GMD Research Series

GMD –
Forschungszentrum
Informationstechnik
GmbH

Stefan Schemmer

Zuverlässige Echtzeit-Gruppenkommunikation auf einem lokalen Funknetz

N° 4/2000

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

Anschrift des Verfassers/Address of the author:
Stefan Schemmer
Institut für Autonome intelligente Systeme
GMD – Forschungszentrum Informationstechnik GmbH
D-53754 Sankt Augustin
E-mail: Stefan.Schemmer@gmd.de

Die Deutsche Bibliothek - CIP-Einheitsaufnahme:

Schemmer, Stefan:
Zuverlässige Echtzeit-Gruppenkommunikation auf einem lokalen Funknetz / Stefan Schemmer. GMD – Forschungszentrum Informationstechnik GmbH. - Sankt Augustin : GMD – Forschungszentrum Informationstechnik, 2000
(GMD Research Series ; 2000, No. 4)
Zugl.: Bonn, Univ., Diplomarbeit, 2000
ISBN 3-88457-376-4

ISSN 1435-2699
ISBN 3-88457-376-4

Kurzfassung[1]

Durch die Fähigkeit zu kommunizieren und zu kooperieren, kann die Effizienz, Robustheit und Flexibilität autonomer Systeme beträchtlich gesteigert werden. Die vorliegende Diplomarbeit leistet einen Beitrag zu einer Architektur zur Unterstützung der Kooperation von autonomen Systemen, die aufgrund ihrer Mobilität Echtzeitanforderungen erfüllen müssen. Ein Kernstück dieser Architektur ist ein Stapel von Protokollen, die vorhersagbare Kommunikation und Konsensbildung auf einem drahtlosen Medium erlauben.

In der vorliegenden Arbeit wird ein echtzeitfähiges Gruppenkommunikationsprotokoll vorgestellt, das auf dem IEEE 802.11 Standard basiert. Aufbauend auf einem Modell der Fehler- und Zeiteigenschaften des Mediums wird dynamische Zeitredundanz eingesetzt, um die Zuverlässigkeit und Rechtzeitigkeit der Nachrichtenübertragung in effizienter Weise sicherzustellen. Das Protokoll ermöglicht es, durch das Einschränken der Zuverlässigkeit der Nachrichtenübertragung bessere Zeitgarantien zu erreichen, so daß Trade-Off-Entscheidungen in anwendungs- und umgebungsspezifischer Weise getroffen werden können. Die Diplomarbeit beinhaltet eine formale Beschreibung des Protokolls in SDL, die für die Herleitung von Zeitschranken und zum Beweis der Korrektheit des Protokolls verwendet wurde. Weiterhin wurde eine Implementierung des Protokolls in er Büro-Umgebung bestehend aus Windows NT PCs und Laptops durchgeführt. Erste Messergebnisse, die im Rahmen dieser Implementierung ermittelt wurden, zeigen, daß das Protokoll die zuverlässige und rechtzeitige Übertragung von Nachrichten auch beim Auftreten von Netzwerkfehlern sicherstellen kann.

Schlagwörter: Echtzeit-Systeme, Verteilte Systeme, Fehler-Toleranz, Gruppenkommunikation, Drahtlose Netzwerke

---

[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

Autonomous systems will significantly gain in efficiency, robustness and flexibility through the ability to communicate and to cooperate. This thesis contributes to an architecture supporting the cooperation of teams of autonomous systems that are mobile and therefore subject to real-time requirements. A core part of this architecture is a stack of wireless communication protocols that allows for predictable communication and consensus among the stations.

In this thesis, a reliable real-time group communication protocol is presented that enhances the IEEE 802.11 Standard. Based upon a model of the fault and timing characteristics of the medium, dynamic time redundancy is applied to ensure reliability and timeliness of message transmission in an efficient manner. The presented protocol allows the application to trade reliability of message transmission for better timing guarantees. This way, trade-off decisions can be taken based on the requirements of the application and its environment. The thesis includes a formal, SDL-based description of the protocol, which has been analyzed in order to derive time bounds for message transmission and prove the protocol's correctness. In addition, an implementation of the protocol for a real office environment consisting of Windows NT PCs and Laptops has been carried out. First measurement results obtained from this implementation are presented that indicate the protocol's ability to provide reliable and timely message transmission in the presence of network faults.

Keywords: Real-Time Systems, Distributed Systems, Fault-Tolerance, Group Communication, Wireless Networks

Zusammenfassung

In Zusammenarbeit mit dem Institut für verteilte Systeme der Universität Magdeburg befasst sich die Gruppe AiS.CAS (Autonomous intelligent Systems – Cooperative Autonomous Systems) der GMD - Forschungszentrum Informationstechnik GmbH mit der Kooperation von autonomen Systemen. Das Ziel der gemeinsamen Arbeit besteht in der Entwicklung einer Systemarchitektur, die autonomen mobilen Systemen die verteilte Durchführung von Aufgaben erlaubt, die unterschiedlichen Zeit- und Zuverlässigkeitsanforderungen genügen müssen. Da autonome mobile Systeme mit ihrer Umwelt interagieren, unterliegt zumindest ein Teil dieser Aufgaben Zeitanforderungen, die durch die Umwelt vorgegeben werden. Bei der Durchführung dieser Echtzeit-Aufgaben müssen die Systeme ein korrektes zeitliches und funktionales Verhalten garantieren, d. h. sie müssen vorhersagbar sein. Lokal, in den einzelnen autonomen Systemen, wird die Vorhersagbarkeit durch Komponenten wie Monitoring oder Echtzeit-Scheduling sichergestellt, die eine unter den Aufgaben koordinierte und effiziente Nutzung der Betriebsmittel gewährleisten. Ein Bestandteil der angestrebten Systemarchitektur sind Kommunikationsprotokolle, die es den autonomen mobilen Systemen ermöglichen, über ein Funknetzwerk Informationen auszutauschen. Diese Protokolle sollen durch eine geeignete Semantik die Entwicklung verteilter Anwendungen vereinfachen. Vor allem müssen sie aber zeitlich vorhersagbar sein, da sie die Verzögerung der Echtzeit-Aufgaben erheblich beeinflussen.

Das Ziel der vorliegenden Diplomarbeit ist die Entwicklung eines zuverlässigen Echtzeit-Gruppenkommunikationsprotokolls für ein lokales Funknetzwerk. Das Protokoll soll in der oben erläuterten Architektur eingesetzt werden. Mit Hilfe des Protokolls können in einer Gruppe von autonomen mobilen Systemen[2], die über ein Funknetzwerk verbunden sind, Broadcast-Nachrichten zuverlässig ausgetauscht werden. Das Protokoll erkennt Stationsausfälle und teilt diese Ereignisse in Form von Änderungsnachrichten den verbleibenden Stationen mit. Alle Stationen, die nicht ausfallen, liefern die gleichen Nachrichten, sowohl Broadcast- als auch Änderungsnachrichten, in der gleichen Reihenfolge aus. So sehen diese Stationen die gleiche, geordnete Folge von Ereignissen (Nachrichten und Stationsausfälle). Durch diese Eigenschaften werden durch das Protokoll viele der durch die Verteiltheit entstehenden Probleme für den Anwendungsentwickler transparent und es ist daher ein starkes Werkzeug für die Implementierung von Kooperationskonzepten auf höheren Ebenen. Um die zeitliche Vorhersagbarkeit zu gewährleisten, stellt das Protokoll sicher, daß sowohl das Übertragen der Broadcast-Nachrichten als auch das Erkennen von Stationsausfällen mit bekannten maximalen Verzögerungen geschieht.

---

[2] Da die Gruppe als ein Funknetzwerk betrachtet wird, werden die autonomen mobilen Systeme im Folgenden als Stationen bezeichnet.

Der Realisierung des Protokolls wird ein Funknetzwerk nach dem IEEE 802.11 Standard zugrunde gelegt. Mit einem zentralisierten Verfahren, das im Standard spezifiziert ist, können die Stationen auf das Funkmedium kollionsfrei zugreifen, so daß sie die Möglichkeit haben, Broadcast-Nachrichten mit bekannter maximaler Verzögerung, d. h. zeitlich vorhersagbar, zu übertragen. Der Standard löst für die Übertragung der Broadcast-Nachrichten aber nicht das Problem der hohen Unzuverlässigkeit von Funkmedien. Funkmedien sind nicht abgeschirmt und daher äußeren Störungen in besonderem Maße ausgeliefert. Weiterhin ist aufgrund der Mobilität der Stationen ihre gegenseitige Erreichbarkeit nicht immer gewährleistet. Das besondere Problem, das in dieser Arbeit zu lösen ist, besteht darin, auf einem solchen Medium die zuverlässige Übertragung der Nachrichten zu garantieren und dabei gleichzeitig sicherzustellen, daß die zeitliche Vorhersagbarkeit der Übertragung erhalten bleibt, d. h., daß auch die zuverlässige Übertragung von Broadcast-Nachrichten mit einer bekannten maximalen Verzögerung vonstatten geht.

Unter der Annahme, daß die Anzahl der Nachrichtenverluste, die in einem bestimmten Zeitintervall auftreten, begrenzt ist, kann dieses Problem durch den Einsatz von statischer oder dynamischer Redundanz gelöst werden. Beim Einsatz statischer Redundanz wird jede Nachricht mehrmals gesendet, unabhängig davon, ob tatsächlich Nachrichtenverluste aufgetreten sind oder nicht. Beim Einsatz dynamischer Redundanz werden Nachrichten nur dann erneut übertragen, wenn es tatsächlich zu Verlusten gekommen ist. Daher kann dynamische Redundanz erheblich effizienter eingesetzt werden als statische Redundanz, wenn der bei dynamischer Redundanz notwendige Mechanismus zur Fehlererkennung effizient realisiert wird. In der vorliegenden Arbeit wird durch eine geeignete Strukturierung des Protokollablaufs die effiziente Erkennung und Behebung von Nachrichtenverlusten erreicht. Die Zuverlässigkeitsanforderungen an die Übertragung können eingeschränkt werden, um geringere Nachrichtenverzögerungen zu erreichen.

Vorwort

Die vorliegende Arbeit entstand in der Gruppe Cooperating Autonomous Systems des Institutes für Autonome intelligente Systeme der GMD. Bei den Mitarbeitern dieser Gruppe bzw. bei den Mitarbeitern des ehemaligen Forschungsbereiches Responsive Systeme des Institutes für Systementwurfstechnologie möchte ich mich an dieser Stelle für ihre stete Bereitschaft mich zu unterstützen bedanken. Ich möchte mich bei Herrn Prof. Dr. Nett bedanken, der das Thema für diese Arbeit gestellt und mit viel konstruktiver Kritik zu ihrer jetzigen Form beigetragen hat; bei Herrn Dr. Mock, meinem Betreuer, der sich immer die Zeit nahm, meine vielfältigen Fragen zu beantworten; bei meinem Kommilitonen Spiro Trikaliotis für die vielen Diskussionen; bei meinen Eltern, sie haben mir mein Studium ermöglicht und mir immer das sichere Gefühl gegeben, daß sie mich in jeder möglichen Weise unterstützen. Ich möchte mich besonders bei meiner Verlobten Kerstin Kraus bedanken. Sie mußte vor allem vor Prüfungen viel Verständnis zeigen und sie hat mir in solchen Situationen immer sehr viel Kraft gegeben.

St. Augustin, Januar 2000

Stefan Schemmer

Inhaltsverzeichnis

1 Einleitung

1.1 Kontext der Diplomarbeit

Autonome Systeme sind in der Lage, selbständig Entscheidungen zu treffen, die der Erreichung vorgegebener Ziele dienen. Dabei stellen sie auch in einer dynamischen und a priori unbekannten Umwelt Fortschritte bei der Erreichung dieser Ziele sicher. Autonome Systeme zeichnen sich daher durch ihre Flexibilität und Robustheit aus.

Diese Eigenschaften können weiter verbessert werden, wenn man autonome Systeme mit der Fähigkeit ausstattet, zu kooperieren. Hierdurch schafft man die Möglichkeit, daß komplexe Aufgaben von Teams von autonomen Robotern bewältigt werden. Bei diesem Ansatz werden aus einer Menge von heterogenen Robotern, die über unterschiedliche Fähigkeiten verfügen, Gruppen zusammengestellt, welche die zum Lösen einer Aufgabe notwendigen Eigenschaften in sich vereinen. Da sich diese Gruppen für unterschiedliche Aufgaben immer wieder neu konfigurieren lassen, kann gegenüber dem Einsatz eines einzelnen Spezialroboters ein Flexibilitätsgewinn erzielt werden. Wenn sich weiterhin die Fähigkeiten der Roboter überschneiden, kann die hierdurch in der Gruppe vorhandene Redundanz genutzt werden, um auch beim Ausfall einzelner Gruppenmitglieder sicherzustellen, daß die vorgegebenen Ziele erreicht werden.

