GMD Research Series

GMD –
Forschungszentrum
Informationstechnik
GmbH

Spiro R. Trikaliotis

Uhrensynchronisation in einem lokalen Funknetzwerk

N° 5/2000

GMD – Forschungszentrum Informationstechnik GmbH
Schloß Birlinghoven
D-53754 Sankt Augustin, Germany
Telefon +49 -2241 -14 -0
Telefax +49 -2241 -14 -2618
http://www.gmd.de

Die vorliegende Veröffentlichung entstand im/
The present publication was prepared within:
Institut für Autonome intelligente Systeme (AiS)
Institute for Autonomous intelligent Systems
http://ais.gmd.de/

Anschrift des Verfassers/Address of the author:
Spiro R. Trikaliotis
Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Institut für Verteilte Systeme
Postfach 4120
D-39016 Magdeburg
E-mail: spiro@ivs.cs.uni-magdeburg.de

Die Deutsche Bibliothek - CIP-Einheitsaufnahme:
Trikaliotis, Spiro R.:
Uhrensynchronisation in einem lokalen Funknetzwerk / Spiro R. Trikaliotis.
GMD – Forschungszentrum Informationstechnik GmbH. - Sankt Augustin :
GMD – Forschungszentrum Informationstechnik, 2000
(GMD Research Series ; 2000, No. 5)
Zugl.: Bonn, Univ., Diplomarbeit, 2000
ISBN 3-88457-377-2

ISSN 1435-2699
ISBN 3-88457-377-2

Kurzzusammenfassung[1]

Die Gruppe Cooperative Autonomous Systems des Instituts für Autonome intelligente Systeme (AiS.CAS) der GMD – Forschungszentrum Informationstechnik GmbH befaßt sich in Kooperation mit dem Institut für verteilte Systeme (IVS) der Universität Magdeburg mit der Entwicklung einer Plattform, die es autonomen mobilen Systemen gestattet, kooperierend an der Durchführung von Aufgaben zu arbeiten. Diese autonomen Systeme müssen mit ihrer Umwelt interagieren. Hieraus ergeben sich Echtzeitanforderungen für die Systeme: Diese Interaktionen müssen zeitlich abgestimmt sein, sofern es sich um Eingriffe der Systeme in die Umwelt handelt. Handelt es sich um die Beobachtung der Umwelt, dann müssen die Interaktionen miteinander zeitlich in Bezug setzbar sein. In beiden Fällen benötigen die Systeme eine einheitliche Uhrzeit, die es erlaubt, diese Funktionen zuverlässig auszuführen.

Aus der Mobilität der Systeme ergibt sich, daß die Kommunikation drahtlos erfolgen muß, damit die Mobilität nicht eingeschränkt wird. Daher findet ein Funknetzwerk Anwendung.

Das Ziel der vorliegenden Diplomarbeit ist der Entwurf eines Synchronisationsprotokolls, welches auf einem Funknetzwerk Anwendung finden kann. Dabei wird den inhärenten Eigenschaften des Funknetzwerkes, die sich von klassischen drahtgebundenen Netzwerken in einigen Details unterscheiden, besondere Beachtung geschenkt.

Das entwickelte Protokoll wird analysiert, so daß eine Garantie gegeben werden kann, wie genau die synchronisierten Uhren sind. Das Protokoll ist in der Lage, Nachrichtenverluste bis zu einer konfigurierbaren Obergrenze zu tolerieren, ohne diese Garantie zu verletzen.

Neben der theoretischen Entwicklung und Analyse des Protokolles, welches auf ein Funk-Netzwerk nach dem Standard IEEE 802.11 zugeschnitten ist, bestand ein weiterer wesentlicher Aspekt der Arbeit darin, dieses Protokoll zu implementieren, und zwar erfolgte die Implementierung in einem Gerätetreiber unter Windows NT auf Funk-Netzwerkkarten „WaveLAN/IEEE“ der Firma Lucent Technologies. Mittels dieser Implementierung und eines speziell hierfür entwickelten Messaufbaus wurden hiernach umfangreiche Messungen der Implementierung vorgenommen, die die theoretischen Ergebnisse unterstützen.

Schlagworte: Funknetzwerk, IEEE 802.11, Echtzeitsystem, autonomes System, Uhrensynchronisation, Ratenanpassung.

---

[1] Die vorliegende Arbeit ist im Rahmen einer Kooperation der Gruppe CAS des Instituts für Autonome intelligente Systeme (AiS) der GMD – Forschungszentrum Informationstechnologie – mit dem Institut für Verteilte Systeme (IVS) der Universität Magdeburg entstanden.

Abstract

The research group Cooperative Autonomous Systems of the Institute for Autonomous intelligent systems (AiS.CAS) of the GMD – German national research center for information technology, in cooperation with the institute for distributed systems of the University of Magdeburg, develops a common platform for autonomous mobile systems which allows these systems to reach a common goal in a cooperative effort. To achieve this, these systems have to interact with their own environment. From this, we derive some real-time constraints: These interactions have to be timed exactly in the case the systems have to interact with their environment, in the case these are observations of the environment, we must be able to establish a timely order. In both cases, the system needs a global time allowing to perform these functions.

Since the systems are mobile, the communication between the autonomous systems has to be wireless to preserve the mobility. Thus, a wireless LAN is used.

This diploma thesis develops a time synchronization protocol which can be used on a wireless network. This protocol copes with the inherent properties of a wireless LAN that distinguish it from a classical wired LAN.

This protocol is analyzed in detail, so that we can guarantee the provided precision of the synchronized clocks under any specified circumstances. These circumstances comprise up to a configurable upper limit for messages losses which is crucial for the wireless medium.

Besides the theoretical development und analyzation of the protocol, which is suited for a wireless LAN of the standard IEEE 802.11, this diploma thesis implements this protocol in a device driver under Windows NT, using wireless LAN network interface cards “WaveLAN/IEEE” of Lucent Technologies, Inc. With the help of this implementation and a measurement environment specifically designed for this purpose, this implementation has been evaluated in detail, confirming the theoretical results.

Keywords: Wireless LAN, IEEE 802.11, real-time system, autonomous system, clock synchronization, continuous clock.

Vorwort

Diese Diplomarbeit entstand in der Gruppe Cooperative Autonomous Systems des Instituts für Autonome intelligente Systeme (AiS.CAS) der GMD. Bei allen Mitarbeitern dieser Gruppe und des ehemaligen Forschungsbereiches Responsive Systeme, die mich bei der Erstellung dieser Diplomarbeit auf vielfältige Weise unterstützt haben, möchte ich mich an dieser Stelle herzlich bedanken. Einige Personen sollen dabei besonders erwähnt werden. Bei Herrn Prof. Dr. Edgar Nett möchte ich mich dafür bedanken, daß er das Thema der Arbeit gestellt hat. Seine vielfältig gegebene konstruktive Kritik und Denkanstöße haben zu ihrem Gelingen und ihrer jetzigen Form maßgeblich beigetragen. Bei meinen Betreuern Michael Mock und Reiner Frings möchte ich mich für die viele Zeit bedanken, die sie sich genommen haben, um meine vielfältigen Fragen zu beantworten. Ebenso möchte ich mich auch bei Martin Gergeleit bedanken, der mir vor allem bei programmiertechnischen Fragen jederzeit hilfreich zur Seite stand sowie vorläufige Versionen dieser Arbeit mit vielfältigen Kommentaren versah. Ein herzliches Dankeschön geht auch an meinen Kommilitonen Stefan Schemmer für die vielen Diskussionen. Herzlichst Bedanken möchte ich mich weiterhin bei meinen Eltern, die mir das Studium ermöglichten und mich in jeder Situation unterstützten. Nicht zuletzt bedanken möchte ich mich auch bei meinen Kommilitonen und Freunden, die die Unannehmlichkeiten des Studiums zur rechten Zeit vergessen ließen.

Sankt Augustin, im Februar 2000.

Spiro Trikaliotis

Inhaltsverzeichnis

1 Einleitung

1.1 Kontext der Diplomarbeit

Computersysteme werden häufig so eingesetzt, daß Sie mit der sie umgebenden Umwelt, der realen Welt, in starkem Maße interagieren und steuernd eingreifen. Sie erhalten von Sensoren Informationen über die sie umgebende Welt und greifen über Aktoren in die Umwelt ein. Beispiele hierfür sind das Motorenmanagement oder ein Anti-Blockier-System (ABS) im Auto, Steuercomputer in der Flugzeugsicherung oder die Computersysteme, die ein Flugzeug steuern, aber auch die Steuercomputer in Kraftwerken oder Preßwerken.

Man spricht hier von Echtzeitsystemen um herauszustellen, daß in diesen Anwendungen Programme nicht nur richtige Ergebnisse, sondern diese auch innerhalb genau spezifizierter Zeiten liefern müssen, weil sonst teure Geräte zerstört oder gar Menschenleben gefährdet werden können. Hieraus ergibt sich unmittelbar, daß Echtzeitsysteme auch große Anforderungen an die Zuverlässigkeit stellen, man muß also Garantien abgeben können, daß die Systeme die ihnen zugewiesenen Aufgaben zeitlich und funktional korrekt erfüllen.

Zumeist wird eine Unterscheidung zwischen „weichen“ und „harten“ Echtzeitsystemen (*soft real-time* bzw. *hard real-time*) vorgenommen. Beiden gemeinsam ist, daß es externe Zeitanforderungen gibt, die die Systeme einhalten müssen, um ihre Funktion zu erfüllen. Während bei weichen Echtzeitsystemen aber durch die Nicht-Erfüllung der Anforderungen lediglich eine Einschränkung der Qualität der Funktion einhergeht, können bei harten Echtzeitsystemen durch die Nicht-Erfüllung der zeitlichen Anforderungen große materielle Schäden oder Personenschäden entstehen. Ein Beispiel für ein weiches Echtzeitsystem ist ein Video-on-Demand-Server, bei dem durch die Nicht-Rechtzeitigkeit des Services ein Ruckeln im Bildschirmaufbau möglich ist; Beispiele für harte Echtzeitsysteme sind oben schon gegeben worden. In dieser Diplomarbeit wird durchgehend von den Anforderungen eines harten Echtzeitsystems ausgegangen.

Harten Echtzeitsystemen wird eine hohe Zuverlässigkeit abverlangt, weil ein Defekt in einer Katastrophe enden kann. Aus diesem Grund führt man Echtzeitsysteme häufig verteilt aus, d. h., man setzt mehrere Computer ein, die gemeinsam an der Lösung der gestellten Aufgabe arbeiten. Zwar erhöht sich hiermit die Wahrscheinlichkeit eines Defektes einer Komponente des Systems, da mehr Komponenten vorhanden sind. Andererseits kann durch einen geeigneten Entwurf dieser Systeme sichergestellt werden, daß der Ausfall einzelner Komponenten des Systems nicht zu einem kompletten Ausfall führt, sondern weiterhin eine geeignete Basisfunktionalität zur Verfügung steht, die eine Katastrophe verhindert. Dies ist insbesondere bei Anwendungen wichtig, bei denen der Computer im Falle eines Defektes nicht in einen „sicheren Zustand“ überführt werden kann, beispielsweise bei der Steuerung eines Flugzeuges.

Ein verteiltes Echtzeitsystem ist wie jedes verteilte Computersystem darauf angewiesen, Aufgaben verschiedener Stationen zu koordinieren und gemeinsam genutzte Daten miteinander auszutauschen. Damit dieser Datenaustausch erfolgen kann, ist als eine wesentliche Komponente jedes verteilten Systems ein Netzwerk vorhanden, welches aus Kommunikationsverbindungen für die beteiligten Stationen besteht.

Manche Anwendungen von Echtzeitsystemen verlangen, daß die beteiligten Computer nicht stationär, sondern mobil sind. In solchen Fällen ist es nicht möglich, die Kommunikation in dem Netzwerk über Drähte wie Kupferleitungen oder Lichtwellenleiter vorzunehmen, wie es klassischerweise realisiert wird. Vielmehr werden drahtlose Übertragungstechniken benötigt, die über Licht- oder Funksignale den Informationsaustausch realisieren. Drahtlose Netzwerke besitzen im Allgemeinen gegenüber drahtgebundenen Netzwerken geringere Bandbreiten und höhere Fehlerraten infolge des Nicht-Vorhandenseins eines geschützten Übertragungsmediums. Sowohl das Licht als auch die Funkwellen besitzen nur begrenzte Reichweiten und können durch Gegenstände zwischen oder in der Nähe der kommunizierenden Stationen gestört werden, außerdem können die ihren Ort wechselnden Computer sich aus den gegenseitigen Sende- und Empfangsbereichen entfernen. Daher ist die Erreichbarkeit der Stationen untereinander nicht mit Sicherheit gewährleistet. Hierdurch kann es passieren, daß sich zwei Stationen spontan nicht mehr gegenseitig empfangen können, oder kurzzeitig eine Verbindung möglich wird, die vorher nicht möglich war, wobei beide Fälle mit den gleichen Stationen innerhalb kürzester Zeit auftreten können. Dieses Problem wird bei drahtgebundenen Netzwerken zumeist ignoriert, da es – wenn überhaupt – recht selten auftritt, bei drahtlosen Netzwerken handelt es sich hingegen um ein wesentliches Merkmal.

In einem Echtzeitsystem benötigt man auf allen Stationen eine einheitliche Uhrzeit, d. h., die beteiligten Stationen müssen eine konsistente Sicht der Zeit besitzen. Dies liegt daran, daß in Echtzeitsystemen Ereignisse in kausale Zusammenhänge gesetzt werden müssen. Hierzu benötigt man die Kenntnis der zeitlichen Abhängigkeiten, d. h., einer Vorher- / Nachher-Beziehung, die nur mit einer einheitlichen Uhrzeit ermittelt werden kann. Weiterhin müssen auszuführende Aktionen koordiniert werden, wozu sich ebenfalls die Uhrzeit anbietet (s. [La 78]).

Um diese einheitliche Uhrzeit zu ermöglichen, benötigt man auf jeder Station eine physikalische Uhr. Es ist technisch unmöglich zwei Uhren zu konstruieren, die in absolut identischen Abständen Signale erzeugen; eine von beiden wird daher schneller laufen als die andere. Daraus ergibt sich, daß die Differenz zwischen zwei (physikalischen) Uhren, die zu einem Zeitpunkt den gleichen Zählerstand besitzen, nach endlicher Zeit beliebig groß werden kann.

Um dies zu vermeiden, benutzt man Uhrensynchronisationsprotokolle; diese sorgen dafür, daß die Zählerstände der Uhren in regelmäßigen Abständen verändert werden, so daß eine maximale Obergrenze für die Differenz der Zählerstände der Uhren garantiert wird, die sogenannte *Präzision.*

1.2 Ziele der Diplomarbeit

Im Rahmen dieser Diplomarbeit soll ein Uhrensynchronisationsprotokoll entworfen und implementiert werden, welches auf einem drahtlosen Netzwerk nach dem Standard IEEE 802.11 ([IEEE 97]) eine zuverlässige Synchronisation ermöglicht. Zuverlässig bedeutet in diesem Zusammenhang, daß Garantien über die erreichte Präzision gegeben werden können, sofern spezifizierte Fehlerannahmen nicht verletzt werden.

Als Arbeitsumgebung für das Protokoll wird ein Funknetzwerk nach dem Standard IEEE 802.11 verwendet. Dieser Standard hat gegenüber anderen Standards für lokale Funknetze (wie HIPERLAN) und proprietären Lösungen (z. B. Proxims OpenAir oder Symphony) den Vorteil, daß er verbreitet ist, von vielen Herstellern unterstützt wird und damit den Einsatz von vergleichsweise kostengünstigen Standard-Komponenten ermöglicht. Weiterhin ist durch die weite Verbreitung des Standards und dadurch, daß er weiterentwickelt wird, mit großer Sicherheit auch in Zukunft Unterstützung hierfür zu erwarten.

Im IEEE-Standard ist ein Uhrensynchronisationsprotokoll vorgesehen, welches für interne Zwecke benötigt wird. Dieses hat aber einige Schwachstellen in Bezug auf die erreichbte Präzision. Das hier zu entwickelnde Uhrensynchronisationsprotokoll soll den IEEE-Standard dahingehend erweitern, daß eine Uhrensychronisation vorhanden ist, die folgenden Anforderungen genügt, die sich aus der Echtzeitfähigkeit ergeben:

- eine hohe Präzision muß jederzeit gewährleistet sein;
- es sollen keine Zeitsprünge auftreten;
- das Verfahren muß in der Lage sein, Fehler zu tolerieren.

Die Anforderung der hohen Präzision ergibt sich unmittelbar aus der geforderten Echtzeitfähigkeit des Protokolls. Bei einer hohen Präzision sind die Uhren der beteiligten Stationen garantiert dicht beieinander, dadurch können gemeinsame Aktionen auch zeitlich exakter koordiniert werden.

Aus der Echtzeitanforderung ergibt sich weiterhin, daß keine Zeitsprünge auftreten sollen. Hierzu muß man sich zuerst bewußt machen, daß bei einer Uhrensynchronisation Zeitsprünge deshalb auftreten können, weil die Uhren umgestellt werden müssen. Springt nun die Zeit rückwärts, dann können Ereignisse, die nacheinander aufgetreten sind, mit Zeitstempeln versehen werden, die eine gegenteilige Beziehung implizieren. Es ist nicht möglich, Ereignisse zeitlich zu ordnen, deren Zeitstempel näher beieinander liegen als die maximale Größe eines Zeitsprunges. Ähnliche Probleme ergeben sich, wenn die Zeit vorwärts gestellt wird. Treten hingegen keine Zeitsprünge auf, dann kann eine solche Inversion der Reihenfolge aus diesem Grunde nicht mehr auftreten.

Die Notwendigkeit der Fehlertoleranz schließlich ergibt sich ebenfalls aus der geforderten Echtzeitfähigkeit, da einzelne Fehler nicht sofort das ganze System zusammenbrechen lassen dürfen, da dies hohe Kosten oder Personenschäden verursachen könnte. Sie

ergibt sich aber auch aus dem Umstand, daß das Funknetzwerk – im Gegensatz zu drahtgebundenen Netzwerken – eine relativ geringe Verläßlichkeit besitzt, da kein abgeschirmtes Medium wie beispielsweise eine Kupferleitung oder ein Lichtwellenleiter existiert, sowie aus der Mobilität der zu synchronisierenden Computersysteme.

Weil das hier entwickelte Protokoll eine Erweiterung des IEEE-Standards darstellen soll, ergibt sich weiterhin die Anforderung, daß es so entwickelt wird, daß jede standard-konforme Implementierung von Netzwerkkomponenten so ausgestattet werden kann, daß sie das Protokoll ausführen kann, andererseits aber nicht-erweiterte Komponenten durch das Protokoll nicht in ihrer Funktion beeinträchtigt werden dürfen.

Bevor ein Protokoll, welches Fehlertoleranz-Eigenschaften garantieren soll, entwickelt werden kann, müssen zuerst Annahmen über die möglichen auftretenden Fehler, das sogenannte Fehlermodell, formuliert werden. Ein Fehlermodell, in dem die möglichen auftretenden Fehler beschränkt werden, ist aus dem Grunde notwendig, da sonst im Extremfall gar keine Kommunikation mehr angenommen werden könnte, weil alle Nachrichten verloren gehen könnten, wenn man beliebige auftretende Fehler berücksichtigt. In diesem Fall wäre keinerlei Synchronisation möglich. Andererseits darf ein Fehlermodell auch nicht zu restriktiv sein: Eine Annahme, daß Nachrichtenverluste niemals auftreten, erleichtert zwar das theoretische Arbeiten, ist jedoch für die Praxis kaum relevant.

Daher wird ein „vernünftiges“ Fehlermodell für die oben angegebene Systemumgebung entwickelt, welches eine große Überdeckung (*assumption coverage*) besitzt, d. h. die Wahrscheinlichkeit, daß die getroffenen Annahmen in der Wirklichkeit zutreffen, soll hoch sein.

Über die beteiligten physikalischen Uhren wird die Annahme getroffen, daß diese zu jeder Zeit die richtige Anzahl der lokal vergangenen Zeiteinheiten seit ihrem Start anzeigen und ihre Drift von der realen Zeit begrenzt ist. Da eine Verletzung dieser Voraussetzung in der Praxis sehr selten ist, ist diese Annahme gerechtfertigt.

Weiterhin werden folgende Annahmen über die an der Synchronisation beteiligten Stationen und das Funknetzwerk gemacht:

- Jede Station kann jederzeit den geleisteten Dienst beenden, wobei die Station bis zu diesem Zeitpunkt fehlerfrei gearbeitet hat (*crash*-Fehler, sogenanntes *fail-silent* Verhalten).

- Jede Station kann Empfangsfehler (*receive omissions*) haben, d. h., jeder Station ist es möglich, erfolgreich versendete Nachrichten nicht zu empfangen. Allerdings ist die Anzahl der Empfangsfehler beschränkt.

- Ansonsten treten keine Fehler des Netzwerkes auf.

Diese Fehler sollen von dem entwickelten Protokoll toleriert werden können.

Neben der Entwicklung eines Protokolls, welches obige Anforderungen erfüllt, besteht ein wesentlicher Teil der Diplomarbeit darin, dieses Protokoll zu implementieren und zu vermessen, wobei die Meßergebnisse mit den analytisch bestimmten Eigenschaften verglichen werden.

Der Entwurf und die Implementierung des Protokolls dient dem Zweck, eine Technologie zu entwickeln, die durch drahtlose Kommunikation zwischen Fahrzeugen eine automatisierte Kooperation zwischen den Fahrzeugen und damit eine Verbesserung des Verkehrsflusses erlaubt. Dieses Szenario wird im Rahmen des DECOR- (‚*dependable cooperative real-time systems*') Projektes, einem Kooperationsprojekt zwischen dem Institut für Autonome Intelligente System der GMD (GMD-AiS), dem Institut für verteilte Systeme (IVS) der Universität Magdeburg und den italienischen Forschungsinstituten IEI-CNR und CNUCE-CNR, betrachtet.

1.3 Ergebnisse der Diplomarbeit

Das in dieser Diplomarbeit entwickelte Protokoll besitzt folgende Struktur: Es handelt sich um ein Master-/Slave-Protokoll, d. h., es existiert eine ausgezeichnete Station, der Zeitmaster, dessen Uhrzeit die anderen Stationen, die Slaves, übernehmen. Hierzu versendet der Zeitmaster in regelmäßigen Abständen eine Nachricht an alle Stationen.

Diese Nachricht hat zwei Aufgaben: Zum einen dient der Zeitpunkt, an dem diese Nachricht empfangen wird, als Indikation, d. h. als global beobachtbares Ereignis; dies bedeutet, daß der Empfang jeder dieser Nachrichten von jeder beteiligten Station einschließlich des Masters mit einem Zeitstempel versehen und abgespeichert wird.

Zum anderen werden in jeder Nachricht vom Zeitmaster die Zeitstempel versendet, die er selber den *n* vorherigen Nachrichten zugeordnet hat. Damit kann ein Slave, der eine Nachricht empfängt, die Differenz seiner Uhr zu der Uhr des Masters bestimmen, die galt, als das Ereignis „Empfang der Nachricht" eingetreten ist. Diese Differenz kann nun als Korrekturwert für die Uhr des Slaves benutzt werden. Um Sprünge der Uhrzeit zu vermeiden, wird von dem Protokoll diese Differenz für die Implementierung einer Ratenanpassung genutzt; dies bedeutet, daß die Uhr so beschleunigt oder abgebremst wird, so daß ihre Geschwindigkeit nach einer Einschwingphase der erwarteten Geschwindigkeit der Uhr des Masters entspricht. Hierdurch wird erreicht, daß zukünftige Korrekturwerte kleiner ausfallen.

Da pro Nachricht *n* Zeitstempel, die mit verschiedenen Indikationen korrespondieren, versendet werden, kann der Verlust von bis zu *n*-1 aufeinanderfolgenden Synchronisationsnachrichten toleriert werden. Trotzdem wird pro Synchronisationsrunde nur eine relativ kleine Nachricht benötigt, die vom Zeitmaster versendet wird. Hierdurch wird sparsam mit der vorhandenen Netzwerkbandbreite umgegangen.

Damit das Versenden der Zeitnachrichten im Protokoll des IEEE-Standards garantiert werden kann, werden diese in einer speziellen Nachricht, dem sogenannten Beacon-

Frame, welches Priorität auf dem Netzwerkmedium besitzt, versendet. Damit das hier entwickelte Protokoll diese Garantie ebenfalls geben kann, wird die eigene Nachricht entweder auch in einem Frame mit Priorität versendet oder als Zusatz in den Beacon integriert.

Als wesentlicher Bestandteil dieser Diplomarbeit wird das entwickelte Protokoll analytisch und durch Messungen an einer Implementierung untersucht. Als Zielsystem dieser Implementierung wurde Windows NT 4.0 auf Standard-Intel-Computern und als Funknetzwerk wurden IEEE-konforme Funknetzwerkkarten „WaveLAN-II“ der Firma Lucent Technologies benutzt, wobei das hier entwickelte Uhrensynchronisationsprotokoll in den Gerätetreiber der Netzwerkkarte integriert wurde. Durch die Vermessung der Implementierung können weiterhin in der Analyse offengebliebene Parameter der Zielumgebung bestimmt werden.

Um einen Vergleich mit dem Protokoll des IEEE-Standards vornehmen zu können, wird auch dieses implementiert und ebenfalls vermessen.

Bei den Messungen ergeben sich für die Differenzen der Uhren der beteiligten Stationen folgende Meßwerte bei dem im Rahmen der Diplomarbeit entwickelten Protokoll: Die synchronisierten Uhren der beteiligten Stationen haben untereinander maximale Abweichungen von unter 150 µs bei Verwendung des hier entwickelten Protokolls, während beim Protokoll nach dem IEEE-Standard die maximalen Abweichungen etwa 900 µs betragen.

Damit werden die Forderungen aus Abschnitt 1.2 „Ziele der Diplomarbeit“ wie folgt erfüllt:

- Die gemessene Präzision des Protokolls ist hoch, sie beträgt ca. 150 µs.
- Durch die Ratenanpassung treten keine Zeitsprünge auf, mit der Ausnahme des ersten Synchronisationsvorganges.
- Das Protokoll ist kompatibel zum IEEE-Standard, da lediglich eine weitere Nachricht versendet wird, die von Stationen, die das Protokoll nicht beherrschen, nicht verarbeitet wird.
- Fehler im Rahmen der gestellten Anforderungen können toleriert werden.

1.4 Aufbau der Diplomarbeit

Die Gliederung dieser Diplomarbeit ergibt sich wie folgt: In Kapitel 2 „Grundlagen“ werden die Grundlagen für die Diplomarbeit erläutert, insbesondere wird auf die drahtlosen Netzwerke eingegangen und die spezifischen Aspekte eines drahtlosen Netzwerkes nach dem Standard IEEE 802.11 erläutert.

In Kapitel 3 „Uhrensynchronisation und verwandte Arbeiten“ werden zuerst die Grundbegriffe und Prinzipien einer Uhrensynchronisation vorgestellt, worauf sich die Präsentation verwandter Arbeiten anschließt. Insbesondere werden hier das Protokoll nach dem IEEE-Standard, welches in dieser Diplomarbeit durch ein zuverlässigeres ersetzt werden soll, als auch das GS-Protokoll, dessen Grundgedanke den Ausgangspunkt des hier entwickelten Protokolls darstellt, vorgestellt. Kapitel 4 „Uhrensynchronisationsprotokoll für das Funknetzwerk“ beschreibt dann das Protokoll, welches für die anvisierte Zielumgebung entwickelt wurde. Dabei wird zuerst das eigentliche Protokoll vorgestellt und dann die Art der Ratenanpassung, die speziell für dieses Protokoll entwickelt wurde. Hieran schließt sich eine formale Darstellung des Protokolls an, mit deren Hilfe das Protokoll anschließend analysiert wird.

Das Kapitel 5 „Implementierung“ beschreibt die Implementationsumgebung und die Implementierung des Protokolls. Mit Hilfe dieser Implementierung werden anschließend Messungen vorgenommen, die in Kapitel 6 „Messungen“ präsentiert und bewertet werden. Kapitel 7 „Fazit“ schließlich faßt die Ergebnisse zusammen und gibt einen Ausblick.

2 Grundlagen

Diese Diplomarbeit ist eingebettet in eine Reihe von Forschungsarbeiten, die das gemeinsame Ziel haben, Methoden und Mechanismen bereitzustellen, um in Zukunft ein vorhersagbares Verhalten von Echtzeit-Kontrollanwendungen auch in nicht vollständig determinierten Umgebungen zu erreichen ([KN 98]).

Forschungsgegenstände sind dabei die Verbindung von Echtzeit und Fehlertoleranz, adaptives Scheduling ([GMN 98]), die Kommunikation und Kooperation von verteilten Echtzeitsystemen ([MN 99]) und als einen wesentlichen Aspekt für viele Anwendungen auch Echtzeitkommunikation über Funknetzwerke ([Sch 00]). Dabei tragen die physikalischen Besonderheiten dieser Netzwerktechnologie selbst bereits wesentlich zu den schwer abschätzbaren Umgebungsparametern bei, die es zu beherrschen gilt. In diesem Kapitel wird ein Überblick über diese Umgebung gegeben.

2.1 Verteilte Echtzeitsysteme

Der Begriff des verteilten Echtzeitsystems besteht aus zwei Komponenten, dem verteilten System und dem Echtzeitsystem. Beide sollen hier zunächst für sich dargestellt und danach ihre Kombination erläutert werden.

2.1.1 Verteilte Systeme

Ein verteiltes System besteht aus mehreren Einzelrechnern, die an einem gemeinsamen Ziel arbeiten. Ein solches gemeinsames Ziel kann beispielsweise die Haltung von Daten sein, wie sie in einer sogenannten verteilten Datenbank auftreten würde.

Damit ein verteiltes System sinnvoll ist, muß die Möglichkeit einer Kommunikation der Stationen miteinander vorgesehen werden, da ansonsten jede einzelne Station nur für sich arbeiten könnte. Hierzu bedarf es spezieller Kommunikationskanäle, welche den Informationsaustausch ermöglichen. Diese Kommunikationskanäle und die zugehörigen Stationen werden als Netzwerk bezeichnet.

Gegenüber der Datenhaltung innerhalb eines einzelnen Computers sind die externen Kommunikationskanäle allerdings langsam und fehleranfällig, so daß die Kommunikation gründlich geplant werden muß, um das System effektiv zu halten[2]. Daher ist es zumeist nicht möglich, allen Stationen alle Informationen des gesamten Systems zur Verfügung zu stellen, vielmehr beschränkt man sich zumeist auf die zur Erzielung des Ergebnisses unbedingt erforderlichen Informationen.

---

[2] Bei modernen Hochgeschwindigkeitsnetzen hat sich dieses Verhältnis inzwischen ins Gegenteil verkehrt (s. [Ta 96]). Bei den im Rahmen dieser Diplomarbeit betrachteten Funknetzwerken gilt diese Eigenschaft allerdings.

Daher ist in einem verteilten System der Zustand des Systems, das heißt die Informationen, die das gesamte System ausmachen, über die einzelnen Stationen verteilt, jede Station besitzt somit nur die Kenntnis über einen Teil der Informationen. Jede Station kann nur aufgrund der ihm zur Verfügung stehenden Informationen agieren, so daß die Informationsverarbeitung eine hohe Lokalität aufweist.

Diese hohe Lokalität und der Umstand, daß relativ wenige Informationen in jeder Station benötigt und vorhanden sind, erlaubt den Einsatz von relativ kleinen Computern, die daher kostengünstig sind. In einem Ansatz, in dem ein einzelner Rechner an der Aufgabe rechnen würde, müßte dieser Rechner wesentlich leistungsfähiger (und damit teurer) sein.

Besteht ein Computersystem aus einem einzelnen Computer, dann führt der Ausfall einer einzelnen Komponente, beispielsweise der Stromversorgung oder einer Festplatte mit lebenswichtigen Informationen, zumeist zum kompletten Ausfall des gesamten Systems. Um diesem Risiko zu begegnen werden häufig redundante Geräte in den Computer eingebaut. Allerdings ist der Aufwand solcher Ersatz-Komponenten recht hoch, vor allem, da diese Ersatz-Komponenten im normalen Betrieb nicht mitbenutzt werden können, weshalb ihre (teure) Funktionalität die meiste Zeit ungenutzt bleibt. Bei einem verteilten System ist man hingegen in der Lage, dieses so zu entwerfen, daß die einzelnen Stationen sich gegenseitig kontrollieren und überwachen. Wichtige Aufgaben können redundant auf die Stationen verteilt werden, so daß bei einem Totalausfall eines einzelnen oder gar mehrerer Stationen noch eine Grundfunktionalität zur Verfügung steht. Ein verteiltes System mit dieser Eigenschaft erfüllt die sogenannte *partial failure property*, d. h. es hat die Eigenschaft, daß der Ausfall von Komponenten das System nicht komplett ausfallen läßt, sondern dieses immer noch eine Grundfunktion zur Verfügung stellt.

Häufig werden für ein verteiltes System weitere Forderungen aufgestellt, beispielsweise ein einheitlicher Zugriff auf den Service und die Ressourcen des Betriebssystems, sowie einige global geltende Eigenschaften wie Zugriffsberechtigungen, Namen und Verwaltung des Systems (vgl. [Sch 93]). Diese zusätzlichen Eigenschaften werden im Umfeld dieser Diplomarbeit nicht benötigt, so daß hier nicht näher darauf eingegangen wird.

2.1.2 Echtzeitsysteme

Ein Echtzeitsystem besteht aus einem Echtzeit-Computersystem und seiner Umgebung. Die Umgebung des Echtzeitsystems besteht aus dem Operator und dem kontrollierten Objekt (s. [Ko 97]). Zwischen dem Computersystem und seiner Umgebung liegt eine Schnittstelle, bestehend aus Sensoren und Aktoren, die es dem Computersystem erlaubt, Informationen aus der Umgebung zu sammeln und auf diese einzuwirken.

Ein Beispiel für ein Echtzeitsystem ist eine Fertigungsanlage, die von einem Computersystem gesteuert wird. Das Echtzeit-Computersystem sind dann der oder die Computer, die die Steuerung übernehmen, das kontrollierte Objekt besteht aus den Fertigungsbän-

dern, Stanzanlagen und ähnlichen Geräten, auf die der Computer einwirken kann, und der Operator ist der Mensch, der dem Computer vorgibt, was dieser zu tun hat.

Das Echtzeit-Computersystem besteht aus einem oder mehreren Computern, die Daten aus der Umgebung sammeln, um dann in geeigneter Weise auf die Umgebung steuernd einzuwirken. Damit das Computersystem hierzu in der Lage ist, besitzt es Sensoren, die es ihm erlauben, die Daten der Umgebung zu sammeln, und Aktoren, mit denen es auf die Umgebung einwirkt. Beispiele für Sensoren sind Temperatur-, Feuchtigkeits oder Geschwindigkeitsmeßfühler, Beispiele für Aktoren sind Pumpen und Motoren, die der Computer steuern kann.

Die Korrektheit eines Echtzeit-Computersystem verlangt nicht nur, daß eine bestimmte Funktion erbracht wird, wie es beispielsweise bei mathematischen Rechnungen oder Simulationen verlangt würde, sondern auch, daß diese Funktion innerhalb vorgegebener Zeitschranken berechnet wird (vgl. [Ko 97]).

Diese Zeitanforderungen ergeben sich dadurch, daß das System auf bestimmte Eingangszustände innerhalb gewisser Zeiten reagieren muß. Dies ergibt sich aus der Umgebung, in der das Echtzeitsystem operiert. Beispielsweise kann die Forderung bestehen, daß auf einen Sensor, der ein drohendes Überlaufen eines Gefäßes anzeigt, innerhalb einer festen Zeit reagiert werden muß, indem z. B. die Flüssigkeitszufuhr gedrosselt wird, weil sonst das Gefäß überläuft. Auch kann sich ein Motor, der längere Zeit gegen einen Anschlag arbeitet, überhitzen. Dadurch muß die maximale Zeit, die der Motor eingeschaltet ist, beschränkt sein. In anderen Fällen gibt es auch Mindestzeiten, die ein Aktor aktiv sein muß – beispielsweise, weil die Umgebung oder der Aktor träge ist, so daß eine kurze Aktivierung gar keinen Einfluß nehmen kann – oder die Aktivierungszeit muß sich innerhalb eines genauen Fensters bewegen, d. h., nicht zu kurz, aber auch nicht zu lang.

Echtzeitsysteme kontrollieren häufig Umgebungen, bei denen es um Menschenleben oder zumindest Material mit hohen Gesamtkosten geht. Dadurch, daß das Echtzeitsystem auf diese Umwelt Einfluß nimmt, ergeben sich zumeist hohe Anforderungen an die Zuverlässigkeit eines solchen Systems, da eine Fehlfunktion hohe materielle oder gar Personenschäden bewirken kann. Ein Beispiel für eine besonders kritische Anwendung ist das Computersystem eines Flugzeuges. Da ein Flugzeug während des Fluges nicht in einen sicheren Zustand gebracht werden kann, ist es besonders wichtig, daß das Computersystem jederzeit und unter jeglicher Belastungssituation funktioniert, sogar bei Ausfall von Komponenten des Systems. Diese Eigenschaft läßt sich von einem einzelnen Computer nur mit sehr hohem Aufwand erreichen. Es bietet sich an, solche System mit hohen Verläßlichkeitsanforderungen als verteilte Echtzeitsysteme auszuführen, d. h. als Kombination der Vorteile eines Echtzeitsystems mit den Vorteilen eines verteilten Systems. Besonders wichtig ist dabei, daß die *partial failure property* erfüllt ist.

2.2 Fehlertoleranz

Der Begriff der Fehlertoleranz beschreibt die Eigenschaft, daß ein Computersystem ohne Ausfälle oder Fehlfunktionen arbeitet, d. h. die Fähigkeit, seine nach seiner Spezifikation erwartete Aufgabe zu erfüllen, und zwar auch bei auftretenden Fehlern in seiner Hard- oder Software oder der Anwendungssoftware, die auf dem System läuft ([Ne 91]). Um die dahinter steckenden Konzepte zu verstehen, soll zuerst der Begriff des Fehlers genauer erklärt werden (siehe auch [Lap 92]).

Ein Fehlzustand (*error*) ist ein (interner) Zustand des betrachteten Systems, der eine Abweichung darstellt von dem Zustand, der aufgrund der inneren und äußeren Umstände und der Spezifikation eingenommen werden sollte. Der Fehlzustand wird hervorgerufen von einer Fehlerursache (*fault*); dies ist ein Ereignis, welches so nicht vorgesehen war, damit eine Abweichung von der Spezifikation der möglichen Ereignisse darstellt und somit zum Erhalt eines Fehlzustandes führen kann. Der Fehlzustand wiederum kann zu einem Ausfall (*failure*), das heißt, zu einer Abweichung von der spezifizierten Leistung des gesamten Systems, führen. Man beachte, daß im Deutschen der Begriff „Fehler“ sowohl für den Begriff „Fehlerzustand“ als auch für den Begriff „Fehlerursache“ stehen kann.

Ein System ist fehlertolerant, wenn das System so entworfen wurde, daß eine Fehlerursache keinen Ausfall erzeugt. Dieses Ziel wird zumeist mit dem Einsatz von Redundanz erreicht, d. h., gleichlautende Informationen werden mehrfach gehalten. Es gibt verschiedene Formen von Redundanz: Hardware kann redundant vorhanden sein, so daß beispielsweise Netzteile, Festplatten oder andere physikalische Geräte mehrfach vorhanden sind. Sehr wichtig in diesem Zusammenhang ist auch die Redundanz der Übertragungskanäle, d. h., das Netzwerk und die Verkabelung werden mehrfach ausgeführt, so daß der Ausfall eines kompletten Kabelstranges toleriert werden kann.

Eine weitere Form von Redundanz ist die mehrfache Speicherung oder Übertragung von Informationen, bzw., das Versehen der Daten mit zusätzlichen Informationen, die eine Erkennung oder sogar Behebung von Verfälschungen erlauben. Beispielsweise kann ein Wert mehrfach hintereinander über ein Netzwerk übertragen werden, so daß ein Vergleich der mehrfach erhaltenen Werte möglich ist. Aber auch Prüfsummenverfahren stellen Redundanzverfahren dar: In die Prüfsumme werden Informationen der Daten eingerechnet; die Prüfsumme erhöht den Informationsgehalt der Daten nicht, sondern dient lediglich der Erhöhung der Zuverlässigkeit. Auch ein verteiltes System stellt eine Art von Redundanz dar: Es wird die Hardware mehrfach ausgeführt, und auf den einzelnen Stationen laufen Dienste, die sich gegenseitig ersetzen können, damit die Verläßlichkeit erhöht werden kann

Man unterscheidet bei Redundanzverfahren zwischen statischer und dynamischer Redundanz. Bei statischer Redundanz wird die Redundanz in einem vorher bestimmten Umfang eingebaut. Ein Prüfsummenverfahren ist ein Beispiel hierfür: So wird bei einem 16-Bit-CRC-Prüfsummenverfahren auf einem Netzwerk zu jeder übertragenen Nachricht genau 16 Bit zusätzlicher Information angehängt. Erkennt nun ein Empfänger

(an der Prüfsumme), daß die Übertragung nicht erfolgreich war, und fordert die Nachricht erneut an, dann wird als Folge des statischen Verfahrens ein dynamisches Redundanzverfahren eingesetzt, d. h., die Redundanz, hier ein erneutes Versenden der Informationen, wird nur bei Bedarf eingesetzt.

Der Vorteil statischer Redundanz ist, daß man genau vorbestimmen kann, wie groß der Aufwand ist. Andererseits droht bei statischer Redundanz eine Verschwendung von Ressourcen, da die zusätzlichen Informationen immer übertragen werden müssen. Diesen Nachteil haben Verfahren mit dynamischer Redundanz nicht, da nur bei Bedarf Redundanz ergänzt wird. Andererseits sind dynamische Verfahren schwerer vorauszusehen; so könnte eine wegen eines Fehlers erneut übertragene Nachricht wieder fehlerhaft sein, wodurch wieder ein neue Übertragung stattfindet, die wieder fehlerhaft sein könnte. Somit kann die für die Übertragung notwendige Zeit nicht abgeschätzt werden.

Fehlertoleranzverfahren haben eine wichtige Eigenschaft: Fehler können nur in begrenztem Umfang toleriert werden. Sobald etwa jede Nachricht, die über einen Kommunikationskanal übertragen wird, zerstört wird, kann auch mittels Redundanz auf diesem Kanal nichts mehr übertragen werden, unabhängig davon, wie gut das System entworfen wurde. Daher muß man Annahmen treffen über die möglichen auftretenden Fehler; man stellt ein *Fehlermodell* auf. Aufbauend auf diesem Fehlermodell wird das System entworfen, wobei garantiert werden muß, daß bei Einhaltung der getroffenen Annahmen die auftretenden Fehler toleriert werden können.

Das gewählte Fehlermodell muß realistisch sein. Ein Fehlermodell, bei dem die Annahme getroffen wird, daß gar keine Fehler auftreten, ist erleichtert zwar den Entwurf eines System, hat aber zumeist mit der Realität nicht viel zu tun. Daher untersucht man die Wahrscheinlichkeit, mit der die Annahmen des Fehlermodells mit der Realität übereinstimmen. Diese Wahrscheinlichkeit, die „(Annahmen-) Überdeckung“ (*assumption coverage*) genannt wird, muß sehr hoch sein, damit von einem adäquaten Fehlermodell gesprochen werden kann.

2.3 Drahtlose Netzwerke

Die meisten zur Zeit existierenden Netzwerke sind drahtgebunden aufgebaut. Die Kabel, mit denen sie arbeiten, bilden ein geschütztes Medium, welches technisch leicht zu beherrschen ist und damit eine relativ gesicherte Übertragung ermöglicht. Die Datenübertragungsraten auf drahtgebundenen Netzwerken liegen relativ hoch, angefangen bei dem kleinen Wert 4 Mbit/s (Token-Ring, ARCNet) über 10 Mbit/s und 100 Mbit/s bis hinauf zu 1 GB/s (Ethernet, Fast- bzw. Gigabit-Ethernet). Mehrere Kabel lassen sich parallel verlegen, ohne daß diese großen Einfluß aufeinander haben, wodurch die Verbindungen je nach den Anforderungen frei gestaltet werden und die Übertragungsraten entsprechend erhöht werden können. Weiterhin kann ein Kabel nur dann angezapft werden, wenn man sich in Nähe des Kabels befindet.

In letzter Zeit ergeben sich weitere Anforderungen, die die Notwendigkeit beinhalten, eine drahtlose Kommunikation zur Verfügung zu stellen. Häufig ist es nur schwer möglich, Kabel zur Kommunikation zu verlegen: Teils, weil Kabelschächte schon voll sind, teils, weil Gebäude nicht verändert werden sollen oder dürfen – man denke an denkmalgeschützte Gebäude – oder, weil die Anwendung dies nicht erlaubt. Moderne Laptops sind heutzutage in der Lage, mehrere Stunden autonom zu arbeiten, d. h., es werden keine Kabel beispielsweise zur Stromversorgung benötigt. Diese Laptops benötigen allerdings zumeist eine Verbindung mit einem Netzwerk. Würden nun drahtgebundene Netzwerke hierfür benutzt, so würde man das Stromversorgungskabel, welches man eingespart hat, durch ein Netzwerkkabel ersetzen.

Es gibt weitere Anwendungen, in denen ein Kabel völlig ungeeignet ist. Während in dem gerade besprochenen Szenario der Laptops von portablen Computern ausgegangen wurde, d. h., die Computer sind ortsveränderlich, allerdings wechseln sie ihren Standort nicht während sie arbeiten, gibt es noch mobile Anwendungen, d. h., die Computer verändern ihren Standort während der Arbeit. Ein häufiges Beispiel hierfür stellen mobile Roboter dar, welche zur Erfüllung ihrer Aufgaben ständig ihren Aufenthaltsort wechseln müssen, beispielsweise in Fertigungs- oder Lagerhallen. Hier wäre ein drahtgebundenes Netzwerk problematisch, da die Kabel die Bewegungsfreiheit der einzelnen Roboter stark einschränken würden. Dies gilt insbesondere für das oben beschrieben DECOR-Anwendungsszenario, bei dem autonome Autos selbständige Entscheidungen treffen sollen.

Daher benutzt man für solche Anwendungen drahtlose Netzwerke, die optisch oder per Funk die Informationen übertragen. Hierdurch vermeidet man die oben angesprochenen Probleme der drahtgebundenen Netzwerke. Allerdings sind drahtlose Netzwerke noch relativ neu und unterliegen damit einer starken aktiven Forschung.

Es gibt verschiedene Verfahren, um drahtlos zu kommunizieren. Man unterscheidet zwischen optischen Netzwerken und Funknetzwerken. Erstere, die zumeist mit infrarotem Licht arbeiten, sind auf Verbindungen beschränkt, bei denen der Sender und der Empfänger in unmittelbarer Nähe zueinander stehen und Sichtkontakt miteinander haben. Bei der Übertragung mittels Funk können hingegen größere Strecken überwunden werden.

Die folgenden Ausführungen beschränken sich auf drahtlose lokale Netzwerke (*local area network*, LAN), d. h. Netzwerke, die mehrere Computer innerhalb einer beschränkten Fläche miteinander verbinden sollen. Nicht betrachtet werden hingegen drahtlose Netzwerke wie Richtfunkstrecken, die zumeist als Punkt-zu-Punkt-Verbindungen angelegt sind und dem Zwecke dienen, mehrere LANs miteinander zu verbinden.

Bei den drahtlosen LANs (WLAN für *wireless* LAN) gibt es mehrere verschiedene Ansätze. Es gibt hier proprietäre, d. h. firmeneigene Lösungen, wie beispielsweise das Produkt OpenAIR, eine Entwicklung der Firma Proxim, die vor der Standardisierung die einzige WLAN-Technik war, die von mehr als einer Firma unterstützt wurde. Inzwischen gibt es einige Standards, wobei primär HIPERLAN (High Performance Radio

LAN, [ET 96]) der ETSI („European Telecommunications Standards Institute“) und der Standard IEEE 802.11 ([IEEE 97]) des IEEE („Institute of Electrical and Electronics Engineers“), sowie dessen Erweiterungen 802.11a, 802.11b zu nennen sind.

Wählt man als Grundlage für eine Entwicklung ein prioprietäres System, dann ist man auf die liefernde Firma festgelegt. Sollte das Unternehmen sich entscheiden, das Produkt nicht mehr herzustellen, dann sind bei notwendigen Erweiterungen eigene Investitionen in die Technik zumeist verloren, da keine Unterstützung mehr zu erhalten ist. Bei Standards, die von mehreren Firmen unterstützt werden, besitzt man im Gegensatz hierzu eine höhere Investitionssicherheit. Daher wurde als Grundlage für die vorliegende Arbeit ein Standard gewählt. Obwohl er der ältere der erwähnten Standards ist, besitzt HIPERLAN zur Zeit keine große Unterstützung, erste Geräte kommen zur Zeit auf den Markt. Im Gegensatz hierzu existieren Geräte nach dem Standard IEEE 802.11 schon seit mehreren Jahren. Viele Firmen unterstützten ihn schon vor seiner Verabschiedung. Daher wird er, trotz seiner geringeren Datenraten im Vergleich zu HIPERLAN, als Grundlage der vorliegenden Arbeit benutzt. Erweiterungen wie IEEE 802.11a und IEEE 802.11b, die sich zur Zeit noch im Entwurfsstadium befinden, sorgen für eine Erhöhung der Datenrate des IEEE-Standards auf 11 Mbit/s (802.11b) bzw. bis zu 54 Mbit/s (802.11a) bei weitgehender Kompatibilität zum ursprünglichen Standard, so daß dieser Nachteil des IEEE-Standards in kurzer Zeit nicht mehr vorhanden sein dürfte. Erste Produkte, die diese Erweiterungen implementieren, befinden sich schon auf dem Markt.

Daher beschäftigt sich diese Diplomarbeit im folgenden nur noch mit dem Standard 802.11, obwohl viele der Gedanken auf beliebige Funknetzwerke übertragbar sind.

2.3.1 IEEE 802.11

Der Standard „IEEE 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications“ wurde 1997 von der IEEE verabschiedet und 1999 als nationaler ANSI/IEEE und internationaler ISO/IEC-Standard übernommen ([IEEE 97]). Der Standard legt nach dem ISO/OSI-Referenzmodell die Schichten 1 (Physical layer, PHY) und einen Teil von Schicht 2 (medium access control sublayer, MAC) fest.

Der PHY-Layer legt fest, wie Daten physikalisch über das Medium „Luft“ übertragen werden. Dabei wird hauptsächlich spezifiziert, wie die grundlegenden Dateneinheiten des Funknetzwerks, die sogenannten Frames, von einer Station auf eine andere übertragen werden. Oberhalb des PHY-Layers befindet sich der MAC-Sublayer, der hauptsächlich den Medienzugriff festlegt. In einem Netzwerk mit mehrfachem Zugriff (*multiple access network*), d. h. einem Netzwerk, bei dem mehrere Stationen miteinander verbunden werden, die alle auf dem gleichen Medium Senden und Empfangen können, muß festgelegt werden, wann jede einzelne Stationen auf das Netzwerk sendend zugreifen darf, damit die Nachrichtenübertragung gewährleistet werden kann. Der MAC-Layer hat u. a. die Aufgabe, die Zeitpunkte für den Medienzugriff festzulegen.

Im Standard 802.11 werden drei verschiedene Varianten für den PHY-Layer dargestellt, eine Variante für die Kommunikation über Infrarot, und zwei Varianten für die Funkübertragung. Auf jedem dieser drei PHY-Layer baut der MAC-Layer auf, welcher – bis auf geringe, hauptsächlich den zeitlichen Ablauf betreffende Ausnahmen – für alle drei PHY-Layer identisch ist.

Eine Übertragung mittels Infrarot kommt in dem Kontext dieser Diplomarbeit nicht in Frage, da diese Übertragungsweise zu unzuverlässig ist. Die anderen beiden PHY-Layer zur Funkübertragung sind hingegen beide geeignet. Da ihre Details nichts wesentliches zum Verständnis der vorliegenden Arbeit beitragen, werden diese allerdings nicht näher betrachtet, stattdessen wird das Hauptaugenmerk auf den MAC-Layer des Standards 802.11 gelegt, der im übrigen bei den Erweiterungen 802.11a und 802.11b im jetzigen Entwicklungsstand nahezu unverändert bleibt. Damit bleiben die hier gemachten Betrachtungen auch für die Erweiterungen gültig.

