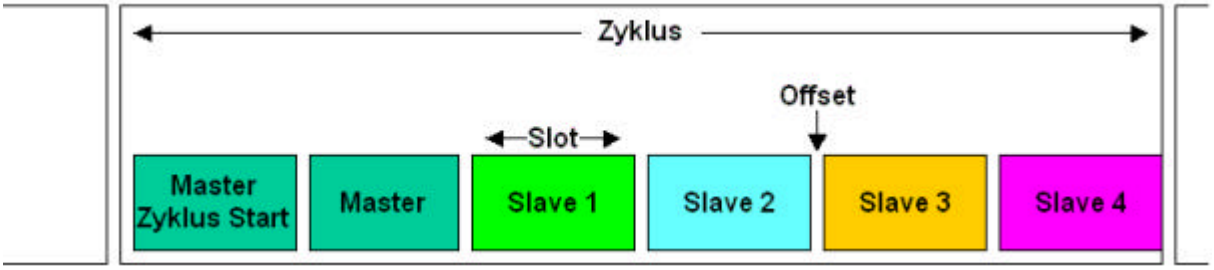


Abbildung 16 TDMA Verfahren allgemein

Zusätzliche Clients können in das laufende Echtzeit Netzwerk hinzugefügt werden. Dazu müssen sie nur in die Liste des Servers eingetragen werden, der Client meldet sich beim Server, wird mit Parametern versorgt und mit in das Zugriffsverfahren aufgenommen.
TDMA Pakete sind im Rtmac Header Frame als Typ 0x0001 spezifiziert.
Durch diese strikte Aufteilung der Übertragungszeit ist RTnet deterministisch.
Um dennoch Anbindungen an nicht deterministische Netze (Internet, Intranet (TCP/IP) zu ermöglichen, muss ein Router die Verbindung zwischen den Netzen regeln.

Weiterhin ist es möglich, mehreren Teilnehmern den gleichen Slot zuzuweisen, so dass die jeweiligen Clients nur jeden n-ten Zyklus den Slot benutzen dürfen.
Diese Einstellungen müssen jedoch den Clients bekannt sein, ist eine zentralisierte Parametrisierung gewünscht, so müssen diese Daten mittels des in Kapitel 3.4.4 beschriebenen RTcfg Konfigurations-Services bekannt gemacht werden.

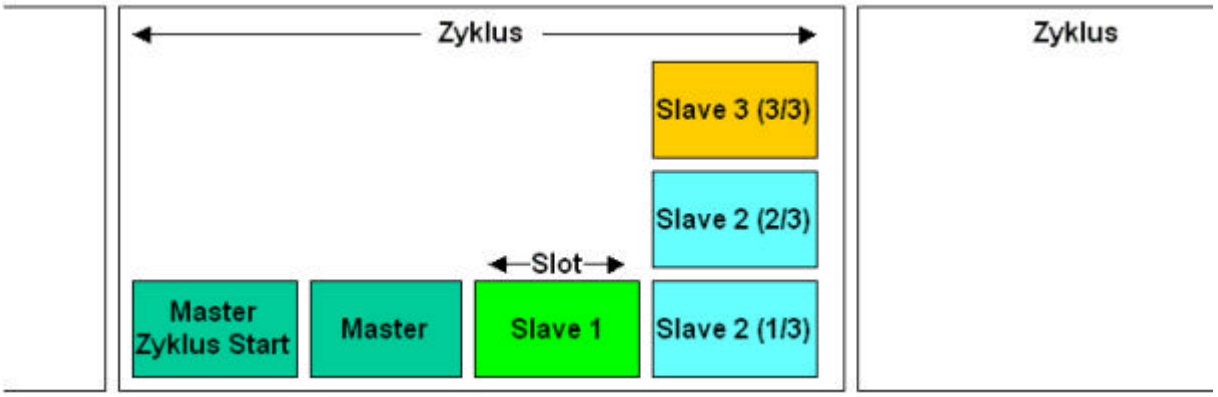


Abbildung 17 - RTnet TDMA Slots

31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik

Startvorgang / Synchronisierung:

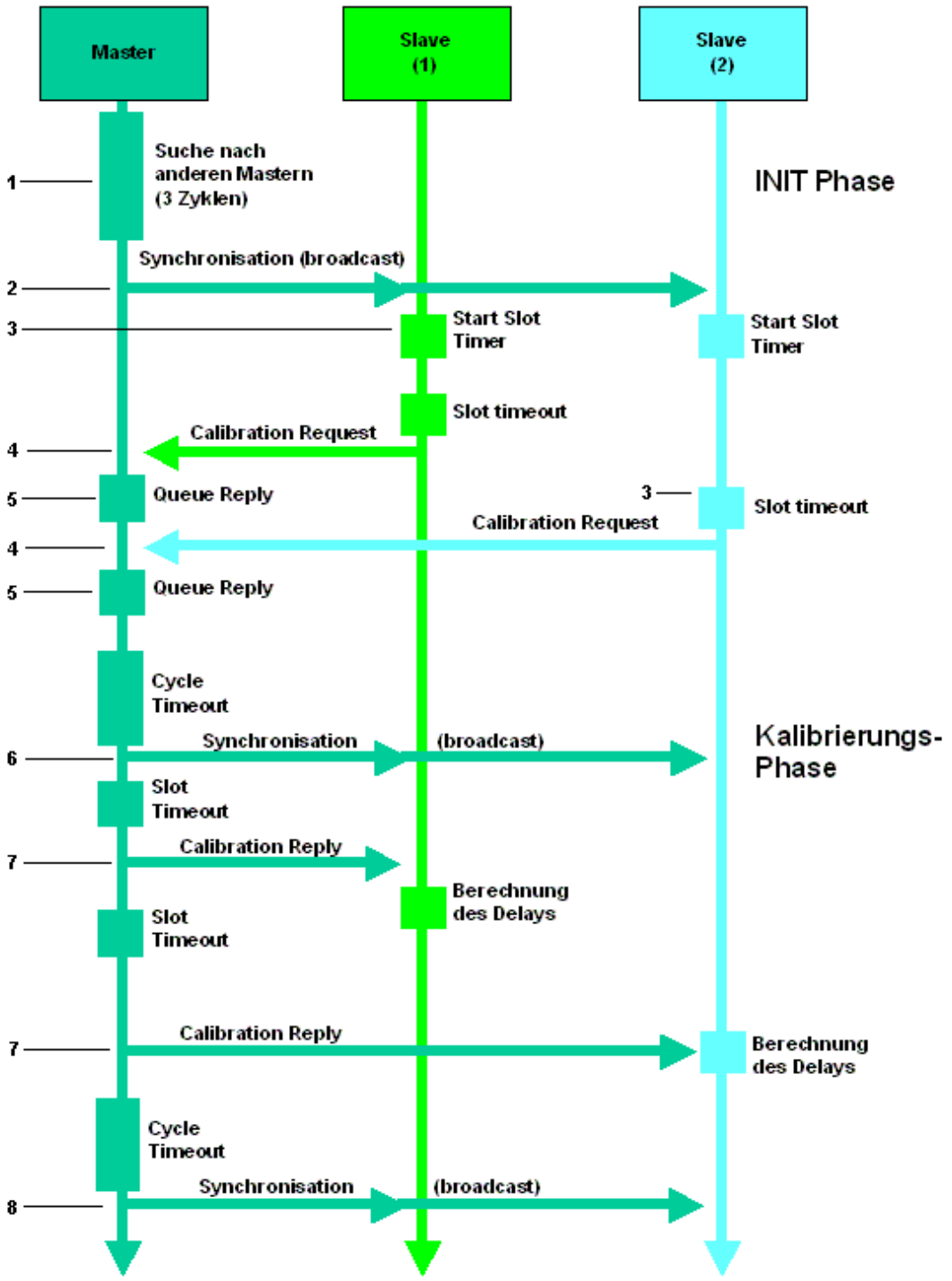


Abbildung 18 TDMA - Startvorgang

1. Der Master sucht drei Zyklen lang nach anderen Mastern – falls es sich um einen Restart handelt und ein Backup Master übernommen hat.
2. Der Master sendet ein Synchronisationsframe.

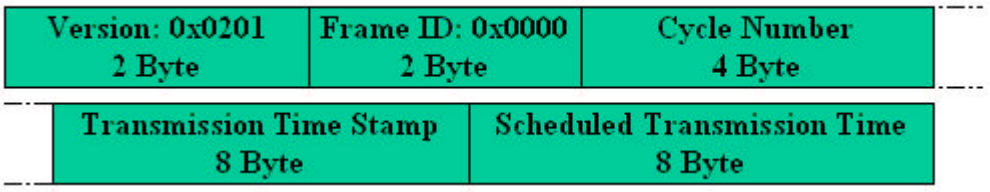


Abbildung 19 TDMA - Frame – Synchronisierungsframe

Die Frame ID identifiziert die Art des TDMA Frames.
Das Feld “Cycle Number“ zählt die Zyklen und wird bei einem Überlauf auf Null gesetzt.
“Transmission Time Stamp“ ist die Master Zeit, wann das Paket gesendet wurde (möglichst zeitnah an der physischen Übertragungsschicht) , die “Scheduled Transmission Time“ ist der Zeitpunkt, als die Sendung auf Anwendungsebene initiiert wurde.
Durch Vergleichen der beiden Zeiten können die Empfänger des Synchronisierungsframes die Bearbeitungszeit des Masters kompensieren.
Backup Master verwenden die “Scheduled Transmission Time“ für ihre Synchronisationsframes, dadurch kompensieren die Slaves automatisch den Zeitunterschied.

3. Die Slaves starten ihre Slot Timer (Beginn des TDMA Zyklus).
4. Nach Ablauf des jeweiligen Slot Timers sendet der jeweilige Slave ein “Calibration Request“ Frame an den Master, um Kalibrierungsdaten zu bekommen.

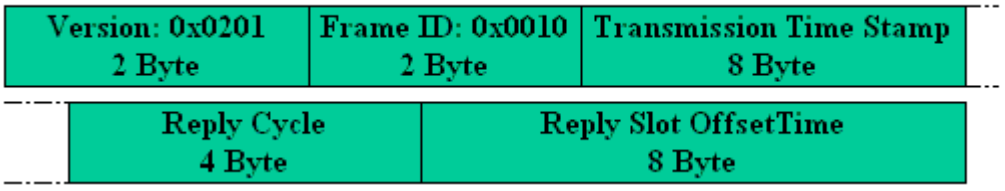


Abbildung 20 TDMA - Frame Anforderung Kalibrierungsdaten (Request Frame)

Der Inhalt des “Transmission Time Stamp“ ist wie in Punkt 2 beschrieben.
Der Slave teilt dem Master mit “Reply Cycle“ und “Reply Slot Time Offset“ mit, in welchem TDMA Zyklus und in welchem Slot die Kalibrierungsdaten gesendet werden sollten. Der Slave kann nur Slots angeben, welche ihm als SendeSlots zugewiesen sind und darf in diesen dann auch nicht senden.

5. Der Master puffert die Kalibrierungsanfragen.
6. Ende des TDMA Zyklus, der Master sendet das Synchronisationsframe für den neuen Zyklus.
7. Der Master schickt die angeforderten Kalibrierungsdaten an die Slaves.