Nicht zuletzt aus diesen Gründen, aber auch, weil es sich um ein Gebiet handelt, das viele interessante Fragen aufwirft, wird die Kooperation von autonomen Systemen in vielen Forschungsbereichen innerhalb, aber auch außerhalb der Informatik untersucht. Die Fragestellungen reichen von ethologischen und soziologischen Untersuchungen zum Gruppenverhalten bis hin zu eher technisch-physikalischen Problemen, z. B. im Bereich der Sensorik. Die Kommunikation ist dabei ein Aspekt, der auf allen Ebenen eine wichtige Rolle spielt. Wenn es auch Ansätze gibt, die versuchen, die Kooperation von autonomen Systemen ohne Kommunikation zu ermöglichen ([Ark92], [KZ97]), so zeigt sich doch, daß Gruppen von autonomen Systemen durch den Einsatz von Kommunikation effizienter werden ([BA94]). Da dies schon bei Aufgaben gilt, die nur eine sehr schwache oder überhaupt keine Synchronität zwischen den Systemen verlangen, wie z. B. der häufig untersuchte Aufgabe der Futtersuche, ist es sehr plausibel, daß für Aufgaben, die mehr Synchronität verlangen, die Kommunikation äußerst wichtig, u. U. sogar unerläßlich ist. Gesteigerte Anforderungen an die Synchronität bei der Aufgabendurchführung, in Form von Fristen und Gleichzeitigkeitsanforderungen, ergeben sich, wenn sich die betrachteten Systeme schnell bewegen. Dies gilt z. B. für die Roboter im RoboCup, einer Fußballweltmeisterschaft für Roboter, oder für Fahrzeuge, die sich in bestimmten Situationen des Straßenverkehrs kooperativ Verhalten sollen, beispielsweise während des Überquerens einer Kreuzung oder beim Einordnen an einer Autobahnauffahrt. In solchen Szenarien muß sichergestellt sein, daß Aufgaben zuverlässig und unter

Einhaltung von Zeitbedingungen durchgeführt werden, die durch die Geschwindigkeit der Systeme vorgegeben sind.

In einer Kooperation der Gruppe AiS.CAS der GMD und des Instituts für Verteilte Systeme der Universität Magdeburg werden solche Szenarien autonomer mobiler Systeme betrachtet, die kommunizieren, um ihr Verhalten zu koordinieren, und die aufgrund ihrer Mobilität Echtzeitanforderungen unterliegen, d. h. Fristen einhalten müssen, die durch die Umwelt vorgegeben werden. Das Ziel der hier verfolgten Arbeit besteht darin, eine Systemarchitektur zu erstellen, die durch die Bereitstellung geeigneter Paradigmen die Entwicklung von verteilten Echtzeitanwendungen auf höheren Ebenen vereinfacht und die Vorhersagbarkeit bei der Ausführung dieser Anwendungen auf verteilten Robotiksystemen sicherstellt. Diese Architektur soll viele Aspekte der Verteiltheit für den Anwendungsentwickler transparent machen. Unter solchen Aspekten sind z. B. die Konkurrenz mehrerer Prozesse um gemeinsame Betriebsmittel, der Zugriff auf entfernte Objekte, die Effekte von Nachrichtenverlusten oder die Folgen unabhängiger Stationsausfälle zu verstehen. Eine Architektur, durch die solche Aspekte transparent werden, erlaubt es dem Anwendungsentwickler, Kooperationskonzepte auf höheren Ebenen leichter und effizienter zu realisieren.

Ein wichtiger Bestandteil der entwickelten Systemarchitektur sind Kommunikationsprotokolle, die es den autonomen mobilen Systemen erlauben, über ein lokales Funkmedium Informationen auszutauschen und dabei verschiedene Service-Eigenschaften in Anspruch zu nehmen. Hierzu gehören auch Protokolle, die für die Kommunikation in verteilten Echtzeitanwendungen eingesetzt werden sollen. Solche Protokolle müssen vor allem zwei Eigenschaften genügen ([KN98]): Sie müssen zum einen vorhersagbar sein, d. h. ein korrektes zeitliches und funktionales Verhalten garantieren, auch dann, wenn es in einzelnen Komponenten des Systems zu Ausfällen kommt. Zum anderen müssen sie die Entwicklung von verteilten, kooperativen Anwendungen durch eine geeignete Semantik unterstützen.

Beide Eigenschaften werden durch zuverlässige Echtzeit-Gruppenkommunikationsprotokolle erfüllt. Die Verfügbarkeit eines solchen Protokolls vereinfacht erheblich die Realisierung von Kooperationskonzepten auf höheren Ebenen. Eine Möglichkeit, den Service eines solchen Protokolls einzusetzen, besteht z. B. in der Realisierung eines verteilten Blackboards. Das Gruppenkommunikationsprotokoll stellt dabei sicher, daß alle Stationen auf dem Blackboard die gleichen Botschaften in der gleichen Reihenfolge lesen, obwohl die Kommunikation nur auf einem unzuverlässigen Medium erfolgt. Gruppenkommunikationsprotokolle ermöglichen es, auch den Ausfall von Systemen auf dem Blackboard anzuzeigen, derart, daß die verbleibenden Systeme bestimmen können, welche Botschaften ein ausgefallenes System noch gelesen haben kann und welche nicht mehr. Schließlich ist durch die Vorhersagbarkeit des Gruppenkommunikationsprotokolls die zeitliche Kohärenz des Blackboards gewährleistet, d. h. eine Botschaft, die ein System auf dem Blackboard einträgt, kann nach einer bekannten maximalen Verzögerung von allen anderen Systemen wahrgenommen werden. Gruppenkommunikation kann aber auch für die effiziente Lösung anderer wichtiger Probleme in verteilten Sys-

temen, wie z. B. Konsensus, Wahl eines eindeutigen Koordinators, Konsistenz von Replikaten, nicht blockierende atomare Transaktionen, eingesetzt werden.

1.2 Ziele der Diplomarbeit

Das Ziel der vorliegenden Diplomarbeit besteht in der Entwicklung eines zuverlässigen Echtzeit-Gruppenkommunikationsprotokolls für Gruppen von autonomen mobilen Systemen (Stationen), die über ein gemeinsam genutztes Funknetz verbunden sind ([MNS99]). Das Protokoll soll sicherstellen, daß alle korrekten, d. h. nicht ausgefallenen, Stationen zuverlässig Broadcast-Nachrichten übertragen können und die gleichen Broadcast-Nachrichten in der gleichen Reihenfolge ausliefern. Weiterhin soll das Protokoll sicherstellen, daß alle Stationsausfälle erkannt und daraufhin Änderungsnachrichten an die korrekten Stationen ausgeliefert werden, derart, daß diese die Änderungsnachrichten zwischen den gleichen Broadcast-Nachrichten ausliefern. Vor allem aber muß das Protokoll vorhersagbar sein, d. h. garantieren, daß sowohl die Übertragung der Broadcast-Nachrichten als auch das Ausliefern der Änderungsnachricht nach einem Ausfall mit bekannten maximalen Verzögerungen geschieht (Rechtzeitigkeit). Genaue Definitionen der Eigenschaften des Protokolls, die sich unter den Oberbegriffen rechtzeitige, atomare Broadcasts und rechtzeitiger Teilnehmerservice zusammenfassen lassen, werden in Kapitel 2 angegeben.

Als Grundlage für die Realisierung dieses Protokolls dient ein lokales Funknetz nach dem IEEE 802.11 Standard ([IEEE97]). Die Verwendung dieses Standards ermöglicht zum einen den Einsatz von Standard-Hardwarekomponenten für die Kommunikation unter den mobilen Systemen, zum anderen ist im IEEE 802.11 Standard neben einem verteilten Medien-Zugriffsverfahren auch ein zentralisiertes Zugriffsverfahren spezifiziert, mit dem die Stationen zeitlich vorhersagbar auf das Medium zugreifen können und das damit eine gute Grundlage für die Realisierung vorhersagbarer Kommunikationsprotokolle bildet. Das verteilte Zugriffsverfahren basiert auf der CSMA-Methode. Es ermöglicht eine effektive Nutzung der zur Verfügung stehenden Bandbreite, stellt aber nicht sicher, daß eine Station nach begrenzter Zeit Zugriff auf das Medium erhält. Bei dem zentralisierten Zugriffsverfahren gewährt eine spezielle Station, der Access Point oder AP, durch das Versenden von Polling-Nachrichten exklusiven Zugriff auf das Medium. Auf einem Funknetz nach dem IEEE 802.11 Standard, das beide Zugriffsverfahren zum Einsatz bringt, alternieren Phasen des verteilten Zugriffs, contention periods (CP), und des zentral gesteuerten Zugriffs, contention free periods (CFP). Die Verfügbarkeit der beiden Zugriffsverfahren im zeitlichen Wechsel erlaubt das Versenden von Nachrichten mit unterschiedlichen Zeitanforderungen auf dem gleichen Medium ([MN99]): Da die verteilte Zugriffsmethode eine effiziente Nutzung des Mediums erlaubt, aber der Medienzugriff zeitlich nicht vorhersagbar ist, kann die CP genutzt werden, um Nachrichten zu übertragen, für die eine begrenzte Verzögerung nicht garantiert werden muß. In der CFP hingegen kann durch das Polling-Verfahren sichergestellt werden, daß die Kommunikation mit einer bekannten maximalen Verzögerung stattfindet.

Daher gründet sich das Gruppenkommunikationsprotokoll auf die CFP, während welcher der Zugriff auf das Medium zeitlich vorhersagbar ist.

Da die Gruppenkommunikation ein Paradigma ist, das die Entwicklung von verteilten Anwendungen erheblich vereinfacht, wurde sie in vielen verteilten Systemen realisiert ([CM84], AAS [CASD85], Amoeba [KT91], Delta-4 [RV92], ISIS [BSS91], RMP [JKN96], Totem [AMMAC95], Transis/ToTo [ADKM92b], TTP [KG93]). Trotz der vielfältigen vorhandenen Gruppenkommunikationsprotokolle stellt die Realisierung eines solchen Protokolls für ein Funknetz nach dem IEEE 802.11 Standard eine neue Herausforderung dar, weil

1. Viele der oben erwähnten Protokolle in zeitunkritischen Systemen zum Einsatz kommen. Daher ist die Rechtzeitigkeit der Nachrichtenübertragung nicht von Belang, so daß Konzepte eingesetzt werden, die die Rechtzeitigkeit nicht explizit berücksichtigen.

2. Systeme, in denen auch die Rechtzeitigkeit der Übertragung berücksichtigt wird, gehen i. A. davon aus, daß Nachrichtenverluste gar nicht oder nur sehr selten auftreten. Diese Annahme wird durch den Einsatz redundanter kabelgebundener Medien und deren relativ hohe Zuverlässigkeit sichergestellt. Funkmedien sind aber von Natur aus unzuverlässig und der Einsatz mehrerer Funkmedien erscheint wenig sinnvoll.

3. Das Funkmedium hat nur eine kleine Bandbreite (1 bis 2 Mbit/s); deshalb muß besonderer Wert darauf gelegt werden, den Aufwand, den das Protokoll in Form von zusätzlichen Nachrichten erzeugt, klein zu halten.

4. Das Protokoll soll die spezielle Kommunikationsstruktur des Standards, bei der der AP als zentraler Koordinator durch das Versenden von Polling-Nachrichten den Stationen exklusiven Zugriff auf das Medium gewährt, berücksichtigen.

5. Vorhandene Teilnehmerprotokolle gehen davon aus, daß sich Gruppen aus Stationen bilden, die sich alle wechselseitig erreichen können. Der Gruppenbegriff des IEEE 802.11 Standards ist aber ein anderer; hier bilden alle Stationen eine Gruppe, die sich im Sende-/Empfangsbereich des AP befinden.

6. Da sich auf dem Medium die Phasen CFP und CP abwechseln, soll der für die CFP benötigte Zeitaufwand möglichst klein gehalten werden, um die der CP zur Verfügung stehende Zeit nach Möglichkeit auszudehnen.

Das hauptsächliche und auch neue Problem, das sich im Rahmer der vorliegenden Arbeit stellt, kann wie folgt zusammengefaßt werden: Wie kann man auf einem Medium, auf dem in erheblichem Maße Nachrichtenverluste auftreten können, gleichzeitig die Zuverlässigkeit und die Rechtzeitigkeit der Nachrichtenübertragung sicherstellen und dabei sowohl der geringen Bandbreite als auch der speziellen Kommunikationsstruktur des IEEE 802.11 Standards Rechnung tragen ?

1.3 Ergebnisse der Diplomarbeit

Um ein Protokoll entwickeln zu können, das sowohl die zuverlässige als auch die rechtzeitige Übertragung der Nachrichten sicherstellt, müssen zunächst Annahmen über das Funknetzwerk getroffen werden, die dieses zum einen zutreffend beschreiben, zum anderen aber ausreichend sind, um die geforderten Eigenschaften zu realisieren. Unter den zwei wesentlichen Annahmen, daß jede Station, die eine Nachricht übertragen möchte, dies mit einer maximalen Verzögerung t_m tun kann und daß die Anzahl der aufeinanderfolgenden Nachrichtenverluste in einem bestimmten Zeitintervall begrenzt ist (sog. Omission Degree *OD*), gibt es grundsätzlich zwei Möglichkeiten, die rechtzeitige und zuverlässige Übertragung der Nachrichten sicherzustellen ([VRR91]):

1. Durch den Einsatz statischer Redundanz. Bei diesem Ansatz wird jede Nachricht *OD+1*-mal gesendet, unabhängig davon, ob tatsächlich Nachrichtenverluste aufgetreten sind oder nicht.