2.3.1.1 Ad-Hoc- und Infrastrukturnetzwerke

Bei einem Funknetzwerk nach dem IEEE-Standard unterscheidet man grundsätzlich zwei verschiedene Netzwerktypen, sogenannte Ad-Hoc-Netzwerke und Infrastrukturnetzwerke.

Zum Verständnis der beiden Begriffe soll zuerst der Begriff des BSS (*basic service set*) erklärt werden. Ein BSS besteht aus allen Stationen, die sich im gegenseitigen Sende- und Empfangsbereich (im folgenden nur noch kurz: Empfangsbereich) voneinander befinden. Ein BSS entspricht damit dem Begriff der Zelle (*cell*), wie er häufig bei anderen Funknetzwerken (beispielsweise Mobilfunk) genutzt wird. Die Abbildung 2-1 zeigt zwei BSS, die aus jeweils 2 Stationen (STA) bestehen.

Ein Ad-Hoc-Netzwerk besteht aus einem BSS. Im allgemeinen erfolgt die Bildung und die Auflösung eines Ad-Hoc-Netzwerkes spontan, je nach der Empfangslage. Stationen, die sich nicht im gegenseitigen Empfangsbereich befinden, können sich nicht erreichen, daher spricht man von einem unabhängigen BSS (IBSS, *independant* BSS). Die zwei BSS der Abbildung 2-1 bilden damit zwei Ad-Hoc-Netzwerke. In einem Ad-Hoc-Netzwerk sind alle Stationen gleichberechtigt.

Im Gegensatz hierzu existiert in einem Infrastrukturnetzwerk pro BSS eine ausgezeichnete Station, der sogenannte *access point* (AP). Der AP ist Teil des BSS und dient als Schnittstelle zwischen diesem BSS und einem drahtgebundenen Netzwerk oder zwischen mehreren APs, die ihrerseits ihre BSS verwalten (s. Abbildung 2-2). Die APs ermöglichen die Kommunikation der Stationen in den BSS untereinander. Um dies zu ermöglichen, sind sie an ein Verteilungssystem (*distribution system*, DS) angeschlossen. Das DS ist ein Netzwerk welches die APs untereinander sowie optional über sogenannte Portale mit einem drahtgebundenen Netzwerk verbinden. Die APs sorgen dafür, daß die Verbindung ihrer verwalteten BSS imtereinander sowie mit dem drahtgebundenen Netzwerk transparent ist, d. h., daß die Stationen keine Kenntnis darüber haben müssen,

wo sich eine Station, mit der sie kommunizieren wollen, befindet. Der AP puffert auch Nachrichten zwischen, die kurzzeitig nicht ausgeliefert werden können. Hieraus ergibt sich, daß dem AP in einem Infrastrukturnetzwerk eine besondere Stellung zukommt, und daß der AP besonders zuverlässig sein muß, um seine Funktion zu erfüllen.

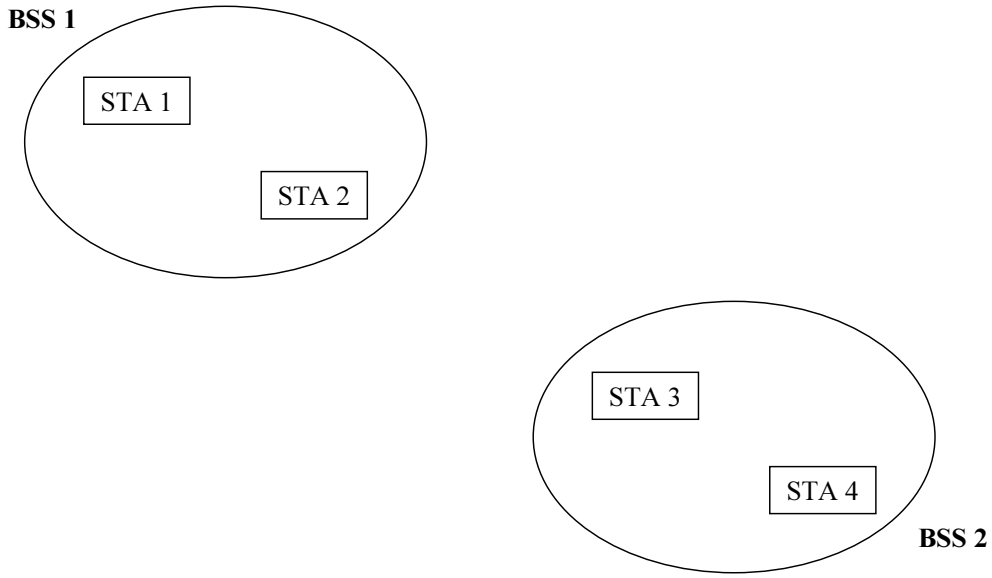


Abbildung 2-1: *Zwei Ad-Hoc-Netzwerke*

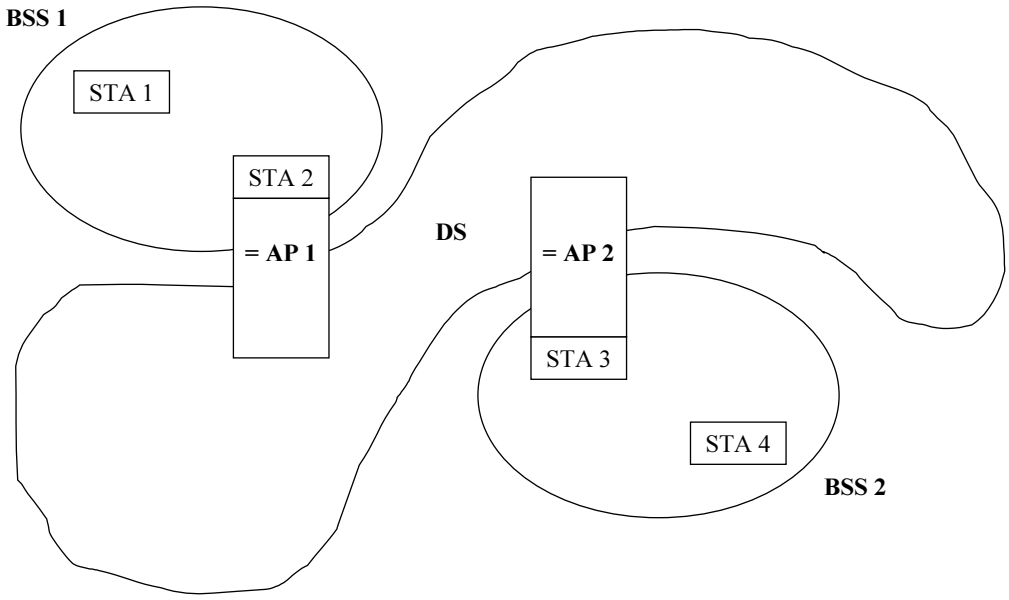


Abbildung 2-2: *Ein Infrastrukturnetzwerk*

Bis hierhin wurde davon ausgegangen, daß der Empfangsbereich klar definiert ist. Dies ist in einem drahtlosen Netzwerk aber nicht möglich, da die Übertragungscharak-

teristiken des Mediums dynamisch und unvorhersehbar sind. Schon kleine Änderungen des Ortes oder der Ausrichtung von Stationen können große Unterschiede der Signalstärke bewirken, ebenso bewirkt die mögliche Bewegung der Stationen, aber auch von Objekten im Emp fangsbereich große Veränderungen der Empfangsqualität. Die Kommunikation kann gar zu einer „Einbahnstraße“ werden, wenn eine Station 1 eine Station 2 empfangen kann, umgekehrt jedoch nicht.

Tatsächlich gibt es in einem BSS gar keinen einheitlichen Empfangsbereich, vielmehr hat jede einzelne Station einen eigenen Empfangsbereich. Wenn sich diese für zwei Stationen überlappen, dann kann Kommunikation zwischen diesen stattfinden, und die betreffenden Stationen befinden sich in einem gemeinsamen BSS.

In einem Infrastrukturnetzwerk hingegen ist weitgehend bekannt, welcher Empfangsbereich von einem AP aus erreicht werden kann. Weiterhin ist im Standard geregelt, daß sich in einem Infrastrukturnetzwerk Stationen beim AP anmelden müssen. Hierdurch ergibt sich ein natürlicher Begriff der Gruppe, welcher aus Stationen besteht, die sich im Empfangsbereich eines APs befinden und bei diesem angemeldet sind. Dieser Gruppenbegriff wird in dieser Arbeit als Gruppenbegriff für die Synchronisation übernommen.

2.3.1.2 Medienzugriff

Beim Funknetzwerk nach dem IEEE-Standard wird der Medienzugriff nach dem CSMA/CA-Verfahren (*carrier sense multiple access* / *collision avoidance*) geregelt, während bei drahtgebundenen Netzwerken häufig wie bei Ethernet CSMA/CD (CD bedeutet *collision detection*) benutzt wird.

Die Begriffe CSMA/CA und CSMA/CD setzen sich aus drei Komponenten zusammen: *Multiple Access* (MA) bedeutet, daß es sich um ein Netzwerk mit mehrfachem Zugriff handelt, daß also mehrere Stationen auf das Medium schreibend zugreifen können. Daher wird *carrier sensing* (CS) eingesetzt: Jede Netzwerkkarte, die etwas senden will, testet das Medium darauf, ob es zur Zeit frei ist, d. h., ob zur Zeit eine Übertragung stattfindet, Man sagt dann, daß ein Trägersignal (*carrier*, kurz: Träger) vorliegt. Ist das Medium nicht frei, also belegt, so wartet die sendebereite Station, bis das Medium wieder frei ist. Das Vorgehen bei freiem Medium wird durch die dritte Komponente, CD oder CA beschrieben.

Bei CSMA/CD wird bei freiem Medium folgendermaßen vorgegangen: Die sendebereite Station beginnt einen Sendeversuch. Alle Stationen, die später senden wollen, erkennen den Träger und warten bis zu Beendigung des Sendevorganges. Es besteht allerdings die Möglichkeit, daß zwei Stationen gleichzeitig das freie Medium erkennen und in der Folge gleichzeitig mit einem Sendeversuch beginnen. In diesem Fall kommt es zu einer *Kollision*, dabei gehen die übertragenen Informationen verloren. Diese Kollision wird von den sendenden Stationen erkannt, worauf diese das Senden abbrechen und

einen sogenannten JAM-Code versenden[3]. Nach einer Wartezeit beginnen die Stationen erneut mit einem Sendeversuch, wobei vorher wieder das Medium getestet wird. Diese Wartezeit hat eine zufällige Länge; mit ihrer Hilfe wird probabilistisch garantiert, daß die Stationen nicht gleichzeitig versuchen, erneut auf das Medium zuzugreifen.

Im Gegensatz zu drahtgebundenen Netzwerken wie Ethernet ist es beim Funknetzwerk technisch nicht möglich, Kollisionen mit Sicherheit festzustellen. Dies liegt daran, daß eine sendende Station in ihrer nahen Umgebung alle anderen sendenden Stationen überdeckt.

Daher versucht das Zugriffsverfahren beim Funknetzwerk, Kollisionen von vorneherein zu vermeiden. Hierzu kennt der IEEE-Standard zwei Zugriffsverfahren, die obligatorische DCF (*distributed coordination function*) und die optionale PCF (*point coordination function*). Die PCF ist ein Zugriffsverfahren, welches nur im Infrastrukturnetzwerk möglich ist und vom AP gesteuert wird. Es dient dazu, Echtzeitdaten mit vorhersehbaren Latenzen zu versenden. Da aber noch kein Hersteller von Netzwerkkarten die PCF implementiert hat oder dies zumindest angekündigt hat, wird diese hier nicht näher betrachtet. Stattdessen wird das Zugriffsverfahren DCF dargestellt.

Zur Vermeidung von Kollisionen werden beim Funknetzwerk zwei Mechanismen eingesetzt: Ein virtuelles Trägersignal, welches mittels eines NAV (*network allocation vector*) implementiert wird, und zufällige Verzögerungen vor dem Versenden von Nachrichten.

Das virtuelle Trägersignal dient dazu, den Zugriff auf das Medium durch sendebereite Stationen stärker zu beschränken, als es durch den physikalischen Träger erfolgen würde. Demnach testet eine Station, welche ermitteln will, ob das Medium frei ist, sowohl den physikalischen als auch der virtuellen Träger. Ist mindestens einer von beiden vorhanden, dann wird das Medium als belegt angesehen.

Der NAV wird dabei folgendermaßen verwaltet: Jede über das Netzwerk gesendete Information, die Frames, enthalten im Kopf ein Feld namens „Dauer“ (*duration*). Dieses Feld gibt an, wie lange die aktuelle Übertragung dauern wird. Jede Station, die das Frame empfängt, weiß also, wie lange das Medium belegt ist, daher wird für diese Zeitdauer der virtuelle Träger von jeder empfangenden Station aktiviert. Das Dauer-Feld wird hierbei von der sendenden Station so groß gewählt, daß auch eventuelle zu erwartende Antworten auf das Frame innerhalb der Dauer berücksichtigt sind.

Der andere Mechanismus zur Vermeidung von Kollisionen sind zufällige Verzögerungen vor dem Versenden von Nachrichten. Jede sendebereite Station, die feststellt, daß das Medium belegt ist, wartet nach dem Freiwerden des Mediums eine zufällige Zeit, bevor der Sendeversuch begonnen wird (s. „backoff-Fenster“ in Abbildung 2-3).

---

[3] Dieser ist notwendig, damit garantiert werden kann, daß wirklich alle kollidierenden Stationen die Kollision bemerken. Die Details sollen hier aber nicht interessieren, da sie unwesentlich für die Arbeit sind.

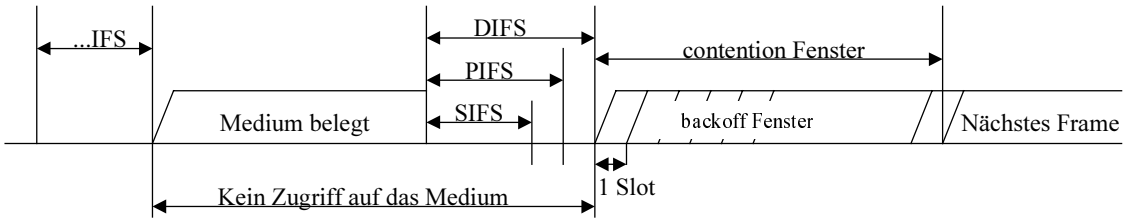


Abbildung 2-3: *Medienzugriff*

Dies hilft, die Auswirkungen des folgenden Problems zu vermindern: Während der relativ langen Zeitspanne, in der das Netzwerk durch die Übertragung eines Frames belegt ist, kann mehr als eine Station sendebereit werden. Würden die Stationen unmittelbar nach Freiwerden des Mediums auf dieses zugreifen, so würde dieser Zugriff gleichzeitig erfolgen und die Frames würden daher kollidieren. Da diese Kollision nicht mit Sicherheit entdeckt werden kann, würde dadurch Netzwerkbandbreite verschwendet.

Durch die Benutzung einer zufälligen Wartezeit wird probabilistisch erreicht, daß die Sendeversuche verteilt werden und daher nicht gleichzeitig stattfinden, so daß die Anzahl von Kollisionen von Frames mit einer hohen Wahrscheinlichkeit gering gehalten wird.

Da Kollisionen auf dem Funknetzwerk nicht völlig vermieden und auch nicht sicher entdeckt werden können, wird auf dem Funknetzwerk ein Mechanismus benutzt, der durch positive Rückmeldungen den erfolgreichen Empfang von Nachrichten bestätigt. Hierzu wird nach dem erfolgreichen Empfang einer Nachricht vom Empfänger ein besonderes Frame, das sogenannte ACK-Frame (Acknowledgement) an den ursprünglichen Sender gesendet. Hierdurch weiß der ursprüngliche Sender, daß das Frame erfolgreich angekommen ist. Bleibt dieses Frame hingegen aus, so weiß er, daß das Frame irgendwo verlorengegangen ist, und versendet nach einer Wartezeit das Frame erneut. Dieses ACK-Frame wird allerdings nur dann gesendet, wenn eine Station an *eine* andere sendet. Bei sogenannten Broad- oder Multicasts, d. h., wenn ein Frame mehrere Empfänger hat, bleibt dieses ACK-Frame aus. Weiterhin gibt es beim Funknetzwerk sogenannte Management-Frames zur Verwaltung des Netzwerks, die auch nicht bestätigt werden.

Um eine Prioritätsregelung für wichtigere Frames zu ermöglichen, wird vor der zufälligen Wartezeit innerhalb der DCF noch eine zusätzliche Verzögerung, der *inter-frame space* (IFS), eingefügt. Die IFS sind genau definierte Wartezeiten, die bei wichtigeren Frames kürzer sind als bei unwichtigeren. Hierdurch können wichtigere Frames früher auf das Medium ausgegeben werden, so daß andere Stationen mit unwichtigeren Frames nicht auf das Medium zugreifen können. Es gibt vier verschiedene IFS, nach aufsteigender Dauer sortiert sind dies PIFS (PCF-IFS), SIFS (*short* IFS), DIFS (DCF-IFS) und EIFS (*extended* IFS), wobei hier nur der SIFS und der DIFS von Interesse sind.

Innerhalb der DCF werden die Frames auf dem Funknetzwerk nach einer Wartezeit der Länge DIFS versendet. Die einzige Ausnahme von dieser Regel sind Frames, die eine

direkte Antwort auf das vorherige Frame darstellen, wie das ACK-Frame. Dies ist notwendig, damit kein anderes Frame vor dem ACK-Frame gesendet werden kann, wodurch der Sender davon ausgehen würde, daß das Datenframe verlorengegangen ist.

Beim Funknetzwerk nach dem IEEE-Standard existieren einige sogenannte Management-Frames, die der internen Verwaltung des Netzwerkes dienen, beispielsweise der An- und Abmeldung von Stationen an den AP (genauer: Authentifizierung und Assoziierung mit dem AP). Von diesen Management-Frames ist im Kontext dieser Diplomarbeit das Beacon-Frame von besonderem Interesse, daher soll es näher betrachtet werden.

2.3.1.3 Das Beacon-Frame

Das Beacon-Frame (kurz: der Beacon) ist ein Management-Frame, welches zur Koordination eines BSS benötigt wird. In einem Infrastrukturnetzwerk wird dieses Frame vom AP generiert und versendet.

Der Beacon wird in regelmäßigen Abständen versendet. Die Frequenz ist ein Parameter des IEEE-Standards, sie wird im Beacon selbst den Stationen mitgeteilt. Der Beacon enthält Informationen über den verwendeten Physical Layer, eine Identifikation für das BSS, ein Feld, welches die implementierten Optionen des APs kodiert, und einen Zeitstempel. Dieser Zeitstempel wird für ein Uhrensynchronisationsprotokoll benötigt, welches im Standard vorhanden ist und in Abschnitt 4.2 „Das IEEE-Protokoll“ dargestellt wird.

Der Beacon wird mittels eines kürzeren IFS versendet, womit er Priorität auf dem Medium erhält. Daher kann weitgehend garantiert werden, daß das Beacon versendet werden kann und nicht durch andere sendebereite Stationen dauernd blockiert wird. Dies ist wichtig für die Uhrensynchronisation, da eine begrenzte Zeit zwischen dem Versenden von Synchronisationsnachrichten benötigt wird. Auf deisem Grund wird beim IEEE-Uhrensynchronisationsprotokoll die Zeitinformation im Beacon versendet.

2.3.2 Vergleich mit drahtgebundenen Netzwerken

Drahtlose Netzwerke unterscheiden sich von drahtgebundenen in einigen Eigenschaften. Daher sollen diese beiden Formen von Netzwerken in diesem Kapitel gegeneinandergestellt werden.

Ein Funknetzwerk besitzt folgende Unterschiede zu drahtgebundenen:

1. Das Medium hat keine absolute oder klar beobachtbare Grenze, außerhalb von der Stationen mit geeigneten Empfängern keine Nachrichten empfangen können.

2. Das Medium ist ungeschützt gegenüber Einflüssen von außen.

3. Das Medium, über das die Kommunikation abläuft, ist wesentlich unzuverlässiger als bei drahtgebundenen Netzwerken.

4. Es gibt dynamische Topologien, weil die Stationen jederzeit ihren Aufenthaltsort wechseln können.

5. Das Medium ist nicht vollständig verbunden, d. h. es kann nicht garantiert werden, daß jede Station jede andere empfangen kann.

6. Das Medium hat zeitabhängige und asymmetrische Ausbreitungseigenschaften.

Hierdurch ergeben sich Implikationen, die für das zu entwickelnde Uhrensynchronisationsprotokoll wichtig sind. Das Fehlen der klar beobachtbaren Grenzen, die dynamischen Topologien und die fehlende vollständige Erreichbarkeit (*connectivity*) des Netzwerkes insbesondere in Verbindung mit den dynamischen Topologien bedeutet, daß die BSS nicht eindeutig auf eine physikalische Fläche festzulegen sind, sich überschneiden und sich jederzeit ändern können. Beim Infrastrukturnetzwerk gibt es eine exakt definierte Regelung, welche Stationen zu einem BSS gehören: Es handelt sich um alle Stationen, die beim AP angemeldet sind. Hierzu müssen sie sich im Empfangsbereich des APs befinden. Hieraus ergibt sich ein natürlicher Begriff der Gruppe. Beim Ad-Hoc-Netzwerk ist eine ähnliche Regelung nicht klar zu treffen, die alle Stationen gleichberechtigt sind. Ein Gruppenbegriff ergibt sich deshalb hier nicht in einer kanonischen Form. Zur Minimierung der Auswirkung dieses Unterschieds in der vorliegenden Arbeit wird die Implementierung des Uhrensynchronisationsprotokolls in einem Infrastrukturnetzwerk betrachtet. Eine Erweiterung auf Ad-Hoc-Netzwerke kann von einer Folgearbeit betrachtet werden. Auch die zeitabhängigen und asymmetrischen Ausbreitungseigenschaften können durch die Verwendung eines Infrastrukturnetzwerks minimiert werden, in dem nur ein einziger Sender für Zeitinformationen, der AP, benutzt wird, wodurch diese Unterschiede alle beteiligten Stationen gleichermaßen betreffen.

3 Uhrensynchronisation und verwandte Arbeiten

Dieses Kapitel befaßt sich mit der Uhrensynchronisation. Zuerst soll geklärt werden, wozu eine Uhrzeit in verteilten Echtzeitsystemen benötigt wird, und wieso einzelne Uhren synchronisiert werden müssen.

In einem Computersystem benötigt man häufig die Information, in welcher Reihenfolge Ereignisse aufgetreten sind, d. h., welches Ereignis vor welchen anderen stattgefunden hat. Dies kann beispielsweise in einer Datenbank notwendig sein, wenn von zwei sich gegenseitig widersprechenden schreibenden Zugriffen nur der erste ausgeführt werden soll. Außerdem kann diese Information zur Bestimmung benötigt werden, welches Ereignis in kausaler Abhängigkeit von welchem anderen steht, denn ein frühes Ereignis kann nicht von einem erst später auftretenden kausal beeinflußt worden sein. Diese (potentiellen) kausalen Abhängigkeiten benötigt man wiederum, wenn man beispielsweise in einem Fehlerfall die Ursache bestimmen will, da in diesem Fall häufig viele verschiedene Fehlerereignisse auftreten, die voneinander abhängig sind. Die Kenntnis des „ersten“ Ereignisses erleichtert die Diagnose. Diese Notwendigkeit der Kenntnis der kausalen Abhängigkeiten gilt insbesondere in Echtzeitsystemen, in denen meist ein Fehler viele Folgefehler nach sich ziehen kann. Das Echtzeitsystem benötigt zumeist die Kenntnis des auslösenden Ereignisses, um geeignet reagieren zu können. Bei den externen Ereignissen, die von den Sensoren beobachtet wurden, ist die Zeit zumeist der einzige Anhaltspunkt, mit dem diese Abhängigkeiten erkannt werden können. Außerdem benötigt man in Echtzeitsystemen häufig die Messung von Zeitintervallen, oder Aktoren müssen zeitlich koordiniert Aufgaben erledigen. Weitere Nutzungsmöglichkeiten von synchronisierten Uhren finden sich in [Li 91].

In einem Ein-Computer-System kann man die Uhrzeit benutzen, um eine Vorher- / Nachher-Beziehung feststellen zu können: Jedes auftretende relevante Ereignis wird mit einem sogenannten „Zeitstempel“ versehen, d. h. der Prozessor liest die Zeit aus, die er zum Zeitpunkt hat, an dem er von dem Ereignis Kenntnis erhält, und assoziiert diese Uhrzeit mit dem Ereignis. Wenn die Uhrzeit mit jedem Ereignis monoton steigt, dann kann der Computer durch Vergleich der Zeitstempel die temporalen Abhängigkeiten bestimmen, denn ein Ereignis mit einem kleineren Zeitstempel ist vor einem Ereignis mit einem größeren aufgetreten.

Man beachte, daß man in diesem Fall außer der strengen Monotonie der Uhrzeit keine weiteren Eigenschaften benötigt. Als „Zeit“ würde hier also ausreichen, daß man einen Zähler verwaltet, dessen Wert jeweils die Uhrzeit darstellt, und der mit jedem auftretenden Ereignis um eins erhöht wird (siehe hierzu auch [La 78]). Hier ist lediglich die eindeutige Zuordnung von Zeitstempeln wesentlich.

In einem verteilten System ist die Erkennung der temporalen Abhängigkeiten schwerer. Man verwaltet einen Zähler auf einer Station, der von den anderen Stationen mittels Nachrichtenaustausches gelesen werden kann, oder man verwaltet auf mehreren Stationen (im Extremfall allen) jeweils einen Zähler und sucht Mechanismen, um die Zähler auf einem gemeinsamen Stand zu halten. In jedem Fall ist der globale Zähler nicht so

einfach zu handhaben wie beim Ein-Computer-System, da lediglich mittels des langsamen Austausches von Nachrichten kommuniziert werden kann.

Der Ansatz mit einem einzelnen zentralen Zähler hat zwei wesentliche Nachteile. Jedes Ereignis, für welches ein Zeitstempel benötigt wird, muß den zentralen Zähler abfragen, daher werden die Kommunikationskanäle sowie die Station mit dem zentralen Zähler belastet. Weiterhin ist das Verfahren ungenau, da die Ereignisse in schnellerer Abfolge eintreten können, als die Zeitstempel ermittelt werden. Ein verteiltes System wird aber gerade mit dem Ziel entwickelt, Informationen möglichst lokal zu verarbeiten und damit die einzelnen Stationen und die Kommunikationskanäle zu entlasten.

Bei der Lösung mit verteilten Zählern muß geklärt werden, wie diese Zähler synchron gehalten werden können. Hierzu bedarf es eines Protokolls, welches durch Austausch von Nachrichten die Zähler synchron halten soll.

In [La 78] wird ein Verfahren vorgeschlagen, mit dessen Hilfe ohne zusätzlichen Kommunikationsaufwand kausale Abhängigkeiten bestimmt werden können. Hierzu wird von einer Station, die eine Nachricht versenden will, mit jeder versendeten Nachricht der aktuelle Stand des lokalen Zählers mitgesendet. Eine empfangende Station setzt beim Empfang einer Nachricht ihren eigenen lokalen Zähler auf den Wert des empfangenen Zählers, sofern der lokale Zähler einen kleineren Wert besitzt. Dieses Verfahren beruht auf der Beobachtung, daß sich interne kausale Abhängigkeiten nur durch den Austausch von Nachrichten ergeben können. Genau dies ist aber der Nachteil des Verfahrens: Es können lediglich *interne* kausale Abhängigkeiten bestimmt werden, d. h. kausale Zusammenhänge, die innerhalb des Computersystems entstehen. Dies liegt daran, daß die Synchronisierung nur mittels der in dem System versendeten Nachrichten erfolgt. Kommunizieren zwei Stationen für eine längere Zeit nicht miteinander, so können sie ihre Uhren auch nicht synchronisieren. Dies ist unproblematisch, da Ereignisse in dem Fall auch nicht intern direkt voneinander abhängen können. In Echtzeitsystemen existieren allerdings auch *externe* Abhängigkeiten, d. h. kausale Abhängigkeiten, die durch die Umgebung des Computersystems bewirkt werden. Beispielsweise bewirkt das Öffnen eines Zulaufventils zu einem Wasserbec??ken, daß der Wasserstand in dem Becken erhöht wird. Die Aktor „Ventil“ ist externe kausale Ursache dafür, daß der Sensor „Wasserstandsanzeige“ einen höheren Wasserstand erkennt. Ohne zusätzliche Kommunikation lassen sich solche Abhängigkeiten mit dem Verfahren nach [La 78] nicht auflösen.

Um auch solche externen Abhängigkeiten erkennen zu können benutzt man eine Uhr, welche selbständig in regelmäßigen Zeitintervallen weitergestellt wird, d. h., deren Zählerwert regelmäßig um eins erhöht wird[4]. Die Uhr besteht demnach aus einem Zähler und einem Zeitgeber. Jede Station besitzt eine eigene Uhr, deren Wert sie jederzeit abfragen kann. Wenn die einzelnen Uhren jeweils den gleichen Wert anzeigen und die Intervalle, nach denen sie weitergestellt werden, gleich groß sind, dann können die Zeit-

[4] wie es auch dem intuitiven Begriff der Uhr entspricht.

stempel zur Bestimmung der temporalen Abhängigkeiten von Ereignissen benutzt werden; dies wird in Kapitel 3.5 „Grenzen für die Genauigkeit der Uhrensynchronisation“ genauer erläutert. Diese Uhr ermöglicht erst die Messung von Zeitintervallen und die zeitliche Koordination von Aktionen, die für Echtzeitsysteme wichtig sind.

Würde man die Zeitgeber der Uhren absolut gleich bauen, dann könnten nach einer Synchronisation beim Start des Systems die Uhren frei weiterlaufen, und sie würden stets dieselbe Uhrzeit anzeigen. Allerdings ist es technisch nicht möglich, zwei absolut gleiche Uhren zu bauen. Man kann zwar durch einen hohen technischen Aufwand zwei Uhren so bauen, daß sie recht genau gleich laufen, ein Restfehler läßt sich aber nicht vermeiden. So können Atomuhren, die radioaktive Zerfälle zur Bestimmung der Zeit benutzen, eingesetzt werden. Allerdings ist diese Lösung sehr teuer und nicht für den mobilen Einsatz geeignet.

Weiterhin ist es nicht möglich, die Uhr eines entfernten Computers mit absoluter Genauigkeit zu lesen. Diese Ungenauigkeit und die Unterschiede der Taktgeber erzwingen, daß die Uhren in Intervallen immer wieder aneinander angepaßt werden müssen, damit ihre Uhrzeiten sich nicht zu weit voneinander entfernen.

Nachdem die intuitive Idee erklärt wurde, werden im folgenden die Begriffe genauer definiert und die Details der Uhrensynchronisation erörtert.

3.1 Begriffsbestimmungen

Für die Betrachtungen in dieser Arbeit wird davon ausgegangen, daß es einen allwissenden außenstehenden Beobachter gibt, der alle relevanten Ereignisse im betrachteten System identifizieren kann, und der eine Referenzuhr besitzt. Diese Referenzuhr sei chronoskopisch, d. h., sie ist stetig und besitzt keine Diskontinuitäten, entsprechend sind alle Zeitintervalle, die numerisch die gleiche Länge haben, auch tatsächlich gleich lang[5].

Diese Referenzuhr wird als Zeitbasis herangezogen, beispielsweise entspreche sie der Internationalen Atomzeit (TAI – *Temps Atomique Internationale*)[6]. Die angezeigten Werte dieser Referenzuhr werden als „reale Zeit“ bezeichnet.

Diese Referenzuhr ist ein Gedankenkonstrukt; tatsächlich kann man keine entsprechende Uhr bauen. Stattdessen baut man *physikalische Uhren*, die in Stationen einbaut werden.

---

[5] hier wird natürlich davon ausgegangen, daß ein intuitiver Begriff der Uhr vorhanden ist. Ansonsten könnte auch die Definition der Sekunde des Systéme Internationale d’Unités, die sich auch in einer DIN-Norm ([DIN 85]) niederschlägt, gewählt werden.
In der gesamten Arbeit wird von dem Begriff der Zeit nach Newton ausgegangen, d. h., relativistische Effekte werden außer Acht gelassen.

[6] Die andere bekannte offizielle Zeit, UTC (*universal time coordinated*), welche Grundlage der offiziellen Zeit ist, ist hierfür nicht geeignet, da sie Schaltsekunden enthält und damit Diskontinuitäten besitzt.

Eine *physikalische Uhr* besteht aus einem *Zeitgeber* und einem *Zähler*. Beim Start der Uhr steht der Zähler auf Null. Der Zeitgeber erzeugt in regelmäßigen Abständen ein Signal, den sogenannten *Tick*, welches von dem Zähler als Aufforderung zur Addition einer Eins verstanden wird.

Die Differenz zwischen zwei Signalen des Zeitgebers, gemessen mittels der Referenzuhr, bezeichnet man dann als *Granularität* oder *Auflösung* der physikalischen Uhr.

Mathematisch läßt sich die physikalische Uhr folgendermaßen definieren:

Sei g_p die *Auflösung* der physikalischen Uhr einer Station p in der Einheit der Zeit. Dann ist die *physikalische Uhr* der Station p eine monoton steigende surjektive Funktion $PC^{(p)}$: $\mathbb{R}^{\geq 0} \rightarrow \mathrm{PT}_p$, welche reale Zeiten in die Menge der physikalischen Zeiten PT_p, mit $\mathrm{PT}_p = \{ n \cdot g_p \mid n \in \mathbb{N} \}$, abbildet, wobei $PC^{(p)}(t) = 0$ für alle t mit $0 \leq t < g_p$ und $PC^{(p)}(t + g_p) = PC^{(p)}(t) + 1$ für alle t mit $t \geq g_p$ gilt. Eine physikalische Uhr zählt demnach im Gegensatz zur Referenzuhr die Uhrzeit in diskreten Schritten.

Der Zeitgeber einer Uhr ist dafür zuständig, alle g_p Zeiteinheiten ein Signal für den Zähler zu erzeugen. Da es technisch nicht möglich ist, Zeitgeber mit beliebiger Genauigkeit zu erstellen, unterscheidet sich die auf dem Zähler gemessene Zeit von der auf der Referenzuhr vergangenen Zeit, d. h., die physikalische Uhr driftet von der realen Zeit. Ein Maß für diese Drift gibt der Term $drift_p(t_1, t_2) = \dfrac{PC^{(p)}(t_2) - PC^{(p)}(t_1)}{t_2 - t_1}$, für zwei Zeitpunkte t_1 und t_2 der realen Uhr, an: Es ergibt sich das Verhältnis aus der Zeit, die auf der physikalischen Uhr zwischen den Zeitpunkten t_1 und t_2 vergangen ist, und der real vergangenen Zeit.

Da physikalische Uhren recht frequenzstabil sind, geht man im Allgemeinen von einer linearen Drift aus, d. h. die Drift ist für beliebige Zeitpunkte t_1 und t_2 konstant. Man bezeichnet diesen Wert daher mit $drift_p$.

Eine physikalische Uhr muß, damit sie brauchbar ist, die reale Zeit so gut wie möglich annähern. Damit ist es notwendig, daß $drift_p$ ungefähr gleich 1 ist. Um ein besseres Maß für die Drift zu erhalten, benutzt man daher die Driftrate $driftrate_p = | drift_p - 1|$.

Die Drift bzw. die Driftrate sind Funktionen in Abhängigkeit von der Uhr, die sie beschreiben, d. h., sie hängen von der konkreten betrachteten Uhr ab. Die Hersteller geben daher eine maximale Driftrate an und garantieren, daß die Uhr diese nicht überschreitet. Diese maximale Driftrate wird mit ρ bezeichnet.

Beim Vergleich zweier Uhren zu einem Zeitpunkt im Hinblick auf Uhrensynchronisation interessiert der Unterschied der Werte, den zwei Uhren zu einem Zeitpunkt besitzen. Man bezeichnet diesen Wert als Driftoffset oder Offset $offset_{p,q}(t)$, der sich mittels der Formel $offset_{p,q}(t) = PC^{(p)}(t) - PC^{(q)}(t)$ berechnet. Sofern klar ist, auf welche Station man sich bezieht, werden diese Angaben weggelassen.

3.2 Prinzip der Uhrensynchronisation

Die Uhrensynchronisation dient dazu, eine Menge von Uhren soweit aneinander anzugleichen, daß sie ständig eine vorgegebene Schranke für die maximale Differenz der Werte nicht überschreiten. Um dieses Ziel zu erreichen, werden Protokolle entwickelt, die mittels Nachrichtenaustausches Informationen über die Uhren von einer Station auf eine andere übertragen. Daher können diese Protokolle nur Informationen verwenden, die sie mittels des Nachrichtenaustausches erhalten können. Von besonderer Wichtigkeit ist hier der Begriff der Synchronisationsgruppe. Diese Gruppe besteht aus allen Stationen, deren Uhren miteinander synchronisiert werden sollen. Wie schon in Abschnitt 2.3 „Drahtlose Netzwerke“ ausgeführt, wird für diese Arbeit als Synchronsationsgruppe derjenige Gruppenbegriff gewählt, der schon im IEEE-Standard für ein Infrastrukturnetzwerk vorgesehen ist.

In [Sch 86] wird gezeigt, daß alle existierenden Synchronisationsprotokolle dem gleichen Schema folgen. Die Synchronisation besteht aus einer Endlosschleife auf jeder Station der Synchronisationsgruppe, die aus zwei Schritten besteht: Im ersten Schritt wartet die Ausführung auf das Erreichen eines Synchronisationszeitpunktes, im zweiten Schritt berechnet eine Funktion, die sogenannte Konvergenzfunktion, einen Korrekturwert, der zu der physikalischen Uhrzeit addiert wird, wobei die Konvergenzfunktion als Eingaben maximal die aktuelle Zeit auf der jeweiligen Station und die Zeiten auf den anderen Stationen erhält, die diese miteinander ausgetauscht haben.

Die virtuelle Uhr, d. i. die synchronisierte Uhrzeit einer Station p, besteht demnach aus der Summe aus der physikalischen Uhr $PC^{(p)}(t)$ und Korrekturwerten $FIX^{(p)}(t)$, wobei $FIX^{(p)}(t)$ für t zwischen den Synchronisationszeitpunkten konstant ist, an diesen Zeitpunkten aber seinen Wert potentiell ändert. Die virtuelle Uhr $VC^{(p)}$ ist demnach $VC^{(p)}(t) = PC^{(p)}(t) + FIX^{(p)}(t)$, wobei $FIX^{(p)}(t)$ zwischen den Synchronisationszeitpunkten t_i konstant ist, d. h., daß für alle t mit $t_i < t < t_{i+1}$ gilt: $FIX^p(t) = FIX^p_i$. Die FIX^p_i sind dabei konstante Werte, die jeweils zum Zeitpunkt t_i bestimmt werden.

Offensichtlich bleiben bei diesem Prinzip zwei Parameter offen, die die verschiedenen Protokolle voneinander unterscheiden: Die Bestimmung des Synchronisationszeitpunktes und die angewendete Konvergenzfunktion. In dem Begriff der Konvergenzfunktion ist auch der Teil des Synchronisationsprotokolls enthalten, der festlegt, welche und wie die Zeitinformationen mit den entfernten Stationen ausgetauscht werden.

Wenn die Synchronisationszeitpunkte auf allen beteiligten Stationen gleich sind, dann haben Synchronisationsprotokolle eine Rundenstruktur, d. h., innerhalb einer Runde, der Zeit zwischen zwei aufeinanderfolgenden Synchronisationspunkten, tauschen die Stationen Informationen miteinander aus, um sich dann nahezu gleichzeitig zu synchronisieren.

Man nennt die Zeit zwischen den Synchronisationszeitpunkten auch Synchronisationsintervalle. Bei den meisten Synchronisationsprotokollen sind alle Synchronisationsinter-

valle gleich lang. Die Länge richtet sich nach der geforderten Präzision, wie man der Abbildung 3-1 entnehmen kann.

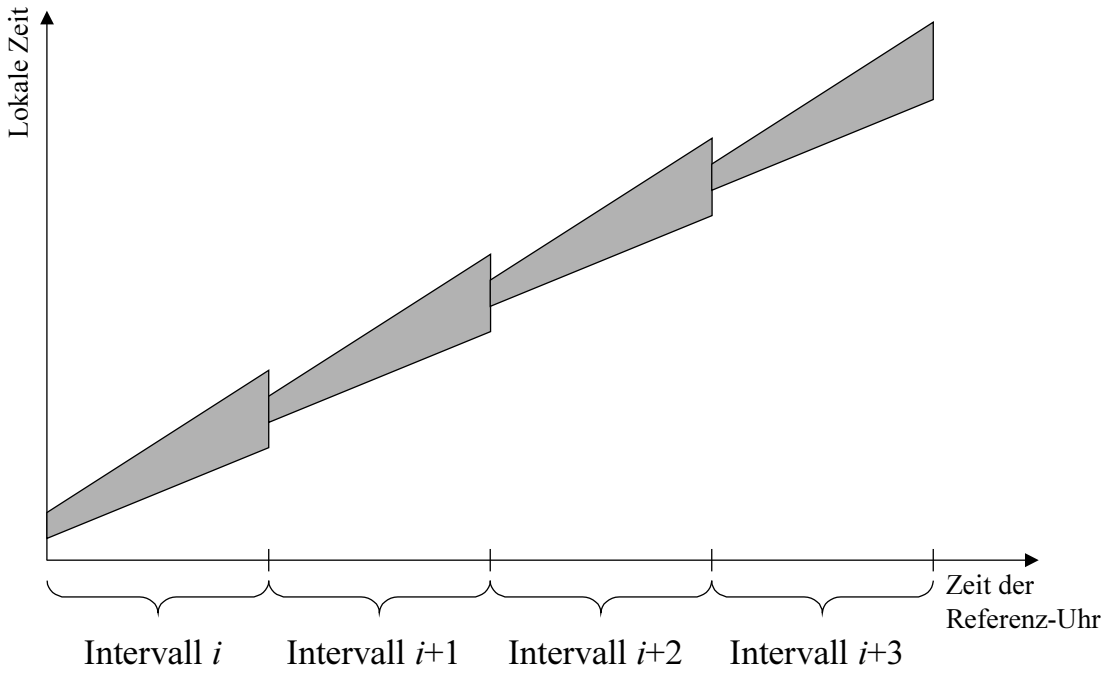


Abbildung 3-1: *Prinzip der Synchronisation*

Die Abbildung zeigt, in welchem Bereich sich die Uhren der beteiligten Stationen voneinander wegbewegen können. Zu Beginn des Intervalles *i* werden die Uhren mittels der Konvergenzfunktion aneinander angeglichen. Während des Synchronisationsintervalles bewegen sich die Uhren durch die Drift voneinander weg, bis sie am Ende des Intervalles (und damit zu Beginn des nächsten Intervalles *i*+1) wieder aneinander angeglichen werden.

Ein großes Synchronisationsintervall bedeutet also, daß die Uhren sich entsprechend lange voneinander wegbewegen können, so daß die garantierbare Präzision entsprechend verschlechtert wird. Andererseits müssen in einem Synchronisationsintervall Informationen über die beteiligten Stationen ausgetauscht werden, so daß kürzere Intervalle bedeuten, daß der Kommunikationsaufwand erhöht wird.

Die Konvergenzfunktion benutzt die Informationen über die (virtuellen) Uhren der anderen Stationen, soweit deren Informationen ausgetauscht wurden, und der virtuellen und physikalischen Uhr der lokalen Station, um einen Korrekturwert zu bestimmen.

Im weiteren Verlauf dieses Kapitels werden noch Beispiele für Konvergenzfunktionen gegeben werden, so daß hier auf später verwiesen wird.

3.3 Werte- und Ratenanpassung

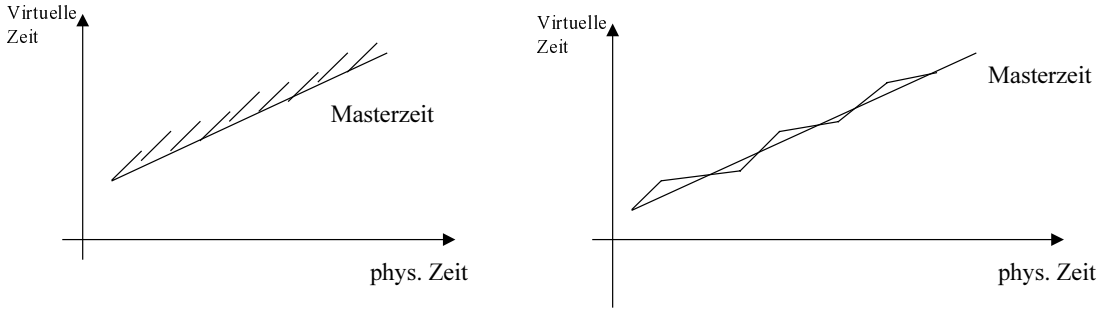
Bei der Uhrensynchronisation wird die lokale Uhr einer Station in Intervallen aufgrund der durch das Protokoll gewonnenen Informationen verändert. Dies kann zu einem bestimmten Zeitpunkt durch einfaches Umstellen der Werte der Uhren erfolgen, man spricht in diesem Falle von einer Werteanpassung (*value correction*). Dabei können Effekte entstehen, wie sie die Abbildung 3-2 verdeutlicht.

In beiden Teilabbildungen wird die virtuelle Uhr eines Slaves wie auch die virtuelle Uhr des Masters in Abhängigkeit von der physikalischen Zeit dargestellt. Die Gerade stellt jeweils die virtuelle Uhr des Masters dar, die Geradenstücke die virtuelle Uhr des Slaves.

In Abbildung 3-2a) erkennt man, daß die Zeit des Slaves schneller läuft als die des Masters, daher muß der Slave seine Zeit regelmäßig zurückstellen. Die Uhr des Slaves läuft immer im gleichen Tempo, also muß die Uhr dauernd zurückgestellt werden. Analog gibt es den Fall, daß die Uhr des Slaves zu langsam läuft und ständig vorgestellt werden muß.

In beiden Fällen erhält man Zeitsprünge, die für die Verarbeitung im verteilten System, insbesondere in verteilten Echtzeitsystemen, schwer zu handhaben sind, da beispielsweise Timeouts zu früh ablaufen oder die Zeitstempel zweier aufeinanderfolgender Ereignisse nicht chronologisch sind. Die zeitliche Reihenfolge wird zumeist benutzt, um einen Kausalzusammenhang zwischen Ereignissen zu erkennen, da später auftretende Ereignisse nicht Ursache für früher aufgetretene Ereignisse sein können. Diese kausale Abhängigkeit wird insbesondere bei Echtzeitsystemen benötigt, beispielsweise, um Fehlerursachen eindeutig bestimmen zu können.

Die Zeit, um die maximal gesprungen werden kann, gibt eine Untergrenze für die Zeit an, die der Zeitstempel eines Ereignisses mindestens größer sein muß als der Zeitstempel eines anderen, damit die Ereignisse nicht als gleichzeitig angesehen werden müssen. Wenn beispielsweise eine Uhr um einem Betrag x springen kann, dann kann ein Ereignis e_1 mit Zeitstempel t_1, der vor einem Ereignis e_2 mit Zeitstempel t_2 aufgetreten ist, trotzdem einen um einen Betrag $x' \leq x$ größeren Zeitstempel t_1 aufweisen, wenn nämlich t_1 unmittelbar vor und t_2 unmittelbar nach einem Zeitsprung rückwärts ermittelt wurde. Daher können die Ereignisse e_1 und e_2 nicht kausal geordnet werden und müssen als gleichzeitig betrachtet werden. Wenn daher Ereignisse mit kleineren Zeitunterschieden auftreten können, so erschwert sich die Erkennung der Ursache sehr.



a) Werteanpassung b) Ratenanpassung

Abbildung 3-2: *Werte- und Ratenanpassung*

Daher führt man die Ratenanpassung (‘*rate correction*‘) ein, wie sie in Abbildung 3-2b) zu erkennen ist: Die Frequenz der Uhr des Slaves wird so angepaßt, daß seine Uhr sich in Richtung der Uhr des Masters bewegt; hiermit vermeidet man, daß der Wert der Uhr sich sprunghaft verändert. Unter der Annahme, daß die Uhren von Master und Slave bei Beginn der Synchronisation nicht zu weit voneinander entfernt sind – was durch eine initiale Werteanpassung erreicht wird –, kann die Ratenanpassung durchgängig angewandt werden, es entstehen somit keine Zeitsprünge. Die Ratenanpassung wird so vorgenommen, daß die Uhr des Slaves sich immer mehr der Uhr des Masters anpaßt, so daß die Beträge der Korrekturwerte geringer werden.

Bei der obigen Definition der virtuellen Uhr wurde von einer Werteanpassung ausgegangen, da kein Mechanismus vorgesehen wurde, die Geschwindigkeit der Uhr zu verändern, sondern stattdessen zur Anpassung der Uhr ein Versatz (Offset) zur physikalischen Uhr addiert wird. Diese Definition ist reicht für die Betrachtungen aus, da man aus einem Verfahren mit Werteanpassung nachträglich ein Verfahren mit Ratenanpassung bilden kann (s. [ST 87]).

3.4 Interne und externe Synchronisation

Bei der Synchronisation der (virtuellen) Uhren einer Synchronisationsgruppe unterscheidet man zwischen interner und externer Synchronisation. Man sagt, daß die Uhren intern synchronisiert sind, wenn zu jedem Zeitpunkt die maximale Differenz zwischen jeweils zwei Werten der beteiligten Uhren einen vorgegebenen Wert Π, die Präzision, nicht überschreitet[7]. Hierdurch wird garantiert, daß Zeitstempel, die zu einem Zeitpunkt t genommen werden, auch auf verschiedenen Stationen maximal eine Abweichung von

[7] Dieser Begriff ist ungünstig, da er der Intuition widerspricht: Eine (numerisch) große Präzision bedeutet eine schlechte Präzision. Daher wird folgende Sprachregelung getroffen: Eine (numerisch) große Präzision ist eine niedrige Präzision, eine hohe Präzision ist hingegen numerisch klein. Der Begriff der Präzision wird benutzt, da er in der Literatur üblich ist.

Π besitzen. Umgekehrt können zwei Zeitstempel, die eine Differenz von mehr als Π besitzen, eindeutig zeitlich in Beziehung zueinander gesetzt werden.

Wenn ein Uhrensynchronisationsprotokoll mit einer Präzision Π ausgeführt wird, dann erhält man eine globale Zeit. Im Idealfall wäre $\Pi = 0$. In der Praxis ist dieses Ideal allerdings nicht erreichbar.

Häufig benötigt man die zusätzliche Eigenschaft, daß die Uhrzeit der Synchronisationsgruppe mit einer äußeren Zeit übereinstimmt, beispielsweise mit der Internationalen Atomzeit TAI. Dann reicht eine interne Synchronisation nicht mehr aus. Der Grund hierfür liegt darin, daß bei der internen Synchronisation lediglich gefordert ist, daß die Uhren der Gruppe untereinander synchronisiert sind.

Um die internen Uhrzeiten mit einer äußeren Uhrzeit zu synchronisieren, benutzt man eine sogenannte externe Synchronisation. Bei dieser benötigt man die Existenz eines Zeitservers, d. h. einer externen Instanz, die diese äußere Uhrzeit zur Verfügung stellt. Der Uhrzeit dieses Zeitservers muß bei der Anwendung der Konvergenzfunktion besondere Relevanz gegeben werden.

Der erwähnte Zeitserver erhält seine Uhrzeit zumeist von einer weiteren Quelle, beispielsweise einem GPS- (Global Positioning System, ein satelliten-gestütztes System zur Ermittlung der eigenen Position auf der Erdoberfläche) Empfänger, die die reale Uhrzeit mit der geforderten Genauigkeit ermitteln kann.

Bei der externen Synchronisation wird gefordert, daß die beteiligten Uhren zu keinem Zeitpunkt einen größeren Abstand als die Genauigkeit (*accuracy*) Π' zur externen Zeitquelle besitzen. Daher ist die externe Synchronisation ein authoritativer Prozeß, d. h. der Zeitserver zwingt allen synchronisierenden Stationen seine Zeit auf. Bei einer internen Synchronisation kann hingegen ein kooperativer Prozeß gewählt werden, indem die Uhren aller beteiligten Stationen in der Konvergenzfunktion berücksichtigt werden.