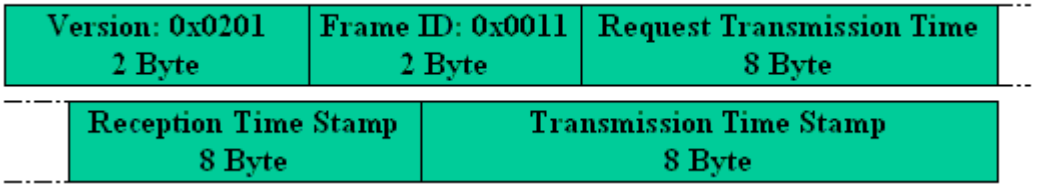


Abbildung 21 TDMA - Frame – Kalibrierungsdaten (Reply Frame)

Der Inhalt des “Transmission Time Stamp“ ist wie in Punkt 2 beschrieben.
Der Inhalt des “Transmission Time Stamp“ des “Calibration Request“ Frames wird in das “Request Transmission Time“ Feld der Kalibrierungsdaten kopiert.
Im Feld “Reception Time Stamp“ wird die Zeit der Annahme (nah am physischen Layer) des “Calibration Request“ Frames gespeichert.
Die Slaves berechnen den Verzug in der Übertragungszeit.
(Abbildung 22)

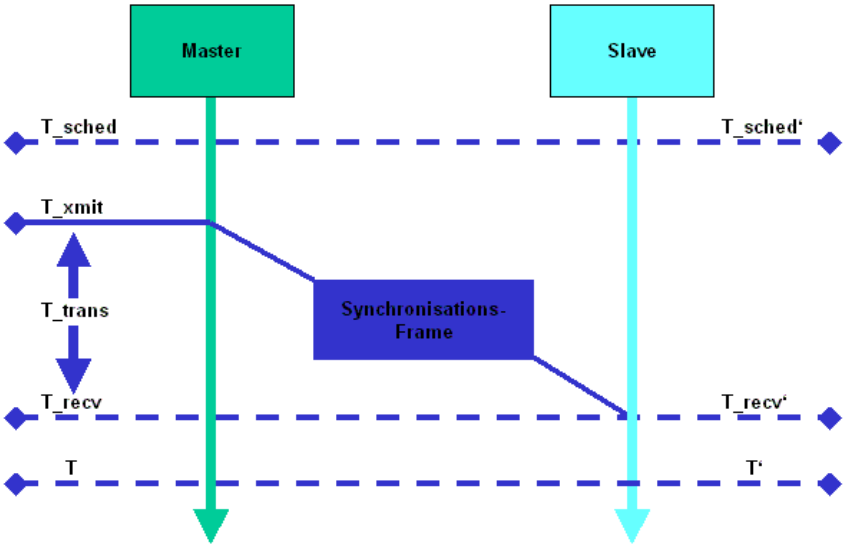


Abbildung 22 TDMA - Synchronisation der Uhren

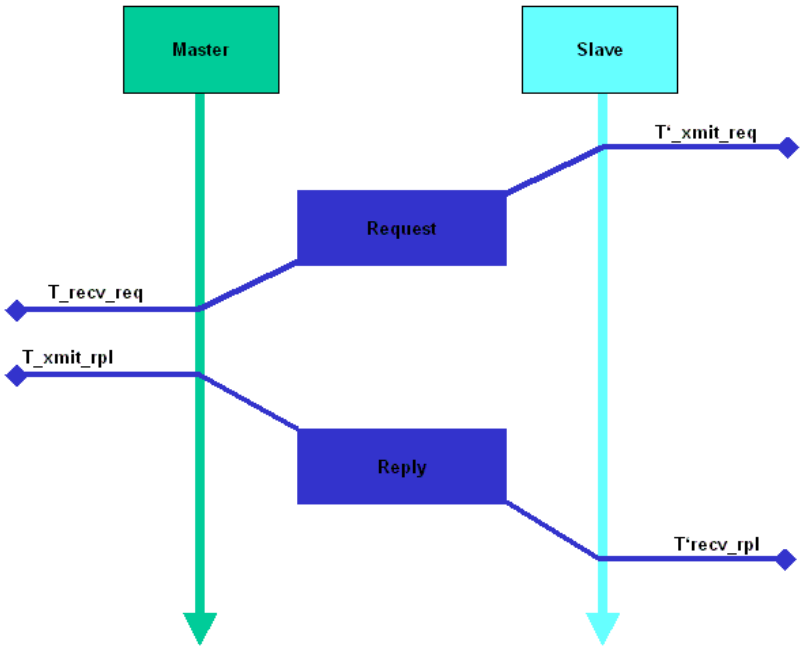
Berechnung des Zeitunterschieds (offset):
offset = T_recv – T’recv = Txmit + t_trans – T’recv

Berechnung der Master Zeit
T = T’ + offset

31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik

Berechnung eines Zeitpunktes relativ zum Synchronisierungsframe
T' = T'_sched + t = T_sched – offset + t

- t_sched:
  Zeitpunkt der Initiierung der Übermittlung auf Anwendungsebene
  "Scheduled Transmission Time"
- t_xmit:
  Zeitpunkt der Übermittlung auf der physischem Übertragungsschicht
  "Transmission Time Stamp"
- t_trans:
  Übertragungszeit auf der physischen Übertragungsschicht
- t_recv:
  Zeitpunkt des Empfangs (Master)
- t_recv':
  Zeitpunkt des Empfangs (Slave)
- offset:
  Zeitunterschied Master / Slave
- T / T':
  Zeitpunkt Master / Slave



31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik

Abbildung 23 TDMA - Berechnung Übertragungszeit

Berechnung der Übertragungszeit:
t_trans = * ((T'_recv_rpl – T'_xmit_req) – (T_xmit_rpl – T_recv_req))

- T'xmit_req :
  Zeitpunkt der Übermittlung des Request Frames vom Slave
  Transmission Time Stamp (Slave – Request Frame)
- T_recv_req :
  Zeitpunkt der Annahme des Request Frames am Master
  Reception Time Stamp (Master – Reply Frame)
- T_xmit_rpl :
  Zeitpunkt der Übermittlung des Reply Frames vom Master
  Transmission Time Stamp (Master – Reply Frame)
- T'recv_rpl:
  Zeitpunkt der Annahme des Reply Frames am Slave

8. Ende des TDMA Zyklus.

Die Kalibrierungsphase wird mehrmals wiederholt, um den Durchschnittlichen Delay (Signallaufzeit) zu ermitteln. Die Anzahl der Durchläufe basiert auf der angenommenen Varianz der Zeiten. Einzustellen mit –c calibration round (8.3.x.x)
Da nur die Übertragungszeit vom Master gemessen wird, sollten Backup Master so gewählt werden, dass sie ähnliche Zeitcharakteristiken aufweisen, d.h. die Übertragungszeiten zu den Slaves, sowie Timerabweichungen (Hardware) sollten ähnlich der Eigenschaften des Master sein.

Undefinierter Zustand des Masters

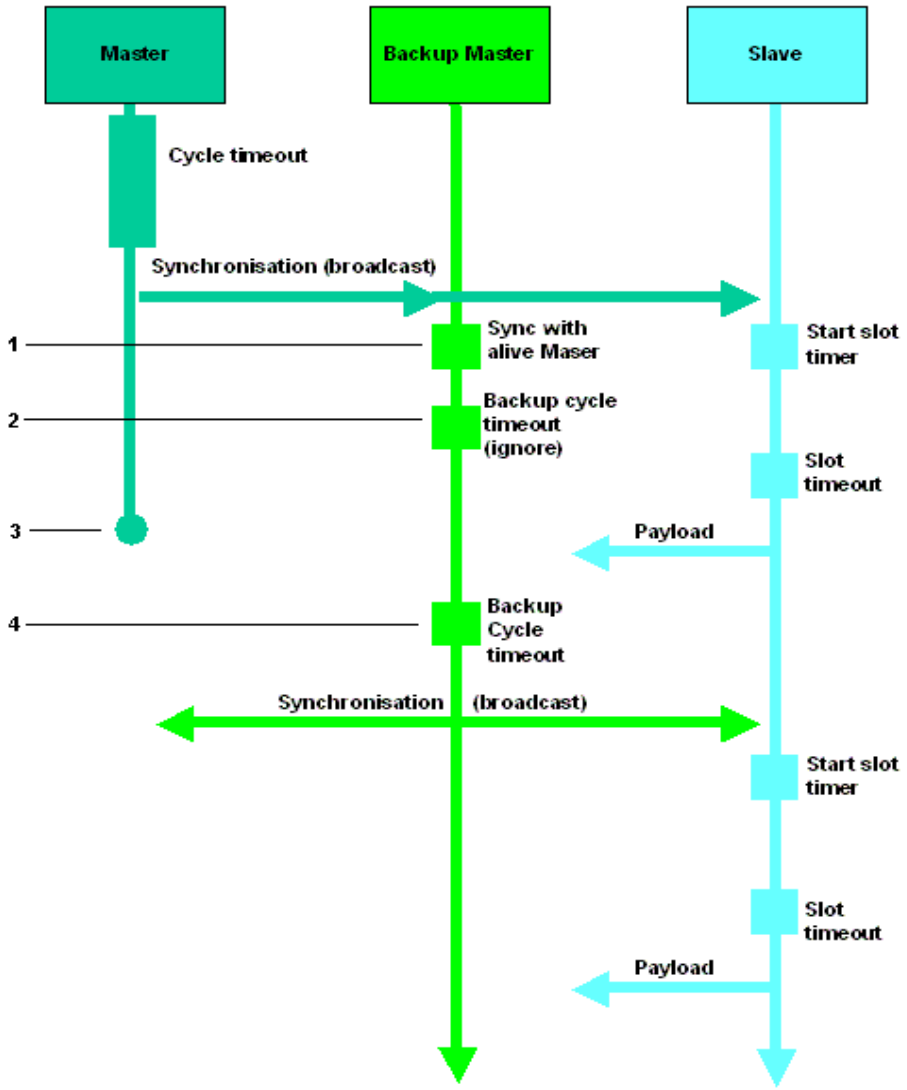


Abbildung 24 TDMA - Absturz des Masters

1. Der Backup Master empfängt den Synchronisierungsframe des Masters und synchronisiert sich.
2. Der Backup Master ignoriert seinen Cycle Timeout (Ende des Backup TDMA Zyklus).
3. Der Master begibt sich in einen undefinierten Zustand.
4. Der Backup Master empfängt kein Synchronisierungsframe des Masters und übernimmt nach Ablauf seines Zyklus Timers (Cycle Timeout) die Synchronisation des Netzwerkes.

Neustart des Masters

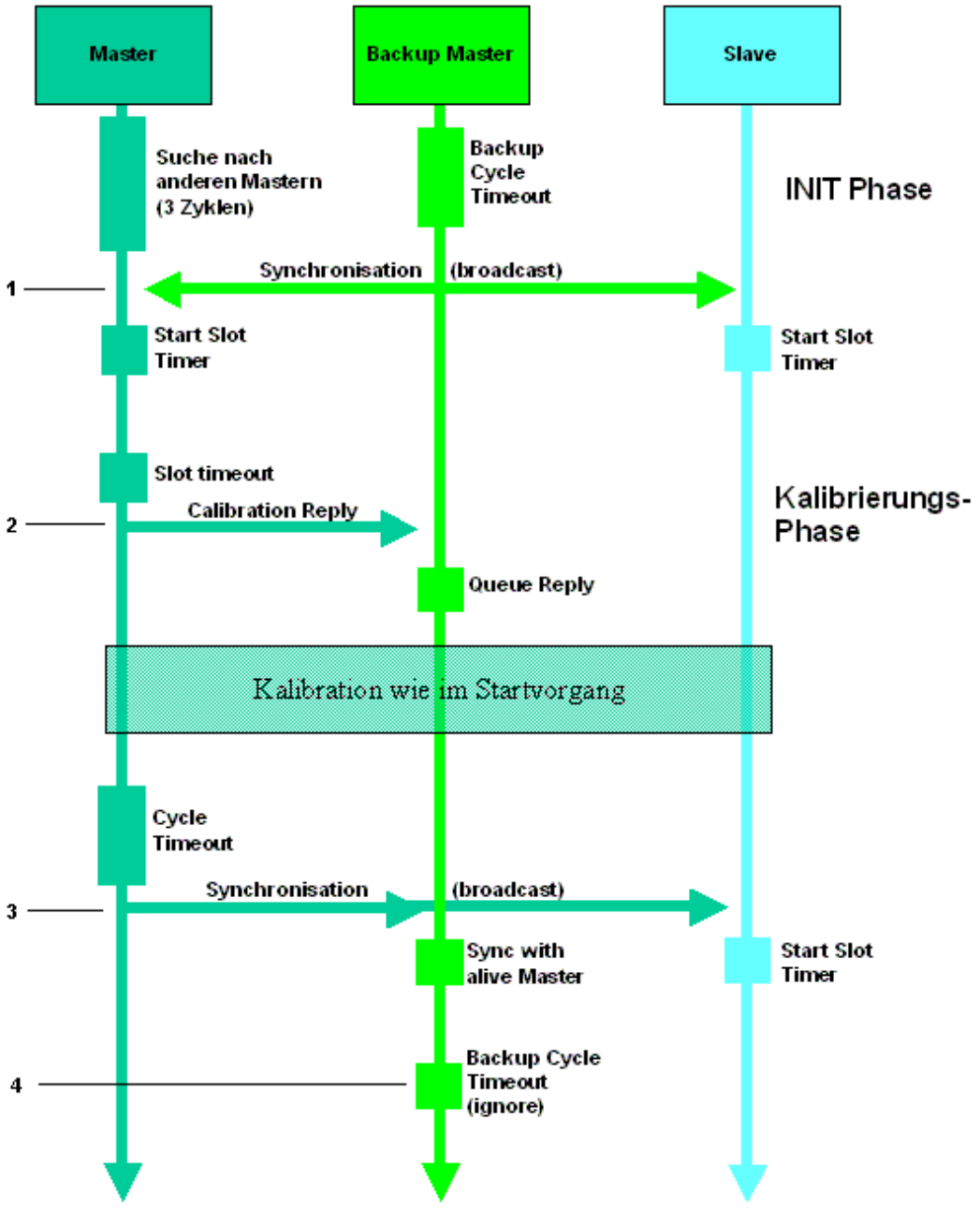


Abbildung 25 TDMA - Master Neustart

1. Der Master empfängt die Synchronisation des Backup Masters und wartet auf seinen Slot, um das Netz nicht zu unterbrechen.
2. Der Master kalibriert sich mittels des Backup Masters.
3. Der Master übernimmt wieder die Synchronisierung des Netzes.
4. Der Backup Master ignoriert seinen Cycle Timeout und überlässt dem Master die Synchronisation.