2. Durch den Einsatz dynamischer Redundanz. Bei diesem Ansatz wird zunächst durch einen Fehlererkennungsmechanismus festgestellt, ob Nachrichtenverluste aufgetreten sind. Nur wenn dies der Fall ist, wird die betroffene Nachricht erneut übertragen.

Sowohl bei statischer als auch bei dynamischer Redundanz wird die maximale Verzögerung durch die maximale Anzahl von Nachrichtenverlusten *OD* bestimmt. Bei statischer Redundanz ist die maximale Verzögerung kleiner, da hier der Aufwand für die Fehlererkennung entfällt. Während beim Einsatz statischer Redundanz diese maximale Verzögerung aber immer eintritt, unabhängig davon, wie viele Nachrichtenverluste tatsächlich aufgetreten sind, richtet sich die tatsächliche Verzögerung der Nachrichten und die benötigte Bandbreite beim Einsatz dynamischer Redundanz nach der tatsächlichen Anzahl von Nachrichtenverlusten. Daher ist auf einem Funknetz der Einsatz dynamischer Redundanz vorzuziehen, da für ein Funknetz eine große Obergrenze *OD* angenommen werden muß, die tatsächliche Anzahl von Fehlern aber i. A. deutlich darunter liegen wird. Auf diese Art kann die ganze Zeit, die für den ungünstigsten Fall eingeplant werden muß, die aber tatsächlich nicht benötigt wird, für die Nachrichtenübertragung in der CP verwendet werden.

Das wesentliche Problem beim Einsatz dynamischer Redundanz besteht im Aufwand der Fehlererkennung. Fehlererkennung beim Übertragen von Nachrichten wird durch den Einsatz von Bestätigungen realisiert. Es muß verhindert werden, daß der durch das Versenden von Bestätigungen implizierte Aufwand die Vorteile der dynamischen Redundanz gegenüber der statischen wieder aufhebt. Der Ablauf des Protokolls wurde in zweifacher Weise gegliedert.

Zum Ersten wurde die Übertragung einer einzelnen Nachricht von einer Station an die Gruppe in zwei Abschnitte unterteilt: Die Station überträgt die Nachricht an den AP und dieser broadcastet die Nachricht an die Gruppe. Auf diese Art kommt der AP als zentraler Router für die Nachrichten zum Einsatz, so daß aufgrund der Definition der Gruppe

sichergestellt ist, daß jede Broadcast-Nachricht von allen Stationen empfangen wird, solange keine Nachrichtenverluste auf dem Medium auftreten. Die Erreichbarkeit der Stationen untereinander, die sich durch die Mobilität der Stationen beständig ändern kann, wird somit für die Realisierung des Gruppenkommunikationsprotokolls unerheblich. Die Fehlererkennung beim Übertragen von Nachrichten von den Stationen an den AP geschieht durch ein implizites Bestätigungsschema, das keine zusätzlichen Nachrichten benötigt. Dabei wird durch den Omission-Degree *OD* sichergestellt, daß jede Nachricht den AP mit einer bekannten maximalen Verzögerung erreicht.

Um auch auf dem zweiten Kommunikationsabschnitt, vom AP zu den Stationen, Nachrichtenverluste effizient erkennen zu können, wurde der zeitliche Ablauf des Protokolls weiter in Runden unterteilt. In jeder Runde pollt der AP jede Station genau einmal. Auf diese Art können die Stationen die Nachrichten, die sie ohnehin an den AP übertragen, nutzen, um via Piggy-Backing für alle Broadcast-Nachrichten der vorhergehenden Runde eine Bestätigung an den AP zu senden. Durch dieses Verfahren kann der AP eine Runde nachdem er eine Broadcast-Nachricht gesendet hat feststellen, ob diese von allen Stationen empfangen wurde, ohne daß hierzu zusätzliche Nachrichten benötigt werden. Weiterhin können durch diese Struktur die Broadcasts der einzelnen Stationen nebenläufig ausgeführt werden.

Durch den effizienten Einsatz dynamischer Redundanz in der oben beschriebenen Weise gelingt es, die tatsächliche Verzögerung der Nachrichten und den tatsächlichen Aufwand der Übertragung nach Möglichkeit klein zu halten. Die maximale Verzögerung bleibt aber dennoch unverändert groß, was der Preis für das garantierte Tolerieren von bis zu *OD* Nachrichtenverlusten ist. Auf der anderen Seite gibt es, vor allem bei Systemen, die sich in einer dynamischen und a priori unbekannten Umgebung bewegen, Anwendungen, die gelegentliche Nachrichtenverluste tolerieren können, solange diese von allen Stationen konsistent wahrgenommen werden, d. h. solange weiterhin entweder alle Stationen eine Nachricht ausliefern oder keine. Auf diese Weise bleibt trotz Nachrichtenverlusten das Verhalten der Gruppe in sich schlüssig. Anwendungen, die die garantierte Zuverlässigkeit der Übertragung nicht in Anspruch nehmen müssen, sollten auch nicht genötigt werden, den dafür entstehenden Preis zu zahlen. Deshalb führt das Protokoll das Konzept der Resiliency von Nachrichten ein, bei dem der Anwender durch die Resiliency einer Nachricht bestimmen kann, wie häufig diese maximal übertragen werden soll. Durch die Wahl einer kleinen Resiliency erhöht der Anwender die Wahrscheinlichkeit, daß eine Nachricht nicht an alle Stationen übertragen werden kann, er verringert aber gleichzeitig die maximale Verzögerung der betreffenden Nachricht.

Da aufgrund der Möglichkeit der eingeschränkten Zuverlässigkeit von Nachrichten nicht mehr alle Nachrichten von allen Stationen empfangen werden, muß das Protokoll sicherstellen, daß die Einigung der Stationen über die ausgelieferten Nachrichten weiterhin gewährleistet bleibt. Aus diesem Grunde wird durch das Protokoll in effizienter Weise ein synchroner Kanal mit kleiner Bandbreite realisiert, der es dem AP erlaubt, Bit-Tupel zuverlässig und rechtzeitig an die Stationen zu übertragen. Diese Möglichkeit benutzt der AP, um die rechtzeitige Einigung unter den Stationen zu erreichen. Dazu

versendet der AP über den synchronen Kanal *accept*- und *reject*- Entscheidungen, mit denen er den Stationen mitteilt, ob sie eine Nachricht ausliefern oder verwerfen sollen.

Schließlich nutzt das Protokoll die Struktur des Protokollablaufs und den synchronen Kanal auch, um Stationsausfälle festzustellen und die verbleibenden Stationen darüber zu informieren. Der AP kann den Ausfall einer Station feststellen, wenn diese mehr als *OD*-mal nicht auf sein Pollen reagiert hat. In diesem Falle benutzt er den synchronen Kanal, um über diesen die Änderung der Gruppe den verbleibenden Stationen mitzuteilen.

Durch die im Vorhergehenden kurz erläuterten Verfahren stellt das entwickelte Gruppenkommunikationsprotokoll alle für das Übertragen von Broadcast-Nachrichten und das Erkennen von Ausfällen geforderten Eigenschaften sicher. Dabei hat das Gruppenkommunikationsprotokoll die folgenden besonderen Vorzüge:

1. Die Tolerierung von Nachrichtenverlusten wird durch dynamische Redundanz erreicht. Auf diese Weise ist der tatsächliche Aufwand und die tatsächliche Verzögerung der Nachrichtenübertragung von der tatsächlichen Anzahl von Fehlern abhängig. Daher kann gegenüber dem bei der Bestimmung der maximalen Verzögerung zugrunde gelegten ungünstigsten Fall in erheblichem Maße Zeit eingespart werden, die für die Übertragung von Nachrichten ohne Echtzeitanforderungen in der CP verwendet werden kann.

2. Die Realisierung des Fehlererkennungsmechanismus, der Grundlage für den Einsatz dynamischer Redundanz ist, erfolgt durch eine geeignete Strukturierung der Kommunikation und durch den Einsatz von impliziten Bestätigungen und Piggy-Backing. So werden für das Versenden von Bestätigungen i. A. keine zusätzlichen Nachrichten benötigt.

3. Das Protokoll erlaubt dem Anwender, die Zuverlässigkeitsanforderungen an die Nachrichtenübertragung einzuschränken und dadurch kleinere maximale Verzögerungen und einen besseren Echtzeitdurchsatz zu erreichen. Dabei stellt das Protokoll sicher, daß die Einigung der Stationen über die ausgelieferten Nachrichten auch dann sichergestellt ist, wenn es zu Nachrichtenverlusten kommt.

4. Um wie unter 3. beschrieben die Einigung unter den Stationen rechtzeitig sicherzustellen, wurde ein synchroner Kanal geringer Bandbreite realisiert, in dem der AP Bit-Tupel an die Stationen übertragen kann. Auch dieser Kanal wurde mit Hilfe von Piggy-Backing realisiert, so daß auch hier i. A. keine zusätzlichen Nachrichten übertragen werden müssen.

5. Das Übertragen von Broadcast-Nachrichten von mehreren Stationen kann nebenläufig erfolgen.

6. Eine aktuelle und konsistente Sicht der Gruppe kann in allen korrekten Stationen mit den für die Übertragung der Broadcast-Nachrichten entwickelten Konzepten realisiert werden, so daß hierfür kein zusätzlicher Nachrichtenaufwand entsteht.

1.4 Aufbau der Diplomarbeit

Die Diplomarbeit ist wie folgt gegliedert. In Kapitel 2 wird die Problemstellung verfeinert, indem zum einen die Servicegarantien des Gruppenkommunikationsprotokolls genau definiert werden und zum anderen die Systemannahmen, auf die sich der Entwurf gründen soll, angegeben werden. Anschließend werden in Kapitel 3 verwandte Arbeiten zu den Themen atomare Broadcasts und Teilnehmerprotokolle vorgestellt und diskutiert. In Kaptitel 4 wird das Protokoll beschrieben und erläutert. Eine formale Darstellung des Protokolls und Beweise der erreichten Eigenschaften finden sich in Kapitel 5. Kapitel 6 enthält Angaben über die Implementierung des Protokolls, die unter Windows NT 4.0 mit der Programmiersprache C++ vorgenommen wurde. Ergebnisse von Messungen, die an dieser Implementierung durchgeführt wurden, sind ebenfalls in Kapitel 6 dargestellt. Die Arbeit schließt mit Kapitel 7, in dem die Schlüsse aus der Arbeit zusammengefaßt werden und ein Ausblick auf mögliche Folgearbeiten gegeben wird.

2 Grundlagen des Entwurfs

Als Grundlage des Entwurfs des Protokolls werden in diesem Kapitel zunächst die Eigenschaften, die das Protokoll gewährleisten soll, sowie die Systemannahmen, auf die sich der Entwurf gründet, genau spezifiziert.

In Unterkapitel 2.1 wird eine erste Abstraktion des Systems vorgenommen. Aufbauend auf dieser werden die geforderten Eigenschaften spezifiziert, wobei auf in der Literatur gebräuchliche Definitionen zurückgegriffen werden kann. In Unterkapitel 2.2 werden die Systemannahmen beschrieben. Systemannahmen, die die Eigenschaften eines Funknetzwerkes nach dem IEEE 802.11 Standard beschreiben und gleichzeitig die Realisierung von Rechtzeitigkeit und Zuverlässigkeit erlauben, sind in der Literatur noch nicht untersucht worden und müssen daher erst noch ermittelt werden. Dazu werden in Unterkapitel 2.2 schon bestehende Modelle erläutert und ihr Einfluß auf die realisierbaren Protokolleigenschaften untersucht. Anschließend wird der IEEE 802.11 Standard näher betrachtet, um schließlich die Systemannahmen detailliert angeben zu können.

2.1 Beschreibung der geforderten Eigenschaften

Gruppenkommunikationsprotokolle erlauben einer Gruppe von Stationen untereinander Broadcast-Nachrichten auszutauschen. Broadcast-Nachrichten, also Nachrichten, die sich nicht an einzelne Stationen richten, sondern an die Gruppe als ganzes, sind für die Kommunikation besonders geeignet, wenn Aufgaben von Gruppen von Stationen gemeinsam durchgeführt werden. Das Gruppenkommunikationsprotokoll stellt dabei sicher, daß alle Stationen die gleiche Sicht auf die ausgetauschten Nachrichten haben und daß somit das Verhalten in der Gruppe immer in sich schlüssig gehalten werden kann. Gruppenkommunikationsprotokolle können konzeptionell und auch tatsächlich in zwei Protokolle unterteilt werden, von denen jedes einen bestimmten Dienst zur Verfügung stellt. Das Teilnehmerprotokoll sorgt dafür, daß alle Stationen eine aktuelle und konsistente Sicht der Gruppe haben; das Broadcastprotokoll stellt sicher, daß die in der Gruppe versendeten Nachrichten bestimmten Eigenschaften genügen. Im Folgenden sollen die Eigenschaften der beiden Protokolle genau spezifiziert werden. Dazu wird zunächst der Begriff der Gruppe sowie die zur Definition der Eigenschaften notwendige Schnittstelle zwischen Protokoll und Benutzer näher erläutert.