Als Seiteneffekt sind extern synchronisierte Uhren auch intern synchronisiert mit einer Präzision $\Pi = 2\Pi'$, da jede Uhr jeweils maximal um Π' zu groß oder zu klein sein kann, also die maximale Differenz zweier Uhren, die Präzision, maximal $2\Pi'$ beträgt. Häufig setzt man kombinierte Verfahren ein, so daß gleichzeitig extern und intern synchronisiert wird, damit man eine Präzision kleiner als $2\Pi'$ erhält.

Bei genauer Betrachtung läßt sich die externe Synchronisation als eine spezielle interne modellieren. Wenn die Kovergenzfunktionen aller Stationen alle eine einzige Station berücksichtigen, dann zwingt diese eine Station alle anderen auf ihre Uhrzeit. Die ausgezeichnete Station nennt man in diesem Fall einen (Zeit-)Master, die anderen Stationen entsprechend (Zeit-)Slaves; das Synchronisationsprotokoll nennt man entsprechend ein Master-/Slave-Protokoll.

3.5 Grenzen für die Genauigkeit der Uhrensynchronisation

Bei der Betrachtung einer Synchronisation mit Rundenstruktur wird die Präzision von zwei Werten bestimmt, der Genauigkeit Φ, mit der eine globale Zeit zu den Synchronisationszeitpunkten t_i bestimmt werden kann, und der Drift Γ, der die Uhren zwischen den Synchronisationszeitpunkten t_i ausgesetzt sind (s. Abbildung 3-1). Da nur abzählbar viele Synchronisationszeitpunkte vorhanden sein können, können diese auch nur diskret angeordnet sein, dadurch gibt es eine minimale Differenz aller t_i voneinander, die größer Null ist. Dadurch ergibt sich zwischen zwei Synchronisationszeitpunkten für den Driftoffset $\Gamma > 0$, und somit kann eine Präzision $\Pi = 0$ nicht erreicht werden.

Auch die Bestimmung einer Globalen Zeit zu den Synchronisationszeitpunkten t_i kann nicht mit beliebiger Genauigkeit Φ erreicht werden, nicht einmal mit der Vereinfachung, daß durch die Kommunikation keinerlei Fehler entstehen können, was in der Praxis nicht erreicht werden kann.

Man betrachte den Fall, daß eine Station einer anderen ihre Uhrzeit übermitteln will. Da sich die Uhren der beteiligten Stationen diskret, d. h. in Ticks, weiterbewegen, ergibt sich folgendes Sznario (s. Abbildung 3-3): Der Empfänger kann bei einer von dem Sender erhaltenen Uhrzeit nicht unterscheiden, ob die übermittelte Uhrzeit unmittelbar nach einem Tick auf der Uhr des Senders (Ereignis e_1 in der Abbildung) oder unmittelbar vor einem Tick auf dessen Uhr (Ereignis e_2 in der Abbildung) gelesen wurde, oder irgendwann dazwischen. Damit ergibt sich, daß die Uhr des Senders s auch im Idealfall nur mit einem Fehler von mindestens der Granularität g_s bestimmt werden kann. Somit ist der Fehler beim Auslesen der Uhr nach unten durch g_s beschränkt.

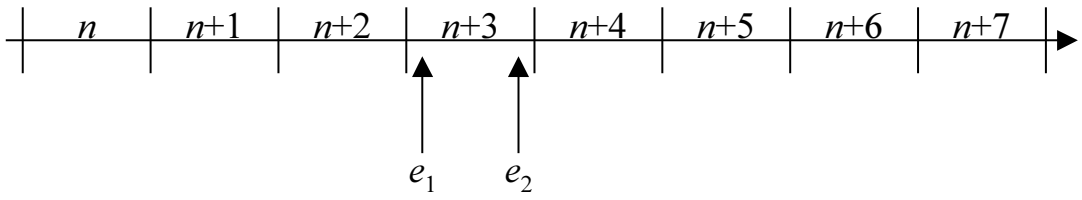


Abbildung 3-3: *Grenzen für die Bestimmung der Vorher-/Nachher-Eigenschaft*

Andererseits unterliegt auch die Kommunikation der Stationen einer Ungewißheit, da die Zeitinformation in einer nicht-vernachlässigbaren Zeitspanne übertragen wird. Diese Zeitspanne unterliegt selbst einer Schwankung, es gibt ein Minimum und ein Maximum[8], wobei der Empfänger nicht unterscheiden kann, ob die Information beim Versenden die maximale oder die minimale Zeit benötigt hat. Die Differenz zwischen

[8] Bei vielen Kommunikationskanälen gibt es kein Maximum, vielmehr kann die Nachrichtenauslieferung beliebig lange verzögert werden. In diesen Fällen werden zusätzliche Mechanismen ergänzt, um ein „künstliches Maximum“ zu erhalten.

dem Minimum und dem Maximum, der sogenannte Netzwerkjitter ε_0, kann demnach als Fehler bei der Bestimmung der Uhrzeit nicht herausgerechnet werden.

Bei einem Master-/Slave-Protokoll, d. h. einem Protokoll, bei dem eine ausgezeichnete Station den anderen ihre Zeit in regelmäßigen Abständen sendet, ist damit der Fehler Φ nach unten beschränkt durch den Jitter $\varepsilon = g_s + \varepsilon_0$, also gilt $\Phi \geq \varepsilon$.

In [LL 84] zeigen die Autorinnen, daß unabhängig von der Wahl einer Konvergenzfunktion mittels eines Synchronisationsprotokolls bei der Synchronisation von n Stationen stets $\Phi \geq \varepsilon(1\text{-}1/n)$ gilt. Um dieses Ergebnis zu zeigen, definieren die Autorinnen Begriffe wie Sichten eines Prozesses und Ausführungen eines Prozesses, wobei im Beweis (ähnlich wie oben, aber allgemeiner) gezeigt wird, daß Ausführungen von den Prozessen nicht unterschieden werden können und sich dadurch Fehler ergeben. Insbesondere ergibt sich, daß der Fehler der Konvergenzfunktion bei großen n auch bei nicht-authoritativen Verfahren nur unwesentlich besser ist als bei einem Master/Slave-Verfahren.

Diese Grenze für die Genauigkeit wurde in [LL 84] bewiesen unter der Voraussetzung, daß ein deterministischer Algorithmus angewendet wird. In [Cr 89] zeigt der Autor, daß die untere Schranke unterschritten werden kann, wenn der Determinismus nicht notwendig ist. Hierzu stellt er ein Verfahren vor, welches probabilistisch die Uhr synchronisiert. Das Verfahren beruht auf der Erkenntnis, daß das Auslesen der Uhr einer entfernten Station nur mit einer geringen Genauigkeit möglich ist, die durch die Varianz bestimmt wird, mit der eine Nachricht von einem Sender zu einem Empfänger gesendet wird. Um die Varianz bestimmen zu können, muß ein Slave, der eine entfernte Uhr lesen will, von der entfernten Station, dem Master, die Uhrzeit anfordern. Der Master antwortet dann mit der Uhrzeit. Der Slave kann durch Messung der Laufzeit von seinem eigenen Sendeaufruf bis zum Empfang der Antwort feststellen, wie groß die Varianz der Nachrichtenübertragung ist, sofern er die minimale Zeit kennt. Benötigte die Antwort zu lange, verwirft er diese und fordert erneut eine Übertragung an. Je genauer der Slave die Master-Uhr auslesen will, desto weniger Zeit gibt er dem Master für die Antwort, allerdings wird der dann auch mit einer größeren Wahrscheinlichkeit die vorgegebene maximale Zeit überschreiten und damit keine Synchronisation möglich. In [OS 91] wiederum wird gezeigt, wie der strenge Master-/Slave-Aufbau bei einem probabilistischen Algorithmus vermieden werden kann.

Die Synchronisation der Uhren dient dem Zweck, Zeitstempel in zeitliche Beziehung miteinander setzen zu können. Wie gezeigt wurde ist es unmöglich, Uhren völlig exakt miteinander zu synchronisieren. Daher stellt sich die Frage, wie genau zwei Zeitstempel miteinander verglichen werden können, wenn sie miteinander in Beziehung zu setzen sind.

Bei den folgenden Überlegungen wird immer davon ausgegangen, daß eine Ratenanpassung erfolgt, weil die bei einer Werteanpassung auftretenden Zeitsprünge die Betrachtung erschweren, aber nicht näher zum Verständnis beitragen.

Zwei Zeitstempel τ_1 und τ_2 zu Ereignissen e_1 und e_2, die auf einer einzigen Station genommen wurden, können durch den numerischen Vergleich der Werte in zeitlichen Bezug miteinander gebracht werden, da die virtuelle Uhr auf der Station monoton steigend ist, d. h.,

- falls $\tau_1 < \tau_2$, dann fand das Ereignis e_1 vor dem Ereignis e_2 statt,
- falls $\tau_1 > \tau_2$, dann fand das Ereignis e_1 nach dem Ereignis e_2 statt,
- falls $\tau_1 = \tau_2$, dann können die Ereignisse e_1 und e_2 nicht geordnet werden, sondern es muß angenommen werden, daß sie gleichzeitig auftraten. In der Wirklichkeit können sie zwar dennoch nacheinander aufgetreten sein, aufgrund der diskreten Uhr läßt sich diese Beziehung aber nicht ohne weitere Hilfsmittel rekonstruieren.

Interessanter ist der Fall, wenn die Zeitstempel τ_1 und τ_2 zu den Ereignissen e_1 und e_2 von unterschiedlichen Stationen erstellt wurden. In diesem Fall können sich die virtuellen Uhren der jeweiligen Stationen p_1 und p_2 um Π voneinander unterscheiden. Man betrachte die Abbildung 3-4, in der beispielhaft das Verhalten bei der Präzision $\Pi=3\cdot g_p$ dargestellt wird, wobei $g_p = g_q$ angenommen wird. Man erkennt an den vertikalen Linien, wann die Uhren der Stationen p bzw. q weitergestellt werden. Durch die Synchronisation kann eine der beiden Uhren maximal um den Betrag der Präzision Π vorauseilen.

Man betrachte das Ereignis e. Wenn sein Zeitstempel auf der Uhr von p, τ_p, gleich $n\cdot g_p$ ist, dann ist der Zeitstempel für das gleiche Ereignis auf der Uhr von q, τ_q, minimal gleich $(n\text{-}3)\cdot g_p$; analog ergibt sich, daß der Zeitstempel τ_q maximal gleich $(n+3)\cdot g_p$ sein kann. Damit wurde durch die Abbildung beispielhaft gezeigt:

- falls $\tau_2 - \tau_1 > \Pi$, dann fand das Ereignis e_1 vor dem Ereignis e_2 statt,
- falls $\tau_1 - \tau_2 > \Pi$, dann fand das Ereignis e_1 nach dem Ereignis e_2 statt,
- falls $|\tau_1 - \tau_2| \leq \Pi$, dann können die Ereignisse e_1 und e_2 nicht geordnet werden, sondern es muß angenommen werden, daß sie quasi-gleichzeitig auftraten.

Dies ist ein weiterer Grund, warum die Präzision Π möglichst hoch (also numerisch klein) sein soll. Außerdem liefert eine hohe Präzision die Möglichkeit, verteilte Prozesse besser synchronisieren zu können. Hierdurch kann die Länge einer steuernden Kontrollschleife gering gehalten werden.

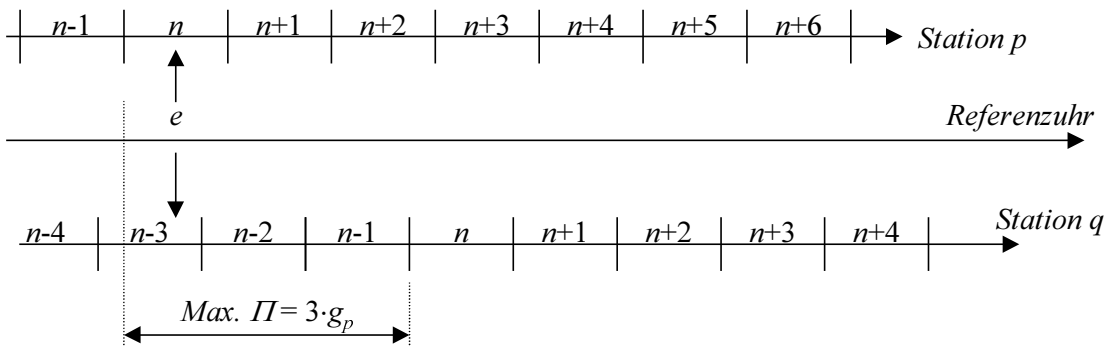


Abbildung 3-4: *Präzision als begrenzender Faktor beim Sortieren von Zeitstempeln*

3.6 Das A-Posteriori-Agreement Protokoll

Das A-Posteriori-Agreement-Protokoll ([RV 92], kurz: APA-Protokoll), welches eine Idee aus [BD 87] aufgreift, ist ein Protokoll, welches auf Broadcast-LANs angewendet werden kann. Es benutzt die inhärenten Eigenschaften dieser Netzwerke: geringe Fehlerrate, begrenzte Nachrichtenübermittlungszeit, durchschnittliche Übertragungszeit nahe am Minimum und der Empfang von Nachrichten ist „dicht“ (*tight*), d. h. der physikalische Empfang der Nachrichten erfolgt nahezu zeitgleich in allen empfangenden Stationen. Diese letzte Eigenschaft ist äußerst wichtig für den Mechanismus des Protokolls.

Um den Mechanismus genauer zu verstehen, ist es zweckmäßig, einige Betrachtungen über das Versenden von Nachrichten voranzustellen. Die Zeit, die eine Nachricht auf einem Netzwerk benötigt, um von einem Sender *S* zu einem Empfänger *R* zu gelangen, teilt sich in vier Teile auf (s. Abbildung 3-5):

- die Sendezeit λ_{send}, die benötigt wird, um die zu sendende Nachricht zusammenzustellen und den Sendeaufruf zu tätigen;
- die Medienzugriffszeit λ_{access}, die der Sender benötigt, um auf das Medium zuzugreifen;
- die Übertragungszeit λ_{prop}, die auf dem physikalischen Medium benötigt wird, um die Distanz zu überbrücken; und
- die Empfangszeit λ_{recv}, die der Empfänger benötigt, um die Nachricht zu bearbeiten.

Der Netzwerkjitter ε_0 und damit die Präzision eines Synchronisationsprotokolls hängt von den Differenzen der maximalen[9] und minimalen Werte der jeweiligen Zeiten ab.

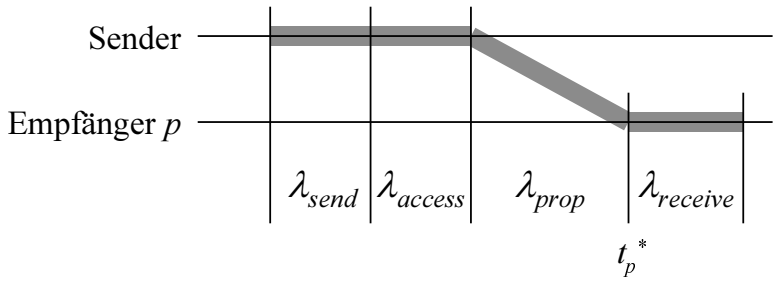


Abbildung 3-5: *Zeitlicher Ablauf beim Versenden einer Nachricht*

Zum Empfangszeitpunkt einer Nachricht kann der Empfänger nicht feststellen, ob bei der Übertragung die Werte der angesprochenen Zeiten λ_{send}, λ_{access}, λ_{prop} und λ_{recv} die maximalen oder die minimalen Werte oder irgendwelche Zwischenwerte angenommen haben. Der Empfänger kann also nur Schätzungen für die Werte ansetzen, die bestenfalls jeweils um die Hälfte der Differenz des Maximums und des Minimums fehlerhaft sein können, sofern er den Mittelwert aus Maximum und Minimum als Schätzwert ansetzt.

Da der Empfänger diese Zeiten also nur mit einem großen Fehler behaftet schätzen kann, ist eine Betrachtung notwendig, welche Fehler in der Folge Auswirkungen bei der Uhrensynchronisation besitzen, da hierdurch Fehler bei der Bestimmung der Uhr des Senders bewirkt werden. Die Zeitspanne zwischen dem Beginn und dem Ende der relevanten Aktionen, die zeitkritisch sind und daher einen möglichst kleinen Fehler besitzen dürfen, da der Empfänger Schwankungen nicht entdecken kann, bilden den sogenannten *(zeit-) kritischen Pfad*. Der Netzwerkjitter besteht aus der Differenz der maximalen und minimalen Länge des kritischen Pfades. Um eine hohe Präzision zu erreichen ist es notwendig, den Fehler, der durch die Laufzeitunterschiede des kritischen Pfades entstehen kann, soweit wie möglich zu minimieren. Hierzu versucht man, den kritischen Pfad insgesamt zu minimieren.

Beim Senden einer Nachricht, die einen Zeitstempel für einen Zeitpunkt „jetzt“ enthält, ergibt sich der kritische Pfad als die Zeitspanne zwischen dem Nehmen des Zeitstempels auf der sendenden Station und der Verarbeitung des Zeitstempels auf dem Empfänger. Der kritische Pfad hat demnach die Länge $\lambda = \lambda_{send} + \lambda_{access} + \lambda_{prop} + \lambda_{recv}$. Die meisten Synchronisationsprotokolle, bei denen durch Nachrichtenaustausch die Uhren synchronisiert werden, indem die Nachrichten das Erreichen eines Zeitpunktes signalisieren oder einen Zeitstempel für die aktuelle Uhrzeit enthalten, besitzen diesen kritischen Pfad. Dies liegt daran, daß bei solchen Protokollen auf einen festen Zeitpunkt,

[9] Wie schon in Fußnote 8 gesagt, muß häufig durch weitere Mechanismen ein künstliches Maximum erzeugt werden, weil es sonst überhaupt kein Maximum gäbe.

zumeist der Zeitpunkt, an dem der Zeitstempel beim Sender genommen wurde, zurückgerechnet werden muß. Hierzu muß die Zeitspanne λ bekannt sein; Schwankungen können vom Empfänger nicht erkannt und somit auch nicht herausgerechnet werden.

Das APA-Protokoll nutzt hingegen den Umstand aus, daß bei Broadcast-LANs die Zeitpunkte t_p^* des physikalischen Empfangs einer Nachricht auf verschiedenen Stationen p sehr dicht beieinander liegen. Diese Zeitpunkte t_p^* können damit als beobachtbare Ereignisse benutzt werden, die mit Zeitstempeln versehen werden und zur Berechnung der Zeitdifferenzen zwischen den Stationen herangezogen werden, so daß die Zeiten λ_{send} und λ_{access} für die Berechnung des Netzwerkjitters nicht mehr relevant sind.

Das Verfahren geht hierzu wie folgt vor:

Bei Erreichen eines vordefinierten Zeitpunktes auf der virtuellen Uhr einer Station versendet diese eine Broadcast-Nachricht, die zu einem sogenannten „simultanen Broadcast“ wird (hierzu später mehr). Bei Empfang eines simultanen Broadcasts startet jede Station eine neue virtuelle Uhr. Da alle Stationen wegen der geringen Laufzeitvarianz (*tightness*) des Nachrichtenempfangs diesen Broadcast ungefähr zur gleichen Zeit empfangen, sind die virtuellen Uhren damit synchronisiert. Dies liegt daran, daß dieser Zeitpunkt des simultanen Broadcasts auf allen Stationen gleichzeitig erkannt wird und damit die virtuelle Uhr von allen beteiligten Stationen zum gleichen Zeitpunkt mit den gleichen Werten (entsprechend des vordefinierten Zeitpunktes) gestartet wird.

Es stellt sich die Frage, wie in diesem Zusammenhang der simultane Broadcast erzeugt wird, da fehlerhafte Uhren und Stationen mit Empfangsausfällen von dem Protokoll toleriert werden sollen. Daher sendet jede Station bei Erreichen des vordefinierten Zeitpunktes auf ihrer virtuellen Uhr einen Broadcast aus; dieser Broadcast wird von allen empfangenden Stationen bestätigt. Die Stationen starten einen Kandidaten für die neue virtuelle Uhr, d. h., sie berechnen alle notwendigen Parameter, als ob diese Uhr die neue virtuelle Uhr sei, aktivieren sie aber noch nicht. Somit startet jede Station für alle empfangenen Broadcasts einen Kandidaten.

Wenn im Rahmen der geforderten Fehlertoleranz genügend Bestätigungen für einen Kandidaten empfangen wurden, schlägt (mindestens) eine Station vor, daß dieser Kandidat aktiv werden soll. Hierzu wird ein Einigungs-Protokoll (*agreement protocol*, daher der Name *a posteriori agreement*) ausgeführt, an dessen Ende sich alle Stationen auf einen Kandidaten geeinigt haben. Der Broadcast, welcher zu diesem Kandidaten führte, ist damit im Nachhinein zu dem beschriebenen simultanen Broadcast geworden.

Dieses Protokoll nutzt demnach ein auf dem Netzwerk global beobachtbares Ereignis, den simultanen Broadcast, um diese Information zur Synchronisation der Uhren zu benutzen.

Das Protokoll hat eine hohe Präzision und eine gute Fehlertoleranz. Diese Eigenschaften erkauft es sich aber mit einem relativ hohen Nachrichtenaufwand, da im schlechtes-

ten Fall *n* Broadcasts und als Einigungsprotokoll *n* Bestätigungen und *n* Vorschläge für Kandidaten, von jeder Station jeweils eine Nachricht, versendet werden.

3.7 Das GS-Protokoll

Das GS-Protokoll ([GS 94]) kann als eine Spezialisierung des APA-Protokolls angesehen werden. Hier wie dort wird ein global beobachtbares Ereignis genutzt, um einen Kandidaten für eine neue virtuelle Uhr zu erhalten. Die Protokolle unterscheiden sich aber darin, wie sie das globale Ereignis erzeugen und wie sie damit umgehen.

Das GS-Protokoll arbeitet, wie alle bekannten Protokolle zur Uhrensynchronisation, in Runden, d. h., in regelmäßigen Intervallen werden Informationen über die Uhrzeit versendet und die lokalen Uhrzeiten entsprechend angepaßt (vergleiche [Sch 86], [Ko 97]).

Das GS-Protokoll ist ein sogenanntes Master-/Slave-Protokoll, d. h. eine besondere Station, der Master, versendet ihre Uhrzeit, und die anderen Stationen, die Slaves, übernehmen diese. Im Gegensatz zum APA-Protokoll wird beim GS-Protokoll das global beobachtbare Ereignis nicht zu einem vorbestimmten Zeitpunkt generiert, sondern der Master entscheidet, wann er die Indikation erzeugt. Zwar wird in einer realen Implementierung der Master in regelmäßigen Abständen die Indikation verschicken, allerdings verwendet das Protokoll diesen Umstand nicht für die Synchronisation. Da diese Information nicht a priori bekannt ist, muß im Gegensatz zum APA-Protokoll beim GS-Protokoll im Nachhinein der Zeitstempel des Masters versendet werden. Andererseits muß durch die Verwendung eines Masters kein Agreement-Protokoll benutzt werden, so daß der Kommunikationsaufwand auf dem Netzwerk gering ist.

Der Master ist der einzige Sender innerhalb des Synchronisationsprotokolls. Sein grundsätzlicher zeitliche Ablauf kann der Abbildung 3-6 entnommen werden kann, in der eine Synchronisationsrunde dargestellt ist. Die Idee besteht darin, daß alle an der Synchronisation beteiligten Stationen, also sowohl der Master als auch die Slaves, ein physikalisches Ereignis benutzen, welches in der Abbildung zum Zeitpunkt t_3 eintritt, und einen Zeitstempel nehmen, der mit diesem Ereignis assoziiert wird. Das physikalische Ereignis zum Zeitpunkt t_3 wird auf den einzelnen Stationen kurz danach, zum Zeitpunkt t_s, beobachtbar. Der Master versendet hierauf seinen eigenen Zeitstempel per Broadcast an die Slaves (zum Zeitpunkt t_5), die hieraus ihrerseits die Differenz ihrer lokalen Uhren zu der Uhr des Masters bestimmen und ihre lokalen Uhren anpassen können (zum Zeitpunkt t_6).

Das physikalische Ereignis zum Zeitpunkt t_3 wird seinerseits durch das Versenden einer Indikationsnachricht durch einen Knoten zum Zeitpunkt t_1 generiert.

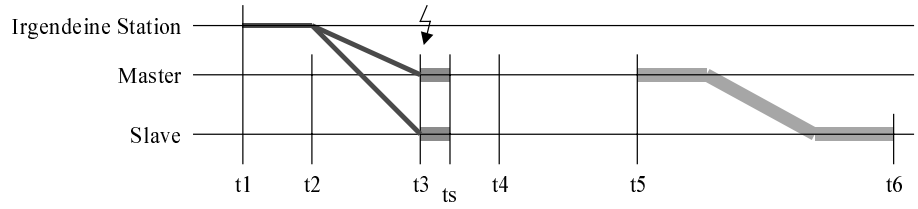


Abbildung 3-6: *Das GS-Protokoll, Ausgangsform*

Der kritische Pfad besteht hier lediglich aus der Zeit zwischen dem physikalischen Ereignis (zum Zeitpunkt t_3) und dem Zeitpunkt, an dem dieses Ereignis beobachtbar wird (zum Zeitpunkt t_s). Sofern man davon ausgeht, daß die Verzögerung auf dem Medium für alle beteiligten Stationen als gleich angesehen werden kann, ist der Zeitpunkt t_3, zu dem der zeitkritische Pfad beginnt, auf allen beteiligten Stationen der gleiche. Diese Annahme kann auf dem CAN-Bus getroffen werden, da dies durch die Charakteristiken des Busses gewährleistet wird. Auf jeder Station müssen hiernach nur noch lokale Verzögerungen (t_s - t_3) berücksichtigt werden, die für jede Station vorab bestimmt werden können.

Durch das bisher beschriebene Verfahren würde für jede Synchronisation das Versenden von zwei Nachrichten benötigt, eine, die als Indikation zum Nehmen des Zeitstempels dient, und eine, um den Zeitstempel zu versenden. Damit nur eine Nachricht benötigt wird, werden beide Nachrichten miteinander kombiniert; es ergibt sich der Ablauf entsprechend Abbildung 3-7, in der nun zwei Synchronisationsrunden dargestellt sind. In jeder Synchronisationsrunde wird eine Nachricht gesendet, welche zum Zeitpunkt t_s bzw. t_s' von den beteiligten Stationen gezeitstempelt wird, gleichzeitig enthält sie den Zeitstempel für die vorherige Indikationsnachricht; d. h., jede Nachricht enthält zwei Informationen: Eine ergibt sich aus dem Inhalt der Nachricht, die andere aus dem Zeitpunkt ihres Empfanges.

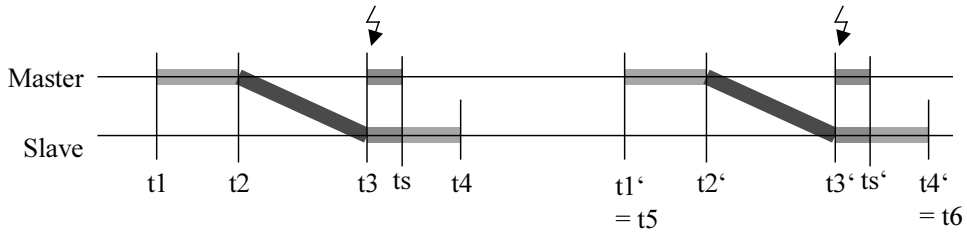


Abbildung 3-7: *Das optimierte GS-Protokoll*

Dieses Protokoll ist stark für den CAN-Bus optimiert. Der CAN-Bus besitzt eine geringe Verlustrate, dafür aber eine geringe Bandbreite. Da auf dem CAN-Bus Prioritäten für Nachrichten vergeben werden, können Garantien gegeben werden, daß und mit welcher maximalen Verzögerung Nachrichten auf dem Medium versendet werden können. Diese Eigenschaften nutzt das Protokoll aus.

4 Uhrensynchronisationsprotokoll für das Funknetzwerk

In dem vorliegenden Kapitel wird das im Rahmen dieser Arbeit entwickelte Synchronisationsprotokoll für das Funknetzwerk beschrieben. Bevor dies geschieht, wird zunächst in dem folgenden Abschnitt das Modell, welches für das Funknetzwerk angenommen wird, beschrieben, und im darauffolgenden Abschnitt das Synchronisationsprotokoll des IEEE-Standards beschrieben und seine Nachteile dargestellt.

4.1 Systemmodell

In Abschnitt 2.2 „Fehlertoleranz“ wurde ausgeführt, daß aufgrund der beliebigen Art möglicher Fehler ein Fehlermodell benötigt wird, wenn man Garantien über das Verhalten eines Protokolls oder Algorithmus abgeben möchte. Das Systemmodell für das Funknetzwerk soll, so weit es für die Uhrensynchronisation relevant ist, in diesem Abschnitt entwickelt werden.

In einem Infrastrukturnetzwerk hingegen existiert eine besondere Station, der sogenannte AP. Diese Station verwaltet einen Gruppenbegriff, der alle Stationen umfaßt, die sich in seinem Empfangsbereich befinden und sich bei ihm angemeldet haben. Durch die Möglichkeit, den AP mit einer stärkeren stationären Antenne auszustatten als die anderen Stationen, ergibt sich weiterhin die Möglichkeit, den Empfangsbereich des APs wesentlich größer zu gestalten als den der anderen Stationen. Weiterhin kann der AP als ortsfeste Station an einer strategischen Stelle plaziert werden, beispielsweise an einer Kreuzung, wenn mobile Systeme synchronisiert werden sollen. Der AP kann ein dynamisches Ein- und Aussteigen von Stationen in seine Gruppe erkennen, wobei er sich hierzu der Mechanismen bedient, die im IEEE-Standard vorgesehen sind (Assoziation und Disassoziation). Diese Möglichkeiten bieten sich in einem Ad-Hoc-Netzwerk nicht. Wegen dieser Vorteile gegenüber einem Ad-Hoc-Netzwerk wird das Funknetzwerk als ein Infrastrukturnetzwerk angenommen. Der AP übernimmt das Versenden der Synchronisationsnachrichten und der Gruppenbegriff des APs ist der Begriff der Synchronisationsgruppe.

Die in dem vorgestellten Protokoll versendeten Synchronisationsframes (*Sync-Frames*) besitzen Priorität auf dem Funkmedium. Dies kann dadurch erreicht werden, daß sie in besonderen Frames versendet werden. Da der AP der einzige Sender bei der Synchronisation sein wird, gibt es dabei zwei Möglichkeiten, dies zu erreichen: Die erste Möglichkeit besteht darin, die Sync-Frames in dem Beacon zu versenden (s. Abschnitt 2.3.1.3 „Das Beacon-Frame“), d. h., von dem Protokoll wird gar kein eigenes Frame versendet, sondern vielmehr wird ein bestehendes Frame, das Beacon, erweitert, so daß es die Informationen für die Synchronisation mit übertragt. Die zweite Möglichkeit besteht darin, die Sync-Frames in einem eigenen Broadcast-Frame zu versenden, welches allerdings – wie das Beacon – mit einem kürzeren IFS gesendet wird, so daß es ebenso

wie das Beacon Priorität hat (s. hierzu Abschnitt 2.3.1.2 „Medienzugriff“). In beiden Fällen gelten im Modell folgende Eigenschaften für ein Sync-Frame:

(W1) Ein Sender kann eine maximale Zeitspanne zwischen dem Versenden zweier Sync-Frames garantieren.

(W2) Ein versendetes Sync-Frame wird von einem Empfänger entweder vollständig und korrekt, oder gar nicht empfangen.

(W3) Ein versendetes Sync-Frame wird von dem Sender maximal einmal physikalisch versendet.

(W4) Auf zwei verschiedenen Empfängern, die das gleiche versendete Sync-Frame empfangen, liegt zwischen dem Aufruf eines Mechanismus, der einen Zeitstempel für den erfolgreichen physikalischen Empfang dieses Frames nimmt, auf den beiden Stationen eine bekannte maximale Zeit. Dabei ist der Zeitpunkt des physikalischen Empfangs der Nachricht der Zeitpunkt, qn dem das letzte Bit des empfangenen Frames am PHY-Layer der Netzwerkkarte ankommt.

(W5) Auf jedem Empfänger liegen zwischen zwei erfolgreich empfangenen Sync-Frames höchstens *OD* (*omission degree*) viele Sync-Frame, die vom Sender versendet, aber nicht empfangen wurden.

Die Annahme (W1) kann garantiert werden, weil die Sync-Frames mit Priorität auf dem Funkmedium versendet werden, sofern keine zwei Stationen, die so eine Nachricht versenden wollen, sich im gegenseitigen Empfangsbereich befinden.

Auf dem Netzwerk wird jedes Frame mit einer Prüfsumme gesendet, damit Übertragungsfehler erkannt werden können. Aufgrund dieses Mechanismus werden unvollständige oder fehlerhafte Frames als fehlerhaft zurückgewiesen und ignoriert. Daher kann die Annahme (W2) garantiert werden.

Um die Annahme (W3) zu garantieren, bedarf es einer Vorüberlegung. In Abschnitt über den Medienzugriff wurde ausgesagt, daß das Funknetzwerk ein Verfahren mit positivem Acknowledgment benutzt, um fehlerhafte Frames zu erkennen. Das Ausbleiben einer Bestätigung für das Sync-Frame wäre ein Zeichen für den Sender, das Frame erneut zu versenden. Hiermit wäre es möglich, daß ein Sync-Frame zweimal versendet würde, was dieser Eigenschaft widerspräche.

Allerdings benutzt das Funknetzwerk dieses Bestätigungs-Schema nur bei Datennachrichten, die Punkt-zu-Punkt, d. h., von einem Sender zu einem Empfänger, gesendet werden. Ist ein Frame hingegen ein Broad- oder Multicast, d. h. es gibt mehr als einen Empfänger, oder handelt es sich bei dem Frame um ein Management-Frame wie das Beacon-Frame, so wird das Bestätigungs-Schema nicht angewandt. Daher versendet ein Sender das Frame auch nur einmal, und die Annahme kann garantiert werden.

Um die Annahme (W4) zu rechtfertigen bedarf es folgender Überlegung: Die Ausbreitung von Funkwellen in der Luft beträgt etwa 2/3 der Lichtgeschwindigkeit, somit also ca. 200.000 km/s. Daher benötigen die Funkwellen zur Durchquerung von 200 m, einer Strecke, die eine Obergrenze für die Reichweite eines IEEE-Netzwerkes darstellt, ca. 1 µs, also erreichen die Funkwellen alle Empfänger innerhalb eines Radius von 200 m um den Sender herum mit einer zeitlichen Differenz von maximal 1 µs, so daß der physikalische Empfang auf den verschiedenen Stationen eine maximale Abweichung von 1 µs besitzt.

Die Annahme (W5) ist eine Abschätzung über die maximal auftretende Anzahl von aufeinanderfolgenden Fehlern im Netzwerk. Der Designer des Systems muß den Wert *OD* sorgfältig wählen, damit diese Annahme in der Praxis nicht verletzt wird. Sofern bekannt ist, daß die Kommunikation nicht in kurzer Folge Nachrichten versendet, sondern diese vielmehr mit größerem zeitlichen Abstand versendet, muß dieser Wert allerdings nicht besonders hoch gewählt werden. Man nutzt hier die Erfahrung, daß neben zufälligen Ausfällen einzelner Bits Fehler in den Nachrichten auf einem Netzwerk meistens in *Bursts* auftreten, d. h., innerhalb einer kurzen Zeitspanne treten gehäuft Fehler auf. Sofern die Länge der zu erwartenden Bursts wesentlich kleiner ist als das Intervall, nach dessen Ablauf die nächste Nachricht versendet wird, muß *OD* nicht besonders groß gewählt werden.

Weiterhin wird davon ausgegangen, daß die Uhren korrekt sind, d. h. weder eine Driftrate aufweisen, welche größer ist als spezifiziert, noch falsche Zählerstände besitzen. Diese Annahmen können in der Realität zumeist getroffen werden, da Uhren sellten fehlerhaft sind. Bei Uhren, bei denen diese Annahme nicht gerechtfertigt zu sein scheint, kann durch zusätzliche Maßnahmen die Wahrscheinlichkeit eines Fehlers minimiert werden. Hierzu werden Redundanz-Mechanismen benutzt: Auf dem Master können mehrere Uhren benutzt werden, deren Ausgaben verglichen werden. Aus der Drift zwischen zwei Uhren ergeben sich allerdings, daß die Uhren aufgrund der unterschiedlichen Drift unterschiedliche Werte für die physikalischen Uhren besäßen. Das Protokoll muß sich darum kümmern, aus diesen Uhren eine Uhrzeit zu erzeugen, die nicht von Fehlern einzelner Uhren gestört werden kann. Eine mögliche Hardware-Lösung hierfür findet sich in [DHM 73]. Dort wird beschrieben, wie durch den Einsatz von $3f+1$ vielen Taktgeneratoren Uhrensignale erzeugt werden können. Aus diesen Uhrensignalen kann dann durch *Konditionierer* genannte Einheiten ein lokales Signal für einen Zähler generiert werden, der als Wert der Uhr benutzt werden kann. Benutzt man mehrere Konditionierer, dann können mehrere Uhren implementiert werden. Das interessante an dem Ansatz ist, daß selbst bei Ausfall von f vielen Taktgeneratoren die lokalen Signale phasengekoppelt sind, d. h., sie besitzen die gleiche Frequenz und damit dieselbe Driftrate von der realen Zeit. Da auch die Zähler redundant ausgelegt werden können, kann damit eine zentrale Uhr erzeugt werden, welche den Ausfall von bis zu f vielen Taktgeneratoren und bis zu einer konfigurierbaren Anzahl von Zählern tolerieren kann. Man beachte, daß die Anzahl $3f+1$ zur Tolerierung von f Fehlern ein allgemeines Ergebnis ist, diese Anzahl ist immer notwendig (in Bezug auf Uhren siehe [DHS 86], allgemein siehe [LSP 82]).

Im folgenden wird davon ausgegangen, daß eine Uhr existiert, die sich korrekt verhält.

4.2 Das IEEE-Protokoll

Das Uhrensynchronisationsprotokoll nach dem IEEE 802.11-Standard ist ein einfaches Master-/Slave-Protokoll, d. h., es existiert eine ausgezeichnete Station, der Master, der in regelmäßigen Abständen die Uhren der anderen Stationen an seine eigene anpassen läßt.

Im Infrastrukturnetzwerk übernimmt der AP die Rolle des Masters: In regelmäßigen Abständen versendet der Master seine aktuelle Uhrzeit, jeder Slave übernimmt diese Uhrzeit unmittelbar beim Empfang und korrigiert seine eigene Uhrzeit entsprechend (s. Abbildung 4-1). Zum Zeitpunkt t_1 ermittelt der Master einen Zeitstempel für die aktuelle Uhrzeit und versendet ihn zum Zeitpunkt t_2 physikalisch über das Netzwerk. Zum Zeitpunkt t_3 kommt die Nachricht physikalisch bei einem Slave an, dieser stellt seine Uhrzeit entsprechend der Zeit des Masters um (t_4).

Da der Master das Versenden vorverarbeitet und die Slaves den Empfang nachbearbeiten müssen, ergibt sich eine Zeit zwischen dem Senden und dem Empfang der Nachricht von $\lambda = t_4 - t_1$. Diese Zeitspanne muß bekannt und möglichst konstant sein, damit das Protokoll eine hohe Präzision erreichen kann; d. h. diese Zeitspanne den zeitkritischen Pfad, der alle relevanten Aktionen enthält, deren zeitlicher Ablauf bekannt sein muß um eine präzise Synchronisation zu erhalten.

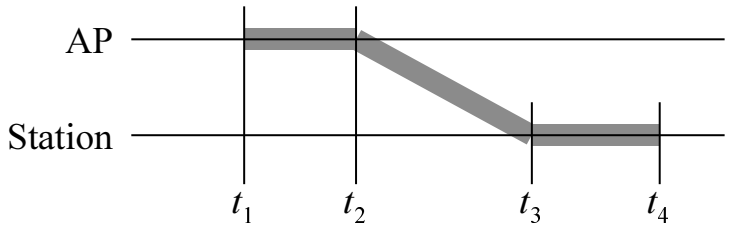


Abbildung 4-1: *Zeitlicher Ablauf beim IEEE-Protokoll*

Damit diese Zeitspanne möglichst exakt bestimmt werden kann, wird beim Protokoll nach dem IEEE-Standard nicht der Zeitstempel versendet, der zum Zeitpunkt t_1 aktuell ist, vielmehr wird vom Master der Zeitpunkt t_2 vorausberechnet. Der Gedanke dabei ist, daß sowohl der Master als auch die Slaves lediglich lokale Verzögerungen berücksichtigen müssen, um den Zeitpunkt t_2 zu bestimmen. Dennoch wird dabei weitgehend vernachläßigt, daß sowohl t_2-t_1 als auch t_4-t_3 sehr variabel sein können. Immerhin wird die Zeitinformation im Beacon gesendet, wodurch das Frame Vorrang auf dem Medium hat, so daß die Varianz von t_2-t_1 gemindert wird.

Die Variabilität von t_2-t_1 ergibt sich primär aus dem Grund, daß die Medienzugriffszeit λ_{access} ein Teil dieser Zeitspanne ist. Ein Frame, welches auf dem Medium versendet wird während die Synchronisationsnachricht zum Versenden bereit wird, verzögert die Synchronisationsnachricht um ihre restliche Übertragungsdauer. Die Zeit kann im schlimmsten Fall mehrere Millisekunden betragen und ist damit sehr hoch. Hier ergibt sich eine extrem große Ungenauigkeit, wobei die konkrete Implementierung auf eine durch den Standard nicht spezifizierte Weise dafür sorgen muß, daß der Einfluß der Medienzugriffszeit gering gehalten wird.

Der Slave seinerseits muß zum Zeitpunkt t_4 auf den Zeitpunkt t_2 zurückrechnen. Auch er muß hierbei (unter der Annahme, daß t_3-t_2 konstant ist und für alle Stationen den gleichen Wert hat) lediglich lokale Verzögerungen berücksichtigen. Allerdings ist die Zeitspanne t_4 - t_3 recht lang und unterliegt einer potentiell großen Schwankung.

Die Schwankungen aller einzelnen Zeiten summieren sich auf und führen dazu, daß das IEEE-Protokoll lediglich eine Präzision erreicht, die verbessert werden kann.

Damit die Schwankungen der Medienzugriffszeit λ_{access} erheblich gemindert werden, wird beim IEEE-Protokoll die aktuelle Zeit im Beacon versendet. Dieser hat durch die Verwendung eines kurzen Inter-Frame-Spaces Priorität auf dem Medium, wodurch der Effekt der Arbitrierung des Mediums gemindert wird. Allerdings reicht diese Maßnahme nur eingeschränkt aus, um die potentiell große Varianz von λ_{access} zu minimieren.

4.3 Ein verbessertes Protokoll

Das in dieser Arbeit entwickelte Protokoll basiert auf dem GS-Protokoll, wie es für den CAN-Bus entwickelt wurde (s. Abschnitt 3.7 „Das GS-Protokoll“). Wie beim GS-Protokoll und dem Protokoll nach dem IEEE-Standard handelt es sich auch hier um ein Master-/Slave-Protokoll, d. h., eine Station, der Master, versendet in regelmäßigen Abständen ihre Zeitinformation, die von den anderen Stationen, den Slaves, übernommen wird. Das verbesserte Protokoll besteht aus einem erweiterten GS-Protokoll, welches den Eigenschaften eines Funknetzwerkes gerecht wird. Dabei wird dieses erweiterte GS-Protokoll so ergänzt, daß es als eine Erweiterung des IEEE 802.11-Standards aufgefaßt werden kann. Insbesondere sollen standardkonforme Implementierungen so erweiterbar sein, daß sie das Protokoll ausführen können, andererseits sollen standardkonforme Implementierungen in der Lage sein, auch ohne Kenntnis der Erweiterung mit erweiteren Komponenten zusammenzuarbeiten.

Die Präsentation des Protokolls erfolgt in zwei Teilen. Zuerst wird im Abschnitt 4.3.1 „Austausch der Zeitinformationen“ das Prinzip präsentiert, mit dem die Informationen über die Uhren zwischen dem Master und den Slaves ausgetauscht werden. Im darauffolgenden Abschnitt 4.3.2 „Ratenanpassung im verbesserten Protokoll“ wird dann erläutert, wie diese Information genutzt wird, um eine Ratenanpassung zu erreichen, die speziell auf die Struktur des Protokolls zugeschnitten ist.

4.3.1 Austausch der Zeitinformationen

Das Protokoll geht in Runden vor: Eine besondere Station, der Master, versendet in regelmäßigen Intervallen eine Nachricht als Broadcast oder als Teil des Beacons, wodurch die Annahmen aus Abschnitt 4.1 „Systemmodell“ angewendet werden können. Insbesondere wird die versendete Synchronisationsnachricht mit Priorität auf dem Medium versendet. Diese versendete Nachricht hat folgenden Inhalt, wobei *m* ein noch festzulegender Protokollparameter ist:

1. einen Zähler, der die aktuelle Rundennummer *N* angibt;

2. *m* viele Zeitstempel.

Diese Nachricht trägt eine weitere Information, die abhängig von dem Empfänger der Nachricht ist, nämlich den Zeitpunkt, an dem sie von dem Empfänger empfangen wurde. Auch der Master selber gilt in diesem Zusammenhang als Empfänger. Dies bedeutet, daß jede Nachricht für den Empfang einer Synchronisationsnachricht einen Zeitstempel ermittelt und sich diesen merkt. Dieser Vorgang wird im Folgenden *Indikation* genannt. Wenn die aktuelle Runde die Nummer *N* besitzt, dann werden jeweils alle die Zeitstempel gepuffert und als gültig erklärt, die in den Runden *N* bis *N*-*m* ermittelt wurden, sofern die entsprechende Nachricht empfangen wurde, anderenfalls wird der entsprechende Zeitstempel als ungültig markiert. Beim Start des Protokolls werden dabei alle Zeitstempel als ungültig markiert.

Der Zähler wird vor dem Versenden einer Synchronisationsnachricht um eins erhöht. Er dient der Koordination zwischen dem Sender und dem Empfänger, damit sich beide einig sind, in welcher Runde sie sich befinden. Anhand dieser Information kann der Empfänger wiederum erkennen, ob, und wenn ja, wie viele Nachrichten er nicht empfangen hat und deshalb die entsprechenden Zeitstempel als ungültig markieren. Der Zeitstempel, der mit der aktuellen Runde korrespondiert, wird hingegen als gültig markiert.

Die *m* Zeitstempel in der versendeten Nachricht korrespondieren mit den *m* vorherigen Runden (s. Abbildung 4-2). Besitzt die aktuelle Runde die Nummer *N*, dann enthält die Nachricht die Zeitstempel der Runden *N*-1 bis *N*-*m*, die der Master für den Erhalt der Synchronisationsnachricht der entsprechenden Runde ermittelt hat. Auf jedem Slave liegen die Zeitstempel für die Nachrichten der entsprechenden Runden vor, sofern der Slave diese Nachrichten empfangen hat. Nun sucht der Slave ein Paar von korrespondierenden Zeitstempeln, d. h. ein Paar von Zeitstempeln des Masters und des Slaves, die zu der gleichen Runde gehören. Wenn mehrere solcher Paare existieren, dann wird das Paar gewählt, welches die numerisch größte Rundennummer *N′* besitzt.

Obwohl die Uhrzeit mit der Kenntnis der Differenz zwischen der Uhrzeit der Masters und der Uhrzeit des Slaves der Runde $N' \in \{N\text{-m}, N\text{-}1\}$ angepaßt werden könnte, indem die Uhrzeit um eben diese Differenz vor- oder zurückgestellt wird, wird der Prozeß des Anpassens der Uhrzeit etwas aufwendiger gestellt, damit keine Zeitsprünge auftreten, eine höhere Präzision erreicht werden kann und vor allem berücksichtigt wird, daß diese

Differenz zu dem Zeitpunkt, an dem sie bekannt wird, veraltet ist. Dieser Prozeß wird im folgenden Abschnitt 4.3.2 „Ratenanpassung im verbesserten Protokoll“ genauer beschrieben.

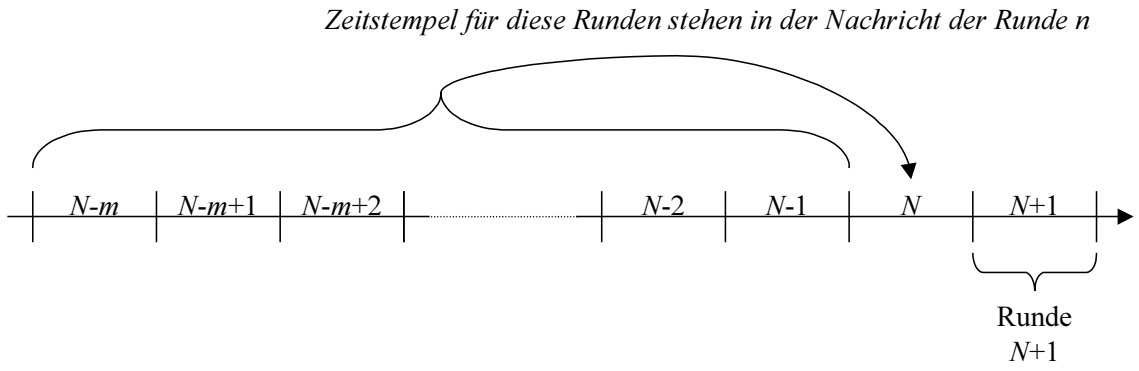


Abbildung 4-2: *In einer Synchronisationsnachricht enthaltene Zeitstempel*

4.3.2 Ratenanpassung im verbesserten Protokoll

Wie schon in Kapitel 3.3 „Werte- und Ratenanpassung“ ausgeführt, gibt es für die Anpassung der Uhren bei einem Synchronisationsprotokoll prinzipiell zwei Möglichkeiten: Die Uhr kann bei jeder Anpassung umgestellt werden, man spricht dann von einer Werteanpassung, oder man beschleunigt die Uhr oder bremst sie ab, je nach den erhaltenen Informationen, in diesem Fall spricht man von einer Ratenanpassung. Eine Ratenanpassung ist einer Werteanpassung vorzuziehen, da bei ihr keine Zeitsprünge entstehen und die Möglichkeit besteht, die Drift der Uhren auszugleichen, so daß in der Folge keine großen Zeitkorrekturen mehr notwendig sind.

Beim Uhrensynchronisationsprotokoll nach dem IEEE-Standard erfolgt ein unmittelbares Umsetzen der Uhrzeit des Slaves auf die Uhrzeit des Masters bei Empfang der Zeitinformation, also wird dort eine Werteanpassung vorgenommen. Bei dem hier entwickelten Protokoll wird stattdessen eine Ratenanpassung vorgenommen, die der speziellen Struktur des Protokolls angepaßt ist.

Bei dem hier benutzten Uhrenmodell existiert in jedem Knoten p eine physikalische Uhr $PC^{(p)}$, welche in regelmäßigen Abständen ihren Zähler erhöht. Die Frequenz und der Wert der physikalischen Uhr lassen sich nicht anpassen. Da dieses Modell meistens eine große Entsprechung mit der Realität aufweist, wird es auch in der Literatur häufig benutzt.

Aufbauend auf dieser physikalischen Uhr $PC^{(p)}$ wird dann für jeden Knoten eine virtuelle Uhr $VC^{(p)}$ definiert, welche eine Abbildung von Werten der physikalischen Uhr in virtuelle Zeiten für diesen Knoten ist. Die virtuellen Zeiten sind das Ergebnis der Uhrensynchronisation, sie bilden die globale Zeit, mit der andere Protokolle und die Ap-

plikationen arbeiten. Da der Master *ms* seine Uhrzeit nie umstellt, ist seine physikalische Uhr $PC^{(ms)}$ gleich seiner virtuellen Uhr $VC^{(ms)}$, daher wird diese auch nur als die Uhr des Masters bezeichnet.

Damit man von einer globalen Zeit sprechen kann, muß das Uhrensynchronisationsprotokoll dafür sorgen, daß die virtuellen Zeiten aller beteiligten Knoten zu jedem Zeitpunkt einen Offset von höchstens der Präzision Π besitzen.