3.4.4 RTCFG

RTcfg ist ein Konfigurationssystem für RTnet. Es ermöglicht das Aufsetzen und Versorgen von RTnet mit beliebigen benutzerspezifischen Daten, z.B. Parameter und Befehle für nicht RTnet Anwendungen.
RTcfg Pakete sind als Ethernet Typ 0x9022 spezifiziert und dürfen nicht größer sein als die MTU (Maximum Transmission Unit) des Netzwerks. (zumeist: 1500Byte)

Der Startvorgang

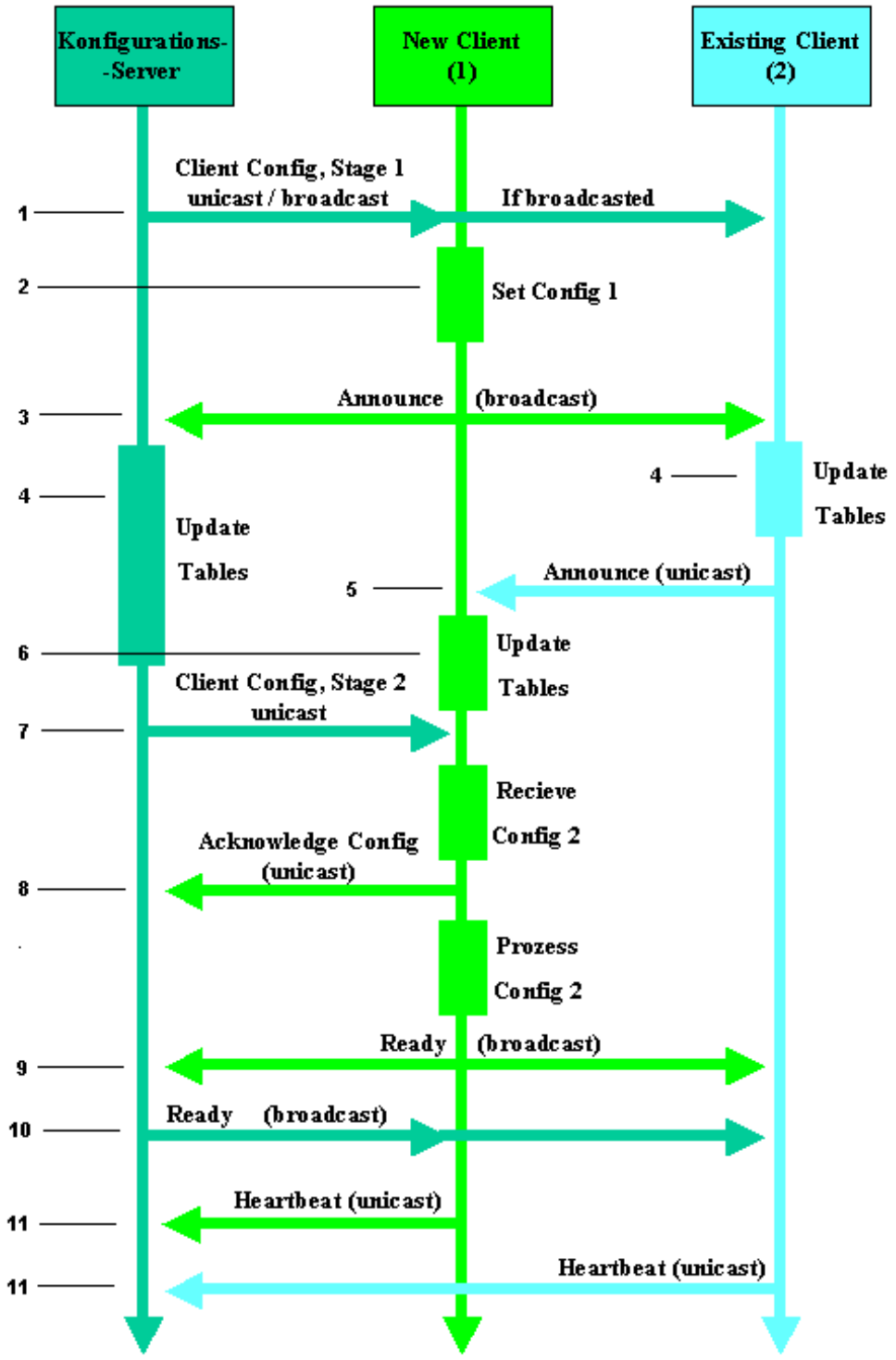


Abbildung 26 RTcfg - Startvorgang

1. Der Konfigurations-Server sendet die Stage 1 Konfigurationsdaten an den neuen Client, wenn die MAC Adresse bekannt ist, sonst sendet er an alle eingetragenen Clients.

ID: 0 1 Byte	Client Adressen Typ 1 Byte	Client Adresse variabel	Server Adresse variabel

Stage 2 Burst Rate 1Byte	Konfigurationsdaten Länge 2 Byte	Konfigurationsdaten variabel

Abbildung 27 RTcfg - Frame - Konfiguration Stage 1

Die Stage 2 Burst Rate bestimmt über die Anzahl der Pakete, welche der Server sendet, ohne eine Acknowledge Nachricht bekommen zu haben.
Die Stage 1 Konfigurationsdaten enthalten Parameter oder Shell Kommandos, welche vom Client benötigt werden, um Teil des RTnet Netzes zu werden. Werden keine Konfigurationsdaten gesendet (z.B.: Rtmac wird nicht genutzt), wird das Feld Länge auf Null gesetzt.

2. Der Neue Client (Client 1) übernimmt die Konfiguration
3. Der neue Client sendet ein “new-announce“ Paket (announce = bekannt machen) an alle.

ID: 1 1 Byte	Client Adressen Typ 1 Byte	Client Adresse variabel	Flags 1Byte	Stage 2 Burst Rate 1Byte

Abbildung 28 RTcfg - Frame - new announce

Flags: 0 - Anforderung verfügbarer Stage 2 Daten
1 – Client fertig (keine explizite Fertig Meldung wird gesendet)

4. Der Server und der schon existierende Client (Client 2) erneuern ihre Listen (ARP und routing)
5. Client 2 sendet ein “reply-announce“ Paket an Client 1

ID: 2 1 Byte	Client Adressen Typ 1 Byte	Client Adresse variabel	Flags 1Byte	Padding Field variabel

Abbildung 29 RTcfg - Frame - reply announce

Flags: 0 - Anforderung verfügbarer Stage 2 Daten
1 – Client fertig (keine explizite Fertig Meldung wird gesendet

6. Client 1 erneuert seine Listen (ARP und routing)

31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik

7. Der Server sendet die Stage 2 Konfiguration an Client 1

Abbildung 30 RTcfg - Frame - Konfiguration Stage 1

Das Feld “Aktive Stationen“ enthält die Anzahl der Aktiven Stationen mit Server allerdings ohne den Client 1 (in diesem Beispiel). Es wird genutzt, um festzustellen, ob alle anderen Stationen ein Paket gesendet haben (Fertig Pakete, etc).
Das Feld “Heartbeat Periode“ enthält die Periode des Clients, nach welcher er Heartbeat Pakete schicken soll. Wird dies nicht genutzt, so steht das Feld auf Null.
Die Stage 2 Konfiguration kann genutzt werden, um anwenderspezifische Daten, Konfigurationen und Applikationen zu übermitteln. Sind die Daten größer als die maximale Paketgröße, so wird es in weitere Pakete aufgeteilt:

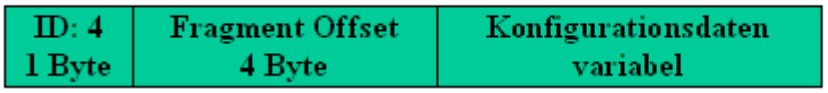


Abbildung 31 RTcfg - Frame - Subsegment -Frame

8. Client 1 bestätigt den Erhalt der Stage 2 Konfiguration mit einem “Acknowledgeconfig“ Paket und verarbeit die Konfiguration

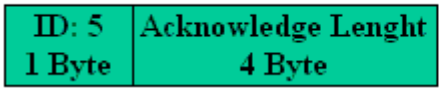


Abbildung 32 RTcfg - Frame - Acknowledge Config

Das Feld “Acknowledge Length“ enthält die Länge der korrekt erhaltenen Stage 2 Konfigurationsdaten

9. Client 1 sendet dann eine “Fertig“ Nachricht an alle.

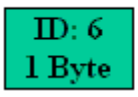


Abbildung 33 RTcfg - Frame - Ready-Frame

31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik

10. Der Server sendet, wenn alle Clients seiner Liste fertig sind, ein “Fertig“ Paket an alle. Diese Nachricht wird nur gesendet, wenn das Ready Flag im “Announcement“ Paket nicht vorher gesetzt wurde.

11. Alle Clients senden regelmäßig ein Heartbeat Paket an den Server, um zu signalisieren, dass sie noch aktiv sind.

Abbildung 34 RTcfg - Frame - Heartbeat

Ausfall eines Clients

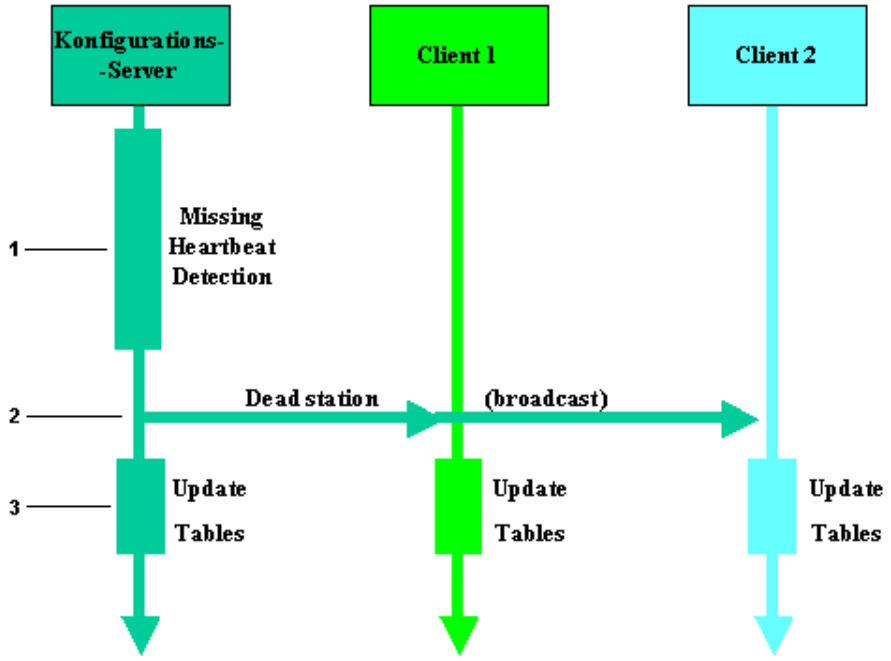


Abbildung 35 RTcfg - Ausfall eines Clients

1. Der Server bemerkt das Ausbleiben eines “Heartbeats“.
2. Nach einer definierten Anzahl (threshold; siehe 4.2) von ausbleibenden “Heartbeats“ sendet der Server ein “Dead station“ Signal.

ID: 8 1 Byte	Client Adressen Typ 1 Byte	Client IP Adresse variabel	Client MAC Adresse 32 Byte

Abbildung 36 RTcfg - Frame - Dead Station

3. Nach Erhalt bzw. Senden des Signals erneuern alle Teilnehmer ihre ARP und routing Listen

Server Neustart

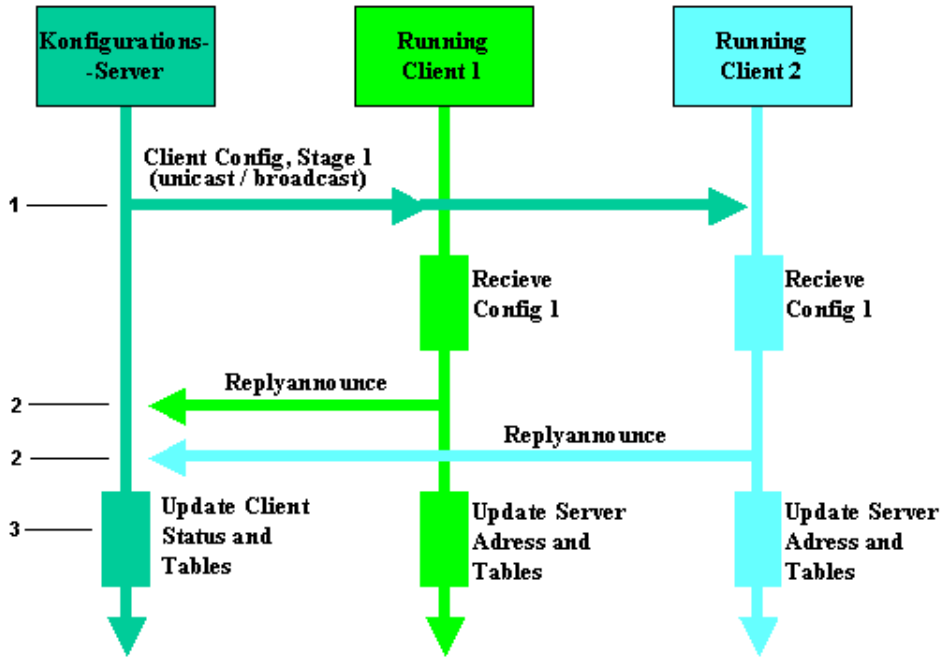


Abbildung 37 RTcfg - Server Neustart

1. Der Server sendet die Stage 1 Konfigurationsdaten an die Clients.
2. Die Clients antworten mit einem “Replyannounce“ Paket.
3. Der Server aktualisiert die Client Status , alle Teilnehmer erneuern ihre ARP und routing Listen.