2.1.1 Die Gruppe als Funk-Rechnernetz

Die physische Struktur

Wenn man eine Gruppe von autonomen mobilen Systemen auf der Ebene der Kommunikationsprotokolle betrachtet, dann stellt sie sich als eine Menge von Stationen[3] dar, die alle über eigene Hard- und Software verfügen. Diese Stationen haben keinen gemeinsamen Speicher und keine gemeinsame Uhr. Alle Stationen sind aber über entsprechende Hardware an ein Funkmedium angeschlossen, über das sie Nachrichten austauschen können. Auf dieser Ebene betrachtet bildet also eine Gruppe von autonomen mobilen Systemen ein Rechnernetz.

Das Funkmedium stellt für die Stationen ein gemeinsam genutztes Betriebsmittel dar. Dieses kann jeweils nur von einer Station zum Übertragen einer Nachricht genutzt werden. Alle Stationen können aber gleichzeitig lesend auf das Funknetz zugreifen und jede Nachricht, die auf diesem Übertragen wird, empfangen, solange keine Fehler auftreten (Broadcast-Eigenschaft). Aufgrund der begrenzten Reichweite des Funknetzes und der Mobilität der Stationen ist die Broadcast-Eigenschaft bei einem Funknetz aber nicht immer vollständig gegeben; so kann es vorkommen, daß eine Station eine Nachricht sendet, aber nicht alle anderen Stationen in ihrer Reichweite liegen. Greifen zwei Stationen gleichzeitig auf das gemeinsame Medium zu, d. h., versuchen zwei Stationen zur gleichen Zeit, Nachrichten auf dem Medium zu übertragen, dann kommt es zu einer Kollision. Bei einer Kollision überlagern sich die physischen Signale, die die beiden Nachrichten repräsentieren, und keine der beiden Nachrichten kann korrekt empfangen werden. Um dieses zu vermeiden, werden Zugriffsverfahren angewendet, sog. Medium Access Control (MAC) Protokolle.

Als Grundlage für die Entwicklung des Gruppenkommunikationsprotokolls werden ein Medium und ein Zugriffsverfahren für dieses Medium verwendet, wie sie im IEEE 802.11 Standard beschrieben sind (s. Paragraph 2.2.2). Da im Speziellen das in diesem Standard definierte Zugriffsverfahren der PCF eingesetzt werden soll, besteht die betrachtete Gruppe aus allen Stationen, die sich im Sende-/Empfangsbereich eines bestimmten Access Point, des zentralen Koordinators der PCF, befinden. Das Rechnernetz bildet sich daher an bestimmten physischen Orten, an denen ein AP positioniert ist, z. B. auf einer Kreuzung oder an einer Autobahnauffahrt. Das Rechnernetz kann sich dynamisch verändern, wenn sich Stationen aus dem Empfangsbereich des AP entfernen oder sich in diesen hineinbegeben.

Die abstrakte Schnittstelle des Gruppenkommunikationsprotokolls

Das Gruppenkommunikationsprotokoll realisiert eine Schicht in einem Stapel aufeinanderfolgender Kommunikationsprotokolle, die die Aufgabe haben, von Schicht zu

---

[3] In Kontext verteilter Systeme werden diese häufig als Stellen oder Knoten bezeichnet. In der Diplomarbeit wird aber die im IEEE 802.11 Standard übliche Bezeichnung "Station" übernommen.

Schicht mehr Eigenschaften des Kommunikationsmediums zu verbergen und dafür vermehrt anwendungsorientierte Eigenschaften zur Verfügung zu stellen. Jede Schicht verwendet den Dienst der nächst niedrigeren Schicht und realisiert darauf aufbauend einen Dienst für die nächst höheren Schicht. Jede Schicht wird durch ein Protokoll realisiert, das zwei Schnittstellen hat, eine zur nächst niedrigeren Schicht, um auf deren Dienst zugreifen zu können, und eine zur nächst höheren Schicht, über die diese auf den Dienst des Protokolls zugreift (s. Abb 2.1).

Da für das Protokoll nur die unmittelbar benachbarten Schichten relevant sind, sind in Abbildung 2.1 nur drei Schichten dargestellt, wobei die Schicht „Benutzer“ diejenige Schicht bezeichnet, die sich unmittelbar über dem Gruppenkommunikationsprotokoll befindet, d. h. die auf den Dienst des Gruppenkommunikationsprotokolls zugreift. Der Benutzer muß aber keine Anwendung auf der obersten Schicht, sondern kann durchaus selbst wieder ein Protokoll sein. Die Schicht, die sich unmittelbar unter dem Gruppenkommunikationsprotokoll befindet, wird als „Kommunikationsmedium“ bezeichnet, was allerdings nicht ausdrücken soll, daß es sich bei dieser Schicht tatsächlich um die unterste Schicht, also die physische Verbindung handelt. Wie der Dienst des Kommunikationsmediums realisiert ist, ist für die Funktionsweise des Protokolls nicht relevant.

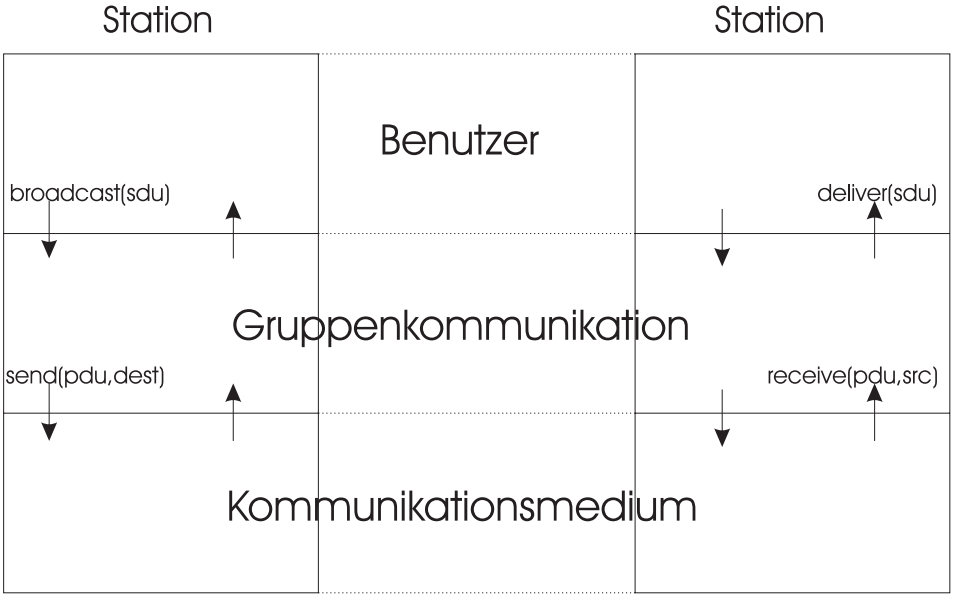


Abbildung 2.1 Darstellung der Schichten

Um das Gruppenkommunikationsprotokoll und seine Eigenschaften beschreiben zu können, werden zwei abstrakte Schnittstellen definiert: Das Gruppenkommunikationsprotokoll kann durch die Aufrufe *send(m, dest)* und *receive(m, src)* auf den Dienst des Kommunikationsmediums zugreifen. Durch Aufruf von *send(m, dest)* kann eine Station *s* eine Protokollnachricht *m* (Protocol Data Unit, PDU) an die Station *dest* versenden (im Folgenden als „Station *s* sendet *m* an *dest*“ bezeichnet). Als Zieladresse kann auch *all* angegeben werden, so daß sich die Nachricht an alle Stationen des Rechnernetzes

richtet. Durch den Aufruf von *receive(m, src)* können die Protokolle Nachrichten empfangen, die an ihre Station gerichtet sind, wobei ihnen in *src* auch der Absender der Nachricht übermittelt wird (im Folgenden als „Station *s* empfängt *m* von *src*“ bezeichnet). Der Dienst, auf den das Gruppenkommunikationsprotokoll durch den Aufruf von *send* bzw. *receive* zugreift, wird in Kapitel 2.2 durch die Systemannahmen beschrieben.

Der Benutzer kann durch Aufrufen des *broadcast(m)*-Primitivs eine Nachricht *m* an das Gruppenkommunikationsprotokoll übergeben mit dem Auftrag, diese Nachricht atomar und rechtzeitig an die Gruppe zu versenden. Der Aufruf von *broadcast(m)* durch die Anwendung auf Station *s* wird im Folgenden als „*s* broadcastet *m*“ bezeichnet; die Station *s* wird dabei als Urheber von *m* bezeichnet. Die Anwendungsnachrichten werden aus Sicht des Broadcastprotokolls als Service Data Units (SDUs) bezeichnet. Durch den Aufruf von *deliver(m)* können das Broadcast- und das Teilnehmerprotokoll der Anwendung Nachrichten an der Schnittstelle zum Empfangen zur Verfügung stellen. Ob und wann eine solche Nachricht tatsächlich von der Anwendung empfangen wird, liegt im Bereich der Anwendung und wird hier nicht betrachtet. Wenn das Teilnehmer- oder das Broadcastprotokoll einer Station *s deliver(m)* aufruft, dann wird dies im folgenden als „*s* liefert *m* aus“ bezeichnet.

Da zwischen dem Primitiven *send* und *receive* und den Primitiven *broadcast* und *deliver* das Broadcastprotokoll operiert, kann eine Station *s* eine Nachrichten empfangen, ohne diese sofort oder überhaupt an die Anwendung auszuliefern. Umgekehrt kann sie aber auch Nachrichten ausliefern, die sie nicht empfangen hat. Aus Sicht der Anwendung ist die Verzögerung einer Nachricht *m* die Zeit, die zwischen dem Aufruf von *broadcast(m)* und dem Aufruf von *deliver(m)* beim letzten Empfänger von *m* vergeht. Aus Sicht des Protokolls ist die Verzögerung einer Nachricht *m* die Zeit, die zwischen dem Aufruf von *send(m, dest)* und dem Aufruf von *receive(m, src)* beim Empfänger bzw. beim letzten Empfänger vergeht.

2.1.2 Eigenschaften des Broadcastprotokolls

Das im Rahmen der Diplomarbeit entwickelte Broadcastprotokoll soll sicherstellen, daß die Broadcast-Nachrichten den Eigenschaften Atomarität und Rechtzeitigkeit genügen. Die Atomarität von Broadcast-Nachrichten wird nach [HT93] durch die folgenden Eigenschaften beschrieben (In den Definitionen sind mit den ausgelieferten Nachrichten nur die gemeint, die das Broadcastprotokoll an die Anwendung ausliefert).

Integrität: Jede Nachricht (SDU) wird von jeder korrekten, d. h. nicht ausgefallenen[4], Station höchstens einmal ausgeliefert und nur dann, wenn sie gebroadcastet wurde.

Validität: Wenn eine korrekte Station eine Nachricht broadcastet, dann wird diese Nachricht von jeder korrekten Station ausgeliefert.

---

[4] Der Begriff des Stationsausfalls wird in Paragraph 2.2.1 näher erläutert.

Einigung: Wenn eine korrekte Station eine Nachricht ausliefert, dann wird diese Nachricht von jeder korrekten Station ausgeliefert.

Totale Ordnung: Wenn zwei korrekte Stationen zwei Nachrichten m_1 und m_2 ausliefern, dann liefern beide m_1 und m_2 in der gleichen Reihenfolge aus.

Broadcast-Nachrichten, die den ersten drei Eigenschaften genügen, werden als zuverlässige Broadcasts bezeichnet. Die Eigenschaft der Integrität stellt sicher, daß es sich bei einer Nachricht, die von einer korrekten Station ausgeliefert wird, nicht um eine zufällig „aus dem Nichts“ entstandene Nachricht handelt. Um die Integrität der Broadcast-Nachrichten zu erreichen, muß bei der Entwicklung des Protokolls vor allem darauf geachtet werden, daß durch die Algorithmen des Protokolls keine Duplikate von Broadcast-Nachrichten entstehen. Die Validität besagt, daß der Verlust von Nachrichten auf dem Netzwerk oder der Ausfall von Stationen nicht dazu führen darf, daß korrekte Stationen eine Nachricht, die von einer korrekten Station gebroadcastet wurde, nicht ausliefern. Die Validität kann daher auch als die Zuverlässigkeit im engeren Sinne aufgefaßt werden. Das Sicherstellen der Valitdität ist aufgrund der Unzuverlässigkeit des Mediums eines der Hauptprobleme bei der Entwicklung des Protokolls. Wenn daher im Folgenden von der Zuverlässigkeit der Übertragung von Broadcast-Nachrichten gesprochen wird, so bezieht sich dies auf die Validität. Die Validität stellt auch die Einigung zwischen den korrekten Stationen bzgl. solcher Nachrichten sicher, die von korrekten Stationen gebroadcastet wurden. Über Nachrichten von ausgefallenen Stationen trifft die Validität hingegen keine Aussage. Für diese stellt die Eigenschaft der Einigung sicher, daß die korrekten Stationen einen Konsens erreichen, d. h., daß solche Nachrichten entweder von jeder korrekten Station ausgeliefert werden oder von keiner. Aufgrund der totalen Ordnung liefern schließlich alle korrekten Stationen die Nachrichten, die sie ausliefern, in der gleichen Reihenfolge aus.