Da die Uhrensynchronisation in Runden abläuft, ist es zweckmäßig, die virtuelle Uhr jeweils über dem Synchronisationsintervall der *i*-ten Synchronisationsrunde zu betrachten; wir bezeichnen diesen Teil der virtuellen Uhr durch $VC_i^{(p)} := VC^{(p)} \mid [t_{i-1}, t_i]$, wobei t_{i-1} den Wert der physikalischen Uhr des Knotens *p* am Anfang und t_i am Ende der *i*-ten Synchronisationsrunde darstellen.

Um die Anzahl der benutzten Indizes zu verringern, beziehen sich im Folgenden alle Angaben zu virtuellen und physikalischen Uhren auf den Knoten *p*, sofern nicht explizit etwas anderes angegeben ist.

Zur Erläuterung der Ratenanpassung dient die Abbildung 4-3. Sie stellt die Sicht eines einzelnen Prozesses dar; in ihr werden virtuelle Zeiten eines Prozesses gegen die Werte seiner physikalischen Uhr aufgetragen und der Graph der virtuellen Uhr dargestellt. Gleichzeitig ist als G_{master} der Verlauf der Uhr des Masters in Abhängigkeit von den Werten der physikalischen Uhr des Slaves dargestellt.

Zum Zeitpunkt t_{i-1} wird durch das Protokoll erkannt, daß die virtuelle Uhr zu schnell läuft. Gleichzeitig erfährt es, wie schnell die Uhr statt dessen verlaufen müßte und welchen Wert sie gerade besitzen sollte, also die Gerade G_{master}, die den Verlauf der Uhr des Masters in Abhängigkeit von der eigenen physikalischen Uhr modelliert - wie es dies erfährt, soll zuerst einmal nicht interessieren.

Zunächst werden die Parameter der Geraden G_{master} als neue Parameter der virtuellen Uhr benutzt – allerdings nicht direkt, da sonst ein Zeitsprung erfolgen würde. Stattdessen wird vorher, bis zu einem Zeitpunkt t_i^*, die Uhr stärker abgebremst, damit sie sich der Geraden G_{master} langsam annähert.

Man erkennt also, daß die virtuelle Uhr VC_i über dem Intervall $[t_{i-1},t_i]$ folgendermaßen definiert ist:

$$VC_i(t) = \begin{cases} VC_i^*(t) & \text{falls } t \in [t_{i-1},t_i^*] \\ VC_i'(t) & \text{falls } t \in]t_i^*,t_i] \end{cases}$$

wobei $t_i^* \in [t_{i-1}, t_i]$ ein ansonsten beliebiger Wert der physikalischen Uhr des Slaves ist.

Damit die Ratenanpassung ihren Zweck erfüllt, keine Zeitsprünge zu erzeugen, wird die virtuelle Uhr VC_i^* so gewählt, daß sie stetig die virtuelle Uhr VC_{i-1} fortsetzt und durch die virtuelle Uhr VC_i' stetig fortgesetzt wird. Aus Gründen der Einfachheit wird VC_i^*

linear gewählt. Mit diesen Anforderungen ist also VC_i^* bei gegebenen Uhren $VC_{i\text{-}1}$ und VC_i' sowie gegebenem t_i^* eindeutig bestimmt. Auf die Frage, wie t_i^* gewählt wird, wird im Laufe des Abschnittes noch eingegangen.

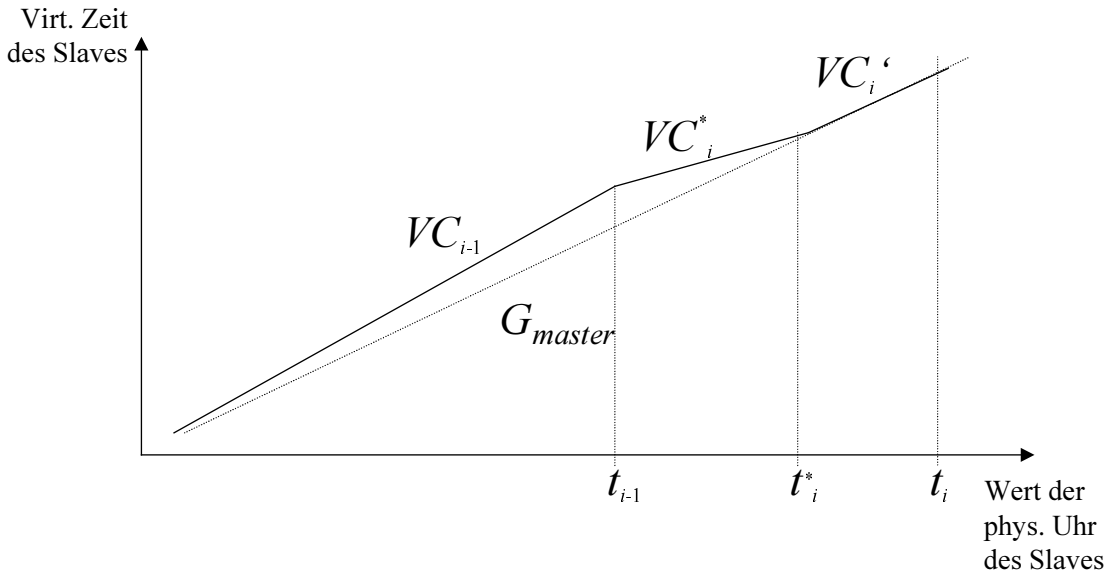


Abbildung 4-3: *Prinzip der Ratenanpassung*

Es bleibt also die Frage offen, wie ein Slave die Gerade G_{master} und damit VC_i' bestimmen kann. Hierzu fallen Überlegungen über die durch das Synchronisationsprotokoll gelieferten Informationen an. Bei unserem Synchronisationsprotokoll ist die Information über den Unterschied der Zeiten zu dem Zeitpunkt, an dem die Uhrzeit gesetzt werden soll, schon veraltet, da die Information über Differenzen der Zeiten mindestens ein Synchronisationsintervall alt ist. Im Folgenden wird ein Verfahren vorgestellt, welches diesem Umstand Rechnung trägt.

Man betrachte die Abbildung 4-4, die zeigt, wie nach Abschluß der i-ten Synchronisationsrunde der weitere Verlauf der virtuellen Uhr VC_i, welche aus VC_i^* und VC_i' besteht, bestimmt wird.

In der Abbildung ist wieder die virtuelle Uhrzeit gegenüber den Werten der physikalischen Uhr des Knotens aufgezeichnet. Die Zeitpunkte t_j und t_{j+k}, gemessen mittels der physikalischen Uhr des Knotens, geben die Zeitpunkte an, an denen der Master seine eigene Nachricht empfangen und die Zeitstempel nahm, die er dann in den Synchronisationsnachrichten versendet hat. Man beachte, daß t_j und t_{j+k} physikalische Zeiten auf dem Slave p angeben. Der Slave kann diese Zeitpunkte beobachten, da der Master die Zeitstempel unmittelbar nach Empfang der eigenen Synchronisationsnachricht nimmt. Es handelt sich dabei um alte Synchronisationszeitpunkte; dabei ist $j+k < i\text{-}1$ und $j,k \geq 0$.

Die (virtuellen) Zeiten T_j bzw. T_{j+k} geben den Stand der Uhr des Masters zu den entsprechenden Zeitpunkten t_j bzw. t_{j+k} an. T_j und T_{j+k} werden den versendeten Synchronisationsnachrichten entnommen und geben damit also „Sollzeiten“ an, d. h., im Idealfall hätte die virtuelle Uhr des Slaves p zum Zeitpunkt t_j den Wert T_j und zum Zeitpunkt t_{j+k} den Wert T_{j+k} gehabt.

Man kann diese Information nutzen um zu bestimmen, wie sich die Uhr des Masters voraussichtlich weiterentwickeln wird: Nach dem aufgestellten Fehlermodell verläuft die Uhr des Masters linear und die Drift zwischen der physikalischen Uhr von p und der des Masters ist konstant. Legt man demnach eine Gerade G_{master} durch die Punkte $P'(t_j, T_j)$ und $P(t_{j+k}, T_{j+k})$, so hat man eine gute Annäherung an das Verhalten der Uhr des Masters. Dadurch läßt sich VC_i' durch $VC_i' := G_{master} \mid [t_i^*, t_i]$ definieren. Nach dem oben Gesagten und der Kenntnis von $VC_{i-1}'(t_{i-1})$ ist damit auch schon VC_i^* eindeutig definiert.

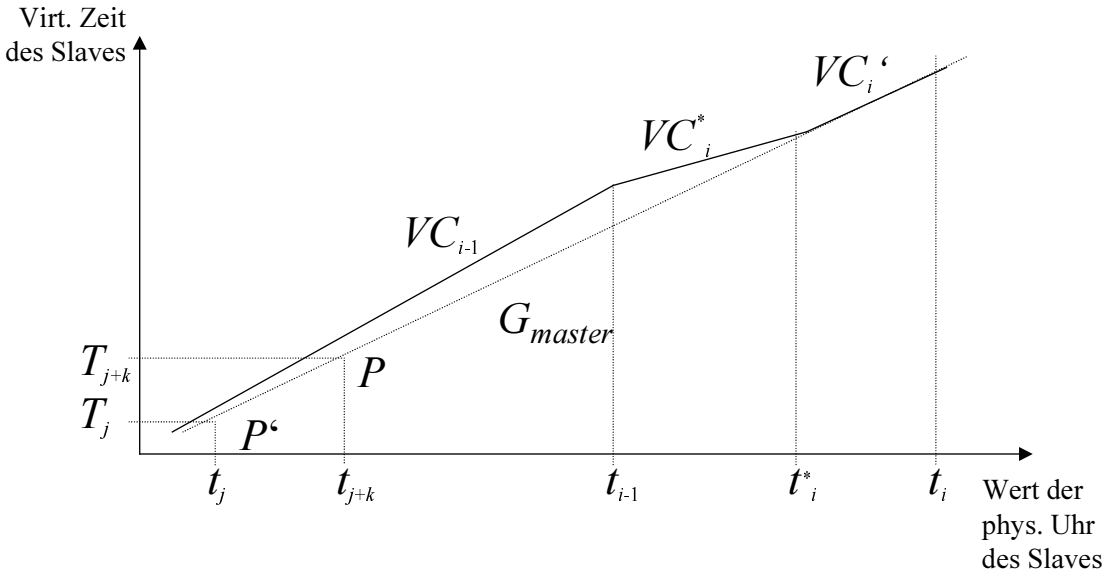


Abbildung 4-4: *Ratenanpassung, Bestimmung der Master-Geraden*

Wegen der auftretenden Meß- und Rundungsfehler sind die Uhren nach dem Zeitpunkt t_i^* nicht synchron, so daß nach Beendigung des nächsten Synchronisationsintervalles diese Anpassung erneut vorgenommen werden muß. Im Idealfall, in dem die Uhr des Masters und die virtuellen Uhren der Slaves nicht mehr driften würden, keine Meßfehler der Uhr des Masters und keine Rundungsfehler beim Rechnen aufträten, wären nach dem Zeitpunkt t_i^* die Uhren von Master und Slave synchron.

Es wurden noch keine Betrachtungen über die Lage der Zeitpunkte t_j und t_{j+k} gemacht. Es liegt nahe, daß diese aufeinanderfolgenden Synchronisationsrunden entstammen, wodurch sie zeitlich recht nah beieinander stehen. Damit wird die Berechnung aber nu-

merisch instabil, da Meßfehler stark ins Gewicht fallen. Wählt man diese Zeitpunkte so, daß sie weiter voneinander entfernt sind, dann wird die Berechnung von G_{master} numerisch stabiler, wie noch gezeigt wird, weil die Fehler bei der Bestimmung der Zeitpunkte T_j und T_{j+k} sowie von t_j und t_{j+k} selbst nicht so sehr ins Gewicht fallen.

Man beachte, daß das Verfahren zur Bestimmung von G_{master} weiter verallgemeinert werden kann, indem mehr als nur 2 Punkte zur Bestimmung der Geraden benutzt werden: Dann kann durch die Punkte eine Ausgleichsgerade gelegt werden, wodurch statistisch die Schwankungen besser ausgeglichen werden können. Während diese Implementierung mittels einer Ausgleichsgeraden den durchschnittlichen Fall verbessern kann, trifft dies auf den worst-case-Fall nicht zu: Wählt man den ältesten und den neuesten Punkt bei beiden Verfahren gleich, so kann jeder Fall, der mit 2 Punkten auftreten kann, auch mit mehr Punkten auftreten, da die Zwischenpunkte auf der Geraden durch die zwei äußeren Punkte liegen können. Umgekehrt gilt die Folgerung nicht, also kann der worst-case-Fall bei der Wahl einer Ausgleichsgeraden schlechter sein als mittels 2 Punkten.

Es soll noch eine Bemerkung zur Wahl von t_i^* gemacht werden: Oben wurde gesagt, daß $t_i^* \in [t_{i-1}, t_i]$ gewählt wird. Allerdings gibt es keinen Grund, die Wahl von t_i^* nicht lediglich durch die Vorgabe $t_i^* \geq t_{i-1}$ zu beschränken. In der obigen Erläuterung ist nirgendwo die Bedingung $t_i^* \leq t_i$ benötigt worden; sie erleichtert lediglich die Präsentation des Verfahrens. Insbesondere sollte man sich vor Augen führen, daß in der tatsächlichen Implementierung der Zeitpunkt t_i gar nicht bekannt ist, da man gar nicht vorhersehen kann, ob die nächste Synchronisationsnachricht überhaupt beim Knoten p ankommt.

Je nach Wahl des Zeitpunktes t_i^* sind demnach folgende Fälle möglich:

1. $t_{i-1} = t_i^*$: In diesem Fall kann VC_i^* nur als vertikale Gerade definiert werden, dies resultiert demnach also als Sprung der Uhrzeit zum Synchronisationszeitpunkt. Tatsächlich wird also in diesem Falle eine Werteanpassung vorgenommen.

2. $t_{i-1} < t_i^* < t_i$: In diesem Fall wird das Verfahren genau wie oben beschrieben ablaufen: Die alte Uhr VC'_{i-1} wird durch VC_i^* ersetzt, welche wiederum durch VC_i ersetzt wird.

3. $t_i^* \geq t_i$: In diesem Fall kommt nur die korrigierende Uhr VC_i^* zum Einsatz, sofern keine Nachrichtenverluste auftreten. Es gilt die Beziehung $VC_i = VC_i^*$ auf dem Intervall $[t_{i-1}, t_i]$.

Die Betrachtung der Fälle ist deshalb wichtig, weil dadurch, daß für die Fälle 2 und 3 keine Besonderheiten bei der Berechnung beachtet werden müssen, es ermöglicht wird, bei der Wahl von t_i^* keine Annahmen über den nächsten Synchronisationszeitpunkt t_i machen zu müssen. Tatsächlich kann es sinnvoll sein, t_i^* hinter den erwarteten nächsten

Synchronisationszeitpunkt t_i zu setzen, falls sonst die Uhr zu stark abgebremst oder beschleunigt werden müßte. Insbesondere ist auch eine dynamische Wahl von t_i^* denkbar, gerade um die Zeitdauer einer virtuellen Sekunde nicht zu sehr von der physikalischen Sekunde abweichen zu lassen.

Wird t_i^* größer als der (erwartete) Wert von t_i gewählt, so ist es zweckmäßig, trotzdem auch die Uhr VC_i' zu bestimmen und sich den Wert von t_i^* zu merken: Sollte nämlich zum (erwarteten) Zeitpunkt t_i keine Synchronisation erfolgen (beispielsweise, weil eine Nachricht verloren gegangen ist), so kann zum Zeitpunkt t_i^* automatisch auf die Uhr VC_i' umgeschaltet werden. Dies ist deshalb notwendig, da es möglich sein soll, mehrere aufeinanderfolgende Nachrichtenverluste zu tolerieren. Eine rapide abbremsende oder beschleunigende Uhr VC_i^* würde die virtuelle Uhr VC des Slaves sonst weit von der Uhr des Masters wegführen.

Satz 1: („Berechnung der Uhrenparameter“)
Die Parameter der Uhren $VC_i^*(t) = m_i^* t + b_i^*$ *und* $VC_i'(t) = m_i t + b_i$ *bestimmen sich mittels* $m_i = \frac{T_{j+k} - T_j}{t_{j+k} - t_j}$, $b_i = T_{j+k} - m_i t_{j+k}$, $m_i^* = \frac{T_i^* - T_{i-1}}{t_i^* - t_{i-1}}$ *und* $b_i^* = T_{i-1} - m_i t_{i-1}$.

Beweis:
Zur Verdeutlichung dient die Abbildung 4-5, die gegenüber der Abbildung 4-4 ergänzt wurde.

Es sollen die Uhren $VC_i^*(t) = m_i^* t + b_i^*$ und $VC_i'(t) = m_i t + b_i$ errechnet werden. Hierzu sind folgende Werte gegeben: t_{j+k}, t_j, T_{j+k}, T_j, t_i^*, t_{i-1} und T_{i-1} (die aktuelle virtuelle Zeit).

Zuerst wird $G_{master} = m_i t + b_i$ berechnet (womit auch VC_i' bestimmt ist).

Dann errechnet sich m_i durch $m_i = \frac{T_{j+k} - T_j}{t_{j+k} - t_j}$, somit b_i durch $b_i = T_{j+k} - m_i t_{j+k}$.

Zur Bestimmung von $VC_i^*(t)$ wird noch die Kenntnis von T_i^* benötigt. Dies läßt sich einfach durch Einsetzen lösen: $T_i^* = VC_i'(t_i^*) = m_i t_i^* + b_i$.

Damit läßt sich m_i^* durch $m_i^* = \frac{T_i^* - T_{i-1}}{t_i^* - t_{i-1}}$ und b_i^* durch $b_i^* = T_{i-1} - m_i t_{i-1}$ bestimmen. ■

Man erkennt an den Formeln zur Berechnung der Steigung m_i, daß die Zeitpunkte t_{j+k} und t_j möglichst weit voneinander entfernt sein sollten, damit die sogenannte *numerische Stabilität* verbessert wird. Dies bedeutet sinngemäß, daß große Werte für die Differenz $\alpha = t_{j+k} - t_j$ Fehler der gemessenen Werte für t_{j+k}, t_j, T_{j+k} und T_j abschwächen, da durch α dividiert wird, kleine Werte hingegen Meßfehler verstärken.

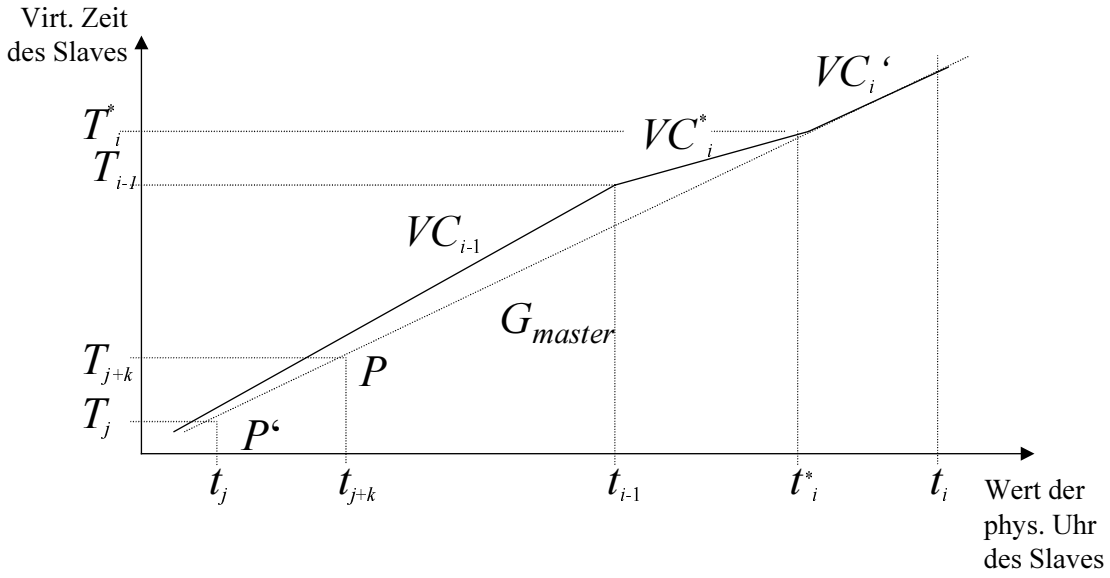


Abbildung 4-5: *Ratenanpassung, Berechnung der Uhrengeraden VC_i‘, VC^*_i*

4.3.3 Diskussion des Verfahrens

Der wesentliche Gedanke des Protokolls ist es, den zeitkritischen Pfad zu minimieren (s. Abbildung 4-6). Dies wird dadurch erreicht, daß ein physikalisches Ereignis, der Empfang einer Nachricht (zum Zeitpunkt t_3), benutzt wird, um zum nächsten beobachtbaren Zeitpunkt (t_s) Zeitstempel auf dem Master und den Slaves zu ermitteln. Später können die Slaves zeitlich unkritisch ihre Zeitstempel mit dem des Masters vergleichen und ihre Uhren entsprechend anpassen.

Der zeitkritische Pfad beim GS-Protokoll ist also kürzer als beim IEEE-Protokoll. Die Verkürzung wird dadurch erreicht, daß die gesendete Nachricht nicht mehr einen Zeitstempel als Inhalt der Synchronisationsnachricht enthält, den der Slave unmittelbar übernehmen muß, wodurch er die komplette Laufzeit von t_1 bis t_4 berücksichtigen muß, sondern stattdessen der Empfang der Nachricht nur noch ein Zeichen für ihn ist, daß er einen Zeitstempel nehmen soll. Daher muß der Slave die Nachricht nicht mehr zu einem genauen Zeitpunkt erhalten, es ist nur wichtig, daß der Master und der Slave den Empfang gleichzeitig erkennen. Häufig ist es möglich festzustellen, *daß* eine Nachricht empfangen wurde, bevor die eigentliche Nachricht komplett vorliegt, d. h., bevor der Inhalt der Nachricht zur Verfügung steht. In diesem Fall liegt der Zeitpunkt t_s vor dem Zeitpunkt t_4, so daß nur noch die Zeit zwischen t_3 und t_s den zeitkritischen Pfad bildet.

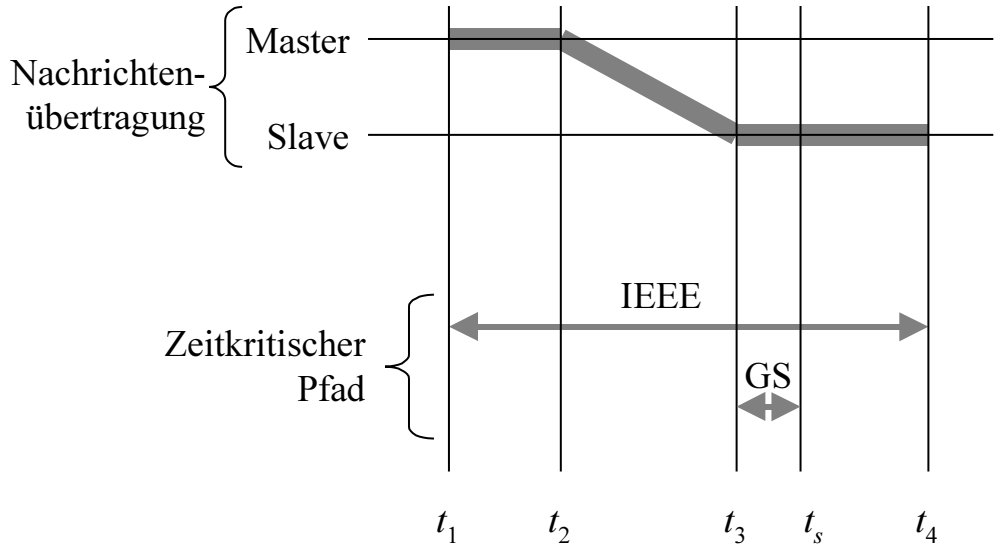


Abbildung 4-6: *Zeitkritische Pfade im Vergleich*

Im optimalen Fall würde der Zeitstempel zu dem Zeitpunkt ermittelt, an dem die Nachricht physikalisch empfangen wurde (t_3). Allerdings ist dieser Zeitpunkt nicht wirklich von dem Empfänger beobachtbar, aber zum nächstmöglichen Zeitpunkt t_s wird der Zeitstempel ermittelt. Lediglich diese kleine Zeitspanne ist kritisch im GS-Protokoll.

Als Master für das Synchronisationsprotokoll wird der AP benutzt, da er durch seine Aufgaben im Funknetzwerk schon eine Sonderstellung einnimmt: Er ist von seiner normalen Funktion her schon ein Master, und er bestimmt die Gruppe, die mit ihm assoziiert ist. In diese Gruppe schließt er alle Stationen ein, die in seinem Empfangsbereich liegen, sofern sie sich bei ihm anmelden. Wie schon in Abschnitt 2.3.1.1 „Ad-Hoc- und Infrastrukturnetzwerke“ ausgeführt wurde, bietet sich sein Gruppenbegriff als Begriff der Synchronisationsgruppe an. Da der AP stationär ist, kann er an einem strategischen Punkt positioniert werden, an dem die Uhrensynchronisation garantiert werden muß. Das Verhalten bei Wahl einer beliebigen mobilen Station wäre hingegen nicht oder zumindest schwer vorhersehbar.

Beim CAN-Bus existiert mit dem Draht, über welches die Übertragung stattfindet, ein abgeschirmtes Medium, daher ist dort die Wahrscheinlichkeit von Nachrichtenverlusten sehr gering. Dies trifft auf ein Funknetzwerk nicht zu, vielmehr treten hier Verluste häufig auf. Daher mußte das Protokoll die Möglichkeit erhalten, Verluste von Nachrichten tolerieren zu können. Dies erfolgt dadurch, daß mehrere Zeitstempel pro Synchronisationsnachricht versendet werden.

Für eine erfolgreiche Synchronisation muß ein Slave sowohl eine Indikation als auch den zugehörigen Zeitstempel vom Master erhalten, da er ansonsten kein Zeitstempelpaar, d. h. ein Paar von zu einer Indikation gehörenden Zeitstempeln des Masters und von sich selbst, findet und deshalb nicht feststellen kann, wie weit seine Uhr von der des

Masters entfernt ist. Damit dieses Zeitstempelpaar nicht aus der Indikation einer Nachricht und dem Zeitstempel der unmittelbar darauffolgenden Nachricht gebildet werden muß, wie es beim GS-Protokoll der Fall ist, was nachteilig in Bezug auf die Fehlertoleranz wäre, wird pro Nachricht mehr als nur ein einziger Zeitstempel versendet. Diesem Zweck dienen die m Zeitstempel, die in der Synchronisationsnachricht enthalten sind. Die m Zeitstempel, die in einer Runde N versendet werden, zeigen den Wert des Zeitstempels an, die der Master zu den Indikationen der Nachrichten der Runden N-m bis N-1 ermittelt hat. Hiermit kann der Verlust von m-1 vielen aufeinanderfolgenden Nachrichten toleriert werden: Wenn maximal m-1 viele aufeinanderfolgende Nachrichten verloren gehen können und die Nachricht einer Runde N empfangen wird, dann ist mindestens eine der Nachrichten der Runden N-m bis N-1 ebenfalls empfangen worden. Somit ist zum Zeitpunkt des Empfangs der Nachricht der Runde N Indikation und Zeitstempel für eine der Runden N-m bis N-1 vorhanden, wodurch eine Synchronisation möglich wird. Aus der Annahme (W5) ergibt sich, daß $m = OD+1$ gewählt wird.

Damit jede Station feststellen kann, ob – und wenn ja, welche – Nachrichten verloren gegangen sind und daher nicht zur Verfügung stehen, ist es notwendig, daß der Master die gesendeten Nachrichten durchnumeriert und diese Nummer mit jeder Nachricht mitsendet.

Damit muß für das Protokoll pro Synchronisationsrunde eine Nachricht mit m Zeitstempeln und einem Zähler übertragen werden. Bei einem Speicherbedarf von jeweils 8 Byte für jeden Zeitstempel und für den Zähler und m=10 ergibt dies einen Aufwand von 88 Byte pro Synchronisationsrunde. Dies stellt auf dem Funknetzwerk einen eher kleinen Aufwand dar, für den CAN-Bus wäre dies jedoch eine große Belastung, da seine möglichen Datenraten wesentlich niedriger sind, auch wenn beim originalen GS-Protokoll die Zeitstempel mit 4 Byte Größe nur halb so groß waren.

Damit das Protokoll angewendet werden kann, ist es notwendig, daß einige Bedingungen erfüllt sind. Diese Bedingungen sind (modifiziert nach [GS 94]):

1. Die gleiche Nachricht wird von zwei Empfängern[10], die sie erhalten, mit einer bekannten maximalen Zeitdifferenz empfangen.

2. Die Nachrichten des Zeitmasters müssen von allen Stationen der Synchronisationsgruppe empfangen werden können.

3. Die Verzögerung zwischen dem Empfang einer Synchronisationsnachricht und der Erzeugung des Zeitstempels auf jeder Station ist ihr bekannt und hat eine geringe Varianz.

4. Eine maximale Zeit zwischen dem erfolgreichen Versenden zweier aufeinanderfolgender Synchronisationsnachrichten kann garantiert werden.

---

[10] wobei der Sender auch als Empfänger zählt, da er den Empfang seiner eigenen Nachricht zeitstempelt

Die Bedingung 1 wird benötigt, um sicherzustellen, daß der Zeitpunkt t_3 auf allen Stationen gleichzeitig gesehen wird. Ohne Bedingung 2 ist es unmöglich, eine Synchronisation durchzuführen. Die Bedingung 3 ist notwendig, damit jede Station aus dem Zeitstempel, den sie zum Zeitpunkt t_s genommen hat, auf den Zeitpunkt t_3 zurückrechnen kann. Bedingung 4 schließlich wird gefordert, da ohne maximale Zeit zwischen dem Versenden von Nachrichten die Uhren der beteiligten Stationen in dieser Zeit beliebig weit voneinander driften können.

Es gilt nun nachzuweisen, daß die geforderten Bedingungen auf dem Funknetzwerk erfüllbar sind:

Die erste Voraussetzung, daß jede ausgesendete Nachricht von allen Stationen, die sie empfangen, ungefähr im selben Augenblick empfangen wird, wird vom Funknetzwerk erfüllt (W4). Innerhalb des Protokolls findet kein Routing statt, d. h., eine gesendete Nachricht wird nicht von einer Station aufgegriffen und weitergesendet, um die Erreichbarkeit oder Zuverlässigkeit zu erhöhen, noch wird eine Nachricht über verschiedene Wege versendet. Da keine Bestätigungen genutzt werden, versendet der AP eine Nachricht auch nicht zweimal (W3). Mit Hilfe von (W2) ergibt sich dann, daß jede Nachricht höchstens einmal empfangen werden kann. Jede Nachricht des Protokolls wird höchstens einmal empfangen, da jede Nachricht entweder von jeder Station in dem Moment, zu dem sie ausgesendet wurde, empfangen wird (unter Vernachläßigung der Verzögerung auf dem Medium), oder gar nicht. Da alle beteiligten Stationen die gleiche physikalische Nachricht sehen, liegt ihr Empfang innerhalb einer bekannten maximalen Zeitdifferenz.

Die zweite Voraussetzung kann garantiert werden, da der Gruppenbegriff gerade so gewählt wurde, daß alle diejenigen Stationen zu der Synchronisationsgruppe gehören, die sich im Empfangsbereich des APs befinden.

Die dritte Voraussetzung über die Erzeugung des Zeitstempels ist nicht abhängig von der Wahl des zugrundeliegenden Netzwerkes, d. h., sie ist unabhängig von CAN-Bus, Funknetzwerk oder anderen Medien. Vielmehr muß die spezifische Implementationsumgebung dies ermöglichen. Lösbar ist dies beispielsweise durch eine Implementierung in der Hardware oder in der Firmware einer Netzwerkkarte. Auf diesen Aspekt wird ausführlich in Kapitel 5 „Implementierung“ eingegangen.

Die vierte Voraussetzung über den maximalen Abstand zwischen erfolgreich gesendeten Synchronisationsnachrichten kann ebenfalls garantiert werden. Sie entspricht der Eigenschaft (W1) und wird dadurch erreicht, daß die Synchronisationsnachricht in dem Beacon-Frame gesendet wird, so wie es auch beim IEEE-Protokoll erfolgt. Da das Beacon-Frame Vorrang auf dem Medium hat, und andere Stationen den Zeitpunkt seiner Aussendung respektieren, kann diese Garantie gegeben werden. Alternativ kann die Synchronisationsnachricht auch anderweitig in einem Frame mit Priorität auf dem Medium vom AP ausgesandt werden, beispielsweise in der PCF.

4.4 Formale Protokollbeschreibung

Zur Analyse eines Protokolls oder Algorithmus ist es notwendig, daß das zu analysierende Protokoll in einer formalen Form vorliegt, damit nicht durch unpräzise oder interpretationsbedürftige Aussagen über das Protokoll die Analyse angreifbar wird. Daher wird in diesem Unterkapitel das bislang vorgestellte Protokoll als finiter Automat formalisiert, damit im nächsten Unterkapitel die Analyse erfolgen kann. Dabei wird die Formalisierung nur soweit getrieben, wie es für die Analyse notwendig ist.

Zuerst werden die verwendeten Darstellungsmittel beschrieben, die sich an die formelle Beschreibungssprache SDL (*specification and description language*, [ITU 92]) anlehnen (wobei nur die graphischen Mittel von SDL, also SDL/GR benutzt werden), hiernach werden die verwendeten Signale des Protokolls beschrieben, und zuletzt der endliche Automat des Protokolls angegeben.

4.4.1 Verwendete Darstellungsmittel

Für die formale Darstellung des Protokolls wird eine graphische Notation verwendet, die sich an SDL/GR anlehnt. Die Sprache SDL/GR wurde vor allem in Hinblick auf die Darstellung von Kommunikationsprotokollen als endlicher Automaten entwickelt; so benutzt auch der IEEE-Standard für Funknetzwerke ([IEEE 97]) diese Darstellungsform.

Das darzustellende System wird in SDL in Blöcke zerlegt, welche wiederum aus Prozessen bestehen. Da hier jeder Block aus genau einem Prozeß besteht, wird auf diese Unterscheidung nicht näher eingegangen, stattdessen werden die Begriffe Prozeß und Block synonym verwendet.

Blöcke kommunizieren in SDL/GR über Signalkanäle miteinander. Signalkanäle werden dargestellt durch Pfeile, wobei Pfeilspitzen angeben, in welcher Richtung die Kommunikation stattfindet, und Beschriftungen neben den Pfeilen angeben, welche Signale über den Kanal ausgetauscht werden können. Ein Beispiel für ein SDL-System, in dem Blöcke und Signalkanäle dargestellt sind, zeigt die Abbildung 4-8.

Die Prozesse in SDL werden als (erweiterte) endliche Automaten dargestellt. Ein endlicher Automat besteht aus Zuständen und Zustandsübergängen, welche von eingehenden Signalen gesteuert werden und ausgehende Signale erzeugen.

Als Beispiel betrachte man die Abbildung 4-7. Der oberste unbeschriftete Kasten ist der sogenannte Startzustand, in dem sich der Automat beim Starten befindet. Aus diesem Zustand wechselt der Automat unmittelbar in den Zustand ZSTND1.

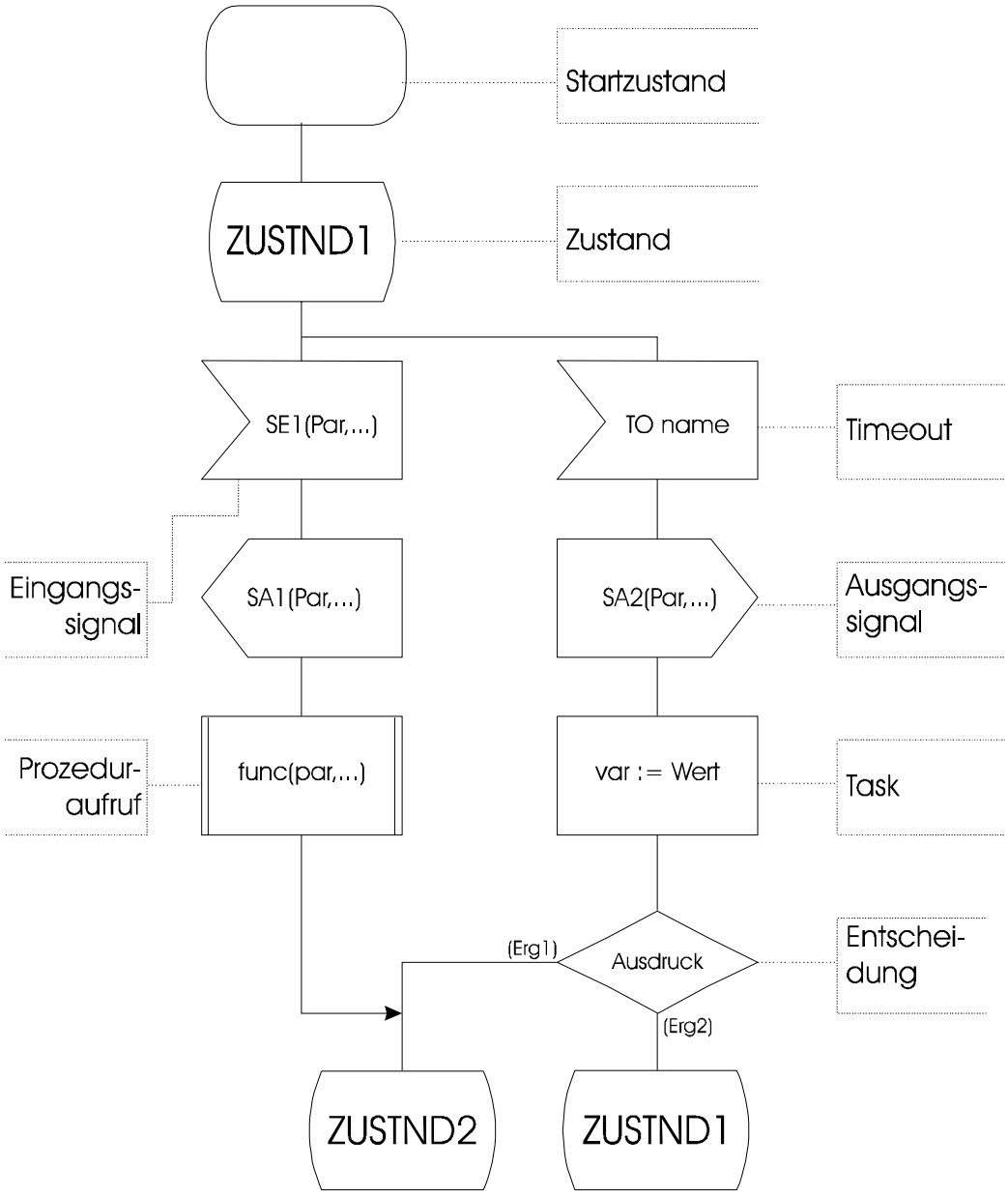


Abbildung 4-7: *Die benutzten Elemente von SDL/GR*

Beim Erhalt von Eingangssignalen wird ein sogenannter Zustandsübergang ausgeführt, der durch eine Linie (von oben nach unten) dargestellt wird. Abhängig vom erhaltenen Signal können unterschiedliche Zustandsübergänge ausgeführt werden, daher liegt an von Zuständen ausgehenden Kanten jeweils ein Eingangssignalblock, der angibt, auf welches Signal durch diesen Zustandswechsel reagiert wird. In der Beispielabbildung reagiert der Automat im Zustand ZUSTND1 beispielsweise auf das Eingangssignal *SE1*(), indem der linke Pfad gewählt wird, und auf das Ablaufen des Timeouts mit dem Namen *name*, indem der rechte Pfad gewählt wird. Damit dieser Timeout nach Ablauf

des Intervalles *INT* einmalig abläuft, muß er mittels der Anweisung „*set*(*now*+*INT*, *name*) “ gesetzt werden.

Ein Zustandswechsel mündet immer in einem Zustand, der auch der Zustand sein kann, aus dem gerade reagiert wurde. Bevor ein neuer Zustand eingenommen wird, können auf dem Weg Ausgangssignale generiert (*SA1*(), *SA2*()), Aufgaben (*Tasks*) ausgeführt werden (im Beispiel die Zuweisung *var* := *Wert*) oder Funktionen aufgerufen werden (*func*()). Weiterhin kann die weitere Ausführung von Bedingungen abhängig gemacht werden, wobei im Beispiel von der Entscheidung *Ausdruck* aus der linke Pfad gewählt wird, sofern *Ausdruck* = *Erg1* gilt, und der rechte Pfad, wenn *Ausdruck* = *Erg2* gilt.

Man beachte, daß bei Eingangs- und Ausgangssignalblöcken die Spitzen links oder rechts am Block angeordnet sein können. Hiermit wird häufig unterschieden, mit wem kommuniziert wird. Beispielsweise wird bei der Beschreibung des Prozesses Medium ein Pfeil auf der linken Seite genutzt, um eine Kommunikation mit dem Master darzustellen, ein Pfeil auf der rechten Seite hingegen, um eine Kommunikation mit einem Slave darzustellen.

4.4.2 Struktur des Protokolls

Das Protokoll besteht aus drei Blöcken, die miteinander interagieren und damit die Synchronisation erst ermöglichen. Abbildung 4-8 zeigt die Struktur.

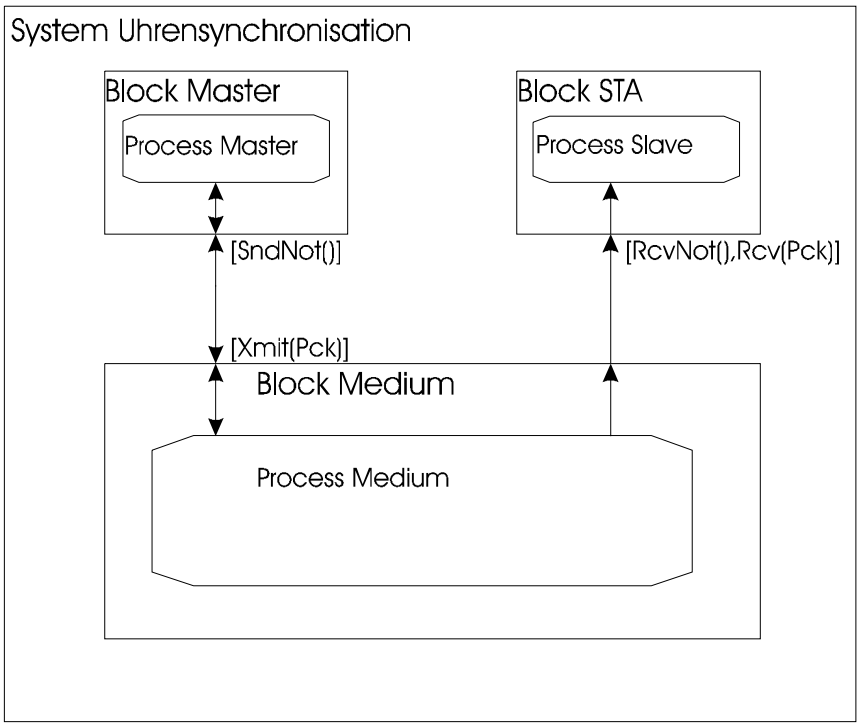


Abbildung 4-8: *Protokollaufbau*

Man erkennt drei Blöcke: Den AP-Block (der Master), den STA-Block (die Slaves) und den Medium-Block. Zwischen den Blöcken existieren Signalkanäle, über welche Informationen ausgetauscht werden können. Dabei ist der Signalkanal zwischen den Blöcken AP und Medium bidirektional, während Signale auf dem Signalkanal zwischen den Blöcken Medium und STA nur vom Block Medium ausgesendet werden können, da bei diesem Master-/Slave-Protokoll der Slave nur Informationen erhält, selber also keine versendet.

Natürlich gibt es innerhalb des Protokolls keinen Block Medium; vielmehr dient dieser der Modellierung der Vorgaben des Funkmediums.

Die Signale, die über die Signalkanäle zwischen den Blöcken ausgetauscht werden, müssen bezüglich ihrer Semantik definiert werden. Dies soll nun erfolgen.

Da die Struktur *Pck* sowohl in den Blöcken AP und STA als auch von den Signalkanälen benutzt wird, soll diese vorab erklärt werden. Es gilt $n := OD+1$, wobei *OD* der Annahme (W5) (s. Seite 50) entnommen wird.

Pck (*packet*) Ein Paket *Pck* besteht aus folgenden Elementen:

- *Ts[0..n-1]* (*timestamp*) Ein Vektor als Ringpuffer für *n* viele Zeitstempel; dabei ist *n* ein Protokollparameter.
- *Val[0..n-1]* (*valid*) Ein Vektor der angibt, ob der korrespondierende Zeitstempel gültig ist;

  d. h., $Val[i] = true$ genau dann wenn $Ts[i]$ ein gültiger Zeitstempel ist.
- *Seq* (*sequence no.*) Sequenznummer, die vom AP für jede versendete Nachricht um eins erhöht wird.

Folgende Signale werden über die Signalkanäle übertragen:

Xmit(pck) (*transmit*) Mit diesem Signal überträgt der AP ein Paket an das Medium bzw. das Medium ein Paket an eine Station STA. Dabei besteht das Paket *pck* aus mehreren Elementen; zur Detail-Struktur: siehe oben.

SndNot() (*send notification*) Mit diesem Signal gibt das Medium dem Master bekannt, daß eine Nachricht empfangen wurde.

RcvNot() (*receive notification*) Mit diesem Signal gibt das Medium den Slaves bekannt, daß eine Nachricht empfangen wurde.

Rcv(pck) (*receive*) Mit diesem Signal wird ein empfangenes Paket an den Empfänger übergeben. Dieses Signal kommt immer und ausschließlich nach *RcvNot*().

Als erstes wird der Prozeß *Medium* erläutert, dessen Struktur Abbildung 4-9 entnommen werden kann. Wie schon erwähnt, ist er nicht wirklich als Automat vorhanden, vielmehr soll er die Eigenschaften des Mediums modellieren.

Man beachte in der Abbildung, daß bei eingehenden oder ausgehenden Signalen anhand der Seite, auf der die Einkerbungen vorhanden sind, unterschieden werden kann, mit welchem Automaten kommuniziert wird: Eine Einkerbung auf der linken Seite der Zeichen beschreibt die Kommunikation mit dem Master, entsprechend eine Einkerbung auf der rechten Seite die Kommunikation mit den Slaves.

Man erkennt, daß nach dem Start der Automat unmittelbar in den Zustand IDLE übergeht. In diesem Zustand wartet er darauf, daß der Master per *Xmit*(*Pck*) ein Paket aussendet. Nach dem Empfang dieses Signales und einer Wartezeit, die die Verzögerungen vor dem Aussenden der Nachricht modelliert (λ_{access}), erhält der Master eine Bestätigung des Aussendens (*SndNot*()) sowie, sofern das Paket nicht verloren geht, auch der Slave eine Bestätigung des Empfanges (*RcvNot*()). Nach einer weiteren Wartezeit, die die Verzögerungen beim Empfang modellieren, erhält der Empfänger das Paket (*Rcv*(*Pck*)).

4.4.3 Der Master-Automat

Der Automat, welcher den Master beschreibt, wird innerhalb des APs eingesetzt. Er sorgt dafür, daß die Synchronisationsnachrichten in regelmäßigen Abständen versendet werden.

Der Master-Automat besitzt folgendes Attribut:

SndPck (*send packet*) Das zu sendende Paket; zur Detail-Struktur: siehe oben.

Der erweiterte Automat des Masters kann der Abbildung 4-10 entnommen werden.

Der Automat startet im Zustand INIT, aus dem heraus er sich initialisiert. Man erkennt, daß zuerst alle Zeitstempel, die versendet werden sollen, als ungültig markiert werden. Dies erfolgt in der Prozedur *clrAllTs*(*SndPck*) (*clear all timestamps*). Diese Funktion, die nicht näher ausgeführt wird, sorgt dafür, daß nach ihrer Beendigung für ihr Argument *Pck* gilt: $\forall i \in \{0,...,n\text{-}1\}: Pck.Val[i] = \text{false}$.

Als nächstes wird das Timeout „Send“ gesetzt, und zwar so, daß es nach *INT* Zeiteinheiten abläuft, wobei *INT* ein Protokollparameter ist. Nachdem die Sequenznummer des zu sendenden Paketes mit Null initialisiert wurde, geht der Automat in den Zustand IDLE über.

Der Rest des Automaten, der die eigentliche Funktionalität enthält, besteht aus den zwei Zuständen IDLE und WAITSEND.

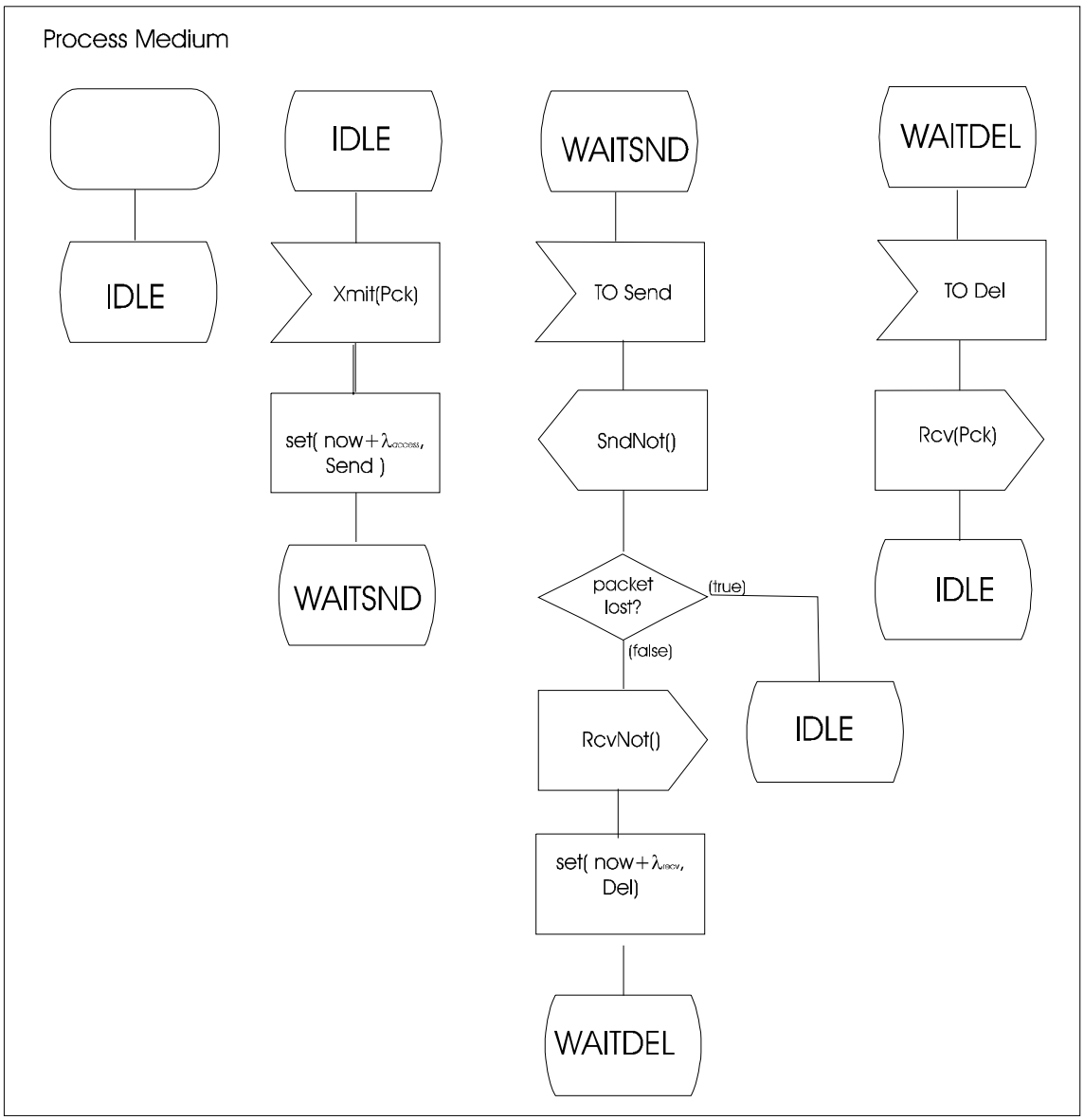


Abbildung 4-9: *Der Prozeß Medium*

Die Funktionsweise ist wie folgt: Im Zustand IDLE wird auf das Timeout „Send“ gewartet. Sobald dieses abgelaufen ist, hat die nächste Synchronisationsrunde begonnen und eine neue Synchronisationsnachricht muß gesendet werden. Vorher wird das Timeout allerdings neu gesetzt, damit es pünktlich für die nächste Synchronisationsrunde ablaufen kann.