Quellen für RTnet: [Kizka 05]

3.5 Profinet

3.5.1 Konzept

“Ethernet hat sich in der Bürowelt als Kommunikationsstandard etabliert. Die Anforderungen an die industrielle Kommunikation sind weitaus höher wie z.B. Echtzeitfähigkeit, Einbindung dezentraler Feldgeräte, industriegerechte Installationstechnik, etc.. Diese Anforderungen werden von PROFINET, dem offenen und herstellerübergreifenden Industrial Ethernet Standard, der PROFIBUS Nutzerorganisation, erfüllt und gewährleistet damit eine durchgängige Kommunikation von der Bürowelt bis in die Feldebene gewährleistet ist.“
Zitat [siemes-homepage]

Die PNO Spezifikation Profinet verfolgt das Konzept, die Vielzahl verschiedener Bussysteme und Netzstandards in einem Industrieunternehmen in sich zu vereinen.
Um das zu bewerkstelligen, erfüllt Profinet Anforderungen von der Büroebene (TCP/IP) bis hinunter zur Feldebene (Echtzeit).
Um auf allen Gebieten die gewünschten Vorgaben zu erfüllen, vom Bürobereich mit verschiedenen Protokollen und großen Datenmengen bis zur Feldebene mit extrem kurzen Latenzzeiten, gliedert sich Profinet in zwei Netzwerkkonzepte.

Profinet CBA
Profinet CBA (Component Based Automation) verfolgt einen durchgängigen Ansatz für alle wesentlichen Ebenen der Automatisierungshierarchie.
Das Konzept von CBA ist, dass eine Industrieanlage in mehrere Komponenten aufgeteilt werden kann. So kann eine komplexe Ablaufkette von Verarbeitungsschritten in überschaubare Teileinheiten aufgeteilt werden. Diese Teilstücke werden nun als Profinet Module standardisiert abgebildet und können so in anderen Anlagen oder Anlagenteilen in gleicher oder abgeänderter Form wiederverwendet werden. Dadurch ist es möglich, Automatisierungsanlagen modular zu strukturieren und zwischen diesen Modulen Datenbeziehungen aufzubauen.
Die Vorteile des CBA Modells sind somit eine übersichtliche, logisch strukturierte Anlage, die Wiederverwendbarkeit einzelner Anlagenmodule, die einfache Einbeziehung bestehender Feldbusse, SPS, etc.

Profinet IO (**I**nput / **O**utput)
Profinet IO ist auf die Kommunikation von Steuerung und dezentralem Feldgerät ausgelegt. Um mit den “einfacheren“ Feldgeräten kommunizieren zu können und die Performance sicherzustellen, wurde für Profinet IO ein auf Ethernet basierendes, an Profibus DP angelegtes Protokoll definiert.
Dadurch ist es möglich, sowohl Ethernet Eigenschaften (Protokollierung über HTML, etc), als auch die Erfahrungen aus der Feldbustechnologie einzusetzen.
[Technologien mit Vorsprung IFAK]

Des Weiteren sind drei verschiedene Performance/Protokoll Stufen definiert.

TCP/IP
Protokollstufe für Profinet CBA (Component Based Automation)/ Profinet IO
Reaktionszeiten im Bereich von 100ms
Kommunikation zwischen Controllern (z.B. SPS)

RT
Protokollstufe für Profinet IO
Reaktionszeiten im Bereich von 5ms
Kommunikation zwischen Controllern und Feldgeräten (z.B. SPS und Sensor/Aktor)

IRT
Protokollstufe für Profinet IO
Reaktionszeiten im Bereich von 250µms
Taktsynchroner Datenaustausch (z.B. Motion control)
[Popp,Weber 2004]

3.5.2 Funktionsweise

TCP/IP

Die erste Protokollstufe basiert auf “normalem“ TCP/IP und ist für die Parametrisierung, Konfiguration und das CBA Modell einsetzbar.
Dieser Ansatz ist nur bedingt echtzeitfähig und wird deshalb auch nur eingesetzt, wenn keine kurzen Latenzzeiten gefordert sind.
Das CBA Modell nutzt TCP/IP, da es nur für die Kommunikation zwischen verteilten Anlagen konzipiert ist, die bedingte Echtzeitfähigkeit wird hier mittels eines kollisionsfreien Ethernets durch den Einsatz von Switches erreicht.
Die Parametrisierung kann ebenfalls auf Echtzeitfähigkeit verzichten, da es keine in der Anlage entstehenden Parameter sind, sondern es sich um Daten vom Benutzer handelt.

RT

Profinet I/O mit der Protokollstufe RT ist für den Echtzeitbetrieb konzipiert.
Vorraussetzung für den Einsatz in beiden Stufen ist eine kollisionsfreie Ethernetstruktur, welche ausschließlich Switches verwendet.
Damit die Echtzeitdaten in den Switches nicht unnötig verzögert werden, arbeitet Profinet IO mit Priorisierungsmechanismen, harte Echtzeitdaten bekommen die Prioritätsstufe 7, weiche Echtzeitdaten (Video, Internet Telephonie) 5-6, usw.

IRT

Im isochronen Echtzeit Betrieb IRT (**I**sochrounous **R**eal**t**ime) wird das Netz zusätzlich in einen Echtzeitbereich und einen Nichtechtzeitbereich aufgeteilt. Des Weiteren kommen als Switches nur ASICS (kundenspezifische Integrierte Schaltung) mit extrem geringen Latenzzeiten zum Einsatz. Die Zeitsynchronisation erfolgt nach dem Synchronisationsmodell der IEEE 1588.

4 Versuche

4.1 Installation von RTAI/Fusion und RTnet:

Die Installation von RTAI/Fusion erwies sich aufgrund von Konflikten zwischen Versionen als umfangreicher als erwartet und zeigt, welchen Arbeitsaufwand automatisierte Distributionen wie Suse dem Benutzer abnehmen.
Wenn wir uns das fertige System als Pyramide vorstellen, so bilden die Bibliotheken (glibc, usw.) zusammen mit dem Kernel die Basis.
Je nach Version der Bibliotheken erhält man leichte Unterschiede im Kernel, welcher wiederum erst mit dem RTAI/Fusion Patch modifiziert werden muss.
Allerdings unterstützt jede RTAI/Fusion Version nicht jeden Kernel, da meist nur die neuesten Kernel weitergepflegt werden.
Die Spitze der Pyramide ist RTnet, welches nicht jede Fusion Version unterstützt.
Als kleine Hürde erwies sich an dieser Stelle, dass sich RTnet wie auch Fusion in der Entwicklung befinden und nicht jede Versionskombination, welche in der Dokumentation angegeben ist, auch wirklich funktioniert.
Zum Zeitpunkt der Erstellung dieser Diplomarbeit befand sich das RT4Linux Projekt noch in der Entwicklung, wodurch es zu weiteren Versionsproblemen durch verschiede Bibliotheken kam. Dadurch ergab sich die Schwierigkeit, zu erkennen, ob der Fehler selber erzeugt (falsche Parameter, Programmierfehler), RTnet spezifisch, ein Versionsproblem oder ein Fehler im sich in der Entwicklung befindlichen RT4Linux ist.

Die Installationsanweisung befindet sich im Anhang.

4.2 Parametrisierung

Die Parametrisierung von RTnet erfolgt über das Start-Script ../rtnet/sbin/rtnet.

- Bei Übergabe des Parameters start (bash: rtnet/start) werden die Parameter aus der Datei ../rtnet/etc/rtnet.conf ausgelesen und RTnet automatisch mittels RTcfg parametrisiert.
- Bei Übergabe des Parameters stop (bash: rtnet/stop) wird das RTnet beendet.

Die Einstellungen sind nur beispielhaft und müssen auf den jeweiligen Anwendungsfall angepasst werden.
../etc/rtnet.conf:

- Netzwerkkarte
  RT_DRIVER="rt_eepro100"
- IP Adresse und Netzmaske des Masters
  IPADDR="128.0.1.203"
  NETMASK="255.250.0.0"
- Aktivierung Lokalhost
  RT_LOOPBACK="yes"
- Aktivierung des Echtzeitcapturings (für z.B.: Etherreal)
  RTCAP="yes"
- Einstellung des Teilnehmerstatus (Master/Slave):
  TDMA_MODE="master"
- Registrierung der Slaves (nur Master)
  TDMA_SLAVES="128.0.3.201 128.0.3.202 128.0.3.203 "
- Einstellen der Zykluszeit (Mikrosekunden) (nur Master):
  TDMA_CYCLE="450"
- Einstellen des Offsets (Mikrosekunden) (nur Master):
  TDMA_OFFSET="50"
- Pfad zu den erweiterten Parametern:
  (zur Nutzung der erweiterten Parameter, diese Zeile einkommentieren,
  und die Master Parameter: TDMA_SLAVES: TDMA_CYCLE; TDMA_OFFSET auskommentieren)
  #TDMA_CONFIG="@sysconfdir@/tdma.conf"

/etc/tdma.conf:
Die erweiterten Einstellungen in der tdma.conf Datei erlauben die Nutzung eines Slots durch mehrere Teilnehmer in verschiedenen Zyklen.
Parameter in eckigen Klammern [ ] sind optional.

Master:

- IP-Adresse des Masters

ip 10.0.0.1

- Zykluszeit in µs

cycle 450

- Slotkonfiguration <ID des Slots><Offset in Mikrosekunden> [<phasing>/<period>[<Größe des Slots in Byte>]]

slot 0 0 1/1 1000

- weitere Slots

slot 1 1000

Backup Master
Zykluszeit wird vom Primären Master übernommen

- IP Adresse des Backup Masters
  ip 10.0.0.2
- Backup-Offset in Mikrosekunden
  backup-offset 200
- Slotkonfiguration <ID des Slots> <Offset in Mikrosekunden> [<phasing>/<period> [<Größe des Slots in Byte>]]
  slot 0 400

Slave A
MAC ist unbekannt, Der Slave wird über die IP vorkonfiguriert

- IP Adresse des Slaves
  ip 10.0.0.3
- Slotkonfiguration<ID des Slots> <Offset in Mikrosekunden>[<phasing>/<period> [<Größe des Slots in Byte>]]
  slot 0 2000 1/2
  slot 1 2200

Slave B
IP wird dem Slave mittels der MAC Adresse zugewiesen.

- IP Adresse des Slaves
  ip 10.0.0.4
- MAC Adresse des Slaves
  mac 00:12:34:56:AA:FF

  slot 0 2400
  slot 1 2200 2/2

Testen des RTnet Netzwerks:
Deaktivieren der Netzwerkkarte
Entfernen des Moduls
Bash: rmmod [modulname]
Laden der RTAI Modle
Bash: insmod /opt/etx/fusion/modules/rtai_hal.ko
Bash: insmod /opt/etx/fusion/modules/rtai_nucleos.ko
Bash: insmod /opt/etx/fusion/modules/rtai_native.ko
Laden des Treiberabstraktions-Moduls
Bash: insmod /opt/etx/rtnet/modules/rtai_rtdm.ko
RTnet Start Script starten:
Bash: /opt/etx/rtnet/modules/rtnet start
Echtzeitping:
Bash: ../rtnet/sbin/rtping <IP-Adresse>
Testprogramme:
../src/rtnet-X.X/addons/examples

Zentrale Konfiguration von RTnet mittels RTcfg
Über das Kommandozeilen Werkzeug RTcfg können die Server und Clients optional vom Server aus parametrisiert werden.
Die Parametrisierung kann aber bequemer über die Konfigurationsdateien in /rtnet/etc vorgenommen werden.

Parameter in eckigen Klammern [ ] sind optional.

Server:

Mittels **RTcfg** wird das Werkzeug zur Parametrisierung des Netzwerkes aufgerufen
dev verweist auf die Netzwerkschnittstelle (rtethx)
Server gibt an, dass ein Server parametrisiert wird

Starten des Servers:
RTcfg <dev> server [-p period] [-b burstrate] [-h heartbeat] [-t threshold] [-r]

- **period** ist der zeitliche Abstand zwischen den Stage 1 Konfigurationsframes.
- **burstrate** gibt an, wie viele Teilnehmer (Clients) pro Zyklus eingeladen werden und wie viele Stage 2 Konfigurationsfragmente pro Zyklus gesendet werden.
  Default ist 4
- **heartbeat** spezifiziert die Zeit zwischen den Nachrichten der Clients, welche dem Server signalisieren, dass sie noch Teil des Netzwerkes sind.
  Default sind 1000ms, 0 schaltet den Mechanismus ab
- **threshold** gibt an, nach wie vielen ausbleibenden “Heartbeats“ eine Station für inaktiv/tot erklärt wird.
  Ist der Parameter **–r** angegeben, so liefert der Server automatisch eine Nachricht, dass er mit der Stage 1 Konfiguration fertig ist. Dies erübrigt eine explizite “Bereit“ Meldung

.