Die bisher getroffenen Definitionen der Eigenschaften atomarer Broadcasts müssen nur von den korrekten Stationen erfüllt werden. Oft ist es aber von entscheidender Bedeutung sicherzustellen, daß die Eigenschaften auch von ausgefallenen Stationen eingehalten werden. Mit den bisher getroffenen Definitionen ist es einer korrekten Station z. B. nicht möglich festzustellen, welche Nachrichten eine ausgefallene Station erhalten hat. Eine ausgefallene Station könnte z. B. Nachrichten ausliefern, die eine korrekte Station nicht ausliefert. Um die Eigenschaften auch auf ausgefallene Stationen auszudehnen, wird zu jeder Eigenschaft eine entsprechende uniforme Eigenschaft definiert. So besagt die Eigenschaft der uniformen Einigung:

Uniforme Einigung: Wenn eine Station eine Nachricht ausliefert, dann wird diese Nachricht von jeder korrekten Station ausgeliefert.

Durch die uniforme Einigung ist sichergestellt, daß ausgefallene Stationen niemals Nachrichten ausliefern, die von den korrekten Stationen nicht ausgeliefert werden. Die uniformen Entsprechungen zu den anderen Eigenschaften lauten wie folgt:

Uniforme Integrität: Jede Nachricht wird von jeder Station höchstens einmal ausgeliefert und nur dann, wenn sie gebroadcastet wurde

Uniforme Totale Ordnung: Wenn zwei Stationen zwei Nachrichten m_1 und m_2 ausliefern, dann liefern beide m_1 und m_2 in der gleichen Reihenfolge aus.

Eine uniforme Entsprechung zur Validität gibt es nicht, da weder sichergestellt werden kann, daß jede Nachricht, die von einer Station gebroadcastet wird, auch von allen korrekten Stationen ausgeliefert wird (die betreffende Station könnte unmittelbar nach dem Aufruf von *broadcast(m)* ausfallen), noch kann sichergestellt werden, daß eine Nachricht, die von einer Station gebroadcastet wurde, auch von den ausgefallenen Stationen empfangen wird.

Außer der uniformen Atomarität soll das Protokoll aber auch die uniforme Rechtzeitigkeit sicherstellen. Diese ist wie folgt definiert:

Uniforme Rechtzeitigkeit: Es gibt eine bekannte Konstante Δ_{BC}, so daß gilt: Eine Nachricht, die von einer Station zum Zeitpunkt t gebroadcastet wurde, wird von keiner Station nach dem Zeitpunkt $t + \Delta_{BC}$ ausgeliefert.

Durch die Eigenschaft der Rechtzeitigkeit ist gewährleistet, daß das Broadcastprotokoll in seinem zeitlichen Verhalten vorhersagbar ist. Rechtzeitigkeit und Validität stellen gemeinsam sicher, daß jede Nachricht, die von einer korrekten Station gebroadcastet wird, von jeder korrekten Station nach spätestens Δ_{BC} Zeiteinheiten ausgeliefert wird.

2.1.3 Eigenschaften des Teilnehmerprotokolls

Dieser Abschnitt beschreibt die Eigenschaften des Teilnehmerprotokolls. Die Definitionen der Eigenschaften entsprechen weitgehend den von Hiltunen und Schlichting in [HS95] getroffenen.

Das Teilnehmerprotokoll liefert, immer wenn es eine Gruppenänderung feststellt, eine Nachricht an den Benutzer aus, die diesen von der Gruppenänderung informiert. Daher werden im Folgenden drei Arten von Eigenschaften beschrieben: Eigenschaften die sich darauf beziehen, ob und wann das Teilnehmerprotokoll Ausfälle erkennt, Eigenschaften die sich auf das Verhältnis der an den unterschiedlichen Stationen ausgelieferten Änderungsnachrichten beziehen und schließlich Eigenschaften die sich auf das Verhältnis der vom Broadcastprotokoll und der vom Teilnehmerprotokoll ausgelieferten Nachrichten beziehen.

Zwei Eigenschaften beschreiben das Verhalten des Teilnehmerprotokolls beim Erkennen von Stationsausfällen:

Rechtzeitigkeit: Es gibt eine bekannte Konstante Δ_{Mem}, so daß gilt: Wenn eine Station s zum Zeitpunkt t ausfällt, dann liefert jede korrekte Station spätestens zum Zeipunkt $t + \Delta_{Mem}$ eine Nachricht aus, die den Ausfall von s anzeigt.

Auch hier stellt die Rechtzeitigkeit die zeitliche Vorhersagbarkeit des Gruppenkommunikationsprotokolls sicher. Wenn die zeitliche Vorhersagbarkeit nicht von Bedeutung

ist, dann reicht es aus, wenn ausgefallene Stationen nach endlicher Zeit als solche erkannt werden. Diese schwächere Eigenschaft wird als Lebendigkeit bezeichnet.

Genauigkeit: Wenn eine Station zum Zeitpunkt t eine Nachricht ausliefert, die den Ausfall einer Station s anzeigt, dann ist s vor dem Zeitpunkt t ausgefallen.

Da Stationsausfälle nicht vor dem Zeitpunkt t des Ausfalls angezeigt werden, folgt aus der Rechtzeitigkeit, daß die Zeitpunkte, an denen zwei Stationen vom Ausfall einer dritten Station erfahren, beide im Intervall $[t,\ t+\Delta_{Mem}]$ und damit maximal Δ_{Mem} Zeiteinheiten auseinander liegen.

Die nächsten beiden Eigenschaften treffen Aussagen bzgl. der von zwei korrekten Stationen ausgelieferten Änderungsnachrichten:

Einigung: Wenn eine Station eine Nachricht ausliefert, die den Ausfall einer Station s anzeigt, dann liefert jede korrekte Station nach endlicher Zeit eine Nachricht aus, die den Ausfall von s anzeigt.

Da jede Station nur dann den Ausfall von s anzeigt, wenn s tatsächlich ausgefallen ist, und da weiterhin in diesem Falle jede korrekte Station eine Nachricht ausliefern wird, die den Ausfall von s anzeigt, ist die Einigung eine Implikation der beiden Eigenschaften Genauigkeit und Rechtzeitigkeit.

Totale Ordnung: Wenn zwei Stationen s_1 und s_2 zwei Nachrichten ausliefern, die den Ausfall der Stationen s_3 und s_4 anzeigen, dann liefern beide Stationen die Nachrichten in der gleichen Reihenfolge aus.

Durch diese beiden Eigenschaften ist sichergestellt, daß zwei korrekte Stationen die gleiche Folge von Änderungsnachrichten ausliefern. Die letzte Eigenschaft bezieht sich auf die Reihenfolge der Nachrichten des Broadcastprotokolls und des Teilnehmerprotokolls untereinander.

Virtuelle Synchronität: Wenn zwei Stationen s_1 und s_2 zwei Nachrichten ausliefern, die den Ausfall der Stationen s_3 und s_4 anzeigen, dann liefern beide Stationen zwischen diesen Nachrichten die gleiche Menge von Broadcast-Nachrichten aus.

Da sowohl die Nachrichten des Broadcastprotokolls als auch die Nachrichten des Teilnehmerprotokolls in sich total geordnet sind, folgt aus der Eigenschaft der virtuellen Synchronität, daß alle korrekten Stationen die gleiche, total geordnete Folge von Nachrichten ausliefern. Da eine Nachricht, die den Ausfall einer Station s anzeigt, vom Teilnehmerprotokoll nur dann ausgeliefert wird, nachdem s auch tatsächlich ausgefallen ist, weiß jede Station, die eine solche Nachricht in ihrem total geordneten Nachrichtenstrom ausliefert, daß s alle Nachrichten, die nach dieser Nachricht ausgeliefert werden, nicht mehr ausliefern wird.

2.2 Systemannahmen

Die Grundlage für die Konzeption eines Protokolls stellt ein Modell des Systems dar, auf dem dieses Protokoll eingesetzt werden soll. Bisher wurde erläutert, daß das System als Rechnernetz aufgefaßt wird, das aus einer Menge von Stationen besteht, die an ein gemeinsam genutztes Broadcast-Medium angeschlossen sind, wobei sich die Menge der Stationen zur Laufzeit verändern kann. Im vorliegenden Unterkapitel sollen die Annahmen an dieses System detaillierter erläutert werden. Dazu wird in Paragraph 2.2.1 zunächst erläutert, welche Eigenschaften modelliert werden sollen, wie diese Eigenschaften modelliert werden können und welche Modelle für Rechnernetze bereits existieren. In Paragraph 2.2.2 werden dann die Eigenschaften des IEEE 802.11 Standard untersucht. Auf Grundlage dieser beiden Paragraphen werden schließlich in Paragraph 2.2.3 geeignete Systemannahmen hergeleitet, die den Eigenschaften des IEEE 802.11 Standard Rechnung tragen und welche die Realisierung der im vorhergehenden Unterkapitel definierten Eigenschaften erlauben.

2.2.1 Systemmodelle

Betrachtete Eigenschaften

Wenn man die Eigenschaften eines Systems modellieren möchte, dann ist es günstig, dieses System zunächst zu strukturieren. Ein System kann als eine Menge von Komponenten aufgefaßt werden, die verbunden sind und zusammenwirken ([Net91], [Lap92]). Jede dieser Komponenten ist selbst wieder ein System. Diese rekursive Definition der Systemstruktur wird bis zu atomaren Komponenten bzw. atomaren Systemen fortgesetzt, deren Struktur nicht erkennbar oder im betrachteten Kontext nicht relevant ist. In dieser Weise können die Stationen und das Netzwerk eines Rechnernetzes als Komponenten aufgefaßt werden, die zusammenwirken und so das System Rechnernetz bilden. Die Stationen und das Netzwerk sind ihrerseits wieder Systeme, deren innere Struktur im Rahmen der hier durchgeführten Modellierung allerdings nicht weiter untersucht wird.

Jedes System, und daher auch jede Komponente, zeigt an seiner Schnittstelle zur Umwelt ein bestimmtes zeitliches und funktionales Verhalten. Der Benutzer eines Systems, der selbst wieder ein System sein kann, hat bzgl. dieses Verhaltens bestimmte Erwartungen, die in Form einer, am besten formalen, Spezifikation festgelegt sind. Verhält ein System sich nicht gemäß seiner Spezifikation, so sagt man, daß dieses System ausgefallen ist. Der Grund für das Auftreten eines Ausfalles (failure[5]) ist ein Fehler im Systemzustand, ein sog. Fehlzustand (error). Ein Fehlzustand ist wiederum durch ein Ereignis, die Fehlerursache (fault), bedingt. Eine Fehlerursache in einem System kann im Ausfall

[5] Die deutschen Bezeichnungen sind der in [Lap92] vorgestellten deutschen Terminologie entnommen. Dennoch werden in Klammer auch die englischen Fachbegriffe angegeben.

einer Komponente des Systems bestehen. Fällt beispielsweise in einem Rechnernetz eine der Stationen aus, dann ist dies eine Fehlerursache, die dazu führt, daß sich das Rechnernetz in einem Fehlzustand befindet. Dieser Fehlzustand kann wiederum dazu führen, daß das Rechnernetz als ganzes sich nicht gemäß seiner Spezifikation verhält und damit ausfällt.

Aufgrund dieser Betrachtung sind bei der Modellierung des Systems die folgenden Fragen zu beantworten:

1. Ist das Verhalten der Komponenten auch in der Zeit spezifiziert? Es ist also zu klären, ob von den Komponenten nur erwartet wird, daß sie logisch korrekte Ergebnisse liefern, oder ob sie diese Ergebnisse auch zu bestimmten Zeitpunkten (bzw. in bestimmten Intervallen) erbringen müssen. Diese Frage ist aus zwei Gründen von entscheidender Bedeutung:
   - Wenn ein System mit einer Umwelt interagiert, die sich im Zeitverlauf ändert, dann ist ein Verhalten des Systems, das zum Zeitpunkt t der Umweltsituation angemessen war, zum Zeitpunkt t' vielleicht nutzlos oder sogar schädlich. Der Wert eines Verhaltens und damit die Erwartungen des Benutzers an dieses Verhalten kann also nicht unabhängig von der Zeit bestimmt werden.
   - Das Erkennen von Ausfällen ist ohne zeitliche Spezifikation u. U. nicht möglich, da ein Benutzer beim Ausbleiben einer Leistung nicht unterscheiden kann, ob diese Leistung sehr spät erfolgt oder ob sie gar nicht mehr erfolgen wird.