Zum Versenden der Synchronisationsnachricht wird die Sequenznummer des zu sendenden Paketes erhöht und das Paket mittels *Xmit*(*SndPck*) abgesendet.

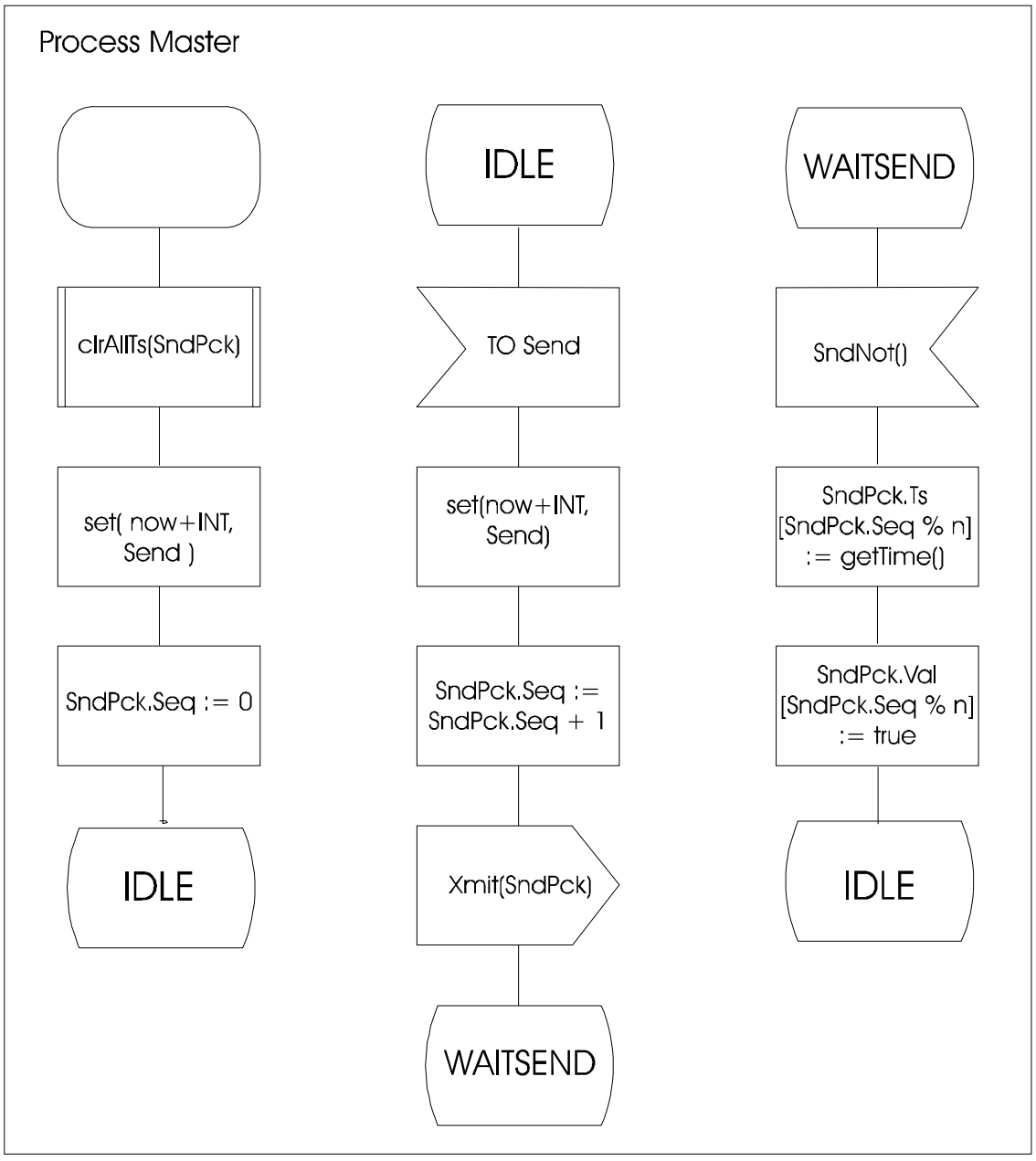


Abbildung 4-10: *Der Prozeß Master*

Der Automat geht dann in den Zustand WAITSEND über, in dem er auf die Bestätigung des Versendens des Paketes wartet. Sobald diese Bestätigung eintritt, wird die aktuelle Zeit in das zu versendende Paket kopiert und der Zeitstempel als gültig markiert. Zum Schluß geht der Automat in den Zustand IDLE über.

4.4.4 Der Slave-Automat

Der Automat, welcher den Slave beschreibt, wird in allen Stationen mit Ausnahme des APs eingesetzt. Dieser Automat nimmt die ausgesendeten Synchronisationsnachrichten des APs an und verarbeitet diese gemäß des Protokolls.

Der Slave-Automat besitzt folgende Attribute:

RcvPck (*received packet*) Das empfangene Paket; zur Detail-Struktur: siehe oben.

LocPck (*local packet*) enthält die Zeitstempel, welche die Station lokal genommen hat, um sie mit den Zeitstempeln des empfangenen Paketes vergleichen zu können; zur Detail-Struktur: siehe oben.

Synch (*synchronized*) = true: Der Slave ist zur Zeit synchronisiert.

NotTs (*notification timestamp*) Der letzte genommene Zeitstempel (für temporäre Speicherung).

Der erweiterte Automat des Slaves kann der Abbildung 4-11 und der Abbildung 4-12 entnommen werden. Auch hier beginnt der Automat im Zustand INIT. Ähnlich wie beim Master werden zuerst alle Zeitstempel durch die Prozedur *clrAllTs*() als ungültig markiert (es handelt sich hierbei um die gleiche *clrAllTs*()-Funktion, wie sie schon beim Master-Automaten beschrieben wurde). Als nächstes wird vermerkt, daß diese Station nicht synchronisiert ist (*Synch = false*) und die Sequenznummer des nächsten erwarteten Paketes auf 1 initialisiert (*LocPck.Seq* := *1*), bevor in den Zustand IDLE übergegangen wird.

Der Arbeitsteil des Slave-Automaten besteht wie auch der Arbeitsteil des Masters aus zwei Zuständen, IDLE und WAITRECV (Wait Receive).

Im Zustand IDLE wartet der Automat auf den Erhalt eines Signals, welches angibt, daß eine Nachricht empfangen wurde (*RcvNot*()). Bei Erhalt dieses Signals wird ein Zeitstempel für die aktuelle Zeit in der Variablen *NotTs* abgespeichert und der Automat begibt sich in den Zustand WAITRECV.

In dem Zustand WAITRECV wartet der Automat auf den Erhalt des empfangenen Paketes (*Rcv*(*RcvPck*)). Zuerst wird getestet, ob die Sequenznummer des empfangenen Paketes diejenige ist, welche erwartet wurde (Variable *LocPck.Seq*). Ist dies nicht der Fall, so werden die den nicht empfangenen Paketen entsprechenden lokalen Zeitstempel als ungültig markiert (*clrTs*()) und der Wert für die nächste zu erwartende Sequenznummer korrigiert.

In beiden Fällen schließt sich nun die Anpassung der Uhrzeit an (*AdjClk*(*RcvPck*)). Hiernach wird der Zeitstempel der aktuellen Nachricht in *LocPck* abgespeichert und der

Zeitstempel als gültig markiert, bevor der Wert für die nächste zu erwartende Sequenznummer inkrementiert wird und der Automat wieder in den Zustand IDLE übergeht.

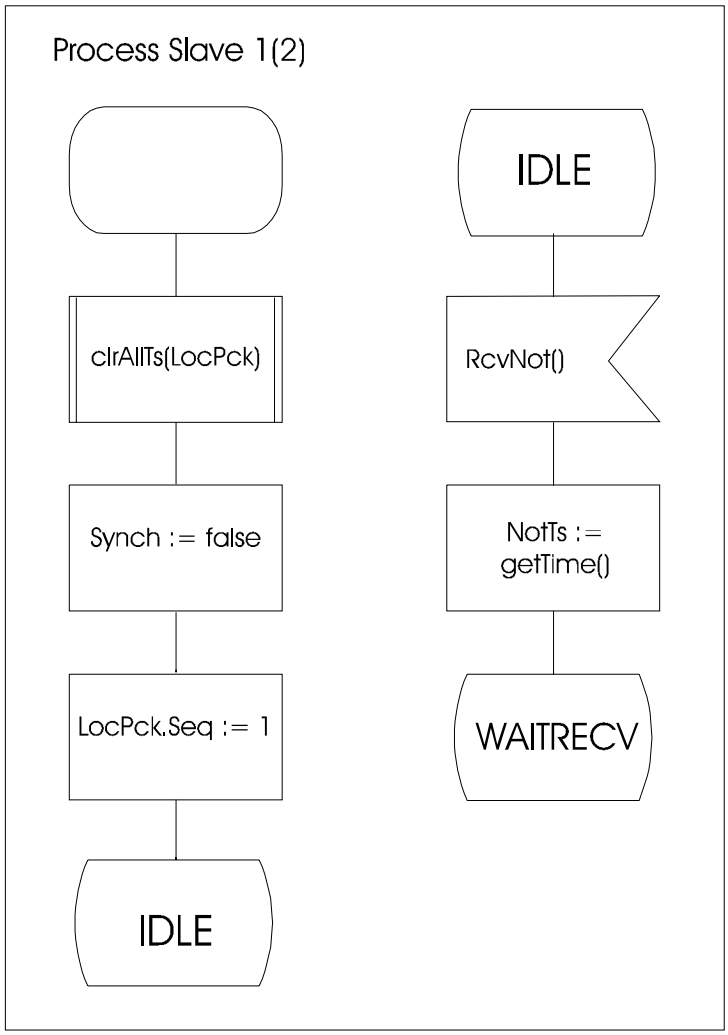


Abbildung 4-11: *Der Prozeß Slave, Teil 1*

Um den Automaten komplett zu beschreiben, fehlen noch die Spezifikationen für *clrTs*() und *AdjClk*().

Die Funktion *clrTs*(*start*, *end*) sorgt dafür, daß bei ihrer Beendigung für das Paket *LocPck* gilt: $\forall i \in \{ start,...,end\text{-}1\}$: $LocPck.Val[i \% n]$ = false, womit die Zeitstempel für die erwarteten, aber nicht erhaltenen Nachrichten gelöscht werden.

Die Funktion *AdjClk*() paßt die virtuelle Uhr des Slaves an die Uhr des Masters an. Dabei wird im Prinzip so vorgegangen, wie es schon in Abschnitt 4.3.2 „Ratenanpassung“ beschrieben wurde.

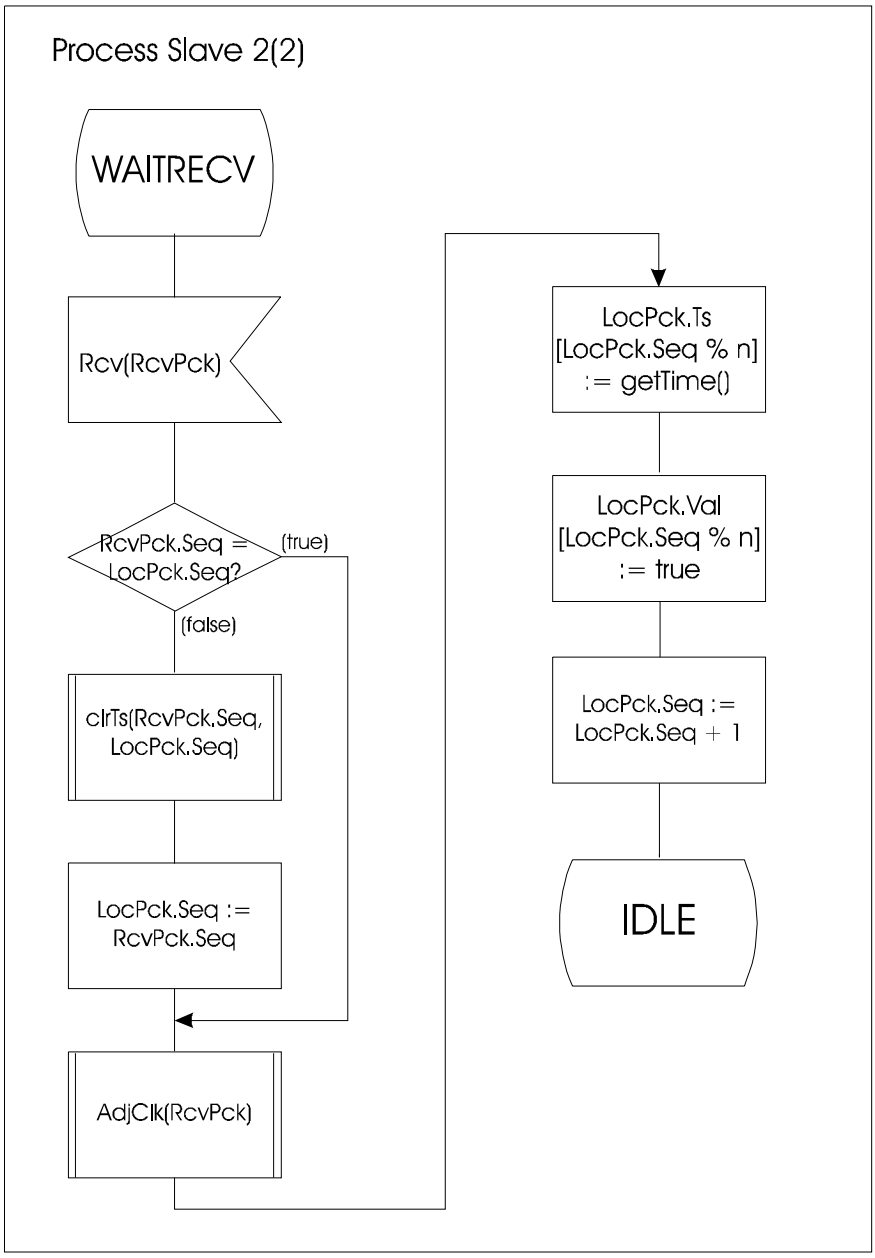


Abbildung 4-12: *Der Prozeß Slave, Teil 2*

Diese Funktion soll anstatt mit SDL/GR in einem Pseudo-Code erklärt werden. Dabei werden folgende Hilfsfunktionen benutzt, die eine Warteschlange implementieren, die statisch eine maximale Größe hat:

A = q.get() Liefert das erste Element derWarteschlange *q* nach *A* zurück; dieses Element wird *nicht* entfernt.

q.remove() Entfernt des oberste Element der Warteschlange *q*.

q.add(*A*) Ergänzt das Element *A* am Ende der Warteschlange *q*.

N = *q.length*() Liefert die Anzahl der Elemente in der Warteschlange *q* zurück.

Mit Hilfe dieser Funktionen kann die Funktionsweise von *AdjClk*() wie in der Abbildung 4-13 gezeigt beschrieben werden, wobei eine an die Programmiersprache C angelehnte Syntax benutzt wird.

Abbildung 4-13: *Die Funktion AdjClk()*

Die Funktion ist folgendermaßen strukturiert: Zu Beginn wird mittels *getFirstPair*() ein Paar von korrespondierenden Zeitstempeln in *LocPck* und *RcvPck* gesucht, d. h. es werden zwei Zeitstempel gesucht, wobei eines vom Master und das anderen vom Slave stammt und die Zeitstempel zur gleichen Indikation gehören. Dieses wird dann, sofern eines gefunden wird, in der Struktur *TimestampsNow* zurückgeliefert (Zeile 13).

Die weitere Verarbeitung erfolgt nur dann, wenn ein Paar gefunden wurde (Zeile 14). Wie später noch gezeigt wird, wird außer in der Initialisierungsphase immer ein Paar gefunden, sofern die Fehlerannahmen nicht verletzt werden.

Wurde ein Paar gefunden, ist die Schlange *queue* mit älteren Paaren aber noch leer, dann befindet sich das Protokoll in der Initialisierungsphase und führt eine Werteanpassung durch (Zeile 18); die Funktion *calcCorrectionTerm*() berechnet hierbei lediglich die Differenz aus *TimestampsNow.master* - *TimestampsNow.slave* und korrigiert seine lokale Uhrzeit um diesen Betrag.

Ist hingegen die Schlange *queue* mit älteren Paaren von Zeitstempeln nicht leer (Zeile 22f), dann sind zwei Paare von Zeitstempeln vorhanden und die Ratenanpassung erfolgt in der Funktion *calcCorrectionClock*() entsprechend der in Abschnitt 4.3.2 „Ratenanpassung“ hergeleiteten Formeln.

Hiernach wird die Schlange *queue* verwaltet, wobei das aktuelle Zeitstempelpaar in die Warteschlange aufgenommen wird (Zeile 30). Falls die Schlange schon voll ist, wird vorher das erste Element entfernt (Zeilen 26ff).

Die Funktion *calcCorrectionTerm*() berechnet lediglich die Differenz der Zeitstempel des Masters und des Slaves, um daraus den neuen Korrekturwert für die virtuelle Uhr zu bestimmen. Die Funktion *getFirstPair*(*RcvPck*, *LocPck*) liefert ein Paar (x,y) zurück, wobei $(x,y) = (RcvPck.Ts[i] , LocPck.Ts[i])$ gilt und i sich aus der folgenden Bedingung bestimmt:

$$i = \max \{ j \in \mathbb{N}: j < RcvPck.Seq \wedge RcvPck.Val[j \% n] = true \wedge LocPck.Val[j \% n] = true \}$$

.

Dies bedeutet, daß (x,y) das neueste Paar von Zeitstempeln ist, welche auf dem Master und dem Slave zu der gleichen Indikation ermittelt wurden.

Die Implementierung der Funktion *calcCorrectionClock*() ergibt sich unmittelbar aus der Beschreibung der Berechnung im Abschnitt 4.3.2 „Ratenanpassung im verbesserten Protokoll“ und wird hier daher nicht näher erörtert.

Es existieren also insgesamt drei Orte, aus denen ein Slave Zeitstempel entnommen kann: Das Attribut *LocPck* enthält die Zeitstempel, welche der Slave selbst für die Indikationen ermittelt hat. Die Variable *RcvPck* enthält die Zeitstempel, die der Master für die Indikationen ermittelt hat. Mittels der Funktion *getFirstPair*() kann der Slave ein Paar von Zeitstempeln ermitteln, wobei jeweils ein Zeitstempel aus *LocPck* und einer

aus *RcvPck* stammt. Dieses Paar wird in der Variablen *TimeStampsNow* gespeichert entspricht dem Punkt $P(t_{j+k},T_{j+k})$ aus Abschnitt 4.3.2 „Ratenanpassung im verbesserten Protokoll“. Bei Beendigung von *AdjClk*() wird dieses Paar in der Schlange *queue* gespeichert. Wenn es bei einem späteren Aufruf von *AdjClk*() aus der Schlange wieder entfernt wird, dann wird es in *TimeStampsLast* gespeichert und entspricht dem Punkt $P(t_j,T_j)$ aus dem Abschnitt 4.3.2 „Ratenanpassung im verbesserten Protokoll“. Der Wert von k ergibt sich demnach aus der maximalen Länge der Warteschlange *queue*, sofern keine Nachrichtenverluste auftreten.

4.5 Analyse des Protokolls

In diesem Kapitel sollen die Eigenschaften des Protokolls untersucht werden. Im folgenden Abschnitt 4.5.1 „Korrektheit des Protokolls“ soll zuerst nachgewiesen werden, daß das Protokoll synchronisiert. Im darauffolgenden Abschnitt 4.5.2 „Erreichte Präzision“ wird dann ermittelt, welche Präzision die Synchronisation garantieren kann.

4.5.1 Korrektheit des Protokolls

In diesem Abschnitt soll gezeigt werden, daß das Protokoll eine Synchronisation erreicht (Satz 9). Hierzu werden drei Eigenschaften gezeigt: das Protokoll bricht nicht ab (Lemma 7), zwischen dem Aufruf von *AdjClk*() auf einem Slave liegt eine maximale, abschätzbare Zeit (Lemma 8) und beim Aufruf von *AdjClk*() auf dem Slave kann eine Synchronisation stattfinden (Lemma 6). Für den Beweis von Lemma 6 wiederum wird benötigt, welche Zeitstempel in einer Synchronisationsnachricht versendet werden (Lemma 2), daß jeder Slave und der Master wissen, in welcher Runde sie sich befinden (Korollar 3) und daß der Slave die Zeitstempel richtig gesammelt hat (Lemma 4). Diese Lemmata werden nun bewiesen:

Lemma 2: („Eine Synchronisationsnachricht enthält Zeitstempel der vorherigen OD vielen Runden“)
*Ruft der AP die Funktion Xmit() mit einem Paket SndPck auf, dessen Rundennummer SndPck.Seq gleich n ist, dann sind alle Zeitstempel der Runden n-OD-*1 *bis n-*1 *in der Nachricht enthalten. Dabei ist für Runden n' mit n' <* 1 *der zugehörige Zeitstempel als ungültig markiert.*

Beweis:
Zu Beginn des Protokolls werden alle Zeitstempel als ungültig markiert. Da die erste Runde die Rundennummer 1 hat, ist bei der ersten Runde die Aussage bewiesen.

Sei die Rundennummer n>1. In der Runde n-1 war die Aussage erfüllt, d. h., die Zeitstempel der Runden n-*OD*-2 bis n-2 waren in der Nachricht enthalten oder als ungültig markiert. Hiernach erhielt der Automat das Signal *SndNot*(), da (wegen der Priorität der Nachricht auf dem Medium) die Nachricht ausgesendet wurde, woraufhin der Zeitstempel, der unmittelbar nach *SndNot*() genommen wurde und damit mit der Nachricht der

Runde *n*-1 korrespondiert, in das Sendepaket geschrieben wurde. Hierbei ersetzte der Zeitstempel der Runde *n*-1 den Zeitstempel der Runde *n-OD*-2, da *n*-1 ≡ *n-OD*-2 (mod *OD*+1). Da Zeitstempel vom *AP* niemals ungültig erklärt werden, gilt damit die Behauptung. ■

Korollar 3: („einheitliche Sicht der Rundennummer von Slaves und Master")
Wenn ein Slave eine Nachricht des Masters empfängt, dann haben der Slave und der Master eine einheitliche Sicht über die Rundennummer, zu der die Nachricht gehört.

Beweis:
Dieser ist trivial, da jede Nachricht die Nummer der Runde, zu der sie gehört, enthält. ■

Lemma 4: („lokale Zeitstempel des Slaves enthalten Informationen über empfangene Synchronisationsnachrichten")
In einer Runde n, in der eine gültige Nachricht empfangen wird, gilt nach dem Empfang der Nachricht unmittelbar vor dem Ausführen von AdjClk() für alle Nachrichten der Runden n-1 bis n-OD-1:

Für jede erfolgreich empfangene Nachricht aus diesen Runden merkt sich der Slave in LocPck den korrekten Zeitstempel, der als gültig markiert ist. Für jede nicht empfangene Nachricht dieser Runden ist der Zeitstempel als ungültig markiert. Nachrichten von Runden n' mit n' < 1 gelten hierbei als nicht empfangen.

Beweis:
Empfange der Slave eine gültige Nachricht in der Runde *n*. Hat er vorher noch keine gültige Nachricht empfangen (weil das Protokoll erst gestartet wurde), dann sind durch die Initialisierung mit *clrAllTs*(*LocPck*) alle Zeitstempel als ungültig markiert, unabhängig davon, ob die Abfrage *RcvPck.Seq* = *LocPck.Seq* wahr oder falsch ist.

Damit ist die Behauptung in diesem Fall bewiesen. Man beachte, daß *LocPck.Seq* hiernach die Nummer der nächsten Runde, d. h. *n*+1, enthält.

Hat der Slave hingegen schon eine Nachricht empfangen, dann wird zuerst getestet, ob die Rundennummer *n* = *RcvPck.Seq* der empfangenen Synchronisationsnachricht gleich der Rundennummer *LocPck.Seq* ist, welche diejenige Runde ist, für die der Slave die nächste Nachricht erwartet. Ist dies der Fall, dann wird *AdjClk*() aufgerufen und hiernach der Zeitstempel der Runde *n-OD*-1 durch denjenigen der Runde *n* ersetzt.

Ist hingegen *RcvPck.Seq* ungleich *LocPck.Seq*, dann werden mit dem Aufruf der Funktion *clrTs*(*RcvPck.Seq*, *LocPck.Seq*) die Zeitstempel, die zwischen der letzten empfangenen Nachricht (der Runde *LocPck.Seq*-1) und dem aktuellen Zeitstempel (der Runde *n* = *RcvPck.Seq*) liegen, gelöscht, bevor *AdjClk*() aufgerufen wird und hiernach der Zeitstempel der Runde *n-OD*-1 durch denjenigen der Runde *n* ersetzt wird.

In beiden Fällen gibt nach dem Inkrementieren von *LocPck.Seq* diese Variable wieder die Nummer der Runde an, zu der der nächste Zeitstempel erwartet wird.

Da beim Aufruf von *clrTs*() genau diejenigen Zeitstempel als ungültig markiert werden, die zu Runden mit Rundennummern n' gehören, wobei $n' < n\text{-}OD\text{-}1$ gilt, andererseits jeder Zeitstempel der Runden mit Rundennummern $n' < n$ in *AdjClk*() unmittelbar nach dem Empfang korrekt gesetzt wurde und erst am Ende von *AdjClk*() nach $OD+1$ vielen Runden oder in *ClrTs*() nach mehr als $OD+1$ vielen Runden gelöscht wird, gilt die Behauptung. ■

Lemma 5: („der Aufruf von getFirstPair() ermittelt immer ein Paar korrespondierender Zeitstempel von Master und Slave“)
Der Aufruf der Funktion getFirstPair() in AdjClk() auf einem Slave findet immer ein neues Paar von korrespondierenden Zeitstempeln des Masters (aus RcvPck) und des Slaves (aus LocPck), sofern es sich nicht um den allerersten Aufruf von AdjClk() handelt.

Beweis:
Nach Lemma 2 enthält die vom Master empfangene Nachricht der Runde n alle Zeitstempel der Runden n-1 bis $n\text{-}OD$-1, sofern diese Runden Rundennummern > 0 haben. Laut Lemma 4 besitzt der Slave beim Aufruf von *AdjClk*() alle Zeitstempel für die Nachrichten der Runden n-1 bis $n\text{-}OD$-1, die er empfangen hat. Wegen des Fehlermodells (Annahme (W5), s. Seite 50) hat er davon mindestens eine Nachricht empfangen (s. Abbildung 4-14). Da laut Korollar 3 der Master und der Slave die Rundennummern eindeutig einander zuordnen können, gibt es mindestens ein Paar von korrespondierenden Zeitstempeln des Masters und des Slaves beim Aufruf von *AdjClk*(). Das Paar, welches die größte Rundennummer besitzt, wird dann von *getfirstPair*() ermittelt. Dieses Paar ist notwendigerweise ein neues, da als lokaler Zeitstemp der lokale Zeitstempel der Indikation beim letzten Aufruf von *AdjClk*() benutzt wird, und bei diesem letzten Aufruf von *AdjClk*() kein passender Zeitstempel des Masters vorhanden wird. ■

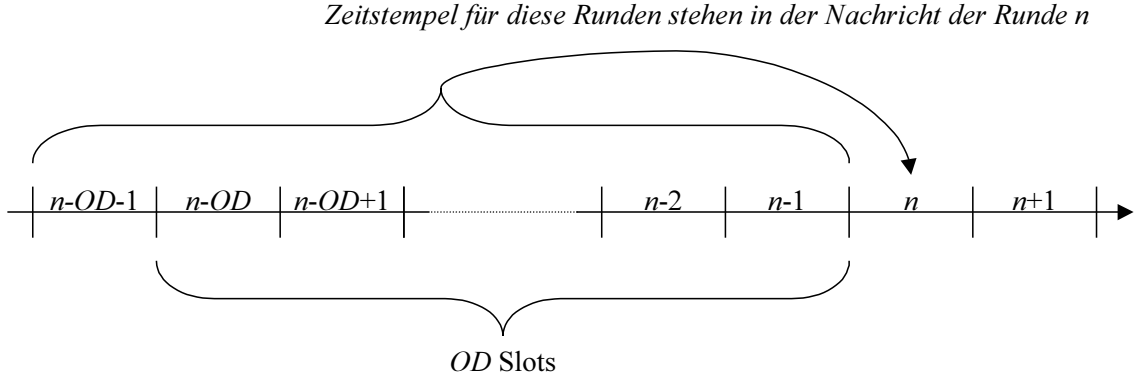


Abbildung 4-14: *In einer Synchronisationsnachricht enthaltene Zeitstempel*

Lemma 6: („Synchronisation bei AdjClk()“)
Mit Ausnahme des ersten Aufrufs kann beim Aufruf von AdjClk() eine Synchronisation erfolgen.

Beweis:
Beim ersten Aufruf von *AdjClk*() sind noch keine gültigen lokalen Zeitstempel vorhanden, also kann keine Synchronisation erfolgen. Beim zweiten Aufruf von *AdjClk*() existiert nach Lemma 5 ein Paar von gültigen Zeitstempeln, welches von *getFirstPair*() ermittelt wird. Da die Warteschlange *queue* leer ist, wird eine Werteanpassung vorgenommen (Zeile 18, Abbildung 4-13 und das Paar in die Warteschlange eingefügt. Ab dem 3. Aufruf von *AdjClk*() existiert ein Eintrag in der Warteschlange. Da laut Lemma 5 das Paar von Zeitstempeln, welches durch getFirstPair() ermittelt wird, ein neues ist, ist es ungleich dem Paar, welches aus der Warteschlange *queue* entnommen wurde, daher wird eine Ratenanpassung vorgenommen (Zeilen 22-24, Abbildung 4-13) entsprechend des in Abschnitt 4.3.2 „Ratenanpassung“ vorgestellten Verfahren. ■

Lemma 7: („Lebendigkeit des Protokolls“)
Der Master-Automat ruft alle INT Zeiteinheiten die Funktion Xmit() auf, wobei die Sequenznummer des gesendeten Paketes mit jedem Aufruf um eins erhöht wird.

Beweis:
Wie der Abbildung 4-10 entnommen werden kann, wird das Timeout Send bei Programmstart so gesetzt, daß es sofort abläuft. Hierdurch wird unmittelbar bei Programmstart das Paket mittels *Xmit*() gesendet. Vor jedem *Xmit*() wird das Timeout auf *INT* Zeiteinheiten nach der aktuellen Zeit gesetzt. Im Zustand IDLE wird das Timeout nach seinem Ablauf neu gesetzt, so daß es nach weiteren *INT* Zeiteinheiten abläuft, und das aktuelle Paket gesendet.

Da unmittelbar vor dem Aufruf von *Xmit*() die Sequenznummer erhöht wird, gilt der zweite Teil der Behauptung. Da keine weiteren Zustände oder Ausführungspfade existieren, gilt damit die gesamte Behauptung. ■

Aus diesen Lemma folgt, daß der Master unablässig seine Synchronisationsnachrichten mit korrekten Zeitstempeln versendet, wodurch ein unabsichtliches „Beenden“ des Protokolls von seiner Seite aus nicht möglich ist.

Lemma 8: („maximale Zeit zwischen dem Aufruf von AdjClk() auf einem Slave“)
Wenn keine Verletzung der Fehlerannahmen auftritt, ruft der Slave-Automat spätestens alle $(OD+2) \cdot$ *INT Zeiteinheiten die Funktion AdjClk() auf.*

Beweis:
Der Abbildung 4-12 kann entnommen werden, daß die Funktion *AdjClk*() immer dann aufgerufen wird, wenn ein Paket empfangen wurde. Es gilt also nachzuweisen, daß spätestens alle $(OD+2) \cdot INT$ Zeiteinheiten ein Paket des Masters beim Slave ankommt.

Man betrachte die Abbildung 4-15. Es sind die einzelnen Runden abgebildet, wobei jede Runde einen Zeitslot bildet, in dem eine Nachricht des Masters versendet wird. Aufgrund der Priorität auf dem Medium kann garantiert werden, daß der Master pro Runde eine Nachricht auf dem Medium versendet. Da er höchstens einen Sendeversuch

unternimmt, liegt innerhalb jedes Slots genau eine Synchronisationsnachricht des Masters.

Laut Annahme (W5) (s. Seite 50) liegen zwischen zwei erfolgreich empfangenen Nachrichten maximal *OD* viele, die nicht empfangen wurden. Dadurch wird immer nach endlich vielen Runden eine neue Nachricht empfangen.

Werde also in der Runde *n* eine Nachricht empfangen, und sei es nicht die erste. Im ungünstigsten Fall können dann die Nachrichten der Runden *n*-1 bis *n-OD* verlorengegangen sein, in der Runde *n-OD*-1 muß dann die letzte Nachricht empfangen worden sein.

Im schlechtesten Fall kann damit die zwischen der vorherigen Nachricht, die frühestens zu Beginn des Slots *n-OD*-1, also zum Zeitpunkt (*n*-OD-1)·*INT*, empfangen worden sein kann, und der aktuellen Nachricht, die spätestens am Ende des Slots *n*, also zum Zeitpunkt (*n*+1)·*INT*, empfangen worden sein muß, maximal die Zeit (*n*+1)·*INT* - (*n*-OD-1)·*INT* = (*OD*+2)·*INT* liegen[11]. ■

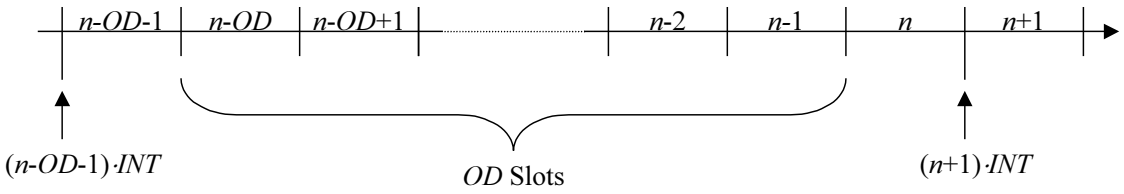


Abbildung 4-15: *Länge des Intervalls zwischen Synchronisationen*

Mit Hilfe der Lemmata ergibt sich folgender Satz:

Satz 9: („Protokoll erreicht Sychronisation“)
Das vorgestellte Uhrensynchronisationsprotokoll sorgt für eine Synchronisation aller Stationen der Synchronisationsgruppe.

Beweis:
Nach Lemma 7 sendet der AP unablässig Synchronisationsnachrichten aus. Nach Lemma 8 empfängt jede Station regelmäßig eine Synchronisationsnachricht und ruft *AdjClk*() auf. Nach Lemma 6 ist durch Aufruf von *AdjClk*() eine Anpassung der betreffenden Station an die Uhrzeit des Masters möglich. Also werden alle Stationen, bei denen das Fehlermodell nicht verletzt wird, synchronisiert. ■

[11] es wird hier die sinnvolle Annahme getroffen, daß die Länge des Intervalles *INT* wesentlich größer ist als die Zeit, die eine Nachricht maximal benötigt, um versendet zu werden.

4.5.2 Erreichte Präzision

In diesem Abschnitt soll die mittels des Protokolls erreichte Präzision, also die maximale Abweichung zweier Uhren zu einem Zeitpunkt, bestimmt werden. Abbildung 4-16 zeigt das Prinzip, wie sich die Präzision errechnet.

Die Abbildung zeigt, in welchem Bereich sich die Uhren der beteiligten Stationen voneinander wegbewegen können. Die Uhren werden zu einem Zeitpunkt t aneinander angepaßt und können sich eine Zeit lang voneinander fortbewegen, bis sie zu einem späteren Zeitpunkt t' wieder angepaßt werden.

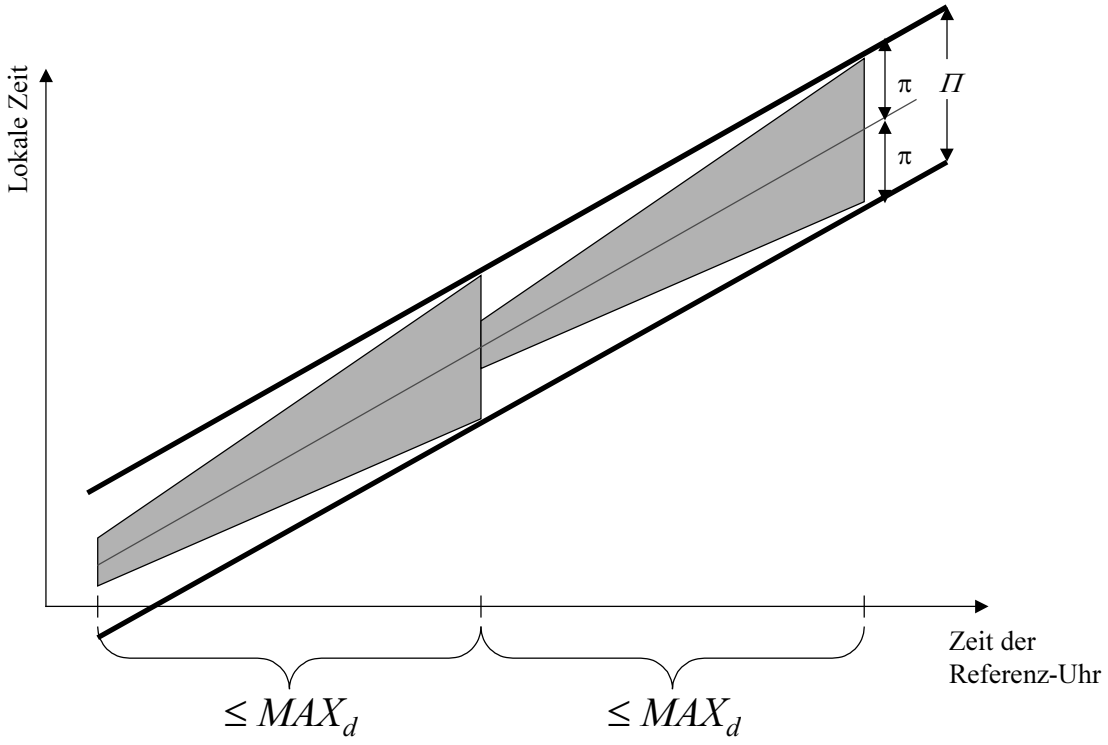


Abbildung 4-16: *Prinzip der Bestimmung der Präzision*

Man beachte, daß die Zeichnung die Verhältnisse beim hier entwickelten Protokoll vereinfacht. Durch die Möglichkeit von Nachrichtenverlusten ist es möglich, daß die Anpassungen der Uhrzeiten auf den verschiedenen Stationen zu unterschiedlichen Zeitpunkten erfolgen. Die Abbildung gibt also demnach nur das Verhältnis *einer* Slave-Uhr zu der Uhr des Masters wieder. Andererseits beziehen sich alle Synchronisationen immer auf die Uhr des Masters, d. h., aus der Annahme, daß eine Uhr sich maximal um einen Betrag π von der Uhr des Masters entfernt, ergibt sich, daß die Präzision des Systems von Uhren $\Pi = 2\pi$ beträgt. Daher kann die Präzision Π bestimmt werden mittels der Annahme, daß alle beteiligten Stationen zum gleichen Zeitpunkt synchronisiert werden, und damit ist die Berechnung nach der die Realität vereinfachenden Abbildung 4-16 ausreichend, um die Präzision des Protokolls zu berechnen.

Für die folgenden Betrachtungen werden folgende Vereinbarungen getroffen: Die Parameter δ_{master} und δ_{slave} geben die Ungenauigkeit an, mit der ein Zeitstempel auf dem Master bzw. dem Slave genommen werden kann; d. h. δ_{master} und δ_{slave} geben die Varianz des zeitkritischen Pfades an. Es wird δ als $\delta = \max\{\delta_{master}, \delta_{slave}\}$ definiert. Unter der Annahme, daß die Ungenauigkeit der Bestimmung der Zeitpunkte Null wäre, ist ΔT die Differenz der beiden Zeitstempel auf der Master-Uhr und Δt die Differenz der beiden Zeitstempel auf der Slave-Uhr, die zur Berechnung der Steigung der Geraden des Masters benutzt werden.

Der Beweis baut auf der Erkenntnis, daß eine Synchronisation stattfindet (Satz 9), auf. Nachdem die maximale Driftrate (Lemma 10) bestimmt wurde, wird die Präzision in Lemma 11 angegeben. In Satz 12 werden nicht ursprünglich gegebene Variablen des Ergebnisses aus Lemma 11 mittels des Ergebnisses aus Lemma 10 ersetzt.

Lemma 10: („Steigung der Uhrengeraden bei Ratenanpassung“)
Bei der Synchronisation mittels der Ratenanpassung hat die virtuelle Uhr eines Slaves maximal die Steigung $m_2 = \frac{\Delta T+\delta}{\Delta t-\delta}$ *und minimal die Steigung* $m_1 = \frac{\Delta T-\delta}{\Delta t+\delta}$ *wobei* Δt *die Differenz aus den zur Synchronisation benutzten Zeitpunkten* t_{j+k} *und* t_j *ist, sowie* ΔT *die Differenz der entsprechenden Zeitstempel des Masters* T_{j+k} *und* T_j.

Beweis:
Zur Abkürzung wird $t' = t_j$, $t = t_{j+k}$, $T' = T_j$ und $T = T_{j+k}$ gesetzt. Die Abbildung 4-17 zeigt die Berechnung der Steigung der Master-Geraden.

Seien $P(t,T)$ und $P'(t',T')$ die beiden Punkte, die bei der Ratenanpassung zur Bestimmung der Uhr des Masters benutzt würden, wenn die Ungenauigkeit der Bestimmung Null wäre. Ohne Einschränkung der Allgemeinheit sei $t' < t$. Es gilt offenbar $\Delta t = t - t'$ und $\Delta T = T - T'$.

Dabei sind P und P' die korrekten Zeitpunkte, d. h., ohne Berücksichtigung der Fehler bei der Bestimmung der Koordinaten.

Die korrekte Steigung der Geraden g_{master}, die den Verlauf der Uhr des Masters angibt, errechnet sich dann aus $\Delta T / \Delta t$.

Da die Bestimmung der Zeitpunkte t und t' auf dem Slave bzw. T und T' auf dem Master jeweils mit einem Fehler von maximal $\frac{1}{2}\delta$ behaftet sind, gilt für den ermittelten Wert von t, der hier t^* genannt wird: $t - \frac{1}{2}\delta \le t^* \le t + \frac{1}{2}\delta$; entsprechendes gilt für t', T und T'. Damit können die für die Punkte P und P' ermittelten Punkte innerhalb der in der Zeichnung grau gerasterten Flächen liegen, und deshalb ist die errechnete Steigung für die Gerade der Uhr des Masters nicht korrekt, sondern kann eine Abweichung haben, wobei sich maximal die Steigung der Geraden g_2 ergeben kann, wenn sich die Abweichungen so akkumulieren, daß der Master als zu schnell angesehen wird, und minimal die Steigung der Geraden g_1 ergeben kann, wenn sich die Abweichungen in die gegensätzliche Richtung akkumulieren.

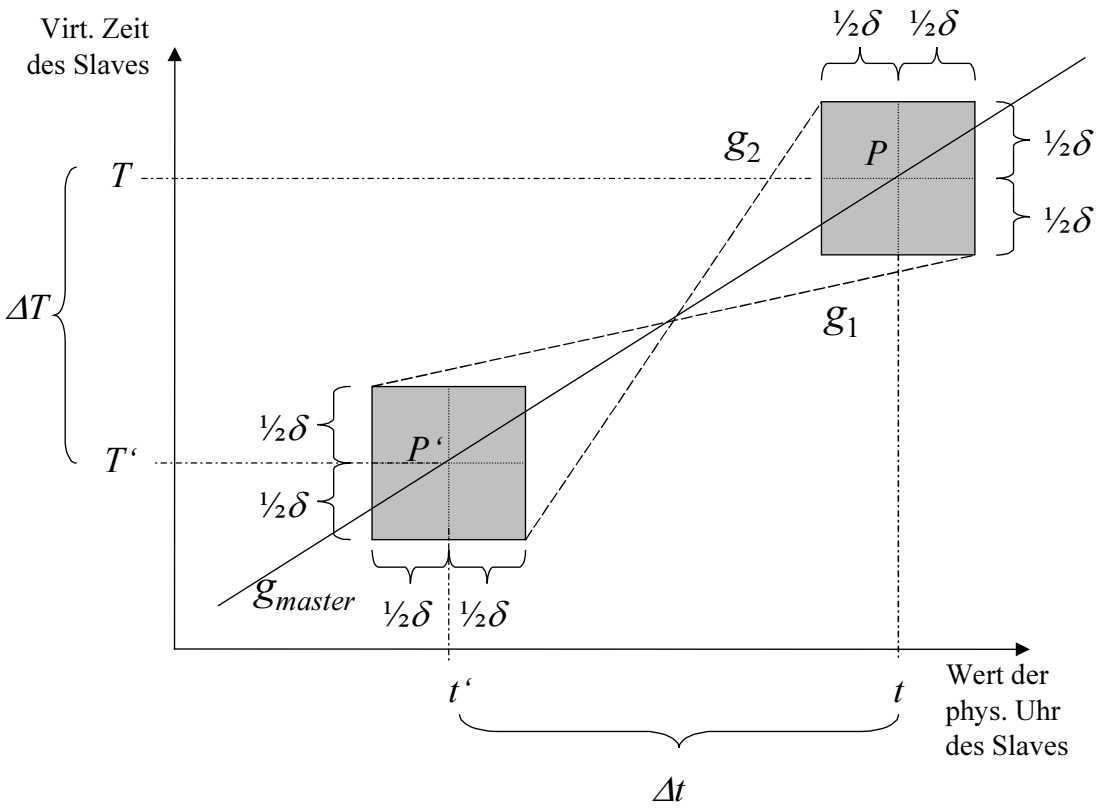


Abbildung 4-17: *Bestimmung der max. und min. Steigung der Uhrengeraden*

Die maximale Steigung ergibt sich nun, wenn der lokale Zeitstempel von P', d. h., t', zu groß, der Zeitstempel T', den der Master genommen hat, hingegen zu klein ist, beim Punkt P hingegen der lokale Zeitstempel t, zu klein und der Zeitstempel T des Masters zu groß ist. Dies liegt darin begründet, daß anhand der Zeitstempel scheinbar die Uhr des Masters, die beim „alten" Punkt P' der Uhr des Slaves noch hinterherlief, beim Punkt P die Uhr des Slaves scheinbar überholt hat, also muß der Slave nachziehen, um dieses Tempo zu halten.

In diesem Falle ergibt sich die Steigung m_2 der Geraden g_2 zu $m_2 = \frac{T'+½\delta - (T-½\delta)}{t'-½\delta - (t+½\delta)} = \frac{\Delta T+\delta}{\Delta t-\delta}$.

Analog ergibt sich die minimale Steigung m_1 der Geraden g_1 zu $m_1 = \frac{T'-½\delta - (T+½\delta)}{t'+½\delta - (t-½\delta)} = \frac{\Delta T-\delta}{\Delta t+\delta}$. ■

Lemma 11: („Präzision")
Bei der Synchronisation mittels der Ratenanpassung ergibt sich eine Präzision von $\Pi = (m_2-m_1)\,(OD+2)INT + ½\delta\,(2+m_1+m_2)$*, wobei INT die Länge des Synchronisationsintervalles ist.*

Beweis:
Man betrachte die Abbildung 4-18. Die obige Abbildung 4-17 zeigte einen Ausschnitt dieser Abbildung. Die Bezeichnungen aus Lemma 10 werden übernommen.

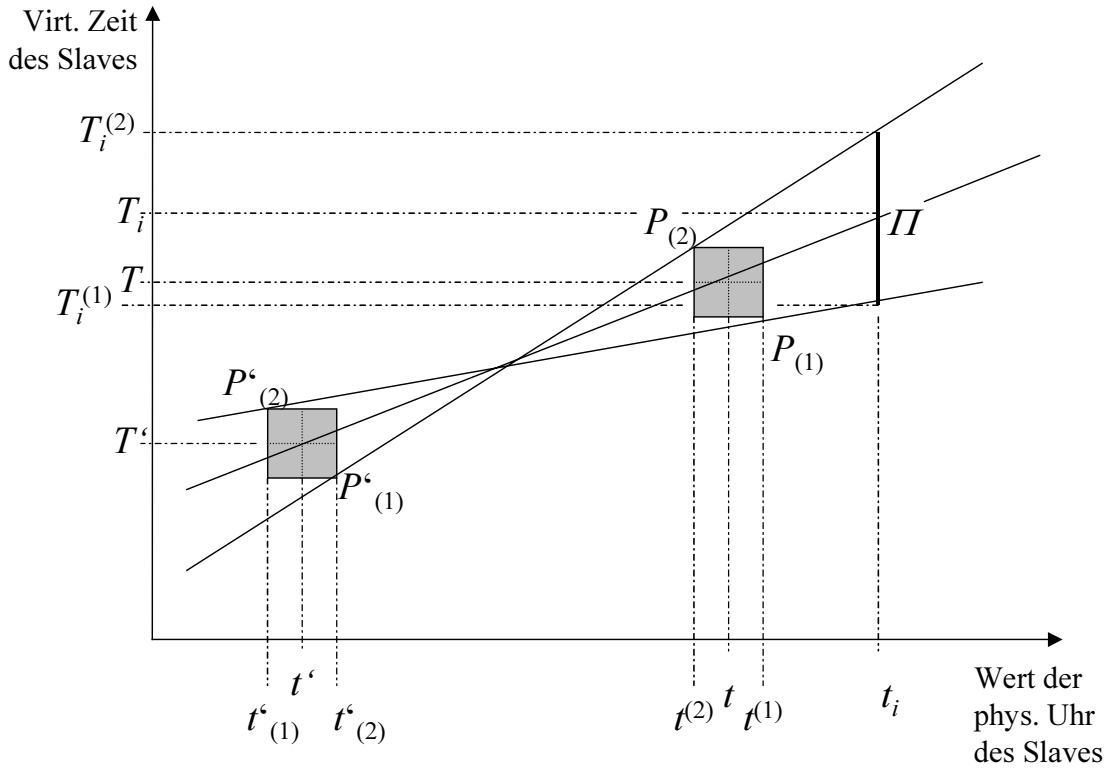


Abbildung 4-18: Bestimmung der Präzision

Sei t_i der Zeitpunkt, an dem spätestens wieder synchronisiert wird. Nach Lemma 8 wird *AdjClk*() spätestens nach der Zeit $MAX := (OD+2) \cdot INT$ aufgerufen, nach Lemma 6 kann dann synchronisiert werden, also existiert t_i, und es gilt $t_i = t + MAX$.

Die Präzision Π errechnet sich aus $\Pi = T_i^{(2)} - T_i^{(1)}$, wobei $T_i^{(2)}$ der maximale Wert ist, den die Uhr des Slaves zum Zeitpunkt t_i annehmen kann, $T_i^{(1)}$ entsprechend der minimale Wert. Seien g_2: $T = m_2\, t + b_2$ bzw. g_1: $T = m_1\, t + b_1$ die Geradengleichungen für die Geraden, die die maximale bzw. minimale Entwicklung der Uhr angeben. Dann gilt für die Präzision Π die Formel $\Pi = (m_2 - m_1)\, t_i + (b_2 - b_1)$.

Es ist $T + \frac{1}{2}\delta = m_2\,(t - \frac{1}{2}\delta) + b_2$, da P_2 auf g_2 liegt, entsprechend $T - \frac{1}{2}\delta = m_1\,(t + \frac{1}{2}\delta) + b_1$.

Daher gilt $b_2 = T + \frac{1}{2}\delta - m_2\,(t - \frac{1}{2}\delta)$ und $b_1 = T - \frac{1}{2}\delta - m_1\,(t + \frac{1}{2}\delta)$, daher bestimmt sich der Term $b_2 - b_1$ durch $b_2 - b_1 = \delta - m_2\,(t - \frac{1}{2}\delta) + m_1\,(t + \frac{1}{2}\delta) = \frac{1}{2}\delta(2 + m_1 + m_2) - (m_2 - m_1)t$.