Client zur Liste des Servers hinzufügen:
RTcfg <dev> add <address> [-hw <hw-adress>] [-stage 1 <stage 1 file>] [-stage 2 <stage 2 file>] [-t timeout]

- **add** fügt einen Client in die Liste potentieller Teilnehmer des Netzes ein, spezifiziert durch die Schnittstelle **dev**
- **adress** ist entweder die IP-Adresse des Teilnehmers oder die MAC Adresse. Wenn die MAC Adresse als **hw-adress** explizit angegeben ist, muss der Parameter **adress** die IP Adresse angeben.
- Optional können Stage 1 und 2 Konfigurationsdateien angegeben werden (**stage 1 file / stage 2 file**)
- **timeout** gibt die Zeit in Millisekunden an, wonach ein nicht fertig konfigurierter Client auf seine Urzustand zurückgesetzt wird.
  Default ist unendlich

Client entfernen
RTcfg <dev> del <adress>

- Entfernt den Teilnehmer mit der Adresse **adress** an der Schnittstelle **dev**

Timeout für die Konfiguration einstellen
RTcfg <dev> wait [-t timeout]

- Wartet bis beide Konfigurationsphasen abgeschlossen sind.
- Wenn ein **timeout** angegeben ist, wird nach Ablauf eine Fehlermeldung ausgegeben.
  Default ist unendlich.

Timeout für die Synchronisation einstellen

- RTcfg <dev> ready [-t timeout]
- Gibt eine Meldung aus wenn der Server sein Setup inklusive der Rtmac Startphase abgeschlossen hat, und wartet bis alle anderen Stationen fertig sind.
- Ist ein t**imeout** angegeben, wird nach Ablauf eine Fehlermeldung ausgegeben
  Default ist unendlich.

Server entfernen

RTcfg <dev> detach

- Stoppt den Server der Schnittstelle **dev**

Client:

Mittels **RTcfg** wird das Werkzeug zur Parametrisierung des Netzwerkes aufgerufen
dev verweist auf die Netzwerkschnittstelle (rtethx)
client gibt an, dass ein Client parametrisiert wird

RTcfg <dev> client [-t <timeout>] [-c | -f <stage1_file>] [-m maxstations]

- wartet, bis die erste Konfigurationsphase abgeschlossen ist
- ist ein **timeout** angegeben, wird nach Ablauf eine Fehlermeldung ausgegeben Default für timeout ist unendlich
- Die eingehenden Konfigurationsdaten vom Server werden entweder auf Standard Out (**-c**) geschrieben oder in eine in **stage1_file** angegeben Datei:
- **maxstations** gibt an, mit wie vielen anderen Stationen, einschließlich Server, sich der Client synchronisieren soll.

RTcfg <dev> announce client [-t <timeout>] [-c | -f <stage2_file>]
[-b burstrate] [-r]

- Sendet ein “New Announcement” Paket über die Schnittstelle <**dev**> und wartet, bis die zweite Konfigurationsphase beendet ist.
- Ist ein **timeout** angegeben, wird nach Ablauf ein Fehler ausgegeben.
  Default ist unendlich
- Ist **–f** oder **–c** angegeben, werden die eingehenden Konfigurationsdaten vom Server angefordert und entweder auf Standard Out (**-c**) oder in eine in **stage2_file** angegeben Datei (**-f**) geschrieben.
- **burstrate** gibt an, wie viele Stage 2 Konfigurationsfragmente pro Zyklus angenommen werden.
  Default ist 4
- Ist **–r** parametrisiert, sendet der Client automatisch eine “Fertig“ Meldung mit seinem “Announcement“ Paket.

RTcfg <dev> ready [-t timeout]

- Sendet eine “Fertig” Meldung und wartet auf alle anderen Teilnehmer.
- Ist ein **timeout** angegeben, wird nach Ablauf ein Fehler ausgegeben.
  Default ist unendlich

RTcfg <dev> detach

- Stoppt den Client auf der spezifizierten Schnittstelle

zentrale Konfiguration des TDMA Verfahrens mittels tdmacfg

Über das Kommandozeilen Werkzeug tdmacfg können die Master und Slaves optional vom Master aus für das TDMA Verfahren parametrisiert werden.
Die Parametrisierung kann aber bequemer über die Konfigurationsdateien in /rtnet/etc vorgenommen werden.

Parameter in eckigen Klammern [ ] sind optional.
Mittels **tdmacfg** wird das Werkzeug zur Parametrisierung des TDMA Verfahrens aufgerufen
dev verweist auf die Netzwerkschnittstelle (rtethx)

Aktivierung und Parametrisierung eines Masters:
tdmacfg <dev> master [cycle_period] [-b <backup offset>]
[-c calibration_rounds] [-i max_slot_id] [-m max_calibration_requests]

- **master** gibt die Parametrisierung des Master an.
- **cycle period ist** die Länge des TMDA Rahmens (Zyklus) bzw. der Abstand zwischen den Synchronisierungspaketen in Mikrosekunden.
- Ist ein **backup offset** angegeben, so ist der Master ein Backup Master. Bei Ausfall des Masters übernimmt der Master mit dem geringsten Backup Offset und sendet seine Synchronisierung Frames mit dem spezifizierten Offset zwischen dem Start des TDMA Zyklus und den Synchronisations- Frames.
- **calibrations_rounds** gibt die Anzahl der Kalibrierungsaufrufe an, welche der Master an potentielle schon aktive Master sendet.
  Default ist 100

- **max_slot_id** ist die größte Slot ID die vergeben wird (Anzahl der Slots).
  Default ist 7.
- **max_calibration_requests** ist die Anzahl der Kalibrierungsanforderungen, die der Master verarbeiten kann.
  Default ist 64.

Aktivierung und Parametrisierung eines Slaves
tdmacfg <dev> slave [-c calibration_rounds] [-i max_slot_id]

- **slave** gibt an, dass ein Slave konfiguriert wird
- **calibration_rounds** ist die Anzahl von Kalibrierungsanfragen, die der Slave sendet. Default sind 100 Anfragen
  Der Konfigurationsvorgang wird gestartet, wenn der erste Slot angefügt wird
- **max_slot_id** Maximale Anzahl der Slots.
  Default ist 7.

Slot (Zeitschlitz) Konfiguration (Master und Slave)
tdmacfg <dev> slot <id> [<offset> [-p <phasing>/<period>] [-s <size>]]
[-l calibration_log_file] [-t calibration_timeout]

- **slot** gibt an, dass ein Zeitschlitz konfiguriert wird.
- **id** wird genutzt, um verschiedene Slots zu unterscheiden.
- **offset** gibt den Offset des Slots relativ zum Zyklusstart in Mikrosekunden an.
- **phasing** und **period** geben an, in welchem Zyklus (phasing) welcher Zyklen Periode (period) der Slot genutzt wird.
  z.B. in jedem ersten Zyklus (phasing) von drei (period) Zyklen [-p 1/3].
- **size** gibt die maximale Größe der Nachricht in Byte an.
  Default: die maximale Größe, welche die Hardware erlaubt.
- **calibration_log_file** aktiviert die Ausgabe der Ergebnisse der Kalibrierung
- **calibration_timeout** erlaubt die Eingabe eines Zeitlimits für die Kalibrierung

Deaktivierung eines Masters Slaves
tdma <dev> detach

4.3 Benchmarkprogramme

Um die Latenzzeiten des Netzwerkes zu testen, mussten Benchmarkprogramme geschrieben werden, welche die Übertragungszeiten des Netzwerkes messen und protokollieren.

Als Prinzip bietet sich die Round-Trip-Time an, d.h. die Zeit, die ein Paket zu einem anderen Teilnehmer und wieder zurück braucht. Dadurch wird die Zeit mit nur einer CPU gemessen, wodurch Synchronisierungsproblemen aus dem Weg gegangen wird. Die Latenzzeit von einem Teilnehmer zu einem anderen, ist die Hälfte der Latenzzeit.
Zuerst wurde das mitgelieferte Round-Trip-Time Programm so modifiziert, dass es keine Zeiten mehr lieferte, sondern die ersten beiden Pins auf dem Parallelport bei senden und empfangen der Pakete setzte. Dies geschah, um die protokollierten Zeiten des Programms zu prüfen.
Da das Programm im nur Kernelspace lief und für RTAI konzipiert war, konnte es unter Fusion keine Protokolldaten in ausreichender Menge liefern.

Um Benchmarktest über längere Zeit zu fahren, musste also folglich ein Programm geschrieben werden, das Daten protokollieren kann.
Mit fortschreitenden RTnet Releases wurde auch die Userpace Unterstützung besser, so dass ein solches Programm implementiert werden konnte.
Die Implementierung erfolgte als Server/Client Userspace Programm.
Der Client misst seine Zeit, und sendet sie an den Server, welcher das Paket quittiert und zurück an den Client schickt.
Dieser misst nun wieder seine Zeit und vergleicht sie mit der Zeit im Paket.
Dadurch können die Pakete auch in falscher Reihenfolge ankommen, ohne das Ergebnis zu verfälschen.
Die Zeiten werden auf der Konsole ausgegeben, so können die Zeiten direkt zur Laufzeit geprüft oder zur Dokumentation und Weiterverarbeitung in eine Datei umgelenkt werden.

Bash: client –d 128.0.3.204	Ausgabe auf der Konsole
Bash: client –d 128.0.3.204 > Datei	Ausgabe in Datei

Implementierung Client

Das Client Programm besteht aus zwei Echtzeit - Tasks und der Main Routine. Die Main Routine verarbeitet und prüft die Eingaben des Anwenders (IP, Größe des Datenpaketes, etc), erzeugt die Sockets und die (Echtzeit-) Tasks und startet diese.

Die Sende-Task xmit_msg:
Nach der Deklaration der Variablen startet die Task eine zyklische Endlosschleife (1), welche den Zeitstempel ausliest (2), in den Sendepuffer der Nachrichten Struktur schreibt (msg) und diese Struktur abschickt(3). Nach dem Abschicken wartet die Task die angegebene Zykluszeit(4) und beginnt die Schleife von neuem zu durchlaufen. Dadurch lässt sich einstellen, wie oft Pakete pro Sekunde gesendet werden.

```
rt_task_set_periodic(NULL, TM_NOW, cycle);
tx_time = rt_timer_read();
sendmsg_rt(sock, &msg, 0);
rt_task_wait_period();
```

Die Empfangs-Task recv_msg:
Nach der Deklaration der Variablen und des Empfangspuffers startet die Task eine Endlosschleife, welche am angegebenen Socket auf Pakete wartet (1) und diese auswertet. Dazu wird direkt nach Eingang eines Paketes die Zeit aus dem Timer ausgelesen(2) und mit der Zeit im Paket verglichen(3;4).

```
recvmsg_rt(sock, &msg, 0);
rx_time=rt_timer_read();
tx_time = strtoll(buffer_in,NULL,10);
diff=rx_time-tx_time;
```

Implementierung Server

Der Server ähnelt dem Client im Aufbau. Die Main Routine erzeugt und startet, nachdem Variablen und Semaphore deklariert wurden, das Socket und die Tasks.
Die Echtzeit-Tasks des Servers arbeiten in entgegengesetzter Reihenfolge wie die des Clients. Während der Client ein Paket sendet und auf den Empfang wartet, wartet der Server auf ein Paket, um es dann zurück zu senden.
Das Zurücksenden wird hier über einen Semaphor gesteuert.

Die Empfangs-Task recv_msg:
Die Empfangs-Task wartet am Socket auf eingehende Pakete(1), analysiert die Größe des Paketes(2) und quittiert den Eingang(3).
Dann löst die Task den Semaphor aus(4), welcher dem Sende-Task signalisiert, dass ein Paket zurückgesendet werden soll.

```
recvmsg_rt(sock, &msg, 0);
echosize=ret-sizeof(msgsize);
sprintf(buffer_out,"%s - echo",buffer_in);
rt_sem_v(&g_sem);
```

Der Sende-Task xmit_msg:
Der Sende-Task des Servers arbeitet ähnlich wie der des Clients. Die Unterschiede zwischen beiden sind, dass die Paketgröße dynamisch von der Empfangs-Task übermittelt wird und dass die Sendung von Paketen nicht zyklisch, sondern durch den Semaphor ausgelöst wird(1).

```
rt_sem_p (&g_sem,TM_INFINITE);
```

4.4 Benchmark Ergebnisse

Die Parameter der Tests mit zwei Teilnehmern (Server/Client) entstanden aus folgenden Überlegungen.