2. Wie häufig treten Komponentenausfälle auf? Ein System kann so entworfen werden, daß es sich auch beim Ausfall einzelner Komponenten noch entsprechend seiner Spezifikation verhält (diese Eigenschaft wird als *Fehlertoleranz* bezeichnet). Dies erreicht man durch den Einsatz von Redundanz. Redundanz bezeichnet ein mehr an Betriebsmitteln im System gegenüber dem Maß, das zum Erreichen des spezifizierten Verhaltens im fehlerfreien Falle notwendig wäre. Das Maß an Redundanz, das zum Erreichen von Fehlertoleranz notwendig ist, wird durch die Rate, mit der Ausfälle auftreten, entscheidend mitbestimmt.

3. Welche Annahmen kann man bzgl. des Verhaltens einer ausgefallenen Komponente treffen? Die Frage, ob das Verhalten einer Komponente, die sich nicht mehr entsprechend ihrer Spezifikation verhält, dennoch bestimmten Annahmen genügt, hat entscheidenden Einfluß auf die benutzten Fehlertoleranzverfahren und das notwendige Maß an Redundanz.

Die dritte Frage beschäftigt sich mit dem für eine Komponente gewählten Fehlermodell. Um solche Modelle bestimmen zu können, wird zunächst ein Modell des Verhaltens von Systemen angegeben, wie es in [Pow92] vorgestellt wird. In diesem Modell wird das Verhalten eines Systems als eine Folge von Zeit-Wert-Tupeln betrachtet, die von einem allwissenden Beobachter wahrgenommen werden. Dieser Beobachter kennt die Spezifikation des Verhaltens und kann daher feststellen, wenn ein solcher Tupel vom spezifizierten Verhalten abweicht. Ein tatsächlicher Beobachter ist dazu u. U. nicht in

der Lage. So kann es vorkommen, daß ein Benutzer einen fehlerhaften Dienst in Anspruch nimmt ohne dies festzustellen, wodurch sein eigener Systemzustand fehlerhaft werden kann (sog. Fehlerpropagierung). Das Verhalten des Systems kann auf zweierlei Weise fehlerhaft sein: Die Wertkomponente des Tupels kann von ihrer Spezifikation abweichen, ein sog. Wertfehler, und die Zeitkomponente des Tupels kann von ihrer Spezifikation abweichen, ein sog. Zeitfehler. Auch überhaupt nicht erbrachte Leistungen werden als Zeitfehler betrachtet, wobei der Wert der Zeitkomponente ∞ ist. Wenn ein System einen Dienst für mehrere (n) Benutzer erbringt, dann werden die Tupel n-fach repliziert. In diesem Falle ist auch die Konsistenz der Tupel von Bedeutung, d. h. die Frage, ob alle replizierten Tupel den gleichen Wert und „in etwa“ die gleiche Zeit enthalten.

Mit Hilfe dieser Beschreibung des Systemverhaltens können Fehlermodelle in Form von Annahmen an die Folge von Zeit-Wert-Tupeln eines ausgefallenen Systems beschrieben werden (ebenfalls [Pow92]). Die folgenden Fehlermodelle sind die am häufigsten in der Literatur verwendeten:

Stabilität: Alle Tupel entsprechen ihrer Spezifikation.

Totalausfall (crash failure): Alle Tupel mit einem Zeitwert kleiner t, dem Ausfallzeitpunkt des Systems, entsprechen der Spezifikation, alle anderen enthalten den Zeitwert ∞. Das bedeutet, daß das System bis zum Zeitpunkt t seines Ausfalls korrekt funktioniert und danach überhaupt keinen Dienst mehr erbringt.

Auslassungsausfall (omission failure): Jeder Tupel, der von der Spezifikation abweicht, enthält den Wert unendlich in der Zeitkomponente, d. h. das System fällt aus, indem es zwischenzeitlich keinen Dienst erbringt.

Zeitausfall (timing failure): Tupel weichen nur in der Zeitkomponente von ihrer Spezifikation ab. Es kann in verfrühte Zeitfehler, die Zeit weicht nach unten ab, und verspätete Zeitfehler, die Zeit weicht nach oben ab, unterschieden werden.

Byzantinischer Ausfall (byzantine bzw. arbitrary failure): Die Tupel können in beliebiger Weise von ihrer Spezifikation abweichen. Man muß sogar davon ausgehen, daß das System böswillig die Fehler so produziert, daß es den maximalen Schaden erzeugt.

Eine weitere Möglichkeit beim Definieren von Fehlermodellen besteht darin, anzunehmen, daß Wertfehler als solche erkennbar sind, weil fehlerhafte Werte außerhalb eines bestimmten Codebereiches liegen. Dies ist z. B. möglich, wenn fehlererkennende Codes zum Einsatz kommen. Wenn Systeme ihren Dienst mehreren Benutzern erbringen, dann können konsistente und inkonsistente Fehler unterschieden werden. Bei konsistenten Fehlern ist bei einem Ausfall immer noch sichergestellt, daß die Werte aller n Replikate eines Tupels gleich und die Zeitwerte „in etwa“ gleich sind. Dies bedeutet z. B. für eine Nachricht, die auf einem Broadcast-Medium versendet wird, daß sie entweder von allen Stationen oder von keiner empfangen wird. Byzantinische Ausfälle schließen auch die Möglichkeit der Inkonsistenz mit ein.

Die Fehlermodelle nehmen vom Totalausfall bis zum Byzantinischen Ausfall an Schweregrad (severity) zu, d. h. die Annahmen an das Verhalten des Systems bei einem Ausfall werden immer schwächer. Dies führt zum einen dazu, daß die Wahrscheinlichkeit, daß ein System dem Fehlermodell tatsächlich genügt, die sog. Überdeckung ([Pow92]), wächst, zum anderen wächst aber i. A. auch die zum Realisieren der Fehlertoleranz notwendige Redundanz.

Die Überdeckung ist bei der Wahl eines geeigneten Fehlermodells von großer Bedeutung. Die Überdeckung begrenzt die Wahrscheinlichkeit, daß ein System nicht ausfällt, nach oben, da davon ausgegangen werden muß, daß eine Abweichung einer Komponente von ihrem Fehlermodell immer auch den Ausfall des Systems nach sich zieht. Um eine hohe Überdeckung zu erreichen, kann man entweder nur sehr schwache Annahmen treffen oder die Gültigkeit der Annahmen durch den Einsatz von Redundanz in den Komponenten verhindern (z. B. selbstprüfende Hardware oder fehlererkennende Codes). Die Annahmen sollten aber nur so schwach getroffen werden, wie dies notwendig ist, weil bei einem größeren Schweregrad die Anzahl der redundanten Komponenten i. A. erhöht werden muß, was der Zuverlässigkeit des Gesamtsystems durchaus abträglich sein kann ([Pow92]). Auf einem Medium mit hinreichend großer Prüfsumme ist es z. B. plausibel anzunehmen, daß alle Wertfehler erkannt werden können.

Bei Systemen, die zeitlich nicht spezifiziert sind, ist jedes Verhalten, bei dem die Tupel richtige Werte und endliche Zeitwerte haben, nach Definition korrekt. Dies macht es unmöglich, Auslassungsausfälle und selbst Totalausfälle zu erkennen, da immer die Möglichkeit besteht, daß es sich lediglich um eine spät, aber nicht zu spät, erbrachte Leistung handelt. Bei Systemen mit zeitlicher Spezifikation können diese Fehler aber sehr wohl erkannt werden, wenn die Komponenten in der Lage sind, die Zeit zu messen. Eine wichtige Frage bei der Modellierung verteilter Systeme besteht daher in der Verfügbarkeit lokaler Uhren in den Stationen. Diese Uhren können für die Fehlererkennung eingesetzt werden, sofern sie nicht beliebig von der Referenzzeit der Spezifikation (i. A. der realen Zeit) abweichen können.

Existierende Systemmodelle[6]

Die stärksten Annahmen liegen synchronen Systemen zugrunde. In synchronen Systemen sind alle Komponenten zeitlich spezifiziert und es wird angenommen, daß das Netzwerk stabil ist. Daher kann die Kommunikation unter den Stationen ohne Verluste und mit einer bekannten maximalen Verzögerung erfolgen. Weiterhin wird angenommen, daß in allen Stationen Uhren vorhanden sind, deren Raten nur begrenzt von der Rate der realen Zeit abweichen. Aus diesen Gründen können Stationsausfälle, bei hinreichender Redundanz auch byzantinische, rechtzeitig und genau erkannt werden.

[6] [GP96] ist ein sehr guter Übersichtsartikel zu diesem Thema

In synchronen Systemen können Broadcastprotokolle, die rechtzeitige und atomare Broadcasts implementieren, realisiert werden. In synchronen Systemen lassen sich weiterhin die lokalen Uhren synchronisieren, so daß für alle Stationen ein Begriff der globalen Zeit entsteht. Diese globale Zeit kann genutzt werden, um ein hohes Maß an Gleichzeitigkeit zwischen den Stationen zu erreichen.

Andererseits ist die Gewährleistung der starken Annahmen synchroner Systeme mit erheblichen Kosten verbunden:

- Massive Redundanz, z. B. in Form mehrfach vorhandener Übertragungsmedien, muß die Stabilität des Netzwerkes gewährleisten.

- Es müssen immer genügend Betriebsmittel zur Verfügung stehen, um die garantierten Zeiten einhalten zu können.

- Die Betriebsmittel müssen durch einen Echtzeitscheduler verwaltet werden, der die Belegung der Betriebsmittel so plant, daß alle Aufgaben innerhalb der vorgegebenen Zeit durchgeführt werden können. Dabei muß der Scheduler bei der Planung immer den ungünstigsten Fall zu Grunde legen, so daß nur relativ wenige Aufgaben mit den vorhandenen Betriebsmitteln zur Ausführung gelangen können.

Wegen der hohen Kosten ihrer Realisierung werden synchrone Systeme i. A. eingesetzt, um harte Echtzeitsysteme zu realisieren. Dabei handelt es sich um Systeme, die Aufgaben durchführen, die garantiert vollständig innerhalb der vorgegebenen Frist ausgeführt werden müssen, weil anderenfalls katastrophale Folgen für teures technisches Gerät oder sogar die Gesundheit von Menschen drohen. Im Gegensatz dazu spricht man von weichen Echtzeitsystemen, wenn das nicht fristgerechte Durchführen einer Aufgabe nur zu einer Beeinträchtigung des Dienstes führt, wie z. B. bei einem Video-Server.

Sehr schwache Annahmen liegen hingegen zeitlosen asynchronen Systemen zugrunde. Der wesentliche Unterschied zu synchronen Systemen besteht darin, daß zeitlose asynchrone Systeme zeitlich nicht spezifiziert sind. Da häufig zusätzlich noch davon ausgegangen wird, daß die Kommunikation unzuverlässig ist, d. h., daß das Netzwerk Auslassungsfehler erleiden kann, lassen sich asynchrone Systeme mit einem erheblich geringeren Aufwand realisieren. Das wesentliche Problem in zeitlosen asynchronen Systemen besteht darin, daß aufgrund der fehlenden Zeitspezifikation ein ausgefallener Prozeß von einem sehr langsamen Prozeß nicht unterschieden werden kann. Daher kann es in zeitlosen asynchronen Systemen kein deterministisches Protokoll für das Konsensusproblem[7] ([FLP85]) und ebenso kein determinsitisches Protokoll für atomare Broadcasts geben ([DDS87]). Weiterhin kann aufbauend auf den Annahmen zeitloser asynchroner Systeme niemals die Rechtzeitigkeit der Nachrichtenübertragung sichergestellt werden.

[7] Eine Definition des Konsensusproblems findet sich in [HT93].

Da Konsensus und atomare Broadcasts in verteilten Systemen sehr wichtig sind, die Realisierung synchroner Systeme aber hohe Kosten verursacht, wurden Systemmodelle vorgestellt, die über ein geringeres Maß an Synchronität verfügen, in denen es aber dennoch deterministische Protokolle für das Konsensusproblem und atomare Broadcasts geben kann, sog. partiell synchrone Systeme.

Durch die Verwendung unzuverlässiger Fehlerdetektoren kann in Systemen ein hinreichendes Maß an Synchronität bereitgestellt werden, um atomare Broadcasts zu realisiern. Bei der Verwendung unzuverlässiger Fehlerdetektoren steht in jeder Station ein Fehlererkennungsmodul zur Verfügung, das auf Anfrage eine Menge von Stationen liefert, von denen es vermutet, daß sie ausgefallen seien. Diese Mengen können in den verschiedenen Sitationen unterschiedlich sein und auch korrekte Stationen enthalten. Dennoch können in Systemen mit unzuverlässigen Fehlerdetektoren atomare Broadcasts realisiert werden, selbst dann, wenn der Fehlerdetektor eine unendliche Anzahl von Fehlern macht ([CT96]). Da die Synchronität aber nur implizit in den Eigenschaften der Fehlerdetektoren steckt, kann auch in asynchronen Systemen mit Fehlerdetektoren ohne weitere Annahmen die Rechtzeitigkeit der Nachrichtenübertragung nicht gewährleistet werden.