Damit ergibt sich für die Präzision Π:

$\Pi = (m_2\text{-}m_1)\, t_i + (b_2\text{-}b_1)$
$= (m_2\text{-}m_1)\, t_i + ½\delta(\, 2 + m_1 + m_2) - (m_2 - m_1)t$
$= (m_2\text{-}m_1)\, (t + MAX) + ½\delta(\, 2 + m_1 + m_2) - (m_2 - m_1)t$
$= (m_2\text{-}m_1)\, MAX + ½\delta(\, 2 + m_1 + m_2).$ ■

Satz 12: („Präzision des Protokolls“)
Bei der Synchronisation mit der Ratenanpassung ergibt sich eine Präzision von $\Pi = \frac{2\delta\Delta t(2+\rho)}{(\Delta t)^2\text{-}\delta^2}(OD{+}2)INT + \delta\left(1 + \frac{(\Delta t)^2(1+\rho)+\delta^2}{(\Delta t)^2\text{-}\delta^2}\right)$*, wobei* ρ *die Driftrate der Uhr ist.*

Beweis:
Dies ergibt sich aus Lemma 11 und der Abschätzung der folgenden Terme:

$$m_2\text{-}m_1 = \frac{\Delta T+\delta}{\Delta t\text{-}\delta} - \frac{\Delta T\text{-}\delta}{\Delta t+\delta} = \frac{2\delta(\Delta T+\Delta t)}{(\Delta t)^2\text{-}\delta^2} \le \frac{2\delta\Delta t(2+\rho)}{(\Delta t)^2\text{-}\delta^2} \text{ und}$$
$$m_1{+}m_2 = \frac{\Delta T+\delta}{\Delta t\text{-}\delta} + \frac{\Delta T\text{-}\delta}{\Delta t+\delta} = \frac{2\Delta T\,\Delta t+2\delta^2}{(\Delta t)^2\text{-}\delta^2} \le \frac{2\,(\Delta t)^2(1+\rho)+2\delta^2}{(\Delta t)^2\text{-}\delta^2}.$$ ■

In Kapitel 6 „Messungen“ wird diese Formel näher diskutiert, nachdem die noch offenen Parameter gemessen wurden.

5 Implementierung

Das in Kapitel 4 „Uhrensynchronisationsprotokoll für das Funknetzwerk“ beschriebene Protokoll sollte im Rahmen dieser Diplomarbeit implementiert werden. Eine Implementierung dieses Protokolls wie auch des Protokolls nach dem IEEE 802.11-Standard erfordert eine möglichst geringe Schwankung des zeitkritischen Pfades. Dies ist nur möglich, indem die Implementierung des Protokolls möglichst „nah“ an der Hardware erfolgt. Diese hardwarenahe Implementierung verlangt wiederum die Unterstützung eines Herstellers von solchen Karten. Die Firma Lucent Technologies konnte dazu bewegt werden, die Gerätetreiber für ihre Funknetzwerkkarten mit der Produktbezeichnung „WaveLAN/IEEE“ und die zugehörige Dokumentation ([Lu 98]) zur Verfügung zu stellen. Diese Treiber sind zum Betrieb unter dem Betriebssystem Windows NT ausgelegt, daher wurde das Protokoll unter diesem Betriebssystem auf Standard-Intel-kompatiblen Computern implementiert, wobei das Uhrensynchronisationsprotokoll in die Gerätetreiber der Netzwerkkarten eingebaut wurde.

Im vorliegenden Kapitel wird die Implementierung beschrieben. Bevor dies in Abschnitt 5.2 „Implementierung des Protokolls“ erfolgt, wird in Abschnitt 5.1 „Implementierungsumgebung“ erläutert, wie sich die Umgebung für die Implementierung darstellt.

5.1 Implementierungsumgebung

5.1.1 Die Windows NT Netzwerk-Architektur

Ein modernes Betriebssystem wie beispielsweise Windows NT ist in hierarchischen Schichten aufgebaut. Jede Schicht hat eine definierte Aufgabe, die sie übergeordneten Schichten über eine klar definierte Schnittstelle zur Verfügung stellt, wobei sich eine Schicht der Hilfe der untergeordneten Schicht bedienen kann. Eine Schicht baut demnach auf der unter ihr liegenden auf, indem sie Aufgaben an diese delegiert, und liefert für die oberhalb von ihr liegende eine neue Abstraktionsebene, da diese sich nicht um die internen Details kümmern muß. Hierdurch erreicht man eine Verringerung der Komplexität jeder Schicht, da diese nur über wohldefinierte Schnittstellen auf die unteren Schichten und letztendlich auf die Hardware zugreifen kann.

Die unterste Schicht besteht aus der Hardware. Hierauf aufbauend gibt es Schichten mit Gerätetreibern, einige Schichten, die interne Verwaltungen des Betriebssystems erledigen, und letztendlich die Applikationsschicht als höchste Schicht, in der die Anwendungsprogramme ausgeführt werden.

Dieses Schichtenmodell besitzt einige essentielle Vorteile, die insbesondere beim Entwurf, der Wartung und der Erweiterung solcher Systeme hilfreich sind. Für die Implementierung eines Uhrensynchronisationsprotokolls benötigt man einen möglichst genau vorhersagbaren zeitlichen Verlauf einiger wichtiger Systemdienste, vor allem beim

Versenden und Empfangen von Nachrichten, da diese Aktionen im kritischen Pfad des Protokolls liegen. In einem Schichtenmodell ist jede zusätzliche Schicht oberhalb der Hardware, die Einfluß auf das Protokoll hat, eine Quelle für zusätzliche Ungenauigkeiten. Daher ist es wichtig, das Protokoll in eine möglichst niedrige Schicht einzubetten, um eine hohe Präzision zu erreichen.

Ein Nachteil eines Schichtenmodells in Bezug auf die Uhrensynchronisation besteht darin, daß eine Kommunikation zwischen den Schichten nur mit der unmittelbar ober- oder unterhalb liegenden Schicht vorgesehen ist, und diese Kommunikation häufig sehr zeitintensiv ist. Der Zeitaufwand dieser Kommunikation soll in dieser Implementierung minimiert werden, damit eine präzise synchronisierte Uhr auch mit einer hierzu passenden Auflösung gelesen werden kann, d. h., daß es möglich sein muß, zwei aufeinanderfolgende Zeitstempel zu nehmen, die sich um einen Wert unterscheiden, der in der Größenordnung der Präzision liegt.

Bei modernen Prozessoren gibt es Mechanismen, die die Implementierung von Schichtenmodellen unterstützen. Die Prozessoren besitzen zumeist mindestens zwei Modi[12], einen speziellen Kernel-Modus (auch Superuser-Modus o. ä. genannt) und einen User-Modus. Im Kernel-Modus hat ein auf dem Prozessor ablaufendes Programm vollen Zugriff auf das komplette System, insbesondere darf es alle Prozessorbefehle ausführen. Dies ist im User-Modus hingegen nicht der Fall: Hier werden als besonders kritisch angesehene Befehle nicht zugelassen. Insbesondere zählen zu solchen verbotenen Befehlen der direkte Zugriff auf die Hardware oder die Veränderung des Betriebssystems. Eine Applikation läuft grundsätzlich im User-Modus ab, lediglich durch den Aufruf von Systemfunktionen erlangt sie indirekt Zugriff auf den Kernel-Modus und damit die Hardware. Diese Trennung dient dem Zweck, das Betriebssystem und die Hardware vor der Applikation zu schützen, so daß ein fehlerhaftes Programm nicht das ganze Computersystem in Mitleidenschaft zieht. Diese Unterstützung mittels Prozessormodi erlaubt es, ein Betriebssystem sicherer zu machen als wenn diese Unterstützung nicht vorhanden wäre.

Im Folgenden soll betrachtet werden, wie sich die Situation konkret unter Windows NT darstellt.

Die Abbildung 5-1 zeigt vereinfachend das Schichtenmodell, welches Windows NT zugrundeliegt (vgl. [MS 96]). Hier ist nur der Teil abgebildet, der für den Betrieb von Netzwerken benötigt wird.

Eine Applikation, die sich im User-Modus befindet, greift über definierte Schnittstellen (dem NetBIOS- oder dem Windows Sockets-Emulator) auf das Netzwerk zu. Dabei ist NetBIOS ein von Microsoft und IBM zusammen entwickelter Industriestandard, die Windows Sockets sind eine erweiterte Umsetzung der BSD Sockets, welche normaler-

[12] Die Familie der Intel-Prozessoren ab dem iAPX286 besitzt sogar vier Modi, die sogenannten „Ringe“; da NT hiervon aber nur zwei nutzt, wird nicht näher auf die anderen Ringe eingegangen.

weise unter Unix für den Zugriff auf TCP/IP benutzt werden. Beide Schnittstellen bestehen jeweils aus einem User-Mode- und einem Kernel-Mode-Teil. Der Kernel-Mode-Teil kommuniziert über das *Transport Driver Interface* (TDI), der Windows NT-eigenen Netzwerkschnittstelle, mit dem NDIS (*Network Driver Interface Specification*). NDIS definiert eine Schnittstelle, mit der festgelegt wird, wie Gerätetreiber für Netzwerkkarten mit ihren zugehörigen Karten, die Gerätetreiber mit den zugehörigen Protokollen und diese wiederum mit dem Betriebssystem kommunizieren – d. h., NDIS selbst besteht auch wieder aus mehreren Schichten. NDIS bietet eine betriebssystemunabhängige Umgebung für den Entwurf von Netzwerkkartentreiber – NDIS existiert für DOS, Windows 3.1, OS/2, Windows 95, 98 und Windows NT –, daher gehören zu NDIS einige eigene Verwaltungsfunktionen, wodurch die Entwicklung von Gerätetreibern, die keine Kenntnisse über das zugrundeliegende Betriebssystem besitzen müssen, ermöglicht wird. Für den Entwickler von NDIS-Treibern *ist* NDIS das Betriebssystem.

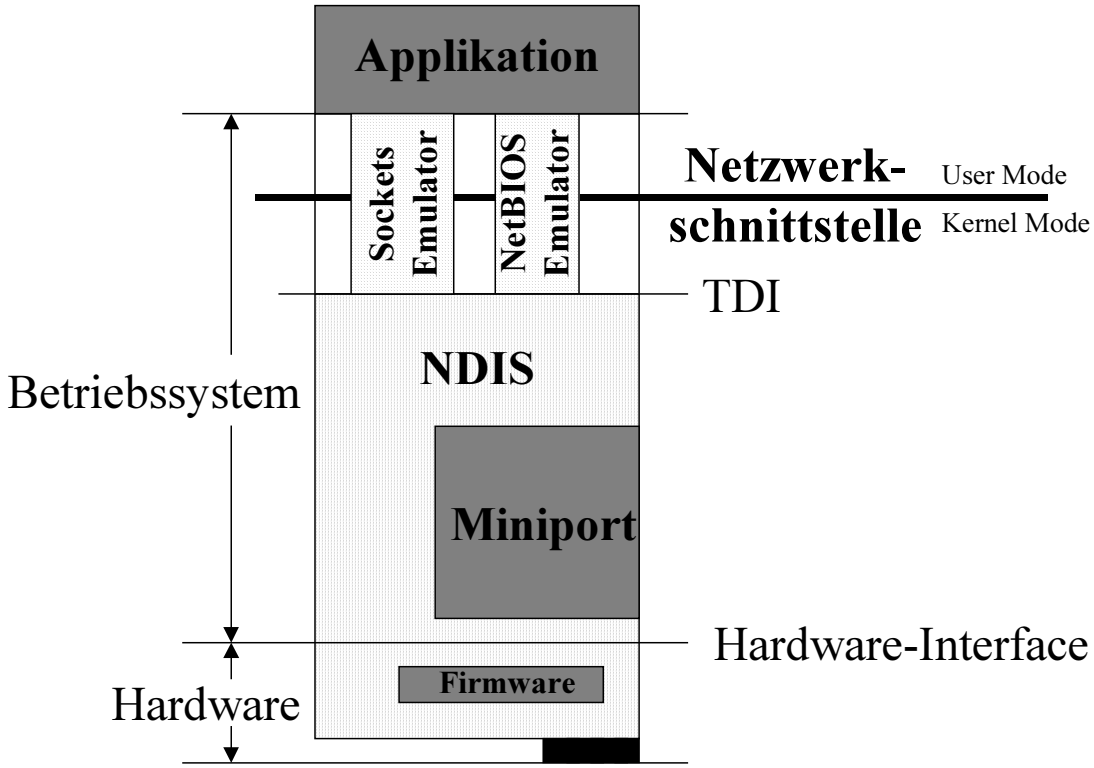


Abbildung 5-1: *Netzwerk-Architektur unter Windows NT*

Der eigentliche Gerätetreiber für die Netzwerkkarte wird in NDIS *Miniport* genannt. Diese Bezeichnung hat historische Gründe. In älteren NDIS Versionen (bis Version 2, wie es unter DOS,OS/2 und Windows 3.1 benutzt wird) bestand ein Netzwerkkartentreiber aus einem „vollwertigen“ Gerätetreiber (*full legacy driver*), der sich auch um Verwaltungsfragen des Betriebssystems kümmern mußte. Mit der NDIS Version 3, die unter Windows 95, 98 und NT eingesetzt wird, wurden diese Funktionen, die für jeden Netzwerkkartentreiber immer gleich sind, in NDIS integriert. Der Gerätetreiber muß nur noch diejenige Funktionalität erbringen, die die unterstützte Netzwerkkarte von anderen

unterscheidet. Um diese „abgespeckten“ Gerätetreiber von den „vollwertigen“ zu unterscheiden, wurde der Begriff *Miniport* geprägt.

Ein sauber entworfener Miniport greift nicht selbständig auf die Hardware, d. h. die Netzwerkkarte, zu, sondern bemüht hierzu wieder NDIS, welches den Zugriff erledigt. Dies ist notwendig, um die geforderte Betriebssystemunabhängigkeit zu erreichen. Aus diesem Grund umklammert in der Darstellung der Abbildung 5-1 der Block NDIS den Miniport.

Unterhalb des Miniports liegt die Netzwerkkarte als physikalisches Gerät, und zwar außerhalb des Betriebssystems. Da moderne Netzwerkkarten häufig einen eigenen Prozessor und ein eigenes Betriebssystem (die sogenannte Firmware) besitzen, empfiehlt es sich, die Netzwerkkarte als aus (mindestens) zwei Schichten bestehend zu betrachten, der Firmware und der eigentlichen Hardware.

Der Miniport-Treiber der im Rahmen dieser Diplomarbeit benutzten Netzwerkkarte der Firma Lucent besteht wiederum aus zwei Schichten, den *Hardware Control Functions* (HCF), einer betriebssystemunabhängigen Schicht zum Zugriff auf die Netzwerkkarte, und den *Module Specific Functions* (MSF), die betriebssystemabhängige Funktionen verarbeiten. In diesem Kontext gilt NDIS als das „Betriebssystem“.

Damit ergeben sich die für die konkrete Implementierung relevanten Schichten, wie sie in der Abbildung 5-2 dargestellt sind. Man beachte, daß NDIS in den Schichten 2 und 5 zu finden ist. Dies liegt an der oben beschrieben „Umklammerung“ des Miniports, der aus den Schichten 3 und 4 besteht, durch NDIS.

Es stellt sich die Frage, in welcher Schicht das eigene Protokoll implementiert wird. Optimal wäre natürlich die Schicht 0 oder alternativ die Schicht 1, weil dort die nichtvorhersagbaren Einflüsse am geringsten sind. Eine Implementierung in der Schicht 0 bedeutet jedoch, daß die Hardware, eine Implementierung in der Schicht 1, daß die Firmware geändert werden muß. Beides verlangt, daß Informationen über die vorhandene Hardware und Entwicklungsumgebungen zu ihrer Veränderung vorhanden sind. Dies ist im Rahmen dieser Diplomarbeit nicht der Fall, somit scheidet diese Lösung zunächst aus.

Die Schicht 2 gehört zum NT Betriebssystem; damit fehlen auch hier die notwendigen Informationen (wie Quelltexte), um diese Änderungen vorzunehmen.

Die Quelltexte für die Schichten 3 und 4 wurden von der Firma Lucent Technologies zur Verfügung gestellt und liegen ausreichend dokumentiert vor. Damit bietet sich die Implementierung in einer dieser beiden Schichten an. Obwohl eine Implementierung in der Schicht 3 möglich wäre, ist es nicht zweckmäßig, dies zu tun, da diese Schicht hauptsächlich Low-Level-Funktionen zum vereinfachten Zugriff auf die Netzwerkkarte enthält. Diese Low-Level-Funktionen werden von dem Protokoll selbst benötigt, andererseits ist beim Umgang mit ihnen einige Sorgfalt notwendig, da einige Randbedingungen bezüglich einer internen Tasksynchronisation erfüllt sein müssen – Randbedingun-

gen, deren Einhaltung von der Schicht 4 garantiert werden. Daher müssten bei der Implementierung in der Schicht 3 große Teile der Funktionalität der Schicht 4 in der Schicht 3 implementiert werden. Um diese Redundanz zu vermeiden, die zudem keinen großen Vorteil in Bezug auf Vorhersagbarkeit verspricht, wurde das Protokoll also in der Schicht 4, der MSF als Teil des Miniports, implementiert.

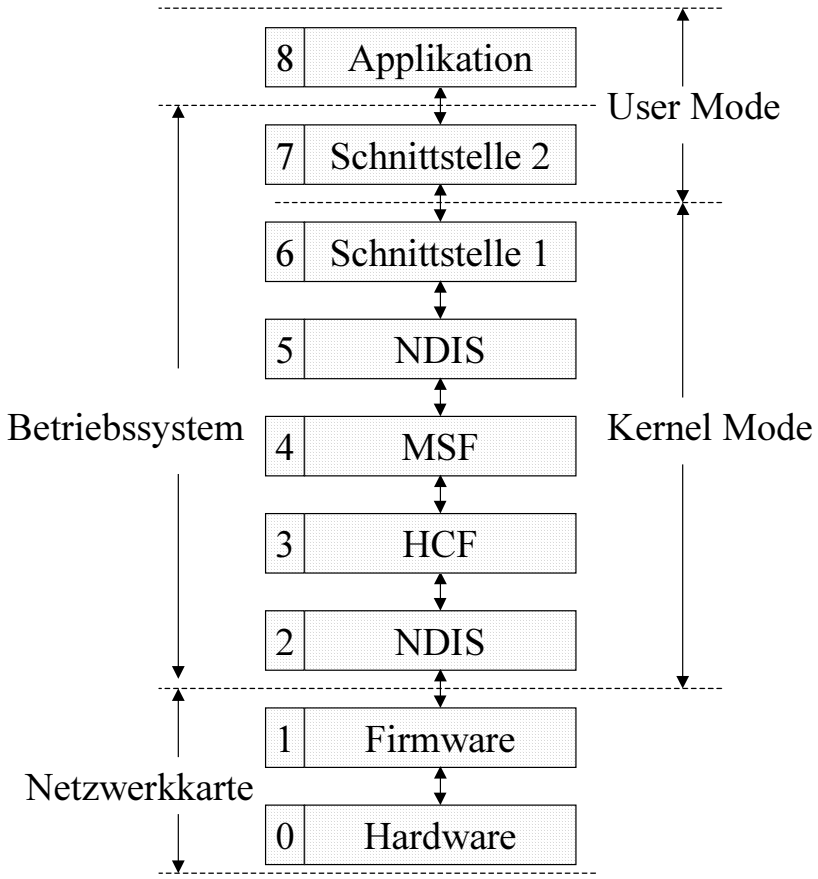


Abbildung 5-2: *Die für die Implementierung relevanten Schichten*

Nach dem oben beschriebenen ergibt sich die Anforderung der Implementierung, daß die Synchronisation möglichst „tief“ in den Schichten implementiert werden muß, weil jede weitere Schicht zwischen der Implementierung der Synchronisation und der Hardware Ungenauigkeiten bei der Bestimmung des zeitlichen Verhaltens bewirkt. Weiterhin muß die Applikation (auf der Schicht 8) jederzeit schnell auf die aktuelle Zeit zugreifen können. Diese beiden Ziele widersprechen sich scheinbar: Je tiefer die Implementierung erfolgt, desto genauer kann die Synchronisation werden, da das zeitliche Verhalten besser bestimmt werden kann. Allerdings ist die Implementierung von der Applikation dann weiter entfernt, so daß die Applikation nur noch mit großer (und ungenau bestimmbarer) Verzögerung auf die Uhrzeit zugreifen kann: Das Ergebnis ist eine genaue Uhr, die aber nur ungenau ausgelesen werden kann!

Die Lösung für das Problem ergibt sich, wenn eine Möglichkeit gefunden wird, an dem Schichtenmodell vorbei auf tiefe Schichten zuzugreifen. Da bei der Uhrensynchronisation nur das Synchronisationsprotokoll die Uhren verändern kann, benötigt man ausschließlich einen Informationsaustausch von „unten nach oben“, also reicht es, einen Mechanismus zu finden, mit dem lesend auf niedrige Schichten zugegriffen werden kann.

Diese Kommunikation erfolgt in der Implementierung, indem die tiefe Schicht und die Applikation über einen gemeinsam benutzten Speicher miteinander kommunizieren. Dieser Speicher kann von der tiefen Schicht gelesen und beschrieben, von der Applikation aber nur gelesen werden. Nähere Details hierzu finden sich in Abschnitt 5.1.3 „Kommunikation zwischen Gerätetreiber und Applikation“.

5.1.2 Realisierung der Uhren

Um eine Uhrensynchronisation mit einer hohen Präzision zu erreichen, benötigt man eine physikalische Uhr, deren Auflösung hoch genug ist und die als anpaßbare Zeitbasis dient. Dieser Abschnitt beschäftigt sich mit der Frage, wie diese Zeitbasis realisiert werden kann.

Wie schon in Abschnitt 3.1 „Begriffsbestimmungen“ erwähnt, besteht eine physikalische Uhr aus einem Zeitgeber, der in regelmäßigen Abständen Ereignisse erzeugt, und einem Zähler, der auf jedes auftretende Ereignis reagiert, indem er seinen Wert um eins erhöht. Es gibt verschiedene Wege, diese Uhr zu realisieren. Folgende Optionen existieren:

1. Nutzung der NT-eigenen Uhr;
2. Anschluß eines externen Zeitgebers;
3. Nutzung einer Uhr auf der Netzwerkkarte;
4. Nutzung anderer interner Mechanismen.

Windows NT besitzt eine eigene Implementierung einer Uhr, die für die Uhrensynchronisation geeignet zu sein scheint. Diese Uhr hat allerdings die Nachteile, daß sie nur relativ langsam abzufragen ist, nur schwer gesetzt werden kann und keine Ratenanpassung ermöglicht. Weiterhin ist sie, je nach der Hardware, auf der NT läuft, unterschiedlich implementiert, so daß quantitative Erkenntnisse über ihre Genauigkeit und Präzision nur sehr schwer auf andere Systeme übertragbar sind.

Die Lösung mit externen Zeitgebern erlaubt eine höhere Flexibilität bei der Implementierung, da die Auflösung der Uhr je nach Anforderung bestimmt werden kann und Mechanismen vorgesehen werden können, die Frequenz der Zeitgeber bei der Synchronisation anzupassen. Allerdings hat diese Vorgehensweise Nachteile: Ist der Zähler

extern, dann befindet er sich in der Schicht 0 des Schichtenmodells (s. Abbildung 5-2). Hiermit wird ein Zugriff aus der Applikation, welche sich in Schicht 8 des Schichtenmodells befindet, erschwert. Dies liegt daran, daß unter Windows NT ein direkter Zugriff auf die Hardware aus dem User-Modus nicht gewollt ist, um die Sicherheit des Systems zu erhöhen. Allerdings läßt sich der Zugriff durch den Einsatz von Gerätetreibern freischalten, so daß der Zugriff grundsätzlich möglich ist.

Daher ist auch die Nutzung einer Uhr, welche auf der Netzwerkkarte realisiert wird, grundsätzlich möglich. So eine Uhr stellt sich nur als ein Spezialfall eines externen Zeitgebers dar und muß daher nicht besonders betrachtet werden. Interessant ist dieser Fall allerdings, weil er die Implementierung des Protokolles auf der Netzwerkkarte mit schnellem Zugriff auf die Uhr erlaubt.

Diese Möglichkeit steht aber im Rahmen dieser Implementierung nicht zur Verfügung. Daher muß ein anderer Mechanismus gesucht werden. Auf allen Intel-Prozessoren ab dem Pentium und auf den Prozessoren von AMD gibt es einen eingebauten Zählmechanismus, der die nötige Auflösung besitzt. Der sogenannte *Timestampcounter* (TSC) ist ein internes Register des Prozessors und zählt jeden Takt des Prozessors. Daher hat er eine Auflösung entsprechend der Prozessortaktfrequenz. Die Prozessortaktfrequenz ist recht frequenzstabil, also ist dies auch der TSC, wodurch er sich für die Implementierung einer Uhr eignet. Der TSC besitzt eine hohe Auflösung und läßt sich einfach mittels eines Prozessorbefehls auslesen. Man kann den Prozessor so einstellen, daß der TSC sowohl im Kernel- als auch im User-Modus ausgelesen werden kann, so daß keine Kommunikation zwischen den Schichten erforderlich ist, um den Wert des TSCs zu ermitteln.

Der TSC wird bei einem Reset des Prozessors auf Null gesetzt, hiernach beginnt er unmittelbar mit dem Zählen der Prozessortakte. Die Taktfrequenz ist je nach Computer unterschiedlich, typische Werte heutiger Computer bewegen sich zwischen 100 MHz und bis zu 800 MHz. Damit eine physikalische Uhr mittels des TSC auf verschiedenen Computern mit unterschiedlichen Taktfrequenzen implementiert werden kann, ist es daher notwendig, den Wert des TSC zu „normieren“, so daß die physikalische Uhr auf allen beteiligten Stationen die gleiche Auflösung besitzt und die Zeitstempel damit vergleichbar sind. Diese Normierung erfolgt dadurch, daß der Wert des TSCs durch die Prozessortaktfrequenz dividiert wird.

Hierbei ergeben sich folgende Anforderungen:

1. Um die notwendige geringe Driftrate zu erreichen, muß die Prozessortaktfrequenz ausreichend genau bestimmt werden. Eine Angabe wie 200 MHz ± 0,5 MHz bedeutet schon durch die Fehlerangabe, daß eine Driftrate von $0{,}5 / 200 = 2{,}5 \cdot 10^{-3}$ vorliegt, somit läge die Driftrate weitaus höher als es in den bisherigen Ausführungen angenommen wurde.

2. Es muß ein Mechanismus gefunden werden, der eine Ratenanpassung der Uhr erlaubt, d. h., die Uhr muß beschleunigt und abgebremst werden können.

Die erste Anforderung läßt sich durch genaue Messung lösen: Bestimmt man die Taktfrequenz auf 1 kHz genau, dann ergibt sich in dem Beispiel mit 200 MHz eine durch den Fehler verursachte Driftrate von 1 kHz / 200 MHz = $5 \cdot 10^{-6}$. Diese befindet sich also in der Größenordnung, die theoretisch für die Driftrate der Uhr angenommen wurde.

Es ist zweckmäßig, den Wert der physikalischen Uhr mit einer Auflösung anzugeben, die eine Zehnerpotenz der Sekunde ist, also beispielsweise in Mikrosekunden, da hiermit ein guter Bezug auf die übliche Zeiteinheit Sekunde ermöglicht wird. Um den gewünschten Wert in Mikrosekunden zu erhalten, müßte der Wert des TSCs durch den Wert der Taktfrequenz, angegeben in Megahertz, dividiert werden. Dies bedeutet aber bei der geforderten Genauigkeit, daß die Taktfrequenz in MHz mit Nachkommastellen angegeben werden müßte. Man hat zwei Implementierungsmöglichkeiten: Man kann eine Fließkommadivision vornehmen, oder diese Division mit einer Integerdivision nachbilden, um den Wert $PC := \mathrm{TSC} / f$, wobei f die Taktfrequenz in der Einheit MHz ist, zu erhalten.

Eine Fließkommadivision hat den Nachteil, daß sie nur mit einer beschränkten Anzahl signifikanter Stellen rechnet und damit Rundungsfehler unvermeidlich sind, wodurch die Auflösung der Uhr verringert wird. Dieser Nachteil tritt bei einer Integerdivision nicht auf. Bei der Integerdivision muß allerdings zuerst mit einem normierenden Wert multipliziert werden, bevor die Division ausgeführt werden kann. Anstelle einer Division muß also vielmehr der Term $PC := \mathrm{TSC} \cdot mult / div$ berechnet werden. Dieser normierende Wert muß groß genug sein, damit die signifikanten Stellen der Taktfrequenz, die im Divisor *div* benutzt werden, als ganze Zahlen betrachtet werden können.

Aufgrund der binären Struktur böte es sich an, eine 2er-Potenz als Multiplikator *mult* zu wählen, da hiermit die Multiplikation beschleunigt würde. Insbesondere die Wahl des Multiplikators *mult* als 2^{base}, wobei *base* die Wortbasis des Prozessors ist (bei aktuellen Intel-Prozessoren also $base = 32$), würde die Multiplikation extrem beschleunigen, da dann gar keine Multiplikation stattfinden muß. Allerdings zeigt es sich, daß diese Wahl nur sub-optimal ist: Tatsächlich benötigt man bei der Uhrensynchronisation keine physikalische Uhr, sondern nur eine virtuelle Uhr. Diese sollte in der Geschwindigkeit anpaßbar sein, damit eine Ratenanpassung stattfinden kann. Da die Taktfrequenz und damit die Frequenz des TSCs nicht anpaßbar ist, muß ein Mechanismus diese Anpassung nachträglich durchführen. Hierzu bietet es sich an, den Wert der physikalischen Uhr mit einer Steigung zu multiplizieren, um eine Uhrengerade zu erhalten. Dieser Multiplikator wird einen Wert in der Größenordnung von 1 besitzen, und er besitzt Nachkommastellen. Damit wäre wiederum eine Fließkommamultiplikation notwendig, oder diese müßte wieder mit einer Integermultiplikation und Division nachgebildet werden.

Es bietet sich an, die Multiplikation und die Division mittels *einer* Integermultiplikation und *einer* Integerdivision nachzubilden. In den Multiplikator fließt die Steigung der Uhrengeraden ein, in den Divisor die Prozessortaktfrequenz. Dann gibt es keinen Grund

mehr, den Multiplikator als 2er-Potenz zu wählen, da die Multiplikation nicht eingespart werden kann.

Um Überläufe in den Rechnungen zu vermeiden, müssen die Größen der auftretenden Zwischenergebnisse abgeschätzt werden. Der TSC hat eine Größe von 64 Bit. Angenommen, der Multiplikator hat eine Größe von 32 Bit, dann hat das erste Zwischenergebnis nach der Multiplikation mit dem Multiplikator eine Größe von 96 Bit. Dieses Zwischenergebnis wird hiernach durch den Divisor geteilt. Unter der Annahme, daß der Divisor größer ist als der Multiplikator, hat das Endergebnis wieder eine Größe von maximal 64 Bit. Diese Annahme kann garantiert werden, wenn die Geradensteigung kleiner als 2 und die Taktfrequenz größer als 2 MHz ist.

Es werden also Algorithmen benötigt, die eine Multiplikation von 64 Bit mit 32 Bit und eine Division von 96 Bit durch 32 Bit ermöglichen, sogenannte Multi-Word-Algorithmen. Solche Algorithmen wurden [Kn 81] entnommen und wie dort angegeben implementiert.

Die virtuelle Uhr, wie sie in Abschnitt 4.3.2 „Ratenanpassung“ beschrieben wurde, besteht aus einer Geradengleichung $VC = m \cdot PC + b$, d. h. es existiert eine Steigung m und ein Achsenabschnitt b. Gerade wurde ausgeführt, wie die Steigung zur Berechnung der Uhr herangezogen werden kann. Der Achsenabschnitt wird nach der Multiplikation und Division der TSCs zu dem Ergebnis hinzuaddiert, damit ergibt sich die virtuelle Zeit. Demnach ergibt sich eine virtuelle Uhr durch die Steigung und den Achsenabschnitt. Sind diese Werte bekannt, so kann durch Auslesen des TSCs die aktuelle Zeit berechnet werden.

Aufgrund der Struktur der Berechnung der virtuellen Uhr ist ihr Wert immer kleiner als der Wert des TSCs. Daher ist für die Frage, wann die Uhr einen Überlauf hat, einzig der Überlauf des TSCs maßgeblich. Aktuelle Prozessoren haben Taktfrequenzen von unter 1 GHz, d. h., es werden 10^9 Signale pro Sekunde erzeugt. In hundert Jahren ist die Anzahl der erzeugten Taktsignale auf $10^9 \cdot 60 \cdot 60 \cdot 24 \cdot 365.2425 \cdot 100$ angestiegen. Der TSC hat eine Größe von 64 Bit, damit läuft die Uhr nach $\frac{2^{64}}{10^9 \cdot 60 \cdot 60 \cdot 24 \cdot 365.2425 \cdot 100} > 5$ Jahrhunderten über.

In Abschnitt 4.3.2 „Ratenanpassung“ wurde ein Zeitpunkt t_i^* eingeführt, der angibt, wann von einer virtuellen Uhr VC_i^* auf die nächste virtuelle Uhr VC_i' umgeschaltet wird. Dieser „Umschaltezeitpunkt“ ist notwendig, da die virtuelle Uhr aus zwei Teilen besteht: Zuerst wird versucht, der Uhr des Masters näherzukommen („Korrekturphase“ VC_i^*), und danach läuft die Uhr ungefähr mit der Geschwindigkeit des Masters weiter („Freilaufphase“ VC_i'). Dieser Zeitpunkt t_i^* wird demnach in der Implementierung benötigt.

Die virtuelle Uhr *VIRTUAL_CLOCK* mit allen notwendigen Informationen besteht daher aus folgenden Elementen:

m_i^* Die Steigung der virtuellen Uhr während der Korrekturphase
b_i^* Der Achsenabschnitt der virtuellen Uhr während der Korrekturphase
t_i^* Der Umschaltezeitpunkt zwischen Korrekturphase und Freilaufphase
m_i Die Steigung der virtuellen Uhr während der Freilaufphase
b_i Der Achsenabschnitt der virtuellen Uhr während der Freilaufphase

Der Umschaltezeitpunkt t_i^* ist dabei der einzige Wert innerhalb der Implementierung, der die Uhrzeit nicht in der Einheit Mikrosekunden, sondern in Taktzyklen des TSC angibt. Dies wird aus Effizienzgründen so implementiert, da hierdurch ein schneller Vergleich zwischen TSC und t_i^* erfolgen kann.

Beim Auslesen der Uhr wird dann folgendermaßen vorgegangen: Zuerst wird getestet, ob der Umschaltezeitpunkt schon überschritten ist, indem der Wert des TSCs mit t_i^* verglichen wird. Wenn dies nicht der Fall ist, wird mittels des Wertes des TSCs und den Parametern (m_i^*,b_i^*) die Uhrzeit berechnet. Ist der Umschaltezeitpunkt hingegen schon überschritten, wird mittels des Wertes des TSCs und der Parameter (m_i,b_i) die Uhrzeit bestimmt.

Dieses Verfahren zum Ermitteln eines Zeitstempels ist relativ langsam, da ein Vergleich und eine Rechnung erfolgen müssen. Man kann diesen Prozeß aber optimieren. Hierzu nutzt man aus, daß die Implementierung der Uhr als stückweise lineare Funktion erfolgt. Die Beschleunigung erfolgt dadurch, daß als Zeitstempel der „rohe“ Wert des TSCs benutzt wird, wobei zusätzlich die Informationen über die zu dem Zeitpunkt geltende virtuelle Uhr ((m_i^*,b_i^*) oder (m_i,b_i)) mit abgespeichert wird. Dies erlaubt es, lokale Ereignisse mit einer höheren Auflösung in Beziehung setzen zu können, da der Wert des TSCs für die Ermittlung einer Vorher-/Nachher-Beziehung ausreicht. Erst, wenn es notwendig wird, verteilt ermittelte Zeitstempel in Bezug zu setzen, muß dann die Rechnung ausgeführt werden, um die virtuelle Zeit zu erhalten.

5.1.3 Kommunikation zwischen Gerätetreiber und Applikation

Das eigentliche Uhrensynchronisationsprotokoll wird, wie schon in Abschnitt 5.1.1 „Die Windows NT Netzwerk-Architektur“ beschrieben, im Gerätetreiber der Netzwerkkarte implementiert, da er sehr zeitkritisch ist. Andererseits benötigt die Applikation die virtuelle Uhr. Nun benötigt ein Aufruf einer Gerätetreiber-Funktion aus einer Applikation heraus etwa gemessene 250 µs. Daher ist es von Vorteil, wenn die aktuelle Uhrzeit *nicht* vom Gerätetreiber errechnet werden muß, sondern statt dessen die Applikation die Uhrzeit selbst berechnen kann, da hierdurch dieser Kommunikationsaufwand zwischen den Schichten vermindert werden kann und die Geschwindigkeit erhöht wird, sofern die Applikation zeitkritisch ist. Hierzu benötigt die Applikation den Wert des TSCs und die Parameter, die in *VIRTUAL_CLOCK* gespeichert sind. Den Wert des TSCs kann die Applikation einfach durch einen Prozessorbefehl auslesen, sofern der Zugriff im User-Modus freigeschaltet ist, was unter Windows NT die Default-Einstellung ist. Es bleibt

die Frage offen, wie der Gerätetreiber die Parameter in *VIRTUAL_CLOCK* an die Applikation weitergeben kann.

Die einfachste Lösung besteht darin, daß die Applikation den Gerätetreiber nach den Werten abfragt, indem ein Betriebssystemaufruf erfolgt. Dieser Zugriff der Applikation auf den Gerätetreiber ist aber relativ langsam, da mehrere Schichten des Schichtenmodells involviert sind: Zum Zugriff auf die Uhr wäre ein langsamer Zugriff auf den Gerätetreiber notwendig. Daher macht es keinen zeitlichen Unterschied, ob der Gerätetreiber oder die Applikation die Berechnung der virtuellen Zeit vornehmen.

Vielmehr wird eine Lösung gesucht, bei der es möglich ist, den Wert der Uhr aus dem Gerätetreiber abzufragen, ohne den Aufruf einer Betriebssystemfunktion und damit einen Moduswechsel zu benötigen. So ein Mechanismus existiert unter Windows NT, es handelt sich dabei um gemeinsam genutzten Speicher (*shared memory*).

Man betrachte die Abbildung 5-3. Sie zeigt den Gerätetreiber, die Applikation und die Speicherbereiche, auf die sie jeweils im normalen Betrieb zugreifen können. Man erkennt, daß die Speicherbereiche vollkommen getrennt sind und somit kein Zugriff von der einen Instanz auf die Datenbereiche der anderen möglich ist.

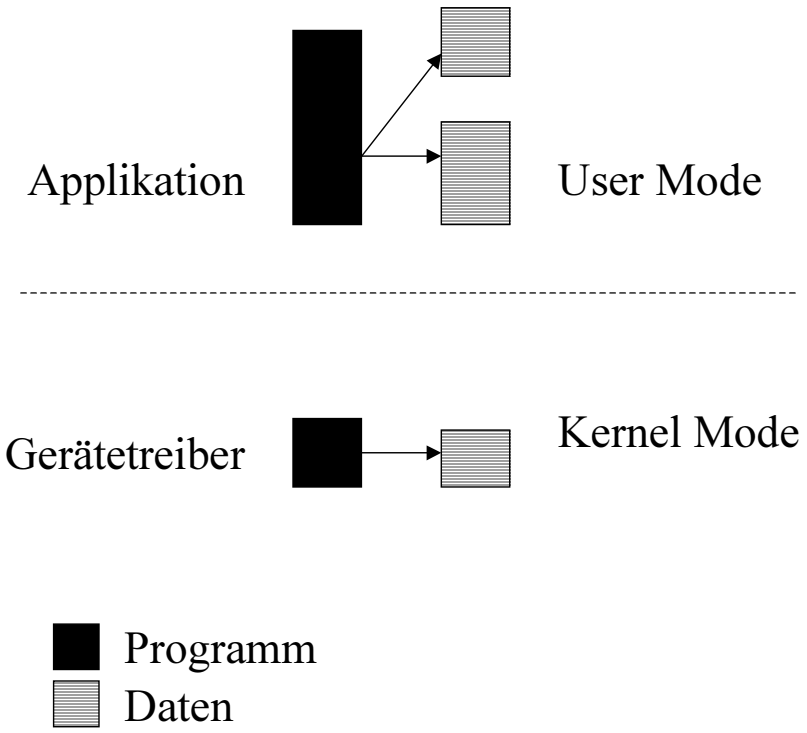


Abbildung 5-3: *getrennte Datenbereiche von Applikation und Gerätetreiber*

In der Abbildung 5-4 wird gezeigt, wie gemeinsam genutzter Speicher benutzt wird. Der Gerätetreiber hat einen Speicherbereich angefordert, der ausschließlich für die Kommunikation mit der Applikation vorgesehen ist. In diesem Speicherbereich wird die Struk-

tur *VIRTUAL_CLOCK* abgelegt. Der Gerätetreiber sorgt nun dafür, daß die Applikation auf diesen Speicherbereich zugreifen kann; dies erfolgt durch das sogenannte *Mapping* des Speicherbereichs in die Speicherbelegung der Applikation. Nach diesem *Mapping* können sowohl die Applikation als auch der Gerätetreiber auf diesen gemeinsamen Speicherbereich zugreifen.

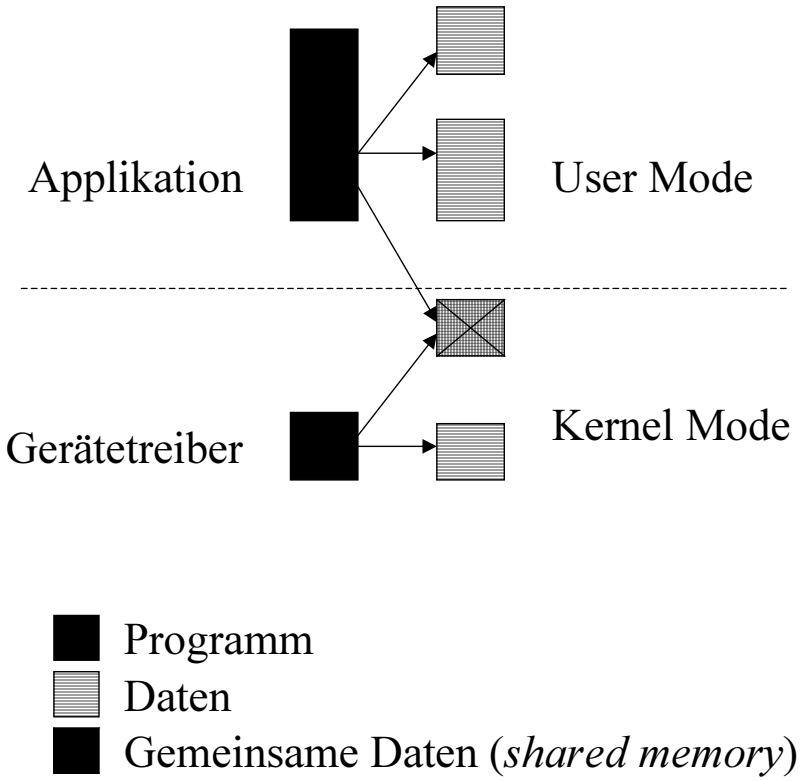


Abbildung 5-4: *gemeinsam genutzter Speicher*

Der Umgang mit dem Mapping darf nicht sorglos erfolgen. Der gemeinsame Speicherbereich ist trotz des Mappings ein Teil des Kernel-Modus. Routinen, die im Kernel-Modus arbeiten, müssen besonders zuverlässig sein, da das Betriebssystem keine Möglichkeiten hat, fehlerhafte Routinen zu erkennen und Schaden zu vermeiden. Damit diese Routinen zuverlässig funktionieren können, ist es weiterhin notwendig, daß die Datenbereiche, auf denen sie operieren, jederzeit konsistent und korrekt sind. Durch das Mapping eines Bereiches des Kernel-Modus in die Speicherbelegung einer Applikation im User-Modus wäre diese Applikation in der Lage, den Speicherbereich zu verändern und damit die Integrität des Betriebssystems zu verletzten. Damit dies nicht möglich wird, wird der Speicherbereich schreibgeschützt gemappt, d. h. die Applikation kann den Bereich lesen, aber nicht beschreiben.

Beim gemeinsamen Zugriff auf den Speicher ergibt sich ein klassisches Synchronisationsproblem (zur Unterscheidung von der Uhrensynchronisation wird diese Synchronisation in folgende *Prozeßsynchronisation* genannt): Die relevanten Informationen sind

länger als ein Wort des Prozessors, damit ist es nicht möglich, atomar auf die Informationen zuzugreifen. Dies bedeutet, daß zwischen dem Zugriff auf ein Element und dem Zugriff auf ein anderes Element der Informationen ein anderer Teil des Computersystems die Kontrolle erlangen kann. Dabei ergibt sich ein Problem, welches sich wie folgt darstellt:

Angenommen, der Gerätetreiber will die virtuelle Uhr aktualisieren. Hierzu aktualisiert er ein Element der Struktur, beispielsweise zuerst die Steigung. Jetzt könnte es passieren, daß die Applikation den Prozessor zugeteilt bekommt, oder bei einem Multiprozessorsystem sogar auf dem anderen Prozessor läuft. Angenommen, die Applikation liest nun die Steigung und den Achsenabschnitt aus, so hat die Applikation inkonsistente Informationen, da die Steigung neuer ist als der Achsenabschnitt und die beiden Informationen nicht zueinander passen.

Die Prozeßsynchronisation muß von der Implementierung unbedingt behandelt werden, damit die Uhrensynchronisation funktioniert. Üblicherweise werden hierzu Semaphoren, Ports oder ähnliche Instrumente der Interprozeßkommunikation genutzt. Diese sind hier aber aus drei Gründen nicht anwendbar: Erstens ist es unter Windows NT nicht vorgesehen, daß eine Anwendung eine Prozeßsynchronisation mit einem Gerätetreiber auf diese Art und Weise erledigt, zweitens könnte eine defekte Applikation den Gerätetreiber endlos blockieren, wodurch die Stabilität des Systems nicht mehr gegeben wäre, und drittens benötigen diese Mechanismen, da sie durch Betriebssystemaufrufe implementiert werden, relativ viel Zeit. Daher wird ein Verfahren gesucht, welches einfach zu implementieren ist, keine unnötige Zeit beansprucht und den Gerätetreiber nicht blockieren kann.

Ein solcher Mechanismus existiert in der Form des sogenannten *double bufferings*. Für jede Information, die zwischen den Ebenen ausgetauscht werden soll, existieren hierbei zwei separate Speicherbereiche, die abwechselnd beschrieben werden. Dabei ist immer genau einer der beiden Puffer aktiv. Solange ein Puffer aktiv ist, darf er nicht beschrieben werden. Beim Lesen der Informationen liest die Leseroutine zuerst aus, welcher Puffer gerade aktiv ist, und holt ihre Informationen ausschließlich aus diesem Puffer. Die Abbildung 5-5 zeigt dies anhand eines an die Programmiersprache C angelehnten Pseudo-Codes. In Zeile 9 wird der gerade aktive Puffer abgefragt und zwischengespeichert. Mit dieser Information wird in Zeile 12 der Wert des aktiven Puffers ausgelesen. Obwohl dieses Auslesen unterbrochen werden kann, hat eine solche Unterbrechung keinen Einfluß mehr auf das Auslesen, da der in der Zeile 9 aktive Puffer auf jeden Fall benutzt wird, um die Informationen zu erhalten, selbst, wenn er vor oder während der Ausführung der Zeile 12 inaktiviert wird.

Beim Schreiben wird der inaktive Puffer beschrieben und danach der aktive Puffer inaktiviert, während der inaktive aktiviert wird. Dies zeigt die Abbildung 5-6 wieder anhand eines Pseudocodes. Vor dem Beschreiben wird bestimmt (Zeile 4), welcher Puffer gerade inaktiv ist. In Zeile 7 wird der inaktive Puffer beschrieben, und erst nach erfolgtem vollständigen Beschreiben wird der inaktive Puffer aktiviert (Zeile 10). Unter der Annahme, daß die Zeit zwischen dem Beschreiben der Puffer wesentlich größer ist als die

Zeit, die zum Auslesen eines Puffers benötigt wird, kann hiermit garantiert werden, daß immer nur konsistente Informationen aus dem aktiven Puffer entnommen werden. Diese Annahme kann im Kontext der Uhrensynchronisation getroffen werden. Die Puffer werden frühestens zum nächsten Synchronisationszeitpunkt neu beschrieben, also etwa alle *INT* Zeiteinheiten[13], das Auslesen hingegen erfolgt innerhalb einer vergleichsweise kurzen Zeit (unter einer Millisekunde).

Abbildung 5-5: *Lesen des Kommunikationspuffers*

Abbildung 5-6: *Beschreiben des Kommunikationspuffers*

[13] Hier wird die Annahme benutzt, daß die maximale Zeit, die zum Empfang einer Nachricht benötigt wird, wesentlich kleiner ist als das Intervall *INT*.

5.2 Implementierung des Protokolls

Bei dem im Rahmen dieser Diplomarbeit entwickelten Protokoll kommt es darauf an, die Länge des kritischen Pfades soweit wie möglich zu minimieren. Bei der Beschreibung des Protokolls in Kapitel 4 „Uhrensynchronisationsprotokoll für das Funknetzwerk“ wurde davon ausgegangen, daß der das Protokoll ausführende Prozeß von dem Erhalt einer Nachricht schon informiert würde, *bevor* die komplette Nachricht vorliegt. In diesem Abschnitt wird erläutert, wie dies in einem Gerätetreiber unter Windows NT erreicht werden kann.

Auf der Ebene der Gerätetreiber gibt es einen Mechanismus, der es erlaubt, daß der Zeitpunkt t_s wesentlich vor dem Zeitpunkt t_4 liegt. Um dies zu erläutern, bedarf es etwas Wissens über die Funktionsweise eines NDIS-Gerätetreibers unter Windows NT.

Die Abbildung 5-7 zeigt die wesentlichen Funktionen, die ein Gerätetreiber für Netzwerkkarten benötigt, um mit den umliegenden Schichten zu kommunizieren. Von den oberen Schichten bekommt der Treiber einen Aufruf *Send*(), wenn er ein Paket über das Medium versenden soll. Andererseits ruft der Treiber eine Funktion *Recv*() auf, die die obere Schnittstelle darüber informiert, daß ein Paket empfangen wurde, dabei wird gleichzeitig auch das empfangene Paket an die obere Schicht übermittelt. Aus der Kenntnis dieses Mechanismus folgt unmittelbar, daß eine übergeordnete Schicht nicht in der Lage sein kann, über das Ereignis des Empfanges informiert zu werden, bevor das Paket vorliegt.

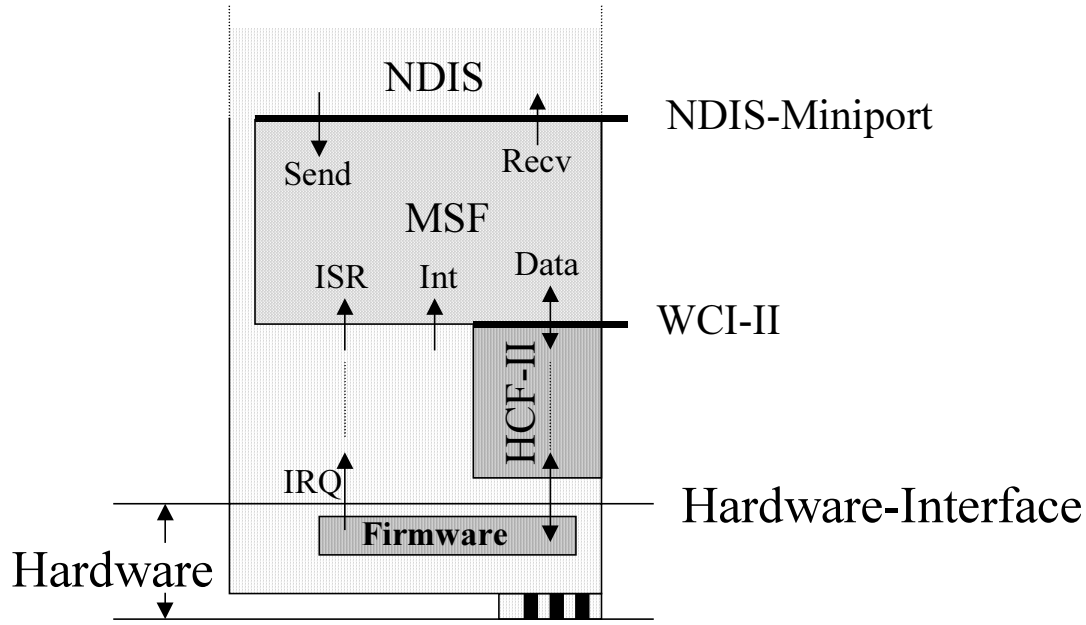


Abbildung 5-7: *Wesentliche Schnittstellen des NDIS-Gerätetreibers*

Die Schnittstelle zur unterliegenden Schicht ist anders organisiert. Will der Gerätetreiber etwas über das Medium versenden, dann ruft er Funktionen der HCF mit dem zu

übertragenden Inhalt auf; die Funktionen der HCF übertragen ihrerseits das zu übermittelnde Paket an die Netzwerkkarte.