Die maximale Paketgröße für Ethernet (MTU) ist 1518Byte
1518Byte = 12144Bit
Fast Ethernet hat eine Übertragungsrate von 100Mbit pro Sekunde
100Mbit = 100 000 000Bit pro Sekunde

12144Bit / 100000000Bit/s = 121,44µs

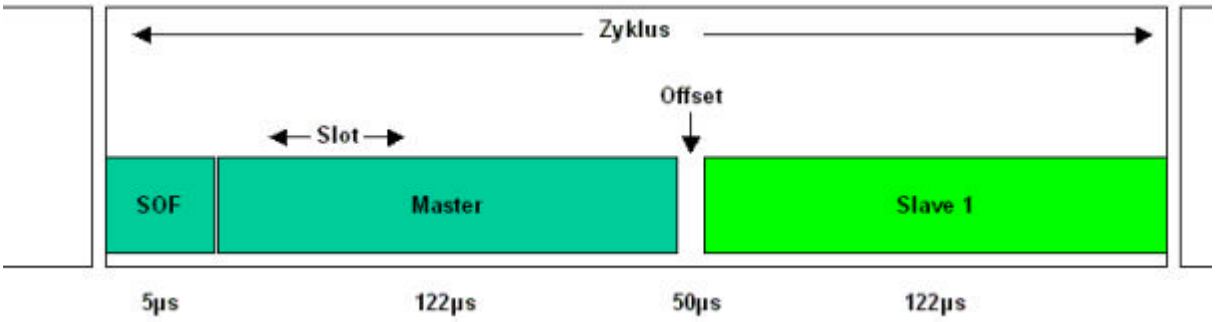


Abbildung 38 Benchmark Parameter

5µs	für Start of Frame
122µs	pro Slot
50µs	pro Offset

Diese Rechnung ergibt eine Zykluszeit von 299µs. Diese Rechnung geht dabei von idealer Hardware und keinem Jittering aus.
Um eine stabile Testparametrisierung zu schaffen wurden aufgrund dessen noch 151µs für Sicherheit (Jitter, CPU Latenzzeiten, etc) mit eingerechnet.

Ergibt eine Zykluszeit von 450µs bei einem Offset von 50µs.

Diese Parameter sind allerdings sehr knapp gewählt, um minimale Latenzzeiten zu erreichen. Erhöht sich nun die CPU Last, durch Echtzeitprogramme wie RTnet selber etc. oder Nicht-Echtzeit Verkehr, kann es vorkommen, dass ein Paket seinen Zyklus “verpasst“, und erst im nächsten Zyklus gesendet wird.

Test mit 450µs Zykluszeit bei einem 50µs Offset
100 Pakete a 1000Bit pro Sekunde

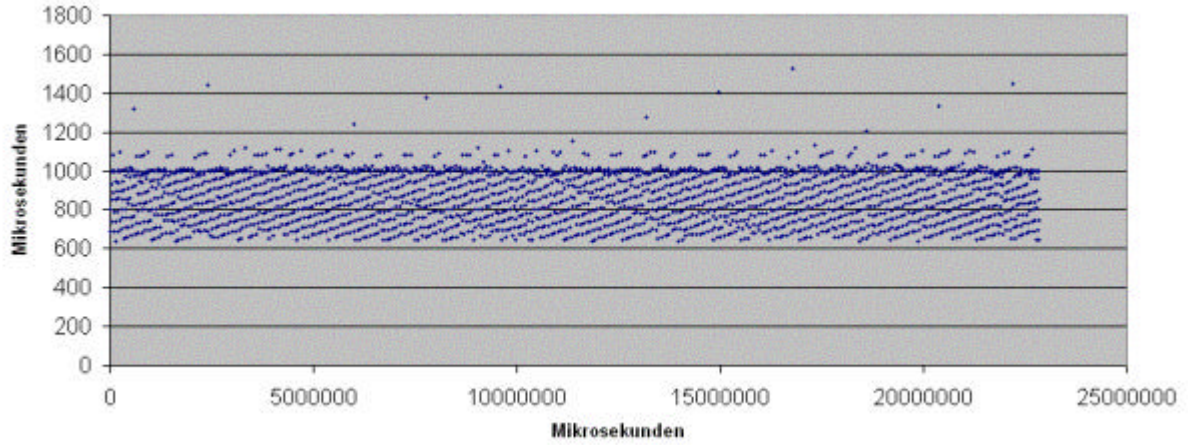


Abbildung 39 Benchmark

Die Round Trip Zeit liegt bei maximal 1531µs.
Auffallend ist die regelmäßige Verteilung der Zeiten, dieses Muster entsteht durch die Überlagerung des TDMA Zyklus mit dem Zyklus des Round Trip Programms(siehe Abbildung 40).

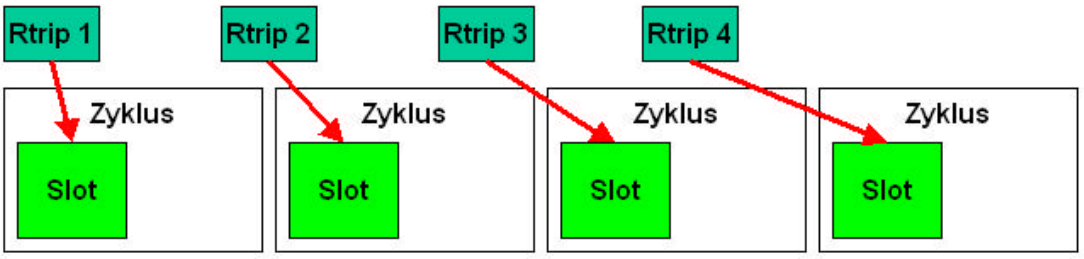


Abbildung 40 Überlagerung Rtrip und Zyklus

Die einzelnen “Ausreißer“ entstehen durch die knapp gewählten Parameter in Verbindung mit CPU Belastungen.
Durch Verkürzung der Zyklus- und Benchmark-Zeiten lassen sich die Zeiten verbessern, allerdings zu Lasten von Stabilität und Jitter.

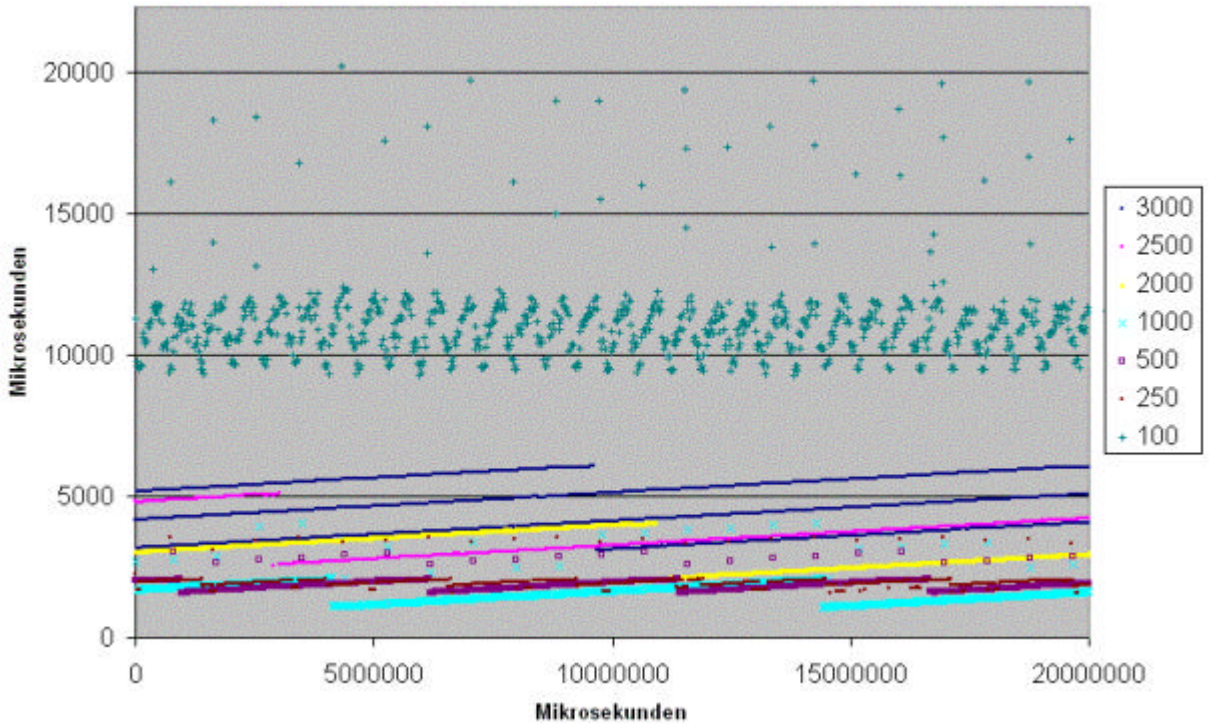


Abbildung 41 Benchmark verschiedene Zykluszeiten

In Abbildung 41 sieht man die Benchmark-Zeiten verschiedener Zykluszeiten mit 100µs Offset, 100 Paketen pro Sekunde bei einer Paketgröße von 100bit.
Bei kleineren Zykluszeiten sinken auch die Round-Trip Zeiten, allerdings steigt die Anzahl der Pakete, welche Zyklen überspringen. In Fällen mit höherer Belastung der CPU und Nicht-Echtzeit Datenaufkommen verschärft sich dieses Verhalten, so dass für die jeweilige Anwendung die Zykluszeiten berechnet und getestet werden müssen.
Die Zeiten im oberen Bereich resultieren auf fehlerhaften Parametern, Zykluszeit und Offset sind gleich. Das Netz ist jedoch stabil genug, um weiterhin zu funktionieren, allerdings mit hohen und unberechenbaren Zeiten.

Mittels dem Ping Befehl wurde eine Nicht-Echtzeit Last zu generiert. Der Befehl wurde so parametrisiert, dass alle 10ms ein 1000Byte Paket an den 2. Teilnehmer gesendet wurde.

Bash: ping –i 0.01 –s 1000 128.0.3.204

Für Zyklen die kleiner als 0.2 Sekunden sind, ist es notwendig als Superuser angemeldet zu sein.

In Abbildung 42 ist zu sehen, dass nach zehn Sekunden das Rtrip Programm beendet wurde, und sich die Latenzzeiten des Ping Programms verbessern.

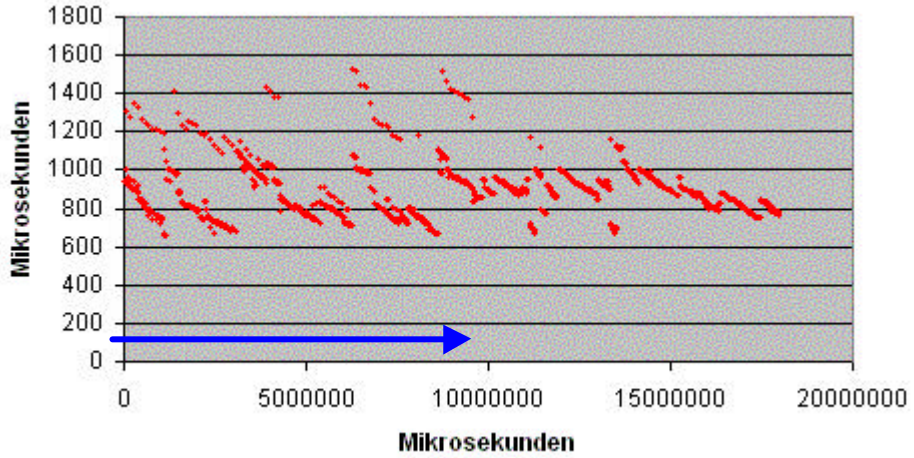


Abbildung 42 Pingzeiten mit Rtrip Last

Dieses Verhalten resultiert daraus, dass die Echtzeit Pakete bevorzugt gesendet werden und die Nicht-Echtzeit Pakete auf freie Übertragungszeit warten müssen.

Abbildung 43 zeigt dagegen, wie nach zehn Sekunden das Netz mit Nicht-Echtzeit Verkehr belastet wird. Die Zeiten erhöhen sich nur gering, was mit der höheren Belastung des Betriebssystems zu erklären ist.

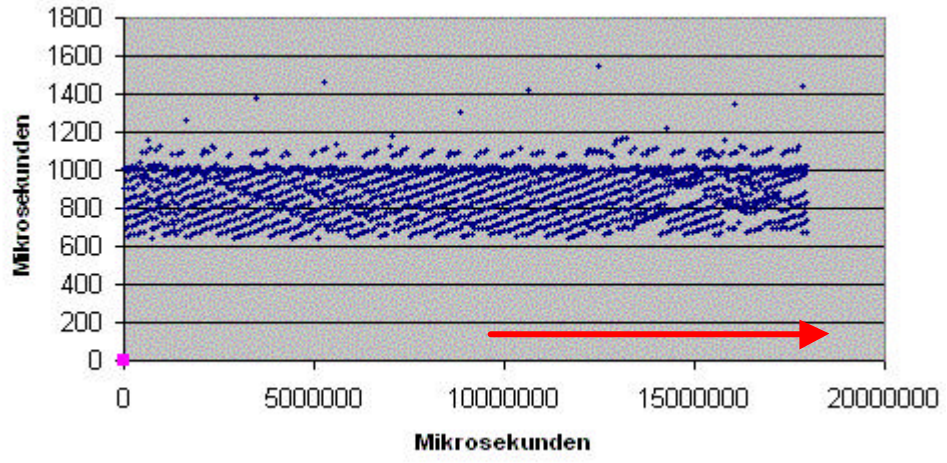


Abbildung 43 Rtrip Zeiten mit Ping Last

31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik

5 Diskussion

5.1 Vergleich beider Systeme

5.1.1 Komplexität und Ausbildungsaufwand

Für den neutralen Anwender, scheint RTnet weniger komplex und "aufgeräumter" als Profinet. Der Medienzugriff (MAC) mit der Implementation des TDMA Verfahrens, und die Konfiguration wirken sehr schlüssig und lösungsorientiert. Durch den freien Sourcecode und die Zusammenarbeit der Entwickler im Internet, ist es relativ einfach eigene Anwendungen zu implementieren und das Netz als Ganzes zu verstehen.
Die vorhandene Dokumentation ist nicht immer auf dem neuesten Stand, da sich RTnet ständig weiterentwickelt. Die Programmierung der Sockets und der Sende- und Empfangs-Routinen entspricht aber zum großen Teil BSD/Linux Standards, so dass ein versierter Programmierer wenig Probleme bei der Einarbeitung hat. RTnet setzt Informatik-Kenntnisse voraus. Sind diese vorhanden, ist die Handhabung von RTnet leicht verständlich, da es durchgehender strukturiert ist.