In zeitgesteuerten asynchronen Systemen ([CS95], [Cri96]) ist das zeitliche Verhalten aller Komponenten spezifiziert. Da aber Total-, Auslassungs- und Zeitausfälle auftreten können, ist das zeitliche Verhalten des Systems nach wie vor unbestimmt. Der große Vorteil dieser Systeme besteht darin, daß die Annahmen an das tatsächliche Verhalten des Systems nicht stärker sind als die in zeitlosen asynchronen Systemen, daß aber durch die zeitliche Spezifikation ausgefallene Stationen von korrekten unterschieden werden können. Daher können in zeitgesteuerten asynchronen Systemen auch atomare Broadcasts realisiert werden. Sogar Zeiteigenschaften lassen sich herleiten, allerdings ist die Rechtzeitigkeit immer durch die Stabilität des Systems bedingt, d. h. von der Annahme abhängig, daß im System hinreichend lange keine Ausfälle auftreten. Da über Häufigkeit und Dauer dieser Stabilitätsphasen keine expliziten Annahmen getroffen werden, können auch keine Garantien für die Rechtzeitigkeit der Nachrichtenübertragung vergeben werden. Zeitgesteuerte asynchrone Systeme sind für die Realisierung von weichen Echtzeitaufgaben geeignet.

Schließlich betrachten auch Veríssimo et al. ([VM90], [VRR91]) Systeme, die zeitlich spezifiziert sind. Im Gegensatz zu [CS95] und [Cri96] lassen diese aber nur Stationsausfälle und Auslassungsausfälle des Netzwerkes zu. Weiterhin nehmen Veríssimo et al. an, daß die Rate, mit der Ausfälle auf dem Netzwerk auftreten, begrenzt ist, so daß in einem bestimmten Zeitraum nur eine begrenzte Anzahl von Nachrichtenverlusten möglich ist. Dieses Modell ist soweit den synchronen Systemen angenähert, daß neben der Atomarität der Broadcast-Nachrichten auch die Rechtzeitigkeit sichergestellt werden kann. So kann aufbauend auf diesem Systemmodell ein synchrones System realisiert werden. Der Vorteil des gewählten Modells gegenüber synchronen Systemen darin besteht, daß die Fehlerbehebung in den entwickelten Protokollen vorgenommen wird anstatt auf unteren Ebenen des Systems.

In diesem Kontext wurde noch ein weiteres Modell vorgestellt [AV95], bei dem für den Großteil der Nachrichten auch Zeitfehler auftreten können. Allerdings beinhaltet das Modell einen synchronen Kanal begrenzter Bandbreite, über den die Stationen zuverlässig und rechtzeitig Nachrichten austauschen können und der benutzt werden kann, um Zeitfehler zu erkennen. Mit Hilfe dieses Kanals können Protokolle realisiert werden, die auch beim Auftreten von Zeitfehlern ein rechtzeitiges und konsistentes Verhalten der Stationen sicherstellen. Die zuverlässige und rechtzeitige Bearbeitung aller Aufgaben kann in solchen Systemen aber nicht mehr gewährleistet werden.

2.2.2 Der IEEE 802.11 Standard

2.2.2.1 Besonderheiten von Funknetzwerken

Die Wahl eines Funkmediums als Grundlage für die Realisierung eines Kommunikationsprotokolls beeinflußt den Entwurf dieses Protokolls, da zwischen einem Funkmedium und einem kabelgebundenen Medium erhebliche Unterschiede bestehen:

- Funkmedien sind nicht abgeschirmt und daher in deutlich höherem Maße externen Störungen ausgesetzt als kabelgebundene Medien. Daher ist die Wahrscheinlichkeit, daß bei der Übertragung auf dem Medium Nachrichten verloren gehen, deutlich höher als bei kabelgebundenen Netzwerken. Weiterhin treten die Verluste relativ häufig in Bursts auf, d. h., daß durch eine externe Störung häufig nicht nur eine einzelne Nachricht sondern mehrere aufeinanderfolgende Nachrichten verloren gehen.

- Die Qualität der Verbindung zwischen zwei Stationen eines Funk-Rechnernetzes wird stark von ihrer räumlichen Position und der Gestaltung des Raumes zwischen den Stationen beeinflußt. Daher kann es beim Versenden von Broadcast-Nachrichten zu asymmetrischen Nachrichtenverlusten kommen, bei denen einige Stationen eine Nachricht erhalten und andere nicht.

- Eine Station kann mit einer anderen Station nur dann über ein Funkmedium kommunizieren, wenn sich diese in einer bestimmten räumlichen Umgebung von ihr selbst, ihrem Sende-/Empfangsbereich, befindet. Wie weit sich diese Umgebung erstreckt, ist nicht genau vorherbestimmbar, vor allem, da die Größe des Sende-/Empfangsbereiches von der Gestaltung der Umwelt abhängig ist und sich daher zeitlich ändern kann. Wenn sich nicht alle Stationen in ihren wechselseitigen Sende-/Empfangsbereichen befinden, dann ist die Broadcasteigenschaft des Mediums nicht mehr gegeben und es müssen u. U. Routing-Mechanismen eingesetzt werden.

- Durch die Mobilität der Stationen und die begrenzte Größe ihres Sende-/Empfangsbereiches kann sich die Topologie des Funk-Rechnernetzes dynamisch verändern. Die Topologie kann man sich als einen Graphen veranschaulichen, in dem die Station die Knoten sind und in dem eine Kante zwischen zwei Knoten angibt, daß sich die betreffenden Stationen gegenseitig erreichen können, d. h., daß je-

de der beiden Stationen sich im Sende-/Empfangsbereich der jeweils anderen befindet[8]. Durch die Mobilität und begrenzte Größe des Sende-/Empfangsbereiches können sich zur Laufzeit beliebige solcher Graphen bilden. Routing-Algorithmen, die in solch dynamischen Topologien die Transitivität der Erreichbarkeit gewährleisten, sind sehr aufwendig.

2.2.2.2 Allgemeines zum IEEE 802.11 Standard

Nachdem die Besonderheiten von Funkmedien erläutert wurden, wird nun auf den IEEE 802.11 Standard ([IEEE97]) im Speziellen eingegangen. Der IEEE Standard 802.11 (im Folgenden auch kürzer als IEEE Standard oder Standard bezeichnet) spezifiziert das physische Medium (PHY) sowie die Medienzugriffsverfahren (Medium Access Control, MAC) für lokale drahtlose Netzwerke. Die spezifizierten Medien und Medienzugriffsverfahren erlauben die Kommunikation zwischen beliebigen ortsfesten, portablen und mobilen Geräten, die mit einer standardkonformen Kommunikationskomponente ausgestattet sind. Solche Geräte werden als Stationen bezeichnet.

Der Standard spezifiziert drei verschiedene Typen physischer Medien: Ein Infrarotmedium und zwei Arten von Funkmedien. Bei allen drei Medien ist die Zeit, die ein Bit benötigt, um vom Sender zum Empfänger zu gelangen, die sog. Bitzeit, begrenzt. Sie beträgt bei den Funkmedien ca. 1µs. Die Funkmedien bieten eine Rohbandbreite, d. i. die Bandbreite, die einer Station zu Verfügung stünde, wenn sie ununterbrochen auf dem Medium Daten übertragen könnte, von 1- oder 2-MBit/s.

Im IEEE Standard werden Mengen von Stationen, die untereinander den Medienzugriff mit den spezifizierten MAC-Protokollen abstimmen, die also ein Medium gemeinsam benutzen, zu *Basic Service Sets (BSS)* zusammengefaßt. Von diesen BSS gibt es im Standard zwei Arten: Das *Independent Basic Service Set (IBSS)* oder auch *Ad-Hoc-Netzwerk* besteht aus einer Menge von Stationen, die sich gegenseitig erreichen können. Das IBSS ist eine in sich abgeschlossene Einheit, ohne die Möglichkeit mit Stationen außerhalb des IBSS zu kommunizieren.

Im Gegensatz zum IBSS ist im *Infrastruktur-Netzwerk* den Stationen eines BSS die Möglichkeit zur Kommunikation mit Stationen, die außerhalb des BSS liegen, gegeben. Hierzu gibt es in jedem BSS eine spezielle Station, den Access Point (AP), der den anderen Stationen den Zugang zu einem Verteilungssystem ermöglicht. Das Verteilungssystem verbindet den AP mit den APs anderer BSS oder mit Stationen in beliebigen anderen Netzwerken. Das Verteilungssystem selbst kann unter Verwendung eines beliebigen Netzwerkes realisiert werden.

---

[8] U. U. muß man sogar davon ausgehen, daß diese Kanten gerichtet sind, was das Problem noch komplizierter macht.

Wenn eine Station über das Verteilungssystem eine Nachricht an eine Zielstation außerhalb ihres BSS sendet, dann muß bekannt sein, welchem AP die Zielstation zugeordnet ist. Nur so kann die Nachricht durch das Verteilungssystem an den richtigen AP und von dort an die Zielstation weitergeleitet werden. Ein Teil der von den APs zur Verfügung gestellten Verteilungsdienste besteht daher aus Protokollen zum Assoziieren, Reassoziieren und Disassoziieren von Stationen mit einem AP, welche die eindeutige Zuordnung von Stationen zu einem AP ermöglichen.

Im IEEE Standard sind zwei MAC-Protokolle spezifiziert, die als „coordination functions“ bezeichnet werden (s. Abb. 2.2). Das Basisverfahren, das für jede standardkonforme Kommunikationskomponente obligatorisch ist, ist die distributed coordination function (DCF), ein verteiltes Zugriffsverfahren nach der CSMA- (carrier sense multiple access) Methode. Bei Anwendung der CSMA-Methode für den Medienzugriff prüft eine Station, die eine Nachricht senden möchte, zunächst, ob das Medium belegt ist, d. h. ob eine andere Station bereits eine Nachricht am übertragen ist. Erst wenn sie feststellt, daß dies nicht der Fall ist, beginnt sie mit dem Übertragen der Nachricht.

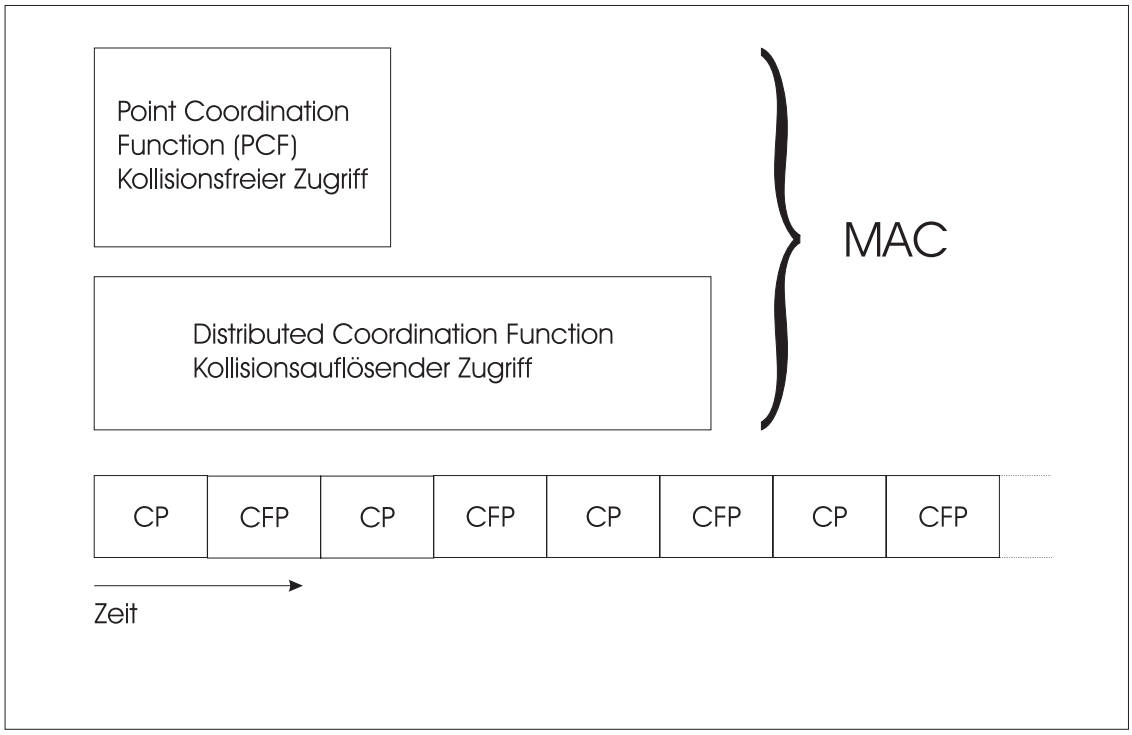


Abbildung 2.2 Architektur der MAC-Schicht im IEEE 802.11 Standard (vgl. [IEEE97] Abb. 47) und zeitlicher Ablauf.