Sobald die Netzwerkkarte eine Behandlung durch den Gerätetreiber erwünscht, erzeugt sie eine Unterbrechungsanforderung (*Interrupt Request*, IRQ), die vom Betriebssystem verarbeitet wird. In Reaktion hierauf ruft das Betriebssystem eine bestimmte Funktion des Gerätetreibers auf, die sogenannte *Interrupt Service Routine* (ISR). Ihre Aufgabe besteht darin, festzustellen, ob der Gerätetreiber den IRQ bearbeiten will oder nicht. Diese Information liefert die Routine an das Betriebssystem zurück. Wenn der Gerätetreiber seine Bereitschaft zur Bearbeitung des Ereignisses angezeigt hat, dann ruft das Betriebssystem später die *Deferred Processing Routine* (DPC) des Treibers auf, worauf die Gerätetreiber den Interrupt bearbeiten kann.

Dieser zweistufige Mechanismus zur Bearbeitung von Unterbrechungen wurde eingeführt, um die gemeinsame Nutzung von IRQs durch verschiedene Geräte zu ermöglichen: Die Gerätetreiber müssen beim Start anmelden, auf welchen Ressourcen sie agieren. Dabei melden sie auch die benutzten Interrupts an, wobei mehrere Gerätetreiber den gleichen Interrupt anmelden können. Wird nun ein IRQ vom Betriebssystem erkannt, dann ruft es die ISR der angemeldeten Gerätetreiber nach und nach auf, um festzustellen, welcher Gerätetreiber zuständig ist. Wenn es einen Zuständigen erkannt hat, dann ruft es dessen DPC auf.

Da das Betriebssystem schnell auf eingehende Interrupts reagieren muß, ist es wichtig, daß die ISR, von denen unter Umständen mehrere aufgerufen werden müssen, möglichst wenig Zeit in Anspruch nehmen. Dies liegt daran, daß der Zeitaufwand zur Bearbeitung der ISR wesentlich bestimmt, wie lange es dauert, bis ein passender Gerätetreiber den Interrupt bearbeiten kann. Daher hat der Aufruf der ISR bei Windows NT höchste Priorität, und die ISR kann auch nicht unterbrochen werden. Damit eine schnelle Arbeitsweise des Systems möglich ist empfiehlt Microsoft, möglichst wenig Funktionalität in der ISR zu nutzen.

Dies sind ideale Voraussetzungen für das hier entwickelte Protokoll. Die Aktion, die beim Protokoll im zeitkritischen Pfad liegt – das Ermitteln eines Zeitstempels –, ist sehr schnell und kann dadurch problemlos in der ISR untergebracht werden. Sobald die Netzwerkkarte ein Paket empfängt, generiert sie einen IRQ, der zum schnellen Aufruf der ISR führt. Innerhalb der ISR kann der Gerätetreiber nun direkt einen Zeitstempel ermitteln, bevor die normale Funktion der ISR ausgeführt wird. Der Zeitpunkt t_s liegt damit innerhalb der Bearbeitung der ISR. Sorgt man dafür, daß der IRQ der Netzwerkkarte nicht mit anderen Geräten geteilt wird, dann kann ein schneller, zeitnaher Aufruf der ISR im Rahmen der Möglichkeiten von Windows NT garantiert werden. Messungen ergaben, daß der Aufruf der ISR für die gleiche Nachricht auf verschiedenen Stationen einen zeitlichen Abstand von unter 50 µs besitzt, der Abstand für den Aufruf hingegen der DPC um eine Zehnerpotenz höher liegt. Die Möglichkeit, die ISR zum Zeitstempeln zu nutzen, ist ein großer Vorteil des hier entwickelten Protokolles gegenüber des IEEE-Protokolles. Dieser Umstand erlaubt es, eine viel niedrigere Varianz des zeitkritischen Pfades zu erhalten als beim IEEE-Protokoll.

Man beachte, daß zum Zeitpunkt des Aufrufs der ISR das empfangene Paket im Gerätetreiber noch nicht vorliegt. Vielmehr muß der Gerätetreiber in der DPC mit vergleichsweise langsamen Zugriffen auf die Netzwerkkarte das Paket aus der Karte herausholen. Dieser Vorgang dauert recht lange, da ein Wort nach dem anderen mit diesem langsamen Zugriff aus der Karte geholt werden muß. Weiterhin gilt:

1. Im Paket liegt eine variable Anzahl von Worten vor;

2. Die DPC kann vom Betriebssystem unterbrochen werden, beispielsweise, weil ein anderer IRQ von einer anderen Karte erzeugt wurde;

3. Interne Synchronisationsmechanismen innerhalb des Gerätetreibers (kritische Bereiche z. B. durch Semaphoren und Spinlocks) müssen sicherstellen, daß durch die Parallelität des Zugriffs auf den Gerätetreiber von oberen Schichten und von unteren Schichten keine Inkonsistenzen der Daten auftreten. Diese Synchronisationsmechanismen können die Bearbeitung erheblich verzögern.

Hieraus ergibt sich, daß die DPC eine große Varianz in der Ausführungszeit besitzt (und damit auch die Verarbeitung des Empfanges in höheren Schichten). Der Zeitpunkt t_4, an dem das Umstellen der Uhr erfolgt, liegt innerhalb der Bearbeitung der DPC und hat damit ebenfalls eine hohe Varianz.

Die Nutzung der ISR hat einen weiteren großen Vorteil. Die Netzwerkkarte erzeugt nämlich nicht nur dann IRQs, wenn sie Pakete empfängt, sondern auch, wenn ein Paket versendet wurde. Auch dieser Zeitpunkt kann mit einer geringen Varianz mit einem Zeitstempel versehen werden. Dieses wird von dem Protokoll benötigt, da der Master einen Zeitstempel erzeugen muß, sobald die eigene Nachricht versendet wurde. Da mehrere Quellen für den Aufruf der ISR möglich sind, ist es notwendig, daß in der DPC getestet wird, aus welchem Grund die ISR aufgerufen wurde, und entsprechend der Quelle muß der Zeitstempel weiterverarbeitet werden.

Daher wäre die Nutzung des ISR zum Zeitstempeln des Versendens ideal, wenn der Sendeinterrupt von der Hardware zu dem Zeitpunkt erzeugt würde, an dem eine empfangende Station einen Empfangsinterrupt erzeugt, was technisch durchaus zu erreichen ist[14] – allerdings zeigten erste Messungen, daß dies bei der für die Implementierung benutzten Netzwerkkarte *nicht* der Fall ist. Stattdessen wird der Sendeinterrupt schon wesentlich früher generiert. Dies hängt offenbar damit zusammen, daß die Netzwerkkarte so entwickelt wurde, daß sie auf Durchsatz optimiert ist. Hierzu besitzt sie interne Puffer, die ein zu sendendes Paket annehmen. Offenbar wird der Sendeinterrupt schon dann erzeugt, wenn der Puffer so weit gelehrt wurde, daß ein neues Paket aufgenommen

[14] In einer ersten Annäherung beispielsweise durch einen „Loopback“ vom (digitalen) Sendeteil zum (digitalen) Empfängerteil in der Hardware der Netzwerkkarte, wobei der Hochfrequenz- (HF-) Sende- und Empfangsteil der Netzwerkkarte umgangen wird. Dabei werden die Verzögerungen durch den HF-Teil ignoriert, diese liegen aber voraussichtlich nur in der Größenordnung von Bruchteilen von Mikrosekunden.

werden kann. Damit kann der Durchsatz der Netzwerkkarte erhöht werden, da bei mehreren zu versendenden Paketen die Zeit, die das Medium ungenutzt bleibt, minimiert werden kann. Diese Minimierung erfolgt dadurch, daß durch den Puffer bei einem freien Medium die nächste Nachricht gesendet werden kann, ohne daß vorher die Nachricht noch in die Netzwerkkarte transportiert werden muß, wodurch Zeit ungenutzt verstreichen würde.

Aus diesem Grunde ist eine 100%ige Implementierung des Protokolls, wie es in Kapitel 4 „Uhrensynchronisationsprotokoll für das Funknetzwerk“ beschrieben wurde, in dieser Umgebung nicht mit der entsprechenden Präzision möglich. Es muß herausgestellt werden, daß dies keine prinzipielle Einschränkung des Protokolls ist, sondern lediglich eine Einschränkung, welche sich durch die Implementierungsumgebung ergibt. Es wäre ohne großen Aufwand möglich, den Interrupt auf den Sendezeitpunkt zu verschieben oder dann zumindest einen weiteren Interrupt zu erzeugen. Hierzu bedürfte es aber des Zugriffs auf die Hardware oder Firmware der Netzwerkkarte, welcher nicht vorliegt.

Um diesen Nachteil der Implementierungsumgebung auszugleichen, wurde eine Variante des Protokolls des Kapitels 4 implementiert. Anstelle einer werden für jede Synchronisationsrunde zwei Nachrichten versendet, eine Indikationsnachricht und eine, die die Zeitstempel enthält. Die Nachricht mit den Zeitstempeln („Synchronisationsnachricht“) wird vom Master versendet; die Indikationsnachricht hingegen von einer an der Synchronisation nicht beteiligten Station, dem Indikationssender (IS). Damit die Eigenschaften des Protokolls des Kapitels 4 erhalten bleiben, wurden folgende Maßnahmen getroffen: Die Indikations- und die Synchronisationsnachricht werden in unmittelbarer zeitlicher Nähe versendet und als eine einzige Nachricht betrachtet (vgl. Abbildung 5-8). Hieraus folgt, daß die Synchronisationsnachricht die Zeitstempel für die vorherigen Runden enthält, nicht für die aktuelle, da dieser Zeitstempel bei der Betrachtung der beiden Nachrichten als Einheit noch nicht bekannt sein könnte. Weiterhin werden der IS und der AP in unmittelbarer räumlicher Nähe plaziert, damit beide so gut wie möglich den gleichen Empfangsbereich überdecken, wie es beim Protokoll des Kapitels 4 der Fall wäre. Es gibt allerdings die Einschränkung, daß eine Synchronisationsnachricht beim Master nicht ankommen könnte, da sie auf dem Weg vom IS zum Master verlorengeht. Dies bedeutet eine Verschlechterung gegenüber dem theoretisch vorgestellten Protokoll, da der Master immer in der Lage wäre, das Versenden seiner eigenen Nachricht zu erkennen und mit einem Zeitstempel zu versehen.

Es sei darauf hingewiesen, daß das Protokoll, wie es in Kapitel 4 „Uhrensynchronisationsprotokoll für das Funknetzwerk“ vorgestellt wurde, auf dem Funknetzwerk auch genau so implementierbar ist. Die oben gemachte Abwandlung der Implementierung ist nur deshalb notwendig, weil die Netzwerkkarte über eigene Logik verfügt die versucht, den Prozessor bei der Arbeit zu entlasten. Sofern man Zugriff auf diese Logik besitzt, also auf die Hard- und Firmware, ist es sehr gut möglich, den Zeitpunkt des physikalischen Versendens genau zu bestimmen, so daß das ursprüngliche Protokoll implementiert werden kann.

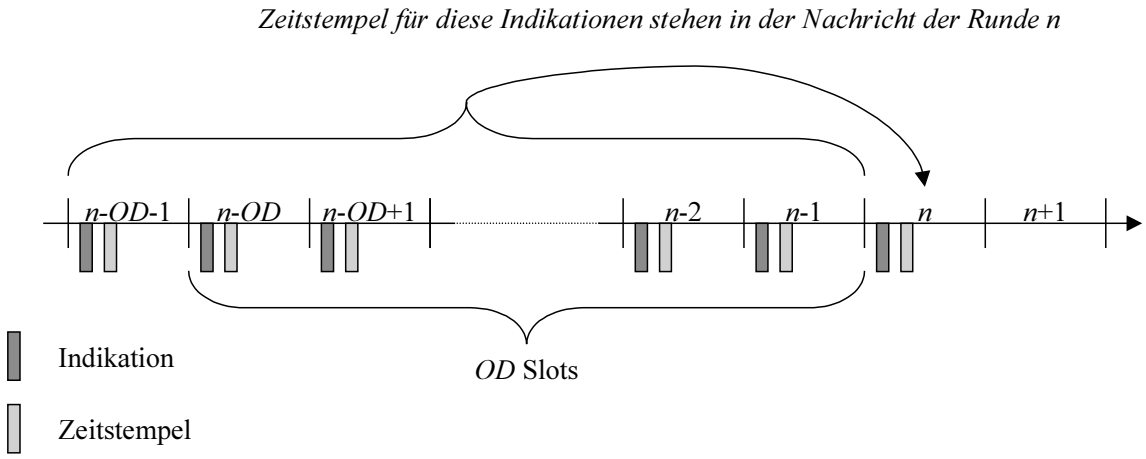


Abbildung 5-8: *In der Synchronisationsnachricht enthaltene Zeitstempel beim implementierten Protokoll*

6 Messungen

In diesem Kapitel wird untersucht, welche Präzision mittels des hier implementierten Protokolls erreicht wird. Das Protokoll nach dem IEEE-Standard wurde ebenfalls implementiert, damit vergleichende Messungen zwischen den beiden Verfahren möglich sind.

Neben der Messung der Präzision werden die Parameter, die für die Bewertung der theoretischen Analyse des Protokolls wichtig sind und im Abschnitt 4.5.2 „Erreichte Präzision“ offengeblieben sind, bestimmt, damit diese Bewertung erfolgen kann. Diese Parameter sind die Schwankungen des zeitkritischen Pfades und die Drift der physikalischen Uhren.

Bevor die Messungen präsentiert werden wird der Meßaufbau erläutert, mit dem die Ergebnisse ermittelt wurden.

6.1 Meßaufbau

Der Meßaufbau, mit dem alle folgenden Messungen durchgeführt wurden, besteht aus IBM-PC-kompatiblen Laptops mit Pentium 133 MHz-Prozessoren. Auf den Stationen lief als Betriebssystem Windows NT, die Stationen waren mittels Funknetzwerkkarten WaveLAN/IEEE der Firma Lucent miteinander verbunden, wobei die Gerätetreiber wie in Kapitel 5 „Implementierung“ beschrieben um die Uhrensynchronisation ergänzt wurden.

Zur Messung einer globalen Zeit und der Ereignisse, die zu ihrer Erlangung benutzt werden, benötigt man eine weitere globale Zeit, welche eine bessere Präzision besitzen muß. Würde eine solche globale Uhrzeit zur Verfügung stehen, so stellte sich die Frage, inwiefern die zu messende globale Uhrzeit überhaupt benötigt wird.

Da im Rahmen dieser Arbeit eine verteilte globale Uhrzeit mit genügender Präzision nicht zur Verfügung stand, wird stattdessen ein zentralistischer Ansatz gewählt, d. h., eine zentrale Station ('Meßrechner') vermißt die verteilten Ereignisse. Der Meßrechner realisiert gewissermaßen die globale Zeit für das SUT. Damit dies möglich ist, müssen die zu vermessenden Stationen die Möglichkeit besitzen, dem Meßrechner das Eintreten von Ereignissen mitzuteilen.

Es wird hier als Lösung der Anschluß der Stationen über die bei jedem Standard-PC vorhandene parallele Schnittstelle gewählt (siehe Abbildung 6-1). Auf dem Meßrechner wird diese Schnittstelle als Eingang programmiert, auf den zu vermessenden Stationen im zu vermessenden System (*system under test*, SUT) entsprechend als Ausgang (zur Programmierung der parallelen Schnittstelle siehe [St 95]). Jede zu vermessende Stationen besitzt nun dedizierte Leitungen zum Meßrechner, die nur er verändern und nur der Meßrechner auslesen kann und mit denen sie dem Meßrechner das Eintreten von den zu vermessenden Ereignissen signalisiert. Auf dem Meßrechner läuft ein Programm ähn-

lich eines digitalen Speicheroszilloskops, welches auf Veränderungen der Eingänge der parallelen Schnittstelle reagiert und diesen Ereignissen lokale Zeitstempel zuweist.

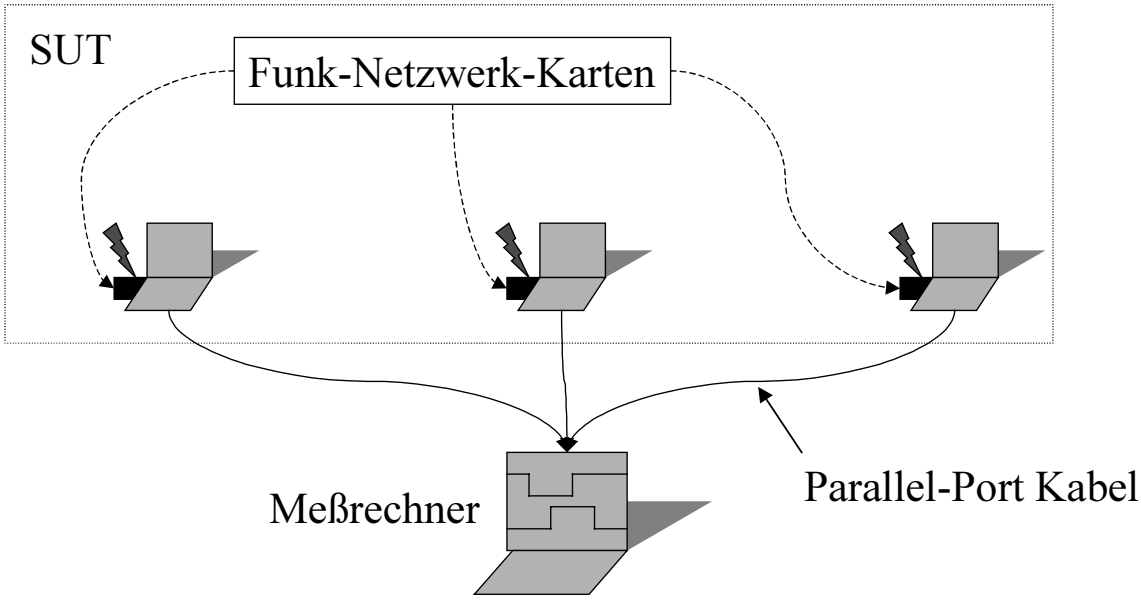


Abbildung 6-1: *Meßaufbau*

In der Meßschleife schreibt das Programm des Meßrechners die Zeitpunkte, an denen die parallele Schnittstelle verändert wurde, in einen Puffer, der nach Beendigung des Messens offline ausgewertet werden kann. Die Struktur der Meßschleife kann der Abbildung 6-2 entnommen werden. Diese Schleife wurde in Assembler kodiert um eine maximale Geschwindigkeit zu erreichen, da sie zeitkritisch ist. Die Endebedingung der Schleife ist dabei das Auftreten eines Interrupts, so daß die Schleife nur aus dem Vergleich des alten mit dem neuen Wert des Einganges der parallelen Schnittstelle besteht, wodurch eine extrem kurze Laufzeit der inneren Schleife erreicht wird. Um eine hohe Geschwindigkeit zu erreichen, wurde in der Meßschleife als Zeitstempel lediglich der Wert des TSC ausgelesen. Die Rechnungen, um aus diesem Wert Zeiten in der Einheit Sekunde zu erhalten, wurden später offline ausgeführt, d. h. nach Beendigung der Meßschleife.

Abbildung 6-2: *Struktur des Auswerteprogrammes auf dem Meßrechner*

Die Abbildung 6-3 zeigt die Struktur des Programmes, welches auf einer Station im SUT ein Ereignis erzeugt. Es wird der alte Inhalt der parallelen Schnittstelle geholt, das Bit, welches die gewünschte zu verändernde Leitung repräsentiert, verändert und der

neue Inhalt der parallelen Schnittstelle geschrieben. Da der Treiber die einzige Stelle ist, die den parallelen Port während des Messens beschreibt, kann eine Optimierung vorgenommen werden: Der alte Wert der parallelen Schnittstelle wird im Speicher gehalten, wodurch die Signalisierung schneller möglich ist, da Lesezugriffe im Speicher schneller sind als Lesezugriffe auf Hardwareports.

Abbildung 6-3: *Erzeugen eines Ereignisses auf einer Station im SUT*

Es ist gewährleistet, daß die Zeitstempel innerhalb der Messung vergleichbar sind, da der Meßrechner als einziger die Zeitstempel für die Vermessung nehmen kann. Die Ungenauigkeiten und damit die Präzision der Messung ergeben sich demnach durch folgende Parameter:

1. die Verzögerung (und Varianz) beim Beschreiben der parallelen Schnittstellen auf den zu vermessenden Stationen;

2. die Laufzeit der Ereignisse der zu vermessenden Stationen auf den Leitungen;

3. die Verzögerung (und Varianz) beim Lesen der parallelen Schnittstellen auf dem Meßrechner;

4. die Größe der inneren Schleife auf dem Meßrechner, die die parallele Schittstelle ausliest;

5. die Drift der Uhr des Meßrechners von der realen Zeit.

Um den Fehler, der durch die Punkte 1 bis 4 verursacht wird, auszumessen, wurde die sogenannte Round-Trip-Delay (RTD) gemessen.

Die RTD ist die Zeit, die ein Signal benötigt, um von einer Station aus über den Versuchsaufbau zu dem Meßrechner und von dort über den gleichen Mechanismus wieder zurück zu gelangen[15]. Unter der Annahme, daß das Signal für den Hinweg genauso viel Zeit benötigt wie für den Rückweg, ist die Hälfte der RTD ein Maß für die mittels dieses Meßaufbaus erreichte Präzision.

Der Meßaufbau hierzu sieht folgendermaßen aus: Eine Station im SUT nimmt einen Zeitstempel t_0, verändert einen Ausgang ihrer parallelen Schnittstelle und arbeitet hiernach die Meßschleife ab, die normalerweise von dem Meßrechner ausgeführt wird. Hierauf reagiert der Meßrechner wie gewohnt, in dem er Daten in seinen Meßpuffer schreibt. Direkt im Anschluß hieran erzeugt er wiederum ein Signal, welches als Ereig-

[15] Man kann auch von einer Ping-Pong-Zeit reden.

nis für die Meßschleife des ursprünglichen Senders dient. Daraufhin schreibt dieser Daten in seinen Meßpuffer, und als Abschluß nimmt er wieder einen Zeitstempel t_1. Die RTD berechnet sich dann als Differenz t_1 - t_0.

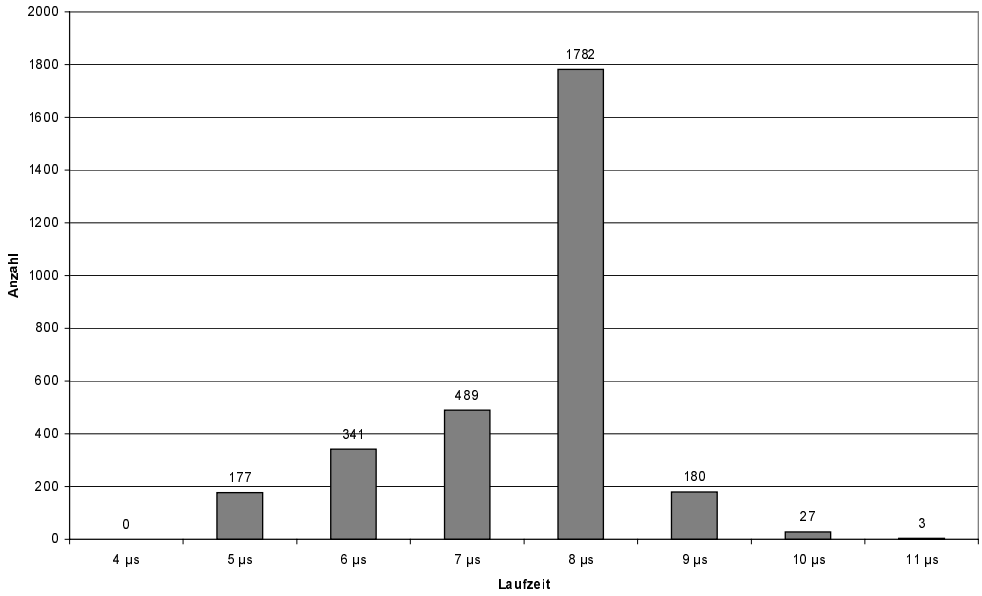


Abbildung 6-4: *Round-Trip-Delay des Meßaufbaus*

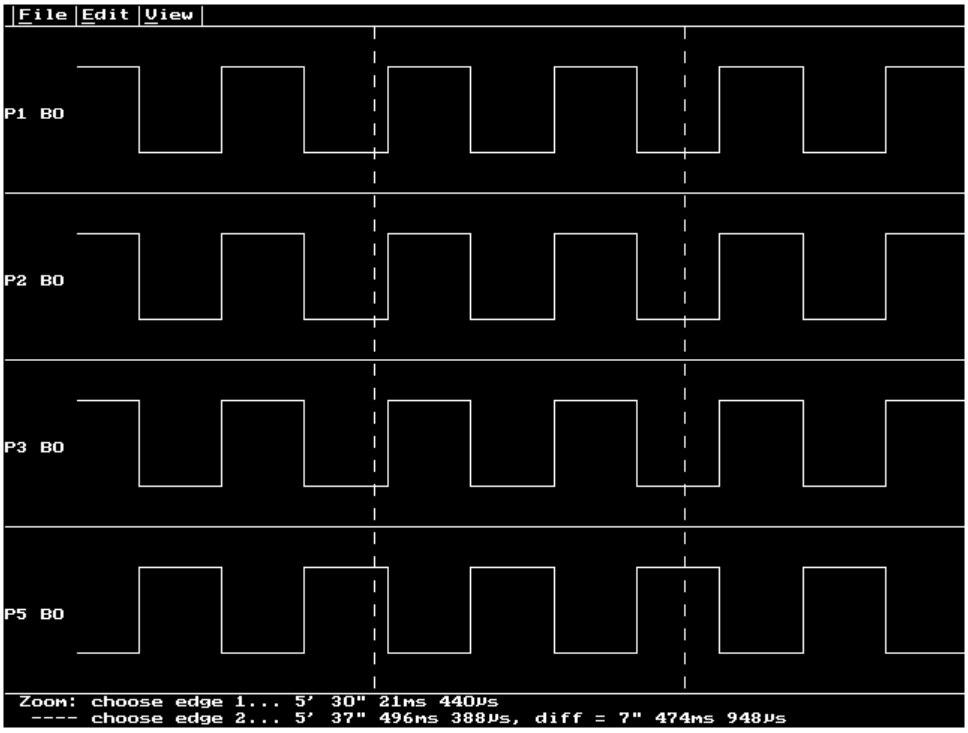


Abbildung 6-5: *Grafische Auswertung der Zustandsänderungen auf dem Messrechner*

Das Ergebnis der Messung kann der Abbildung 6-4 entnommen werden. Es wurde ca. 3000 mal das Signal von der sendenden Station aus versendet. Die RTD betrug in den Messungen zwischen 5 µs und 11 µs, wobei der Durchschnitt bei 7,5 µs lag. Die Verteilungen bei Messungen von anderen Stationen im SUT (und hierzu passend anderen Signalwegen) aus ergaben beinahe identische Verteilungen, also ist die Messung weitgehend unabhängig von der betrachteten Station oder dem benutzten Signalweg. Nach den Messungen ist die Präzision der zentralen Uhr besser als 5,5 µs.

Um die Auswertung der durch den Meßaufbau erhaltenen Werte zu ermöglichen, wurden zwei Module für das Meßprogramm geschrieben. Das erste Modul erlaubt die manuelle graphische Auswertung der Zustandsänderungen auf der parallen Schnittstelle im Dialogbetrieb (s. Abbildung 6-5), das andere Modul exportiert die ermittelten Daten in ein Text-Format, so daß sie von anderen Programmen (beispielsweise Microsoft Excel) weiterverarbeitet werden können.

6.2 Messung der Drift der physikalischen Uhren

Die physikalischen Uhren wurden so implementiert, wie es in Abschnitt 5.1.2 „Realisierung der Uhren“ beschrieben wurde. Die in diesem Abschnitt beschriebene Messung wurde vorgenommen, um die Größenordnung der Driftrate der physikalischen Uhren zu bestimmen. Hierzu wurde für jede Station p zu einem Zeitpunkt t_0 der Wert der physikalischen Uhr der Station ausgelesen, dieser Wert wird $T_0^{(p)}$ genannt. Zu einem späteren Zeitpunkt t_1 wird der entsprechende Wert $T_1^{(p)}$ der physikalischen Uhr ausgelesen. Die Drift der Station p ergibt sich dann aus dem Term $\frac{T_1^{(p)} - T_0^{(p)}}{t_1 - t_0}$, die Driftrate ρ_p entsprechend aus $\rho_p = \left| \frac{T_1^{(p)} - T_0^{(p)}}{t_1 - t_0} - 1 \right|$.

Um eine hohe Meßgenauigkeit zu erreichen ist es zweckmäßig, den Abstand $t_1 - t_0$ möglichst groß zu wählen, damit die Meßfehler bei der Bestimmung der Werte nur einen möglichst geringen Einfluß haben können. Daher wurde ein Abstand von 24 h = 86400 s gewählt.

Die Signalisierung der Zeitpunkte t_0 und t_1 erfolgte wie in Abschnitt 6.1 „Meßaufbau“ beschrieben, allerdings wurde die Richtung der Signalisierung umgekehrt, d. h. der Meßrechner erzeugte jeweils ein Signal, welches von den Slaves erkannt und mit einem Zeitstempel versehen wurde.

Der Meßrechner besaß als Zeitquelle die offizielle Zeit in Deutschland, wie sie von der physikalisch-technischen Bundesanstalt in Braunschweig über eine Atomuhr generiert und von einem Sender in Mainflingen bei Frankfurt/Main über die Frequenz 77,5 kHz über Langwelle ausgestrahlt wird. Hierzu wurde der Meßrechner mit einem Empfänger für dieses Signal ausgestattet, für die der Hersteller eine maximale Abweichung von unter 2 ms angibt. Die Messung der Uhren der Slaves kann nach Abschnitt 6.1 „Meßaufbau“ mit einer Ungenauigkeit von nicht mehr als 6 µs ausgelesen werden.

Damit ergibt sich der maximale Fehler dieser Messungen folgendermaßen[16]: Die Zeitspanne $T_1^{(p)}$-$T_0^{(p)}$ läßt sich mit einem Fehler von 6 µs bestimmen, die Zeitspanne t_1-t_0 mit einem Fehler von 2 ms. Damit ergibt sich eine maximale Driftrate von $\left|\frac{86400\text{ s} + 6\text{ µs}}{86400\text{ s} - 2\text{ ms}} - 1\right| < 3\cdot10^{-8}$ bzw. $\left|\frac{86400\text{ s} - 6\text{ µs}}{86400\text{ s} + 2\text{ ms}} - 1\right| < 3\cdot10^{-8}$, also ist der maximale Fehler dieser Messungen kleiner als 10^{-7}.

Die Messung wurde iterativ durchgeführt, wobei die Angabe der Taktfrequenzen der beteiligten Stationen bei jedem neuen Iterationsschritt angepaßt wurden. Der maximal ermittelte Meßwert aller Stationen betrug in der Messung etwa 86401,473 s, der minimale Meßwert entsprechend etwa 86399,132 s. Dies entspricht gemessenen Driftraten der physikalischen Uhren von $\left|\frac{86401{,}473}{86400} - 1\right| < 2\cdot10^{-5}$ bzw $\left|\frac{86399{,}132}{86400} - 1\right| \approx 1\cdot10^{-5}$. Also wird als maximale ermittelte Driftrate $2\cdot10^{-5}$ angenommen.

6.3 Benötigte Zeit zum Ermitteln eines Zeitstempels

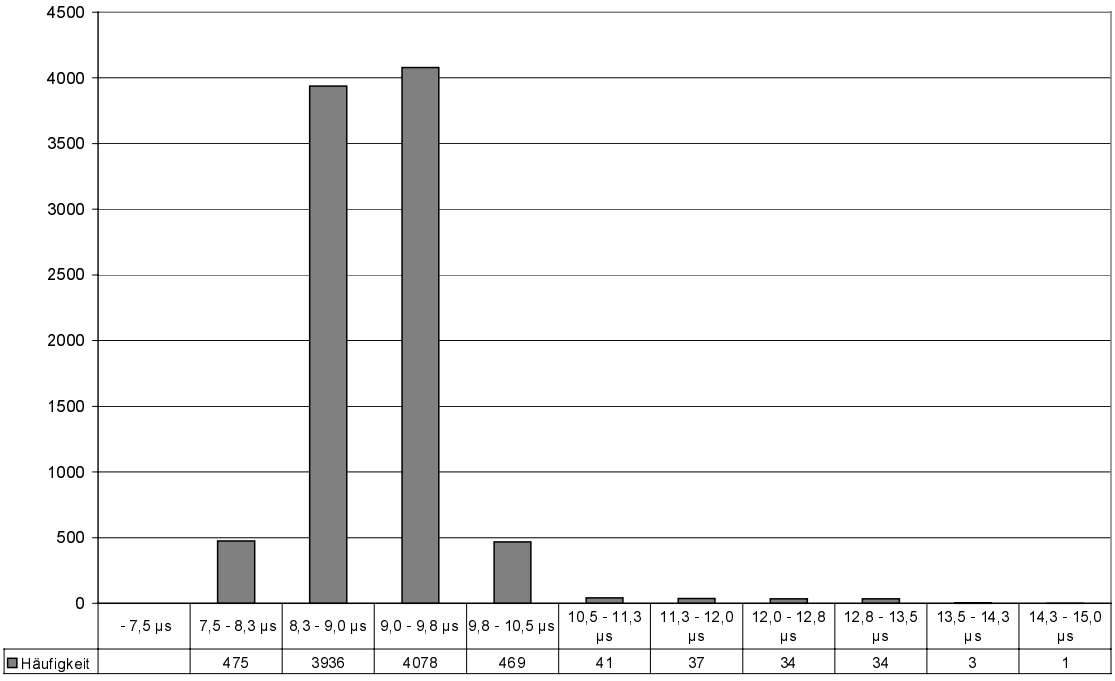
In Abschnitt 5.1.3 „Kommunikation zwischen Gerätetreiber und Applikation“ wurde dargestellt, wie die Kommunikation zwischen einer Applikation und dem Gerätetreiber aussieht. Vorrangiges Ziel dabei war es, daß sowohl die Applikation als auch der Gerätetreiber schnellen Zugriff auf die aktuelle Zeit haben sollten. Daher wurde gemessen, wie lange es dauert, bis ein Zeitstempel zur Verfügung steht.

Diese Messung fand auf einer einzigen Station statt. Es wurde ein Zeitstempel t_0 ermittelt, dann die Funktion zum Auslesen der aktuellen Zeit t' aufgerufen und zum Schluß ein weiterer Zeitstempel t_1 ermittelt. Die Zeitstempel t_0 und t_1 wurden dabei ohne weitere Bearbeitung ermittelt, d. h., es wurde lediglich der Wert des TSCs ermittelt. Die Differenz t_1 - t_0 - *diff*, umgerechnet in Sekunden, ergibt dann die Laufzeit der Routine. Der Korrekturwert *diff* ist deshalb notwendig, weil bei der Messung ohne eingeschlossenen Befehl schon Taktzyklen vergehen, da der Befehl zum Lesen des TSCs einige Zyklen zur Abarbeitung benötigt. Diese Zyklen werden mittels des Parameters *diff* in den Berechnungen berücksichtigt.

Es wurde etwa 9000 mal die Laufzeit ermittelt. Dabei wurde nach einem Timeout von etwa einer Sekunde einmalig ein Zeitstempel ermittelt und die Laufzeit für diese Routi-

[16] Es wird eine näherungsweise Fehlerrechnung vorgenommen, indem lediglich untersucht wird, wie groß die Auswirkungen der Fehler sind, wenn der gemessene und der erwartete Wert für die Zeitspanne gleich sind, also eine Driftrate von 0 vorläge. Diese Betrachtung reicht aus, da sie sehr großzugügig nach oben abgeschätzt wird und die tatsächlichen Werte sich von den angenommenen Werten nicht sehr unterscheiden.

ne bestimmt[17]. Das Ergebnis der Messung in der Applikation zeigt die Abbildung 6-6. Sie stellt dar, wie häufig die Laufzeit in bestimmten Wertebereichen lag. Man erkennt, daß mehr als 98 % der Messungen Laufzeiten von unter 10,5 µs ergaben. Die längste gemessene Laufzeit betrug weniger als 15 µs.



	- 7,5 µs	7,5 - 8,3 µs	8,3 - 9,0 µs	9,0 - 9,8 µs	9,8 - 10,5 µs	10,5 - 11,3 µs	11,3 - 12,0 µs	12,0 - 12,8 µs	12,8 - 13,5 µs	13,5 - 14,3 µs	14,3 - 15,0 µs
Häufigkeit		475	3936	4078	469	41	37	34	34	3	1

Abbildung 6-6: *Benötigte Zeit zum Ermitteln eines Zeitstempels*

Die Unterschiede in den Laufzeiten der Rechnung lassen sich damit erklären, daß die Laufzeit auf modernen Prozessoren mit Cache-Speichern stark davon abhängt, ob eine Routine das erste mal aufgerufen wird oder nicht, und welche Funktionen zwischen den Aufrufen vom Prozessor ausgeführt wurden.

Die gleiche Messung wurde auch in dem Gerätetreiber vorgenommen und lieferte qualitativ und quantitativ in etwa das gleiche Ergebnis. Aufgrund der Messungen ergibt sich, daß das Ziel, mit einer geringen Laufzeit einen Zeitstempel zu ermitteln, von der Implementierung erreicht wurde.

[17] Also: Es wurde keine einfache Schleife programmiert, die 9000mal die Routine aufruft, sondern eine Messung unter „echten“ Bedingungen vorgenommen, wobei andere Betriebssystemfunktionen auf dem Computer arbeiten konnten.

6.4 Messung des zeitkritischen Pfades des Protokolls

Die grundlegende Idee des hier vorgestellten Protokolls ist es, die Länge des zeitkritischen Pfades zu verkürzen. Dies ist der große Vorteil gegenüber dem IEEE-Protokoll; in diesem Abschnitt wird ermittelt, welche Verbesserung dies in Bezug auf die Schwankungen der Länge des kritischen Pfades bewirkt. Der Betrag dieser Schwankung ist weiterhin notwendig, um ihn in die Formel für die erreichte Präzision einzusetzen, womit ein Wert für die Präzision des hier vorgestellten Protokolls erhalten werden kann.

Es soll noch einmal betont werden, daß nicht die Länge des zeitkritischen Pfades minimiert werden muß, sondern vielmehr die Differenz zwischen dem spätesten und dem frühesten auftretenden Empfang auf den verschiedenen Stationen. Daher ist es sinnvoll, genau diese Schwankungen zu messen. Hierzu wird folgender Meßaufbau gewählt: Es wird das implementierte Uhrensynchronisationsprotokoll ausgeführt, wobei ein Master mehrere Slaves synchronisiert. Unmittelbar vor der Stelle, an der der Zeitstempel auf dem Master oder auf dem Slave genommen wird, d. h. zum Zeitpunkt[18] t_s, wurden beim Master und bei den Slaves Meßpunkte eingefügt, an denen Ereignisse für den Meßrechner erzeugt werden. In der Auswertung wurde nun die Differenz aus dem spätesten und dem frühesten aufgetretenen Zeitpunkt einer Synchronisationsrunde berechnet.

Im Rahmen dieser Diplomarbeit wurde auch das Protokoll nach dem IEEE-Standard implementiert. Um eine Vergleichbarkeit zu erreichen, wurde es in der gleichen Schicht wie das hier entwickelte Protokoll implementiert und vermessen. Unmittelbar vor dem Aufruf der Sendeaufforderung wird ein Zeitstempel genommen. Sobald das Paket in der INT-Routine des Treibers verfügbar ist, wird die Synchronisation vorgenommen. Auch dieses Protokoll wurde vermessen, so daß ein Vergleich zwischen den Protokollen stattfinden kann. Hierzu wurde unmittelbar vor dem Zeitpunkt t_4, an dem die Uhrzeit gesetzt wird, ein Ereignis eingefügt, so daß auch hier die Differenz aus dem spätesten und dem frühesten aufgetretenen Zeitpunkt einer Synchronisationsrunde berechnet werden kann.

Die Ergebnisse der Messung können der Abbildung 6-7 entnommen werden. Es wird dargestellt, welche minimalen und maximalen Abweichungen auf den einzelnen Stationen an den Zeitpunkten t_s (für das hier entwickelte Protokoll) bzw. t_4 (für das IEEE-Protokoll) vorliegen. Man sieht, daß beim IEEE-Protokoll die Abweichungen durchschnittlich 76 µs betragen, allerdings einzelne Ausbrecher bis 910 µs vorlagen. Beim hier entwickelten Protokoll waren die Abweichungen mit durchschnittlich lediglich 11 µs wesentlich niedriger, und selbst das Maximum von 43 µs liegt noch klar unterhalb des durchschnittlichen Wertes des IEEE-Protokolls.

In der Abbildung ist weiterhin die Standardabweichung angegeben. Die Standardabweichung ist ein Maß für die Verteilung der empirisch ermittelten Werte um den Mittelwert. Eine kleine Standardabweichung bedeutet, daß die Meßwerte zum größten Teil nah am Mittelwert liegen. Eine große Standardabweichung hingegen zeigt, daß die

[18] Die Bezeichnugen der Zeitpunkte wurde aus der Beschreibung des Protokolles in Kapitel 4 „Uhrensynchronisationsprotokoll für das Funknetzwerk“ übernommen.

Meßwerte stärkere Abweichungen vom Mittelwert besitzen. Die Standardabweichung beim IEEE-Protokoll ist mit 147 µs relativ gering; dies bedeutet, daß viele der gemessenen Werte in der Nähe des Durchschnitts, also relativ wenige nah am Maximum liegen. Beim eigenen Protokoll ist die Standardabweichung mit 7 µs noch wesentlich geringer. Dies bedeutet, daß auch in der Verteilung der Meßwerte das hier entwickelte Protokoll dem IEEE-Protokoll stark überlegen ist.

Man erkennt unmittelbar, welche große Auswirkungen die Verkürzung des kritischen Pfades hat. Da die Varianz des zeitkritischen Pfades eine Unterschranke für die erreichte Präzision angibt, ist eine Synchronisation mit einer Präzision im Sub-Millisekunden-Bereich auf dieser Implementierungsebene mit dem IEEE-Protokoll nicht möglich.

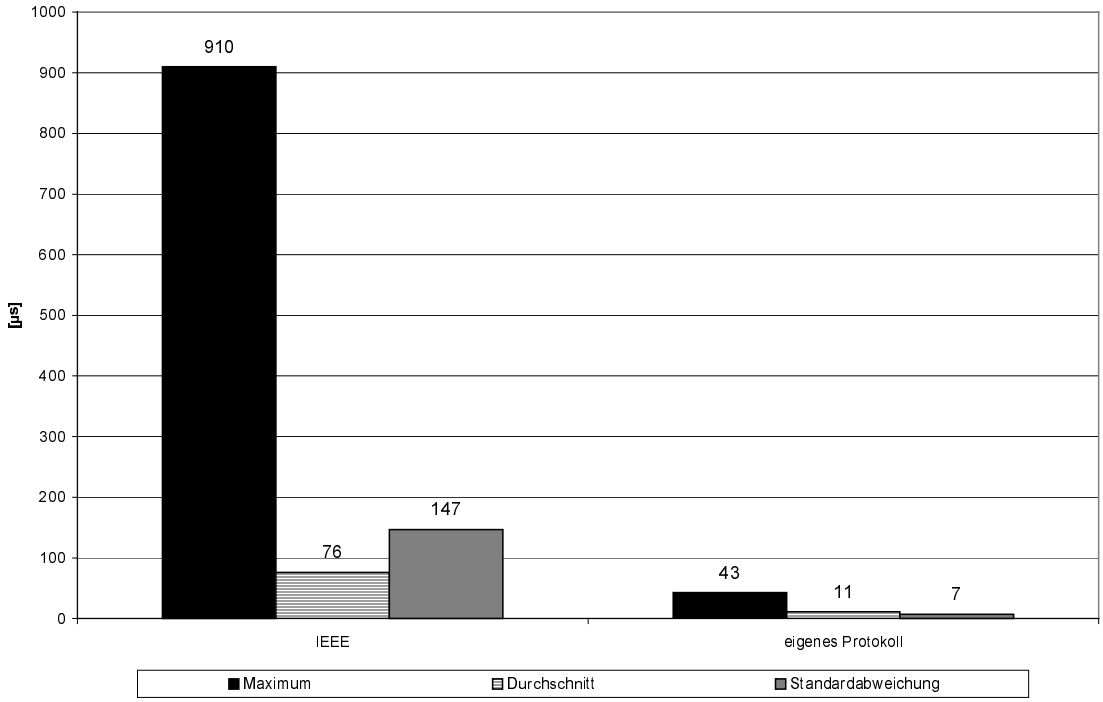


Abbildung 6-7: *Varianz des zeitkritischen Pfades*

6.5 Erreichte Präzision

In diesem Abschnitt werden die Ergebnisse der Messungen präsentiert, mit denen die Präzision der Protokolle bestimmt wurden.

Bei der Messung der Präzision der Uhren einer Menge von Stationen ergibt sich die Frage, wie die verteilten Uhren verglichen werden. In Abschnitt 6.1 „Meßaufbau“ wurde beschrieben, wie ein globaler Meßrechner benutzt werden kann, um Ereignisse zeitlich in Bezug zueinander zu setzen. Dieser Aufbau reicht aus, um beobachtbare Ereignisse auf mehreren Stationen zu vergleichen. Will man hingegen die Präzision bei der

Uhrensynchronisation bestimmen, so stellt sich die Frage, welche Ereignisse in zeitlichen Bezug zueinander gesetzt werden sollen, da keine entsprechenden beobachtbaren Ereignisse existieren.

Als Lösung wird dafür gesorgt, daß alle Stationen zu einem bestimmten Zeitpunkt selbst ein Ereignis erzeugen. Da die Präzision der Uhren bestimmt werden soll, wird dieses Ereignis zu einem vordefinierten Zeitpunkt auf den lokalen Uhren der Stationen erzeugt; d. h., jede beteiligte Station setzt ein Timeout auf einen vordefinierten Zeitpunkt in der Zukunft. Sobald dieser Timeout abläuft, generiert sie ein Signal für den Meßrechner an der parallelen Schnittstelle. Dieser vordefinierte Zeitpunkt auf den lokalen Uhren wurde im Rahmen der Implementierung so gewählt, daß es alle Zeiten sind, die Modulo 2 s den Wert Null ergeben, also wird alle 2 Sekunden ein Signal erzeugt. Da die Uhr in 1 µs-Schritten zählt, wird dieses Signal im Idealfall mit einer ebenso großen Genauigkeit erzeugt.

Es gibt zwei Probleme bei der Implementierung dieses Prinzips, die zuerst gelöst werden müssen: Unter Windows NT kann man zwar Timeouts setzen, allerdings bezieht sich die Angabe der Zeit in dem Timeout auf die Windows NT-interne Uhr, welche zudem eine Auflösung in der Größenordnung von lediglich etwa 10 ms hat. In der Implementierung der Uhrensynchronisation wurde hingegen eine eigene Uhr implementiert, die parallel zur Uhr von Windows NT arbeitet. Es muß also ein Verfahren gefunden werden, um das Timeout auf die Uhr der eigenen Uhr zu eichen und die geforderte Präzision (im Mikrosekunden-Bereich) zu erreichen.

Das zweite Problem ergibt sich dadurch, daß Windows NT keine Garantien abgibt, daß ein Timeout pünktlich abläuft. Vielmehr garantiert Windows NT lediglich, daß *mindestens* die spezifizierte Zeit abgelaufen ist. Dies ist für eine präzise Messung der Genauigkeit nicht ausreichend.

Um beide Probleme zu lösen, wurde folgendes Verfahren eingesetzt: Der Gerätetreiber im SUT setzt ein Timeout, welches rechtzeitig *vor* dem gewünschten Zeitpunkt abläuft. Sobald das Timeout abläuft wird eine Funktion aktiviert, die fortlaufend die synchronisierte Uhr abfragt und testet, ob der vorgegebene Zeitpunkt erreicht wurde (ein sogenanntes *busy waiting*). Sobald er erreicht wurde, wird ein Signal für den Meßrechner erzeugt und ein neuer Timeout gesetzt.

Um eine hohe Meßgenauigkeit zu erreichen, sind weitere Vorkehrungen nötig. Bei jeder Synchronisation wird die Uhr potentiell verstellt, dadurch kann auch der vordefinierte Zeitpunkt auf früher oder später verschoben werden. Aus diesem Grund wird der Timeout bei jeder Synchronisation verstellt, so daß er den veränderten Rahmenbedingungen gerecht werden kann.

Da Windows NT keine Garantien abgibt, daß die zu einem Timeout gehörende Funktion zeitnah ausgeführt wird, muß der Timeout früh genug gesetzt werden. Eine Zeit von 15 ms *vor* dem vordefinierten Zeitpunkt hat sich bei den Messungen als ein geeigneter Wert erwiesen. Dennoch passiert es auch bei diesem frühen Start, daß die Funktion

nicht früh genug aufgerufen wird. In diesem Fall wird gar kein Signal erzeugt, damit sich keine Fehlmessung ergibt, die eine schlechte Genauigkeit nahelegen würde. Wurde die Timeout-Funktion früh genug aufgerufen, dann könnte während des *busy waitings* die wartende Funktion vom Betriebssystem unterbrochen werden, weil eine wichtigere Funktion ansteht. Würde unmittelbar bevor das Ereignis auf den Meßaufbau gegeben wird diese Unterbrechung entstehen, dann würde sich eine Fehlmessung ergeben, die nicht erkannt werden kann. Damit dies nicht passiert, wird während der Phase des *busy waitings* bis unmittelbar nach Erzeugen des Ereignisses auf der parallelen Schnittstelle die Priorität des Programmabschnittes auf das Maximum erhöht (unter Windows NT heißt dies: Der IRQL wird erhöht), so daß Windows NT die Funktion nicht unterbrechen kann. Der gesamte Aufbau der Timerfunktion kann der Abbildung 6-8 entnommen werden.

Abbildung 6-8: *Messung der Präzision*

Es stellt sich die Frage, wie genau dieser Meßaufbau ist. Um dies zu untersuchen, wurde eine erste Messung unternommen, bei der die Abweichung der Uhren der Slaves von der Uhr des Masters gemessen wurde, ohne daß eine Synchronisation erfolgt. Hierzu wurde ein Master mit zwei Stationen mittels des hier entwickelten Protokolls synchronisiert, und kurz vor dem Start der Messung wurde die Synchronisation abgeschaltet, indem die Netzwerkkarte des Masters abgezogen wurde. Hierdurch konnte die Uhr des Masters mit den Uhren der Slaves verglichen werden, ohne daß eine Synchronisation erfolgt. Das Ergebnis dieser Messung kann der Abbildung 6-9 entnommen werden.

In der Abbildung ist dargestellt, welche Abweichungen die Uhren der Slaves zur Uhr des Masters besitzen, und zwar in Abhängigkeit von der vergangenen Zeit. Die Uhr des Masters verläuft demnach auf der Ordinate, da sie immer den Abstand Null zu sich selber hat. Man erkennt in der Abbildung, daß zu Beginn der Messung die Uhr des Masters und die Uhren der Slaves nah beieinander liegen. Im Verlauf der Zeit bewegen sich die Uhren der Stationen stetig von der Uhr des Masters weg. Offensichtlich ist der Verlauf der Slave-Uhren im Verhältnis zur Uhr des Masters linear, also ist die Annahme einer linearen Drift gerechtfertigt. Aufgrund der Messung ergibt sich eine Driftrate der Uhren

der beiden Slaves von $1{,}1 \cdot 10^{-6}$ bzw. $1{,}8 \cdot 10^{-6}$. Unter Brücksichtigung der Driftrate ergibt sich, daß keiner der einzelnen Meßwerte weiter als 10 µs von dem Wert entfernt war, der aufgrund der Driftrate zu erwarten gewesen wäre. Hiermit wird der Meßaufbau gerechtfertigt, da er in der Lage ist, die Differenz zweier Uhren mit einer Genauigkeit in der Größenordnung von etwa 10 µs zu bestimmen.