Profinet hingegen wartet mit 2 grundlegenden Systemen (CBA und IO) und drei Performance Stufen auf (TCP/IP, RT, IRT).
Zusätzlich ist es schwer, Informationen über die grundlegende Funktionsweise des Netzes zu bekommen. Die vorhandenen Informationen beziehen sich zum größten Teil auf die Implementation von Profinet in eine Industrieanlage mit vorhandener Infrastruktur, also Bussystemen, SPS, Sensor/Aktor Ebene und Prozessrechnern. Aber darin besteht auch die Stärke von Profinet. Es wurde von der Industrie für die Industrie entwickelt. Dabei wurde auf vorhanden Strukturen (Feldbusse, etc) Rücksicht genommen, insbesondere auf Profibus, welcher ebenfalls von der PNO entwickelt wurde.
Die Benutzerschnittstellen von Profinet sind sehr Profibus nachempfunden, so dass sich Anwender mit Profibus Erfahrung relativ schnell in Profinet einarbeiten können.
Teile der Profinet Dokumentation sind selbst ohne Beitritt in die PNO teilweise verfügbar, allerdings sind die im Internet frei erhältlichen Daten eher als Werbebroschüre einzustufen, andere Quellen müssen eingekauft werden.
Die Dokumentation ist sehr umfangreich, aber schlecht geordnet und undurchsichtig.

5.1.2 Kompatibilität zu anderen Netzen / Investitionssicherung der vorhandenen Infrastruktur

Grundlegend ist zu sagen, dass ein echtzeitfähiges Netz niemals voll kompatibel zu anderen Netzen sein kann, da die Echtzeitkommunikation deterministisch ablaufen muss. Würde jetzt ein anderes Netzprotokoll auf das gleiche Medium zugreifen, würde es zwangsläufig zu Kollisionen kommen.
Allerdings bieten die meisten Echtzeitnetze die Möglichkeit, über einen Router auf andere Netze zugreifen zu können. Hierzu werden Daten, welche den Router in die eine oder andere Richtung passieren, im Echtzeitnetz getunnelt, das heißt, dass z.B. ein TCP/IP Paket von einem Echtzeit-Netz Teilnehmer in ein UDP Paket “verpackt“, und in einem Zeitbereich, welcher für Nicht-Echtzeit Daten reserviert ist, übertragen wird. Der Router entpackt dieses Paket dann wieder, um es als TCP/IP Paket z.B. über das Internet weiterzuverschicken.
RTnet bietet diese Möglichkeit und kann zusätzlich Pakete fragmentieren, also Daten, welche die maximale Paketlänge überschreiten, in mehrere kleine Pakete aufteilen. Das Routing kann hierbei von einem handelsüblichen PC übernommen werden, welcher 2 Ethernet Karten besitzt.

Für die Integration in eine bestehende Anlage bietet Profinet den Vorteil, gerade Bussysteme sehr gut einbinden zu können. Durch die Erfahrungen der Mitglieder der PNO ist Profinet im Zuge der Investitionssicherung darauf ausgelegt, bestehende Bussysteme mit einzugliedern.
Profibus und Interbus können über vorgefertigte Proxies in das Profinet aufgenommen, und verwaltet werden.
Vorhandene speicherprogrammierbare Steuerungen (SPS) werden ebenfalls weitergenutzt, so dass fast die ganze vorhandene Infrastruktur weiterbenutzt werden kann, und wenig verändert werden muss.

5.1.3 Benötigte Hardware / Installations-/ Hardware-Kosten

Stellt man eine Industrieanlage auf ein Echtzeit Ethernet um, so ist meist ein Bussystem vorhanden. Im Zuge der Investitionssicherung ist man in einem solchen Fall mit Profinet im Vorteil, da es nicht nur bestehende Bussysteme mit eingliedern kann, sondern auch das bestehende Know-how der Anwender, da es in Projektierung und Parametrierung an Profibus angelehnt ist.
Zudem beseht eine breite Unterstützung verschiedener Komponentenhersteller (Siemens, etc), welche Hardware für Profibus entwickeln.
Allerdings muss bei der Eingliederung von Feldbussen und dem Einsatz von Profinet IRT auf spezielle Hardware zurückgegriffen werden. Zudem können in der Profinet Topologie nur Switches eingesetzt werden.

Baut man allerdings ein Echtzeitnetz zwischen mehreren Computern, so zeigt RTnet seine Stärken. Es werden nur generische Ethernet Karten und Hubs eingesetzt.
Das ganze Netz kann mit relativ einfachen Mitteln der jeweiligen Umgebung angepasst werden.

5.1.4 Debugmöglichkeiten zur Laufzeit / Protokollierung der Pakete

RTnet lässt sich mittels Etherreal, eines Ethernet Capturing Programms, zur Laufzeit überwachen. Etherreal analysiert alle Pakete, welche an der Netzwerkkarte des jeweiligen Computers ankommen, unabhängig davon, für welchen Teilnehmer sie bestimmt sind.
Etherreal unterstützt die meisten Protokolle, so auch RTnet, wodurch es möglich ist, zu analysieren, welche Pakete sich wirklich physikalisch auf dem Medium Ethernet befinden.
Des Weiteren werden mehrere Beispiele mitgeliefert, welche sich zur Analyse und ersten Tests eignen.
Da RTnet auch in Userspace Programme genutzt werden kann, können hier die normalen Debug Programme genutzt werden.

5.1.5 Stabilität / Datenverlust / Sicherheit

Da beide Systeme mit Ethernet Frames arbeiten, welche eine CRC Prüfsumme enthalten, werden Pakete, welche bei der Übermittlung beschädigt wurden, mit der Genauigkeit des CRC Prüfsummenverfahrens erkannt
[Davie &Peterson 2000].
Durch den Einsatz von UDP hat RTnet aber keine Möglichkeit zu erkennen, ob ein Paket fehlt.

5.1.6 Benchmark

Bei der Erstellung der Benchmarks ergaben sich zwei Probleme:
Profinet Quellen sind nur durch Beitritt in die PNO zu bekommen.
Mit welchen Umgebungsvariablen (Anzahl der Teilnehmer, Netztopographie, etc) soll getestet werden?

Die Schwierigkeit ein Echtzeit Netzwerk zu testen, liegt darin, die Umgebung, worin es später eingesetzt werden soll zu kennen und die Parameter des Netzes richtig einzustellen.
Da die späteren Einsatzgebiete sehr weit gesteckt sind, habe ich mich dafür entschieden den Idealfall (zwei Teilnehmer, keine verzögernde Topographie) anzunehmen, da dieser eine feste Grenze darstellt (schlechtere Zeiten durch mehr Teilnehmer, bzw. verzögernde Topographie wie Switches etc. sind nach oben kaum begrenzt) und diese Zeiten am ehesten mit den Werbe-Zeiten der proprietären Produkte vergleichbar sind.

RTnet liegt mit seinen Latenzzeiten weit unter 10 Millisekunden und ist somit für Echtzeitanwendungen in der Industrie sehr gut geeignet. Nur für den Bereich der Isochronen Bewegungen mit Latenzzeiten unter 1Millisekunde und Jitter von 1 Mikrosekunde muss auf spezielle Hardwaregestützte Verfahren zurückgegriffen werden.

6 Zusammenfassung und Ausblick

Im Zuge dieser Diplomarbeit wurde RTnet in das Echtzeit Linux implementiert und dokumentiert. Die häufigsten Probleme entstanden durch den experimentellen Status der meisten Komponenten. Durch die ständigen Updates der Versionen war es notwendig, sich ständig auf eine wechselnde Umgebung einzustellen.
RTnet läuft mittlerweile stabil unter RTAI/Fusion und kann zur Echtzeit-Kommunikation genutzt werden. Durch die Möglichkeit der Programmierung von RTnet im Userspace lassen sich Programme komfortabler und sicherer implementieren als durch die Erzeugung von Kernelmodulen.
Die RTnet API entspricht der normalen Posix/BSD API, wodurch die meisten Anwender wenig Einstiegsschwierigkeiten haben sollten.
Die Parametrisierung von RTnet erfordert ein wenig Erfahrung, da die Zeiten und Parameter nicht nur von der Topologie des Netzes und dem Datenaufkommen abhängen, sondern auch von den Eigenschaften des Betriebssystems und der eingesetzten Hardware.
Während der Diplomarbeit wurde klar, dass ein zahlenmäßiger Vergleich zwischen den verschiedenen Systemen kaum möglich ist und auch keinen Sinn macht.
Es gibt also keinen “Gewinner“ in diesem Vergleich.
Profinet ist für die Industrie erschaffen worden, um dort die Bussysteme abzulösen und die Vernetzung zu verbessern. Dabei wurde darauf geachtet, dass dieser Wechsel einfach und unkompliziert vonstatten gehen kann. Bestehende Systeme und Know-how der Angestellten können einfach übernommen werden.
Es muss nur wenig neu entwickelt werden, um Feldgeräte, SPS, etc anzubinden, da viele Firmen Hardware für Profinet anbieten.
RTnet dagegen ist von Informatikern entwickelt worden, um eine Open Source Alternative zu anderen Echtzeit-Ethernet-Systemen zu bieten. RTnet setzt dabei nur auf Standard Hardware (Netzwerkkarten und Hubs) und probiert mit diesen Vorgaben das Optimum zu erreichen.
Die Latenzzeiten von RTnet unterscheiden sich dabei nicht stark von anderen Systemen, die mit ähnlicher Hardware arbeiten
Es fällt auf, dass RTnet nicht darauf ausgelegt ist, etwas anderes als Computer mit Ethernet Karten miteinander zu vernetzen. RTnet bietet allerdings das Potential, um so etwas zu entwickeln, wobei Industrieunternehmen eher auf garantiegedeckte Produkte großer Hardware Hersteller setzen, als selber zu entwickeln.
RTnet eignet sich sehr gut, um eigene Entwicklungen, kleine Insellösungen oder Computer zu vernetzen, aber nicht, um Standard Industrie Hardware im großen Stil

vom Bus zum Netz zu bringen. Allerdings wird RTnet ständig weiterentwickelt und flexibel gehalten, so dass spezielle Anforderungen an das Netz umgesetzt werden können.
Das Echtzeit Linux Projekt der NPN bietet sicherlich noch viel Stoff für weitere Diplomarbeiten. Den Bewerbern sollte aber klar sein, dass weniger programmiert, dafür aber umso mehr konfiguriert und recherchiert wird, was nicht weniger anspruchsvoll ist.
So sollte, sobald die PNO Quellen verfügbar sind, Profinet in das RT4Linux implementiert werden.

7 Glossar

ARP:
Adress **R**esolution **P**rotokoll, setzt die IP-Adresse in Relation zur MAC Adresse

Bash:
Bourne **a**gain **Sh**ell, Standard Befehlsinterpreter unter Linux

Feldgerät:
Sensor oder Aktor auf Feldebene.
Beispiele: Motor einer Maschine, Temperatursensor, Ventil, Roboter, etc.

Frame:
Datenpaket aus Nutzdaten und Header

Fusion:
Erweiterung von RTAI – ermöglicht das Starten von Echtzeitprogrammen im Userspace.

GNU GPL (**G**eneral **P**ublic **L**icense):
Lizenz zur Verbreitung freier Software (Open Source)
Die GNU GPL wurde von der Free Software Foundation herausgegeben und garantiert jedem folgende Rechte:

- Die Freiheit, ein Programm für jeden Zweck zu nutzen
- Die Freiheit, Kopien des Programms zu verkaufen oder kostenlos zu verteilen (wobei der Quellcode zur Verfügung gestellt werden muss)
- Die Freiheit, ein Programm den eigenen Bedürfnissen entsprechend zu ändern
- Die Freiheit, auch nach Punkt drei veränderte Programme unter den Regeln vom Punkt zwei zu vertreiben.

Header:
Bereich im Datenpaket mit Netzwerkinformationen wie Quell- und Zieladresse, Protokoll, etc.