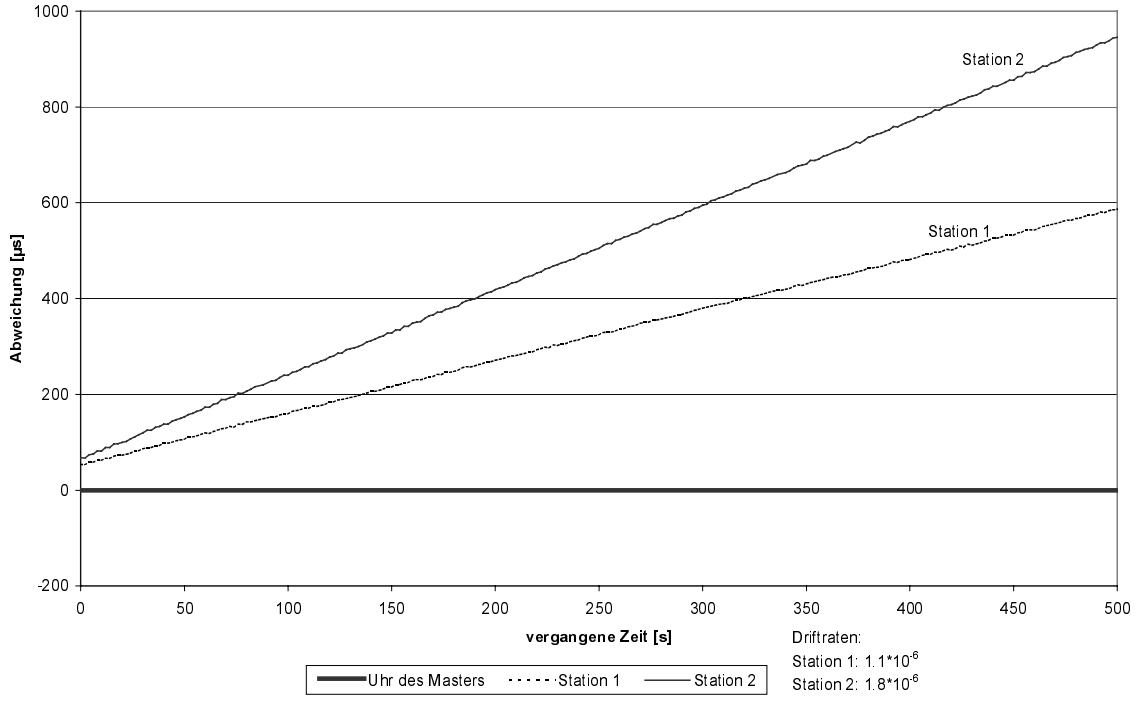


Abbildung 6-9: *Abweichung zur Uhr des Masters ohne Synchronisation*

Bei der Messung wurde nicht die Driftrate der physikalischen Uhren gemessen. Es wurde das Synchronisationsprotokoll mit Ratenanpassung ausgeführt, daher wurde die Drift der virtuellen Uhr gemessen, die sich nach Ausführung der Ratenanpassung ergibt, die also durch die Ratenanpassung nicht kompensiert werden kann. Die Messung in Abschnitt 6.2 „Messung der Drift der physikalischen Uhren“ ergab eine physikalische Driftrate von maximal $2 \cdot 10^{-5}$ für die physikalische Uhr. Die Abbildung zeigt, daß die Driftraten der virtuellen Uhren untereinander deutlich unter der Driftrate der physikalischen Uhren von $2 \cdot 10^{-5}$ liegen (s. Abschnitt 6.2 „Messung der Drift der physikalischen Uhren“), und daß die Driftrate wesentlich unter dem maximalen Wert, der sich nach Lemma 10 ebenfalls als ca. $\frac{10\text{ s}+2 \cdot 50\text{ µs}}{10\text{ s}-2 \cdot 50\text{ µs}} - 1 \approx 2 \cdot 10^{-5}$ ergibt, liegt.

In der Abbildung erkennt man weiterhin, daß die Uhren der Slaves sich linear von der Uhr des Masters wegbewegen. Dies rechtfertigt im Nachhinein die getroffene Annahme, daß die Driftrate konstant ist, was eine wesentliche Voraussetzung für eine gute Ratenanpassung ist.

Da der Meßaufbau hierdurch gerechtfertigt wurde, soll nun die tatsächlich erreichte Präzision des Protokolls gemessen werden. Hierzu wurde ein Master mit mehreren Slaves synchronisiert und mittels des Meßaufbaus die Abweichung zu bestimmten Zeitpunkten gemessen, analog zu der vorherigen Messung. Es wurde OD=8 angekommen, das Synchronisationsintervall betrug INT=1 s und der Abstand Δt der Zeitstempel, die zur Berechung der Uhr des APs benutzt wurden, betrug Δt=10 s. Das Ergebnis der Messung am Beispiel eines Slaves kann der Abbildung 6-10 entnommen werden. Zum Vergleich ist dort das Ergebnis ohne Synchronisation mit aufgetragen.

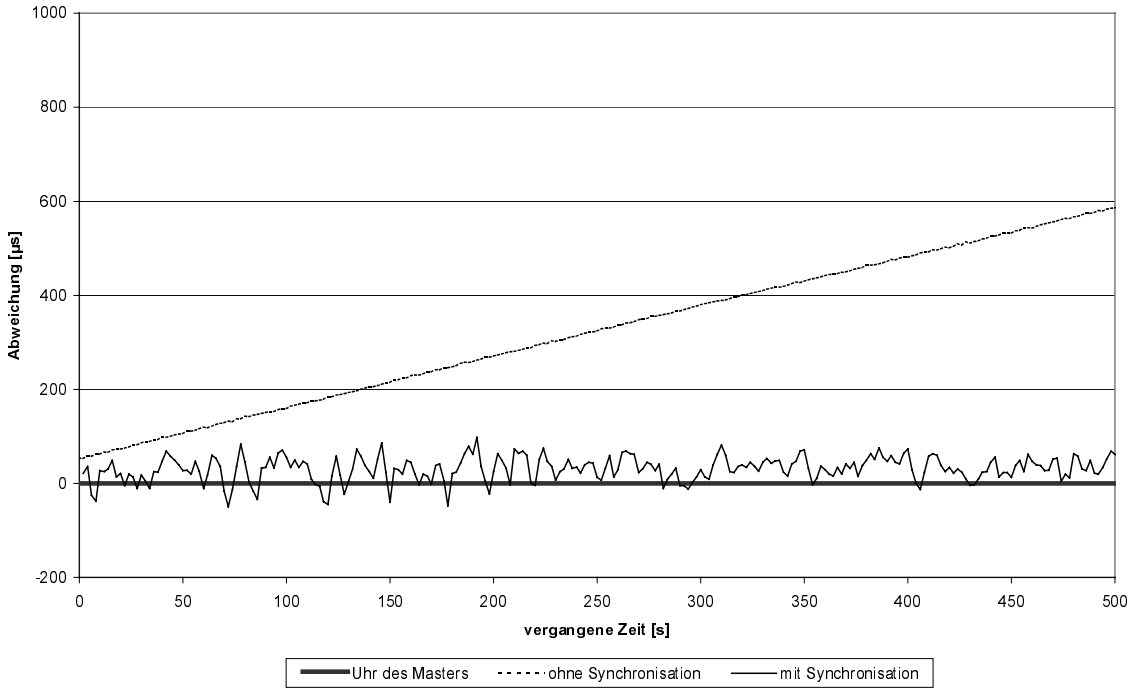


Abbildung 6-10: *Abweichung zur Uhr des Masters, mit und ohne Synchronisation*

Der Abbildung kann entnommen werden, daß die Uhr des Slaves regelmäßig umgestellt wird. Man erkennt weiterhin, daß keine Zeitsprünge auftreten. Als wichtigstes Ergebnis ergibt sich, daß die Uhr des Slaves maximal um 100 µs der Uhr des Masters voraus- und maximal um 50 µs hinterhereilt, sich also in einem Bereich von 150 µs um die Uhr des Masters bewegt.

Um einen Vergleich mit dem IEEE-Protokoll zu ermöglichen, wurde die gleiche Messung an der Implementierung des IEEE-Protokolls vorgenommen. Das Ergebnis kann der Abbildung 6-11 entnommen werden, wobei das Ergebnis der Synchronisation mit dem eigenen Protokoll zum Vergleich auch eingezeichnet wurde. Man kann erkennen, daß die Uhr des Slaves maximal um etwa 30 µs der Uhr des Masters voraus- und maximal um 950 µs hinterhereilt, sich also in einem Bereich von 980 µs um die Uhr des Masters bewegt. Dieses Ergebnis stimmt gut mit dem Ergebnis des Abschnittes 6.4 „Messung des zeitkritischen Pfades des Protokolls“ überein, in dem festgestellt wurde,

daß der zeitkritische Pfad beim IEEE-Protokoll eine Schwankung von etwa 900 µs aufweist.

Der Abbildung kann auch entnommen werden, wie sich das IEEE-Protokoll verhält: Zumeist bleibt die Uhr des Slaves in der Nähe der Uhr des Masters, vereinzelte Aussetzer sorgen aber dafür, daß die Präzision nicht verbessert werden kann. Dies zeigt, daß dieses Verhalten tatsächlich durch lediglich vereinzelt auftretende große Schwankungen der Nachrichtenauslieferung zustande kommt.

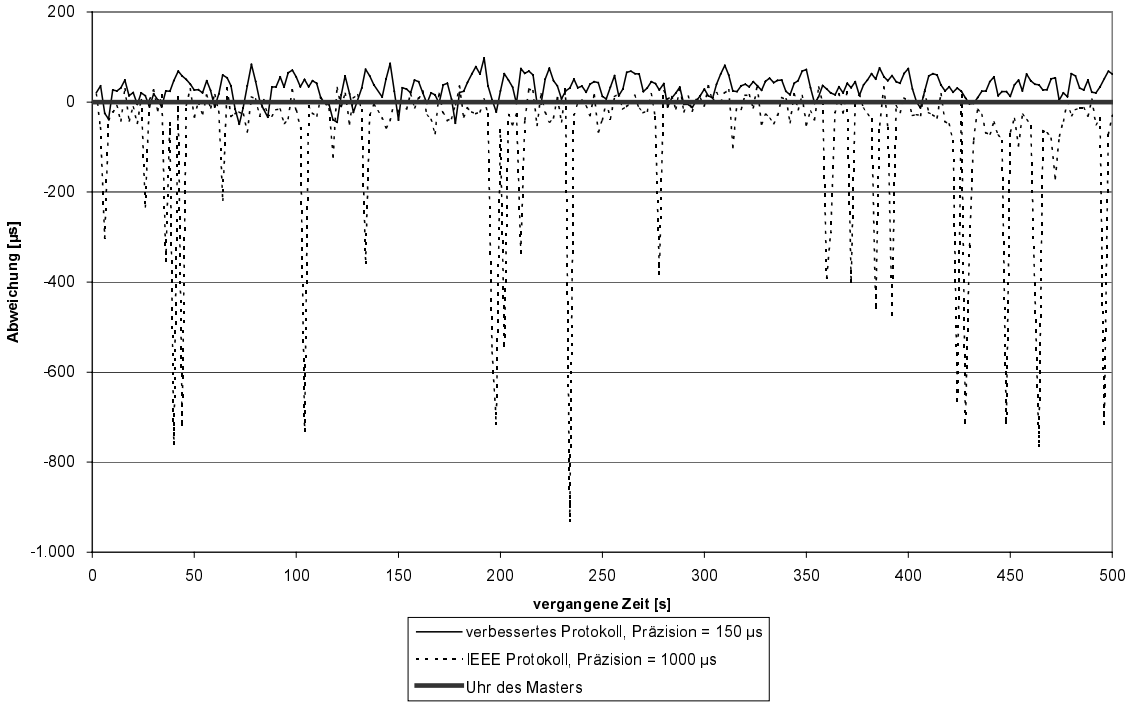


Abbildung 6-11: *Vergleich der verbesserten Synchronisation zur IEEE Synchronisation*

Die bisherigen Messungen wurden mittels eines Meßaufbaues vorgenommen, bei dem keine Nachrichten verloren gehen. Es stellt sich die Frage, wie weit sich die Ergebnisse wiederholen lassen, wenn Nachrichten verloren gehen. Daher wurde eine Messung vorgenommen, bei der die Protokolle mit steigenden Verlustraten arbeiteten, d. h., mit einer Wahrscheinlichkeit von p, mit p=10%, 20%, …, 70%, wurden empfangene Nach-richten von dem Protokoll als verloren angenommen und nicht ausgewertet. Diese Modifikation wurde sowohl für das IEEE-Protokoll als auch für das hier entwickelte Protokoll vorgenommen. Wie schon in Kapitel 4 „Uhrensynchronisationsprotokoll für das Funknetzwerk“ besprochen, ist ein Parameter für das Protokoll die Anzahl der Zeitstempel, die in einer Sync-Nachricht vorhanden sind. Daher wurde das Protokoll jeweils mit 4, 8 oder 16 Zeitstempeln pro Nachricht vermessen, was einer tolerierbaren Omission Degree OD von 3, 7 und 15 entspricht. Das Ergebnis dieser Messung kann der Abbildung 6-12 entnommen werden.

In der Abbildung ist die durchschnittliche gemessene Präzision gegen die provozierte Nachrichtenverlustrate aufgetragen. Man erkennt, daß bis zu einer Verlustrate von 40 % alle drei Instanzen des hier entwickelten Protokolls sich nicht schlechter verhalten als das IEEE-Protokoll. Bei 40 % Verlustrate verschlechtert sich das eigene Protokoll mit der Annahme, daß *OD*=3 wäre. Bis zu einer Verlustrate von 60 % bleiben die Implementierungen für *OD*=7 oder 15 wesentlich besser als das IEEE-Protokoll. Dies zeigt, daß das Protokoll sehr robust gegenüber Störungen ist – man bedenke, daß eine Verlustrate von 50 % eine extrem schlechte Verbindung bedeutet.

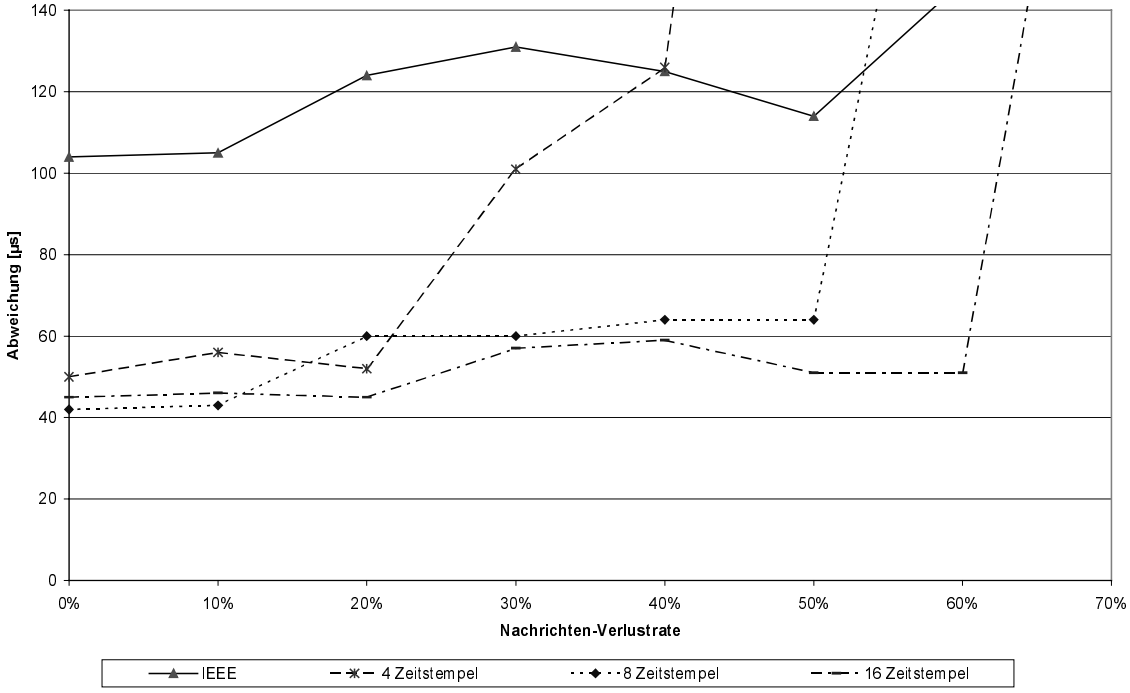


Abbildung 6-12: *Durchschnittliche Präzision bei Nachrichtenverlusten*

Eine interessante Erkenntnis ist, daß sich bei *OD* = 15 gegenüber *OD* = 7 keine wesentliche Verbesserung mehr ergibt, obwohl die Anzahl der Zeitstempel verdoppelt wurde. Man beachte, daß eine Verlustrate von 50 % extrem hoch ist, realistischerweise kann man gar nicht mehr von einer Verbindung sprechen. Da bei *OD* = 7 diese Verlustrate von 50 % noch toleriert werden kann, wird es in den meisten Anwendungen keinen Grund geben, von einem Omission Degree von mehr als 7 auszugehen, daher wird eine Annahme von *OD* = 7 oder *OD* = 8 in den meisten Anwendungen völlig ausreichen.

6.6 Vergleich mit der theoretischen Analyse

Mit den Ergebnissen der in diesem Kapitel vorgenommenen Messungen kann das in Abschnitt 4.5 „Analyse des Protokolls“ gefundene Ergebnis konkretisiert werden, indem die ermittelten Werte für die Parameter eingesetzt werden.

Der Satz 12 besagt, daß die Präzision Π des Protokolls bestimmt werden kann mittels der Formel $\Pi = \frac{2\delta\Delta t(2+\rho)}{(\Delta t)^2-\delta^2}\cdot(OD+2)\cdot INT + \delta\left(1 + \frac{(\Delta t)^2(1+\rho)+\delta^2}{(\Delta t)^2-\delta^2}\right)$, wobei INT die Länge des Synchronisationsintervalles ist. Aufgrund der Messungen können dabei folgende Werte eingesetzt werden: Es ist $\delta = 50$ µs die maximale Schwankung der Nachrichtenauslieferung, und es ist $\rho = 2\cdot10^{-5}$ die maximale Driftrate der physikalischen Uhr. Sei weiterhin OD=8 (die Rechtfertigung für diese Annahme findet sich in Abschnitt 6.5 „Erreichte Präzision“).

Die Abbildung 6-13 zeigt die Präzision in Abhängigkeit von dem Wert von Δt. Der Wert Δt bezeichnete die Zeitdifferenz zwischen den beiden Werten, die zur Bestimmung des Verlaufs der Uhr des Masters benutzt wurden. In einer Implementierung besitzt dieser Wert eine konstante Obergrenze. Er ist also ein Protokollparameter, der davon abhängt, wie groß die Warteschlange ist, in der die „alten“ Zeitstempelpaare gepuffert werden. Während der Initialisierung des Protokolles, d. h., wenn eine neue Station die Synchronisationsgruppe erreicht, ist der Wert von Δt allerdings kleiner, da die Station noch nicht lange genug zurückliegende Informationen besitzt. Mit jeder Synchronisationsrunde nach dem ersten Empfang einer Synchronisationsnachricht steigt der Wert von Δt. Man erkennt anhand der Abbildung 6-13, daß schon mit einem zeitlichen Abstand von mehr als 3 Sekunden zwischen zwei erfolgreichen Synchronisationen eine Präzision von unter 1 ms garantiert werden kann.

In der Implementierung wurde ein Mechanismus vorgesehen, der „alte“ Zeitstempelpaare in der Warteschlange sammelt, damit ein großes Δt erreicht wird. Die Tabelle legt es nahe, keine Warteschlange für die Berechnung zu benutzen. Stattdessen würde sich der Treiber das erste Zeitstempelpaar merken und jede nachfolgende Synchronisation würde dieses erste Zeitstempelpaar als Bezugspunkt benutzen. Hiermit würde Δt unbeschränkt steigen, und hieraus würde man folgern, daß die Präzision beliebig verbessert werden kann. Allerdings hat dieses Verfahren einen Nachteil in der Praxis. Es wurde zwar angenommen, daß die Driftrate konstant ist. In realen Implementierungen verändert sich diese allerdings langfristig, beispielsweise durch Temperatur- oder Luftdruckschwankungen. Solche Veränderungen könnten bei diesem Verfahren nicht berücksichtigt werden und würden die Präzision des Verfahrens in der Praxis verschlechtern, weil sie von dem Protokoll nicht mehr ausgeglichen werden könnten.

Tatsächlich läßt sich die Präzision nicht beliebig verringern. Bei $\Delta t = 100$ s ergibt sich eine Präzision von 120 µs, bei $\Delta t = 1000$ s ergibt sich eine Präzision von 102 µs und bei $\Delta t = 10^9$ ist die Präzision immer noch 100 µs. Anhand der Formel für Π erkennt man, daß 2δ der Grenzwert ist, wenn Δt gegen unendlich konvergiert: In diesem fall konvergiert der erste Summand gegen Null, der zweite gegen $\delta\cdot(1+1)$.

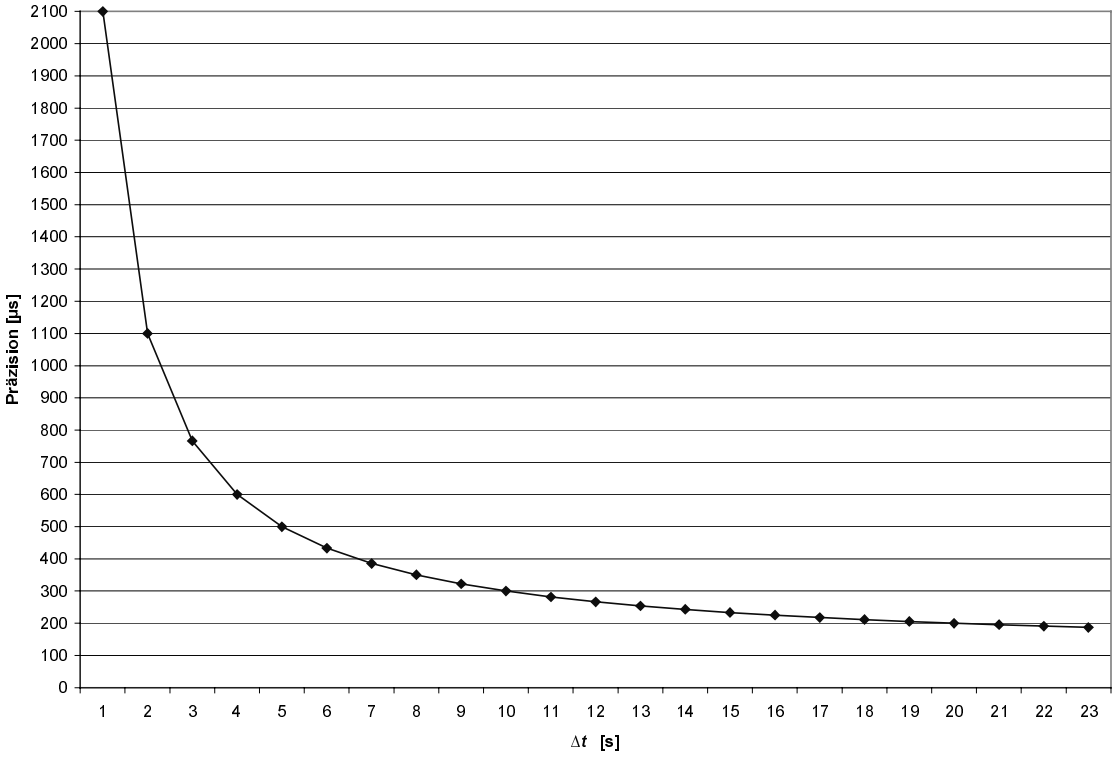


Abbildung 6-13: *Präzision in Abhängigkeit von* Δt

Interessant ist, daß der zweite Summand unabhängig von Δt beinahe konstant 2δ beträgt. Dies kann an der Formel schon erkannt werden: Da δ und ρ im Verhältnis zu Δt sehr klein sind, ist der Bruch im zweiten Summanden beinahe konstant 1. Daran erkennt man, daß bei $\Delta t = 1$ s der große Wert 2,1 ms für die Präzision fast ausschließlich aus dem ersten Summanden besteht. Der erste Summand gibt an, wie weit die Uhr durch die fehlerhaft bestimmte Driftrate in der Zeit, in der sie sich nicht synchronisieren kann, von der Uhr des Masters driften kann.

Daher kann man folgende Verbesserung des Protokolls erwägen: In der Initialisierungsphase des Protokolles wird die Ratenanpassung zwar vorgenommen, allerdings wird die Driftrate der Uhr des Masters erst als Multiplikator übernommen, wenn das Protokoll eine längere Zeit gelaufen ist. Die Auswirkungen dieser Modifikation müßten noch untersucht werden.

Schließlich soll untersucht werden, welche Auswirkungen die Wahl von *OD* auf die Präzision besitzt. Seien die Werte wie oben, $\Delta t = 10$ s fest, nun aber *OD* variabel. Die Präzision kann der Abbildung 6-14 entnommen werden.

Man erkennt die lineare Abhängigkeit der Präzision von *OD*. Ein kleiner Wert für *OD* bedeutet eine hohe Präzision, ein großer Wert hingegen eine geringere Präzision. Der Wert für *OD* sollte nicht zu klein gewählt werden: Wenn das Fehlermodell nicht mehr

erfüllt ist und damit keine Synchronisation mehr möglich ist, dann kann auch die Präzision Π von dem Protokoll nicht mehr garantiert werden.

Sehr interessant ist auch die Abhängigkeit der Präzision von der Länge des Synchronisationsintervalles. Diese Abhängigkeit zeigt die Abbildung 6-15. Dabei ist in der Tabelle $OD = 8$ und $\Delta t = 10$ s angenommen.

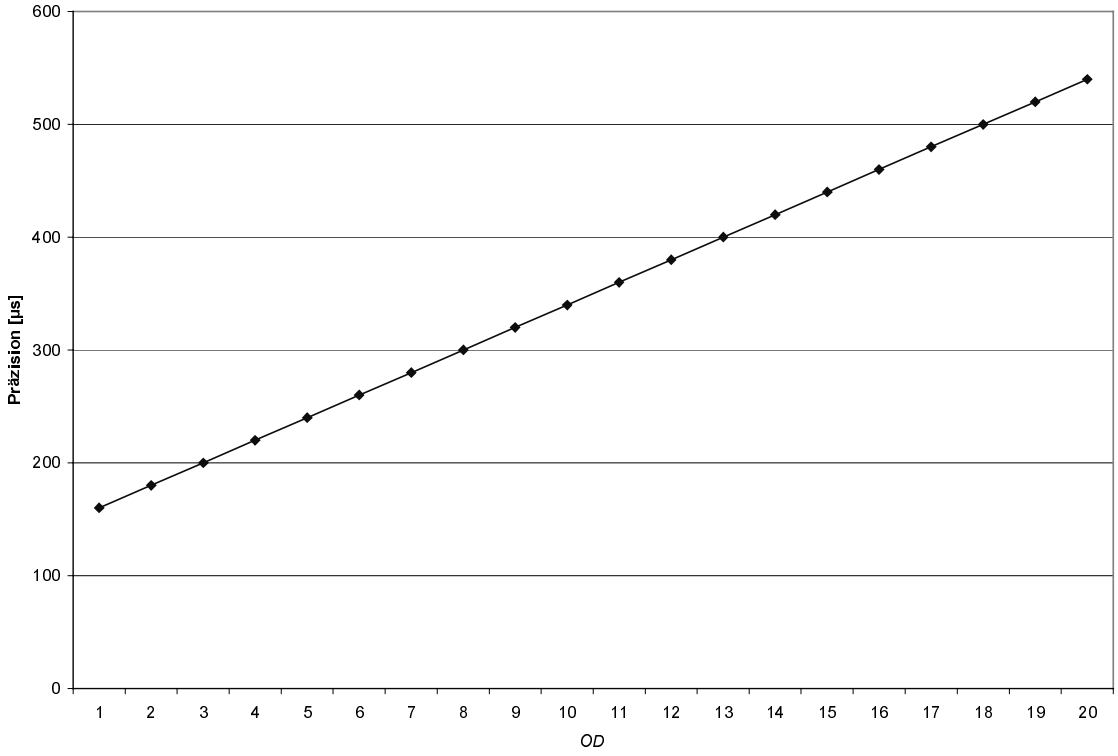


Abbildung 6-14: *Präzision in Abhängigkeit von OD*

INT	Präzision
10 ms	102 µs
50 ms	110 µs
0.1 s	120 µs
0.5 s	200 µs
1 s	300 µs

INT	Präzision
2 s	500 µs
3 s	700 µs
4 s	900 µs
5 s	1.100 µs
10 s	2.100 µs

Abbildung 6-15: *Präzision in Abhängigkeit von INT*

Man erkennt, daß ein Synchronisationsintervall im Sekundenbereich einen guten Kompromiß darstellt zwischen dem Nachrichtenaufwand und der erreichten Präzision: So

erhält man bei einer Intervallänge von 10 ms gegenüber 1 s zwar eine Präzision, die nur ein Drittel beträgt, dafür beträgt der Nachrichtenaufwand das 100fache.

Die Werte zeigen eine weitere Möglichkeit, das Protokoll zu verbessern: Wie der Abbildung 6-13 entnommen werden kann, erzeugt das Protokoll mit zunehmender Laufzeit eine verbesserte Präzision. Beim Erreichen einer ausreichenden Präzision könnte das Intervall *INT* von dem Master vergrößert werden. Die Abbildung 6-16 zeigt dies beispielhaft.

Δt	*INT*	Präzision
10 s	1 s	300 µs
20 s	2 s	300 µs
30 s	3 s	300 µs
40 s	4 s	300 µs
50 s	5 s	300 µs

Δt	*INT*	Präzision
60 s	6 s	300 µs
70 s	7 s	300 µs
80 s	8 s	300 µs
90 s	9 s	300 µs
100 s	10 s	300 µs

Abbildung 6-16: *Gleichzeitige Anpassung von INT und Δt*

Dabei sei wieder OD=8 angenommen. Es wird *INT* immer auf $\frac{1}{10}\,\Delta t$ gesetzt. Man beachte, daß hierbei die Anzahl der in der Warteschlange der Station gepufferten Zeitstempelpaare konstant bleibt. Man erkennt, daß die Präzision ebenfalls konstant bleibt, obwohl das Intervall immer länger wird. Damit kann die Belastung des Netzwerks verringert werden.

Das Protokoll läßt sich entsprechend erweitern. Der AP merkt sich, wie lange er synchronisiert, und verlängert sein Synchronisationsintervall entsprechend. Hierbei müßte der AP, sobald eine Station in seinen Empfangsbereich gelangt, das Intervall *INT* wieder herabsetzen, damit diese Station schnell ihre Initialisierungsphase überbrücken kann. Hierzu muß der AP erkennen, daß eine Station seinen Empfangsbereich erreicht. Da jede Station sich am AP anmelden muß, bevor sie zu seiner Gruppe (und damit zur Synchronisationsgruppe) gehört, kann der AP dies feststellen.

Interessant ist der Vergleich der Theorie mit der Messung. Die Messung für die Präzision in Abschnitt 6.5 „Erreichte Präzision“ (s. Abbildung 6-10) wurde mit INT=1 s, Δt=10 s und OD=8 vorgenommen. In der Messung wurde eine maximale Abweichung zwischen den Uhren von ca. 150 µs festgestellt, die errechnete Präzision beträgt 300 µs. Diese beiden Werte passen gut zueinander, d. h. sie widersprechen sich nicht.

Im folgenden soll auch ein theoretischer Vergleich mit dem IEEE-Protokoll erfolgen. Die Präzision des Protokolls nach dem IEEE-Standard kann folgendermaßen bestimmt werden: Zwischen zwei Synchronisationen liegt maximal die Zeit $(OD+2)INT$, in dieser Zeit kann eine Uhr maximal um den Betrag $\rho\cdot(OD+2)INT$ von der Uhr des Masters driften. Da zwei Uhren in gegensätzliche Richtungen driften können, ergibt sich ein maximaler Offset der Uhren aufgrund der Drift von $2\rho\cdot(OD+2)INT$. Zusätzlich zu diesem durch die Drift bewirkten Offset gibt es noch einen Offset, der seinen Ursprung darin hat, daß die Uhr des Masters nur mit einer begrenzten Genauigkeit bestimmt werden

kann. Diese Ungenauigkeit ist genau die Varianz des zeitkritischen Pfades des IEEE-Protokolles, dieser Fehler wurde in der Implementierung als ca. $var := 900$ µs gemessen. Die Präzision beim IEEE-Protokoll beträgt demnach $\Pi_{ieee} = 2\rho\cdot(OD+2)INT + var$. Die Werte in den oben dargestellten Tabellen zeigen, daß das IEEE-Protokoll bereits aufgrund des Wertes von *var* in fast jeder Situation dem hier vorgestellten Protokoll unterlegen ist. Lediglich beim Beginn der Synchronisation, d. h. kleinen Werten Δt, ist das IEEE-Protokoll manchmal besser. Allerdings kann es nicht von der Laufzeit des Protokolls, also dem wachsenden Δt, profitieren, so daß es nach kurzer Laufzeit dem hier entwickelten Protokoll immer unterlegen ist.

Das Protokoll nach dem IEEE-Standard ist also nur abhängig von der Intervallänge *INT* und der maximalen Anzahl der aufeinanderfolgend verlorengehenden Nachrichten *OD*. Bei INT=1 s und $OD = 0$, d. h. ohne Nachrichtenverluste, beträgt die Präzision des IEEE-Protokolles $\Pi_{ieee} = 80$ µs + $var = 980$ µs. Dieser Wert wird von dem im Rahmen dieser Arbeit entwickelten Protokoll schon bei der ersten Ratenanpassung mit $\Delta t = 1$ s unterboten (s. Abbildung 6-17), dann hat das Protokoll eine Präzision von etwa 500 µs. Mit fortschreitender Dauer der Synchronisation verbessert sich die Präzision beim hier entwickelten Protokoll noch wesentlich, während das IEEE-Protokoll die längere Laufzeit nicht nutzen kann. Daher ist das hier entwickelte Protokoll auch unter der Annahme, daß keine Nachrichtenverluste auftreten, dem IEEE-Protokoll überlegen.

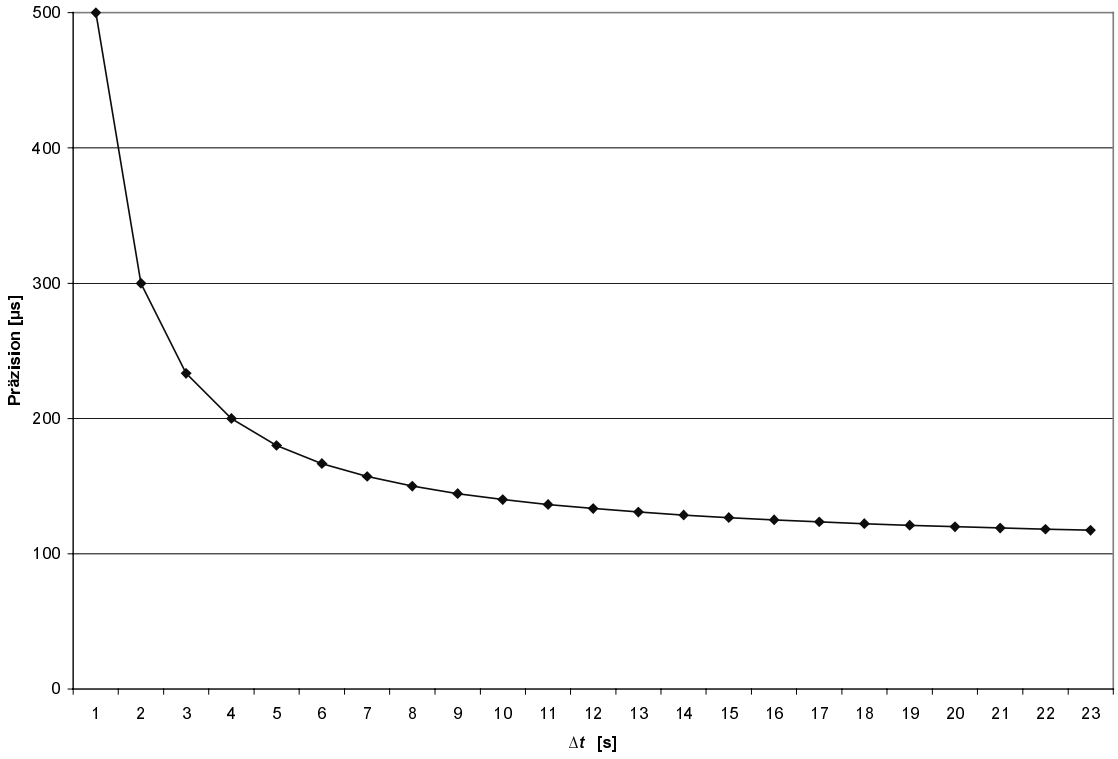


Abbildung 6-17: *Präzision des eigenen Protokolls in Abhängigkeit von Δt bei OD=0*

7 Fazit

In der vorliegenden Arbeit wird ein Uhrensynchronisationsprotokoll vorgestellt, welches auf einem Funknetzwerk nach IEEE 802.11 eingesetzt wird. Dieses Protokoll wird dabei unter besonderer Beachtung der Eigenschaften des Funknetzwerks entworfen, so daß eine hochpräzise globale Uhrzeit erreicht wird. Dabei wird besonderer Wert darauf gelegt, daß das Protokoll kompatibel zum IEEE-Standard ist, so daß es auf jeder standard-konformen Netzwerkkarte implementiert werden kann. Weiterhin sollen Netzwerkkarten, die das Protokoll nicht unterstützen, in ihrer Funktion durch das Protokoll nicht beeinträchtigt werden.

Die Synchronisation von Uhren wird über den Austausch von Nachrichten über das Netzwerk bewerkstelligt. Der sogenannte zeitkritische Pfad beschränkt dabei die erreichbare Präzision nach unten, da durch Schwankungen in der Laufzeit der Synchronisationsnachrichten die Genauigkeit der Uhren der Stationen beschränkt ist. Eine gute Präzision erreicht ein Protokoll, das die Schwankungen des zeitkritischen Pfades minimiert.

Das GS-Protokoll verkleinert den zeitkritischen Pfad auf dem CAN-Bus. Im IEEE-Standard über Funknetzwerke existiert ein Uhrensynchronisationsprotokoll, welches einen großen zeitkritischen Pfad aufweist. Die wesentliche Aufgabe dieser Arbeit bestand darin, durch Übertragung des Prinzips des GS-Protokolls auf das Funknetzwerk diesen Pfad zu verringern, wodurch sich eine höhere Präzision ergibt. Die Struktur des Funknetzwerks kommt dabei einer Adaption entgegen. In einem Infrastrukturnetzwerk nach IEEE-Standard liegt ebenso wie beim GS-Protokoll eine Master-/Slave-Struktur vor. Daher wird diese Master-/Slave-Struktur des Netzwerkes für das eigene Protokoll übernommen, als Master für das Synchronisationsprotokoll wird der Master des Netzwerkes benutzt, der sogenannte AP. Er besitzt eine besondere Stellung im Funknetzwerk und prägt durch seine zentrale Stellung einen Gruppenbegriff, der als Begriff der Synchronisationsgruppe übernommen wird.

Weiterhin läßt sich das Prinzip des GS-Protokolls gut auf das Funknetzwerk übertragen, da das Funknetzwerk wie auch der CAN-Bus keine großen Laufzeitunterschiede zwischen dem Empfang einer Nachricht auf verschiedenen Empfängern besitzt.

Das Funknetzwerk besitzt aber durch das Fehlen eines geschützten Übertragungsmediums im Gegensatz zu drahtgebundenen Netzwerken eine hohe Fehlerrate. Diese muß im Entwurf des eigenen Protokolls berücksichtigt werden. Daher werden durch Anwendung von statischer Redundanz, nämlich dem mehrfachen Versenden von Informationen, Nachrichtenverluste des Protokolls toleriert. Dabei ist die Anzahl der maximal angenommenen Verluste ein Protokollparameter, der an die Anforderungen der Anwendung angepaßt werden kann.

Ferner enthält das Protokoll eine Ratenanpassung, welche über einen bestimmten Zeitraum die Drift der zugrundeliegenden physikalischen Uhr beobachtet. Die dabei gewonnenen Erkenntnisse werden genutzt, um die Drift der physikalischen Uhr zu kom-

pensieren und damit die Drift der synchronisierten Uhren zu vermindern, wodurch die Präzision der Synchronisation gesteigert wird.

Ein minimaler zeitkritischer Pfad läßt sich durch die Implementierung des Protokolls in der Hard- bzw. Firmware der Netzwerkkarte erreichen. Hersteller von Netzwerkkarten geben jedoch im Allgemeinen Details über ihre Hard- und Firmware nicht aus der Hand. Die Firma Lucent Technologies stellt uns jedoch im Rahmen eines Kooperationsvertrages die Treibersourcen für ihre WaveLAN-Netzwerkkarten für das Betriebssystem Windows NT zur Verfügung. Daher wurde sowohl das Protokoll nach dem IEEE-Standard als auch das verbesserte Protokoll auf der Ebene der Gerätetreiber unter Windows NT implementiert und vermessen.

Für die Messung wurde ein Meßaufbau erstellt, welcher durch die Nutzung der in PCs standardmäßig vorhandenen parallelen Schnittstelle erlaubt, verteilte Ereignisse mit einer hohen Präzision, besser als 6 µs, zu messen und auszuwerten.

Die Messung des IEEE-Protokolls ergab eine Präzision von etwa 1000 µs, die Messung des eigenen Protokolles ergab den wesentlich besseren Wert von 150 µs.

Eine theoretische Analyse des hier entwickelten Protokolls ergab, daß in der benutzten Implementierungsumgebung nach einer Initialisierungsphase von 10 s eine Präzision von etwa 300 µs garantiert werden kann, wenn jede Sekunde eine Synchronisationsnachricht gesendet wird und nicht mehr als 8 aufeinanderfolgende Nachrichten verloren gehen. Bei einer Verdoppelung der Initialisierungsphase ergibt sich eine Präzision von 200 µs. Die Meßergebnisse stimmen mit den theoretisch ermittelten Ergebnissen überein.

Der Vergleich mit dem IEEE-Protokoll ergibt, daß das hier entwickelte Protokoll dem Protokoll nach dem IEEE-Standard überlegen ist. Dieser Vorteil ergibt sich aus dem verkürztem zeitkritischen Pfad des Protokolls und daraus, daß das hier entwickelte Protokoll mit der Zeit lernt, die Uhr des Masters genauer abzuschätzen, während das IEEE-Protokoll aus einer längeren Synchronisationsdauer und sich hieraus ergebenden Informationen nicht profitieren kann.

In der beschriebenen Implementierungsumgebung ergeben sich die hohen Varianzen hauptsächlich dadurch, daß das Betriebssystem einen großen zeitlichen Overhead erzeugt, insbesondere, weil es andere Aufgaben wichtiger als das Uhrensynchronisationsprotokoll einstuft. Eine Implementierung in der Hard- oder Firmware der Netzwerkkarte würde es erlauben, die Varianzen zu senken. Hiervon würde sowohl das Protokoll nach dem IEEE-Standard wie auch das eigene Protokoll profitieren. Durch den kürzeren zeitkritischen Pfad und der Möglichkeit, die Drift der Uhr des Masters noch genauer zu bestimmen, würde das eigene Protokoll stärker von diesem Vorteil profitieren als das Protokoll des IEEE-Standards, so daß die Implementierung des Protokolls auf dieser Ebene lohnenswert ist.

8 Abkürzungsverzeichnis

Folgende Abkürzungen wurden in der Arbeit benutzt:

ACK	Acknowledge
ANSI	American National Standards Institute
AP	Access Point
BSS	Basic Service Set, eine Zelle in der Terminologie von IEEE 802.11
CAN	Control Area Network
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access / Collision Detection
DCF	Distributed Coordination Function
DS	Distribution System
DIFS	DCF Inter-Frame Space
EIFS	Extended Inter-Frame Space
ETSI	European Telecom Standards Institute
GPS	Global Positioning System
HIPERLAN	High Performance Radio Local Area Network
IBSS	Independant BSS
IEEE	Institute of Electrical and Electronics Engineers
IFS	Inter-Frame Space
IS	Indikationssender
ISO/IEC	International Organization for Standardization / International Electrotechnical Commission
LAN	Local Area Network
MAC	Medium Access Layer
NAV	Network Allocation Vector
NDIS	Network Driver Interface Specification
OD	Omission Degree
PCF	Point Coordination Function
PHY	Physical Layer
PIFS	PCF Inter-Frame Space
SDL	Specification and Description Language
SIFS	Short Inter-Frame Space
SUT	System Under Test
TAI	Temps Atomique Internationale

TDI	Transport Driver Interface
TSC	Time Stamp Counter
UTC	Universal Time Coordinated
WLAN	Wireless LAN

9 Literatur

[BD 87] Özalp Babaoglu, and Rogério Drummond, *(Almost) No Cost Clock Synchronization*. In: Digest of Papers, the 17th International Symposium on Fault-Tolerant Computing (FTCS), Pittsburgh, USA, July 1987, pp. 42-47.

[CAS 93] Flaviu Cristian, Houtan Aghili, and Ray Strong, *Clock Synchronization in the Presence of Omission and Performance Failures, and Processor Joins.* In: Zhonghuad Yang, T. Anthony Marsland (eds), *Global States and Time in Distributed Systems*, Computer Society Press, 1994, pp. 69-75.

[Cr 89] Flaviu Cristian, *Probabilistic clock synchronization.* Distributed Computing 3, 1989, pp. 146-158.

[DHS 86] Danny Dolev, Joseph Y. Halpern, and H. Raymond Strong, *On the Possibility and Impossibility of Achieving Clock Synchronization.* Journal of Computer and System Science 32(2), April1986, pp. 230-250.

[DHM 73] William M. Daly, Albert L. Hopkins, Jr., and John F. McKenna, *A Fault-Tolerant Digital Clocking System*. In: Digest of Papers, the 3rd International Symposium on Fault-Tolerant Computing (FTCS), Palo Alto, CA, USA, June 1973, pp. 17-22.

[DIN 85] DIN, *DIN 1301 Teil 1: Einheiten – Einheitennamen, Einheitenzeichen*. Deutsches Institut für Normung, Dezember 1985.

[ET 96] European Telecommunications Standards Institute, *HIPERLAN*. ETSI Standard ETS 300 652, 1996.

[GMN 98] Martin Gergeleit, Michael Mock, and Edgar Nett, *T-CORBA: making object-oriented systems time-aware*. Computer Systems Science & Engineering, Vol 13, No. 3, May 1998, pp. 151-160.

[GS 94] Martin Gergeleit, and Herman Streich, *Implementing a Distributed High-Resolution Real-Time Clock using the CAN-Bus.* Proceedings of the 1st international CAN-Conference, September 1994.

[IEEE 97] Institute of Electrical and Electronics Engineers, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications.* IEEE Std. 802.11-1997, November 1997. 1999 als ANSI-Standard und ISO/IEC-Standard 8802-11 übernommen.

[ITU 92] ITU, *ITU Specification and Description Language (SDL).* ITU.T Recommendation Z.100 (SDL-92), March 1993.

[Kn 81] Donald Erwin Knuth, *The Art Of Computer Programming, Vol 2: Seminumerical Algorithms.* 2nd edition, Addison-Wesley, 1981.

[KN 98] Jörg Kaiser und Edgar Nett, *Echtzeitverhalten in dynamischen, verteilten Systemen*. GI Informatik Spektrum 21(6), pp. 356-365, Dezember 1998

[KO 87] Hermann Kopetz, and Wilhelm Ochsenreiter, *Clock Synchronization in Distributed Real-Time Systems*. IEEE Transactions on Computers, Vol. C-36, No. 8, August 1987, pp. 933-940.

[Ko 97] Hermann Kopetz, *Real Time Systems – Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, Boston-Dordrecht-London, 1997.

[La 78] Leslie Lamport, *Time, clocks, and the ordering of Events in a Distributed Real-Time System.* Communications of the Association for Computing Machinery (CACM) 21(7), July 1978, pp. 558-565.

[Lap 92] Jean-Claude Laprie (ed), *Dependability: Basic Concepts and Terminology*. Springer, 1992.

[Li 91] Barbara Liskov, *Practical Uses of Synchronized Clocks in Distributed Systems.* Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing, August 1991, pp. 1-9.

[LL 84] Jennifer Lundelius, and Nancy A. Lynch, *An upper and lower bound for clock synchronisation.* Information and Control 62, 1984, pp. 190-204.

[LL 88] Jennifer Lundelius Welch, and Nancy A. Lynch, *A new fault-tolerant algorithm for clock synchronisation.* Information and Computation 77, 1988, pp. 1-36.

[LMS 85] Leslie Lamport, and P. M. Melliar-Smith, *Synchronizing Clocks in the Presence of Faults.* Journal of the Association for Computing Machinery (JACM), Vol. 32(1), January 1985, pp. 52-78.

[LSP 82] Leslie Lamport, Robert Shostak, and Marshall Pease, *The Byzantine Generals Problem.* ACM Transactions on Programming Languages and Systems 4(3), July 1982, pp. 382-401.

[Lu 98] Lucent Technologies Inc, *DRAFT Software Interface Specification for Wireless Connection Interface for WaveLAN-II*. Confidential, Unpublished Property of Lucent Technologies Inc., Document No. S0005 Rev. 6, 1998.

[MN 99] Michael Mock, and Edgar Nett, *Real-Time Communication in Autonomous Robot Systems*. In: Proceedings of the Fourth International Symposium on Autonomous Decentralized Systems (ISADS '99), Tokyo, Japan, March 1999, pp. 34-41.

[MS 96] Microsoft Corp, *Network Driver Interface Specification*. Part of the Windows NT Device Driver Development Kit (DDK), 1996.

[Ne 91] Edgar Nett, *Supporting Fault Tolerant Computations in Distributed Systems*. Habilitationsschrift, Universität Bonn, 1991.

[OS 91] Alan Olsen, and Kang G. Shin, *Probabilistic Clock Synchronization in Large Distributed Systems*. In: Proceedings of the 11th International Conference on Distributed Computing Systems, 1991, pp. 290-297.

[PSL 80] Marshall Pease, Robert Shostak, and Leslie Lamport, *Reaching Agreement in the Presence of Faults.* Journal of the Association for Computing Machinery (JACM) 27(2), April 1980, pp. 228-234.

[RV 92] Luís Rodrigues, and Paulo Veríssimo, *A posteriori Agreement for Clock Synchronization on Broadcast Networks.* INESC Technical Report RT/62-92, Januar 1992. Gekürzte Version auch in: Digest of Papers, The 22nd International Symposium on Fault-Tolerant Computing (FTCS), Boston, USA, July 1992.

[Sch 86] Fred B. Schneider, *A Paradigm for Reliable Clock Synchronization.* In: Proc. Advanced Seminar Real Time Local Area Network, April 1986, pp. 85-106.

[Sch 93] Michael D. Schroeder, *A State-of-the-Art Distributed System: Computing with BOB.* In: Distributed Systems, 2nd edition, edited by Sape Mullender, New York 1993, pp. 1-16.

[Sch 00] Stefan Schemmer, *Zuverlässige Echtzeit-Gruppenkommunikation auf einem lokalen Funknetzwerk*. Diplomarbeit, Universität Bonn, Januar 2000.

[ST 87] T. K. Srikanth, and Sam Toueg, *Optimal Clock Synchronization*. Journal of the Association for Computing Machinery (JACM), Vol. 34(3), July 1987, pp. 626-645.

[St 95] Startech Semiconductor Inc, *ST87C34 General Purpose Parallel Printer Port with 83 Byte FIFO*. Data Sheet, Revision 1.0, August 1995.

[Ta 96] Andrew S. Tanenbaum, *Computer Networks*. 3rd edition, Prentice-Hall, Upper Saddle River, NJ, 1996.