Ethernet: Definition Wikipedia
Ethernet ist eine rahmenbasierte Computer-Vernetzungstechnologie für lokale Netze (LANs). Sie definiert Kabeltypen und Signalisierung für die Bitübertragungsschicht (physikalische Schicht) sowie Paketformate und Protokolle für die Medienzugriffskontrolle (Media Access Control, MAC)/Sicherungsschicht des OSI-Modells. Ethernet ist weitestgehend in der IEEE-Norm 802.3 standardisiert. Es wurde ab den 1990ern zur meistverwendeten LAN-Technologie und hat alle anderen LAN-Standards wie Token Ring, FDDI und ARCNET verdrängt. Ethernet kann die Basis für Netzwerkprotokolle, wie z. B. TCP/IP, AppleTalk oder DECnet bilden.
[Wikipedia 05 – Ethernet]

Jitter:
Der Jitter ist die zeitliche Abweichung der Differenz zwischen dem Ankommen zweier Pakete.Je kleiner der Jitter, desto regelmäßiger kommen die Pakete an. Im Beispiel: “200ms +- 2ms“ beträgt der Jitter 2ms.
Geringe Jitter sind in Regelungen, elektrisch gekoppelten Achsen, etc notwendig.
Ein Beispiel [Wenk 03] ist der Drehzahlgleichlauf mehrerer Antriebe bei kontinuierlichen Prozessen mit einer durchgehenden Wahrenbahn (Walzwerk, Wickler).

Master:
Teilnehmer eines Netzwerkes, welcher das Netzwerk koordiniert und parametriert.

MTU (**M**aximum **T**ransmission **U**nit)
Maximale Paketgröße in Computernetzen
Bei Ethernet 1518 bzw.1522Byte(VLAN) 1500 Byte Nutzdaten + 18 Byte (22Byte VLAN) Header

Paket
Datenpaket, bestehend aus Nutzdaten und Header.

RTAI:
RTAI ist eine Echtzeiterweiterung für Linux, welche dem eigentlichen Linux Kernel einen eigenen Kernel (ADEOS) vorschaltet, welcher die Kontrolle übernimmt, die Echtzeitprogramme ablaufen lässt und den Linux Kernel als Idle Task (niedrigste Priorität) laufen lässt.

Rtmac
Real**t**ime **M**edia **Ac**ess
RTnet Modul, dass den Zugriff des UDP/IP Stacks, wie auch den Zugriff des Nicht-Echtzeit TCP/IP Stacks auf die Netzwerkkarte regelt.

Rtos:
Real**t**ime **O**peration **S**ystem
Deterministisches Betriebssystem, d.h. es lässt sich einstellen, wie viel Rechenzeit welchem Prozess zugewiesen wird und, es lässt sich voraussagen, wann welcher Prozess gestartet wird.

routing :
Weiterleiten eines Datenpaketes in ein anderes Netz

Slave:
Teilnehmer eines Netzwerkes, welcher vom Master Verhaltensregeln zugewiesen bekommt (Sendezeitpunkt, Dauer, etc).

TDMA
Time **D**ivision **M**ultiple **A**ccess
Ein Buszugriffsverfahren, welches durch Zeitmultiplexen den Netzwerkverkehr regelt.

8 Quellen

[www.beckhoff.com]
-> Ethercat ->principle of operation 21.06.05

[Davie &Peterson 2000] Bruce S. Davie & Larry L. Peterson
Computernetze

[Göhring, Kauffels 90] Hans-Georg Göhring, Franz-Joachim Kauffels
Token Ring –Grundlagen, Strategien, Perspektiven

[Kizka 05] Jan Kizka
www.rtnet.org

[Kofler 2000] Michael Kofler
Linux - Installation, Konfiguration Anwendung

[Könen 93] Peter Lorenz Könen
Netzwerke zur Automatisierung

[Maurer 04] Wolfgang Maurer
Linux Kernelarchitektur – Konzepte, Strukturen und Algorithmen von Kernel 2.6

[Pfeifer 03]
Andreas Pfeifer Zeitschrift: Elektronik 07-2003

[Popp,Weber 2004] Manfred Popp, Karl Weber
Der Schnelleinstieg in PROFINET PNO

[Presseinformation EtherCat Technologiegroup 04]

[Profinet Broschüre 05]
Profinet – more than just Ethernet – Broschüre der Hannovermesse 2005

[siemes-homepage] 03.08.05
www2.automation.siemes.com/profinet/html_00/produkte/produkte_nav.htm]:

[Technologien mit Vorsprung IFAK]
www.ifak.system.de/fbk/profinet-io/index.php 22.06.05]

[Weibel 04] Hans Weibel
Zitat: Fachartikel Bulletin SEV/VSE 2004

[Wenk 03] Dr. Mathias Wenk
IEE 2003 Ausgabe Nr.3

[Wikipedia 05 - Echtzeit]
http://de.wikipedia.org/wiki/echtzeit 29.07.05

[Wikipedia 05 – Ethernet]
http://de.wikipedia.org/wiki/Ethernet 14.07.05

9 Abbildungsverzeichnis

31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik

Anhang:

Grundsätzlicher Ablauf der Installation:

Benötige Quellen für die Installation:
GNU System
Kernel
RTAI/Fusion
RTnet

Ablauf der Installation:
(Pfade und Einstellungen können je nach Projekt modifiziert werden)

Das GNU System wird mittels einer vorhandenen Linux Distribution (Suse, Fedora, etc) auf eine freie Partition kopiert.
Die Kernel-, RTAI-, RTnet- Quellen werden nach /opt/etx/src kopiert und entpackt.

Die Sourcen des Kernels werden mittels des in den RTAI/Fusion Quellen enthaltenen Patches modifiziert.
Dazu wird der für den Kernel passende Patch aus
../src/fusion-X.X/arch/i386/patches
in das Verzeichnis
../src/linux-X.X.X kopiert und der Patch ausgeführt:
Bash: patch –p <Name des patches>

Der Kernel wird aus den gepatchten Sourcen erstellt
Zuerst muss der zu bauende Kernel parametrisiert werden, das geschieht durch den Aufruf des Konfigurationsmenüs, im Verzeichnis der Kernelsourcen:
Bash: make menuconfig
Die Parameter können nach eigenen Vorgaben modifiziert werden, allerdings muss darauf geachtet werden, dass keine Netzwerkmodule fest in den Kernel kompiliert werden, sondern allenfalls als Module erstellt werden. Die Einstellungen für die Netzwerkmodule finden sich im Menü unter:
Device Drivers →
Networking support →
Ethernet (10 or 100 Mbit) →

Die Einstellung des Moduls wird in den spitzen Klammern angegeben: < >
Fest Einkompilierte werden durch ein Asterisk <*>,
modulare mit einem M <M> gekennzeichnet.

Die Kompilierung des Kernels erfolgt durch den Aufruf:
Bash: make bzImage

make bzImage prüft die Quellen auf Abhängigkeiten und kompiliert diese. Treten in dieser Phase Fehler auf, so kann geprüft werden ob alle Bibliotheken vorhanden und an den richtigen Stellen sind.
Um die Module zu erstellen wird make modules aufgerufen
Bash: make modules
Das Verschieben der Module aus den Sourcen in das Gnu System erfolgt durch:
Bash: make modules install

Den Kernel in das Boot Verzeichnis kopieren und neu starten
Der erstellte Kernel (bzImage) liegt unter ../src/linux-X.X.X/arch/i386/boot und muss nach /boot kopiert werden.

System neu starten (um den neuen Kernel zu laden)

RTAI/Fusion installieren:
RTAI/Fusion wird zunächst parametrisiert:
Bash: make menuconfig (in ../src/fusion-X.X.X/)
General →
Installation directory = /opt/etx/fusion
Linux build tree = ../src/Linux-X.X.X

Das kompilieren der Quellen und Anlegen der Verzeichnisse erfolgt durch:
Bash: make
Bash: make install

31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik

RTnet installieren:
Da RTnet über kein Menueconfig verfügt, wird es mit configure konfiguriert:
Bash: ./configure --with-RTAI=<PATH-TO-RTAI> options –prefix=<PREFIX>
Der configure muss im RTnet Source Ordner ../src/rtnet-X.X ausgeführt werden.
Der RTAI Pfad wird durch den Fusion Pfad ersetzt (/opt/etx/fusion), als Ziel (Prefix) wird /opt/etx/rtnet gewählt.
Optionen:

--enable-allpci	kompiliert die PCI NIC Treiber
--enable-fcc-enet	kompiliert den MPC8260 FCC Treiber
--enable-scc-enet	kompiliert den MPC8xx SCC Treiber
--enable-fec-enet	kompiliert den MPC8xx FEC Treiber
--enable-smc91111	kompiliert den SMSC LAN91C111 Treiber
--enable-rtcap	aktiviert die Capturing Unterstützung
--enable-net-routing	aktiviert das Routing
--enable-router	aktiviert IP forwarding
--disable-checks	Interne Fehlersuche wird nicht mit kompiliert

Die Liste aller Optionen erhält man durch aufrufen von:
Bash: ./configure –help

Beispiel:
Bash:
./configure –with-RTAI=/opt/etx/fusion -–enabel-allpci –prefix=/opt/etx/fusion

Das kompilieren der Quellen und anlegen des Verzeichnisses erfolgt durch:
Bash: make
Bash: make install

Das Verzeichnis ../etx/rtnet gliedert sich in die Unterverzeichnisse:

/sbin:	Konfigurations-Werkzeuge
/modules:	Kernelmodule
/include:	Header Dateien
/etc:	Konfigurations-Dateien für das Rtnet Start Script

Gerätedatei erstellen:
Bash: mknod /dev/rtnet c 10 240

31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik

In der Praxis traten auf diesem Wege mehrere Probleme aus oben genannten Gründen auf. Die Nachfolgenden Probleme sind ein Auszug der aufgetretenen Probleme:

Problem:	RTnet *make* findet verschiedene Dateien nicht
Ursache:	RTnet 8.1 ist nicht kompatibel mit Fusion 0.6.7
Lösung:	Neuestes Fusion installiert (0.7)
Problem	RTnet inkompatibel zum neuesten Fusion (0.7)
Lösung:	Fusion 0.6.9 installiert
Problem:	Fusion 0.6.9 Kernel Patch ist nicht kompatibel zu Kernel 2.6.11
Lösung:	Kernel 2.6.9 installiert
Problem	RTnet findet Dateien in RTnet/sbin nicht
Ursache:	Fehler im Startscript
Lösung:	manuell gepatcht (Anleitung aus der Mailinglist)
Problem	Netzwerkkarte inkompatibel
Lösung:	Erste Tests über Loopback (Lokalhost)
Problem:	Lokalhost nicht erreichbar
Ursache:	Lokalhost wird im Startscript 2x aufgerufen da ich ihn zusätzlich als Ersatz für die fehlende Ethernetkarte eingetragen habe. Dadurch wurde das Script nicht ordnungsgemäß bis zum Ende durchlaufen und einige Einstellungen, insbesondere der Start des Systemtimers und die Einstellung der Localhost-IP, nicht übernommen
Lösung:	Erstellung eines eigenen Scriptes.
Lösung:	kompatible Karte eingebaut (DEC 21x4x- based (Tulip)
Problem:	RTnet Startscript sucht im falschen Pfad nach den Modulen
Ursache:	Anderer Kernelname in Fusion menumake
Lösung:	Pfad manuell geändert (im Dateisystem)
Problem:	Error: RTnet line 304: $ TDMA_Config ambigouos redirect
Ursache:	RTnet Startscript findet einen IO nicht. Die Zeilen IP, Netmask, SlaveIP, in der Datei RTnet.config, waren wegen der Loopback Tests auskommentiert.
Lösung:	Einkommentieren

31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik

2. Rechner
Das komplettes Rtos wurde kopiert, allerdings ohne Tools und Sourcen da auf dem Zielsystem zuwenig Platz ist.

Problem:	RTnet Programme starten nicht mehr.
Ursache:	Bibliotheken fehlen!
Lösung:	Mittels eines speziellen Programms die nötigen Bibliotheken auf dem 1.Rechner ermittelt und auf das Zielsystem kopiert. (Die Dateien waren im nicht kopierten Tools Ordner)
Problem:	Dateien in /dev fehlen
Ursache:	/dev enthält Dateien welche die Hardware des Rechners darstellen und lassen sich nicht normal kopieren.
Lösung:	/dev vom TestRtos (andere Partition auf der gleichen Hardware) kopiert.
Problem:	Special Device RTnet fehlt unter /dev
Ursache:	/dev Ordner stammt von einem System ohne RTnet (TestRtos) Und kann nicht mit mknod erstellt werden dem Chassis die nötige Software fehlt.
Lösung:	Special device RTnet vom 1. Rechner kopiert (mit Parameter -a da keine normale Datei

Name: Schulz
Vorname: Jan – Peter
Matr.Nr.: 153797
Studiengang: Ingenieur - Informatik

An den Prüfungsausschuss
Des Fachbereichs Automatisierungstechnik
Der Fachhochschule Nordostniedersachsen

21339 Lüneburg

Erklärung zum Praxissemesterbericht
1 Praxissemester

Ich versichere, dass ich diese Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Scharnebeck, den .....

.....

31.08.2005
Jan-Peter Schulz
Universität Lüneburg
Ingenieur Informatik