Da mehrere Stationen gleichzeitig zu der Ansicht gelangen können, daß das Medium ungenutzt sei, wird durch die CSMA-Methode das Auftreten von Kollisionen nicht ausgeschlossen. Daher werden die Zeitabschnitte, in denen der Zugriff durch die DCF geregelt wird, als *contention period (CP)* bezeichnet. Aufbauend auf der DCF ist ein zentralisiertes Zugriffsverfahren (die *point coordination function*, *PCF*) spezifiziert, das kolli-

sionsfreien Zugriff sicherstellt, indem eine spezielle Station (der *point coordinator*) durch Polling, d. h. durch das Versenden spezieller Nachrichten (Polling-Nachrichten) exklusiven Zugriff auf das Medium gewährt. Da unter der Kontrolle des point coordinators keine Kollisionen auftreten können, werden Zeitabschnitte, in denen der Zugriff durch die PCF geregelt ist, als *contention free period (CFP)* bezeichnet. Kommt die PCF zum Einsatz, dann werden im zeitlichen Wechsel nach CFP und CP auf das Medium zugegriffen (s. Abb. 2.2).

2.2.2.3 Die Distributed Coordination Function

Die DCF stellt ein verteiltes Zugriffsverfahren, basierend auf der CSMA-Methode, dar. Eine Station, die eine Nachricht senden möchte, muß zunächst feststellen, ob das Medium ungenutzt ist oder gerade eine Übertragung stattfindet. Nur wenn keine Übertragung im Gange ist, darf die Station auf das Medium zugreifen. Dabei muß sie sicherstellen, daß nach dem Ende der vorhergehenden Übertragung eine bestimmte Mindestzeit vergangen ist. Aus übertragungstechnischen Gründen ist das Erkennen von Kollisionen während der Übertragung (Collsion Detection) bei Funknetzen nicht möglich. Daher werden im Standard Verfahren eingesetzt, durch die die Wahrscheinlichkeit für das Auftreten von Kollisionen nach Möglichkeit klein gehalten werden soll (Collision Avoidance). MAC-Protokolle, die das Auftreten von Kollisionen nicht gänzlich ausschließen, aber die Wahrscheinlichkeit für das Auftreten aufeinanderfolgender Kollisionen reduzieren, werden als kollisionsauflösende Protokolle bezeichnet (vgl. [KSY84]).

Neben der Möglichkeit, auf physischem Wege festzustellen, ob eine Übertragung stattfindet, ist auch eine virtuelle Möglichkeit, der *virtual carrier*, vorgesehen. Zur Implementierung des virtual carrier wird im Header jeder Nachricht angegeben, wie lange die Übertragung dieser Nachricht dauert, und damit, wie lange das Medium belegt ist. Diese Information speichern die Stationen in ihrem sog. *network allocation vector (NAV)*. Solange der NAV anzeigt, daß das Medium belegt ist, darf eine Station nicht auf das Medium zugreifen.

Auf einem drahtlosen Medium ist die vollständige Erreichbarkeit (jede Station kann mit jeder anderen direkt kommunizieren) nicht immer gegeben. Es kann z. B. der Fall auftreten, daß Station s_1 Station s_2 erreichen kann und Station s_2 Station s_3 erreichen kann, aber die Stationen s_1 und s_3 können sich nicht erreichen (s. Abb. 2.3). Diese Situation, die als "Hidden-Station"-Problem bezeichnet wird, macht es besonders schwierig, festzustellen, ob das Medium belegt ist. Dieses liegt darin begründet, daß Station s_1 nicht erkennen kann, ob bereits eine Übertragung von Station s_3 nach Station s_2 im Gange ist. Will Station s_1 eine Nachricht an s_2 übertragen, wird sie daher mit der Übertragung beginnen und es kommt zu einer Kollision bei Station s_2, die keine der beiden Nachrichten empfangen kann.

Um dieses Problem zu umgehen, ist im IEEE Standard der RTS/CTS-Mechanismus vorgesehen: Bevor eine Station mit der Übertragung einer Nachricht, die Nutzdaten enthält (Dataframe, Datennachricht[9]) beginnt, sendet sie eine RTS- (request to send) Nachricht an die Zielstation, die die Länge der Datennachricht beinhaltet. Ist die Zielstation zum Empfang bereit, dann antwortet sie mit einer CTS- (clear to send) Nachricht, die ebenfalls die in der RTS-Nachricht empfangene Länge enthält. Jede Station die im Sende-/Empfangsbereich der Urheber- oder der Zielstation der Datennachricht liegt, kann aufgrund der Längeninformation in den RTS-/CTS-Nachrichten ihren NAV setzen, so daß das Medium für die folgende Übertragung reserviert ist. Da allerdings auch die RTS- und CTS-Nachrichten kollidieren können, werden Kollisionen durch diesen Mechanismus nicht vollständig ausgeschlossen; nur die Wahrscheinlichkeit einer Kollision wird durch die Kürze der RTS-/CTS-Nachrichten reduziert.

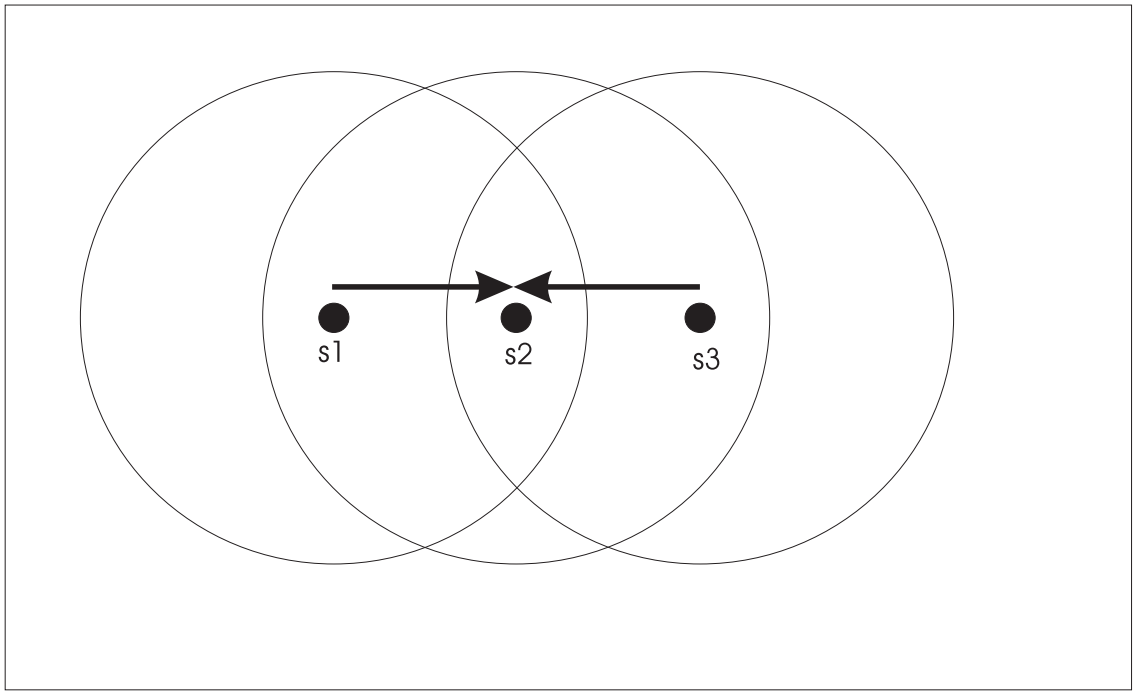


Abbildung 2.3 Das Hidden Station Problem: Nachrichten von s_1 und s_3 kollidieren an Station s_2.

Um die Wahrscheinlichkeit von Kollisionen zu reduzieren, erfolgt der Zugriff auf das Medium nicht-persistent, d. h., daß eine Station, die das Medium als ungenutzt erkannt hat, zunächst eine zufällig zwischen einer Ober- und Untergrenze gewählte Zeit abwarten und währenddessen das Medium beobachten muß. Erst wenn innerhalb dieser Zeit-

[9] Nachrichten, die innerhalb der OSI-Verbindungsschicht ausgetauscht werden, werden häufig als Frames bezeichnet. Überall wo in der Beschreibung das Wort Nachricht steht könnte daher auch Rahmen oder Frame stehen.

spanne keine Nachrichtenübertragung begonnen hat, darf die Station selbst mit der Übertragung beginnen.

Da Kollisionen weder vollständig ausgeschlossen noch während der Übertragung erkannt werden können, werden Bestätigungen eingesetzt, um ausgefallene Nachrichten zu erkennen und ggf. neu zu übertragen. Eine Station, die eine Nachricht korrekt empfängt, muß an den Sender der Nachricht eine Bestätigung senden. Wenn der Sender einer Nachricht nicht innerhalb einer vorgegebenen Zeitspanne nach dem Übertragen der Nachricht die Bestätigung erhält oder vor dem Ende dieser Zeitspanne eine beliebige andere Nachricht empfängt, dann nimmt er an, daß die Nachricht verloren gegangen sei, und überträgt sie erneut. Dieser Vorgang wird wiederholt, bis der Sender entweder eine Bestätigung erhalten hat oder ein vorgegebenes Limit an Sendeversuchen erreicht ist.

Auch wenn eine Station aufgrund von ausbleibenden Bestätigungen den Verlust einer Nachricht vermutet und daher mit der erneuten Übertragung dieser Nachricht beginnt, muß sie zunächst eine zufällig bestimmte Zeitspanne warten. Um nach dem Auftreten einer Kollision, und damit einer verlorenen Nachricht, das Auftreten weiterer Kollisionen zu verhindern, wird das Intervall, aus dem die Station den Wert für ihre Wartezeit bestimmt, vor jeder erneuten Übertragung einer Nachricht vergrößert. So wird die Wahrscheinlichkeit, daß wiederum zwei Stationen zur gleichen Zeit mit der Nachrichtenübertragung beginnen, immer geringer.

Der RTS/CTS- und der Bestätigungsmechanismus, die die Übertragung von Nachrichten angesichts der ungünstigen Eigenschaften des Mediums zuverlässiger machen sollen, kommen nur für Punkt-zu-Punt-Nachrichten, nicht aber für Broadcasts zum Einsatz. Daher ist die Wahrscheinlichkeit, daß Broadcast-Nachrichten aufgrund von Kollisionen oder Störungen verloren gehen noch relativ hoch.

2.2.2.4 Die Point Coordination Function

Ein Funknetzwerk nach dem IEEE 802.11 Standard kann als weiteres Zugriffsverfahren neben der DCF auch die PCF zur Verfügung stellen. Die PCF ist ein zentralisiertes Verfahren, bei dem der Point Coordinator durch das Versenden von speziellen Nachrichten, den Polling-Nachrichten, exklusiven Zugriff auf das Medium gewährt. Da auf diese Weise das Auftreten von Kollisionen verhindert werden kann, handelt es sich bei der PCF um ein sog. kollisionsfreies MAC-Protokoll (vgl. [KSY84]). Die PCF kann nur innerhalb eines Infrastrukturnetzwerkes zum Einsatz kommen. Die Funktion des eindeutigen Point Coordinators, der den Medienzugriff innerhalb eines BSS koordiniert, wird vom AP dieses BSS wahrgenommen. Daher wird im Folgenden das Wort AP auch zur Bezeichnung des Point Coordinators verwendet.

Während der CFP darf eine Station immer nur dann auf das Medium zugreifen, wenn sie eine Polling-Nachricht vom AP erhalten hat. Die Station darf nach Erhalt der Polling-Nachricht eine Datennachricht, als Punkt-zu-Punkt- oder Broadcast-Nachricht, versenden. Der AP selbst kann außer Stationen zu pollen auch Datennachrichten an be-

liebige Stationen versenden. Der AP pollt die Stationen gemäß einer Polling-Liste, die angibt, welche Stationen der AP in welcher Reihenfolge pollt. Die Stationen können die Aufnahme in die Polling-Liste beantragen, wenn sie sich am AP assoziieren oder reassoziieren. Die Ausgestaltung der Polling-Liste ist im IEEE Standard bewußt nicht spezifiziert. Sie kann also weitgehend frei implementiert werden.

Auch in der CFP werden Bestätigungen für Punkt-zu-Punkt-Nachrichten eingesetzt. Dies gilt sowohl für das Übertratgen von Nachrichten zwischen Stationen als auch für das Übertragen von Nachrichten zwischen einer Station und dem AP. Sendet der AP eine Polling-Nachricht an eine Station, dann muß diese Station mit dem Senden einer Nachricht antworten. Hat die Station keine Nachricht zu senden, dann sendet sie eine NULL-Nachricht an den AP. Empfängt eine Station eine Datennachricht, dann muß sie mit einer Bestätigung antworten. Empfängt der AP keine Bestätigung, dann steht es ihm frei, die Nachricht unmittelbar oder erst dann erneut zu übertragen, wenn er die nächste Polling-Nachricht an die Zielstation sendet. Eine Station, die eine Nachricht an den AP oder eine andere Station sendet, erwartet ebenfalls eine Bestätigung. Bleibt die Bestätigung aus, dann darf sie die Nachricht aber erst dann erneut übertragen, wenn sie wieder vom AP gepollt wird, oder in der CP. Auch in der CFP werden Broadcast-Nachrichten nicht bestätigt. Der RTS/CTS-Mechanismus wird in der CFP nicht eingesetzt.

Um den Nachrichtenaufwand beim Einsatz von Bestätigungen zu reduzieren, können in der CFP verschiedene Nachrichtenarten kombiniert werden, d. h. Bestätigungen und Polling-Nachrichten können via Piggy-Bagging auf anderen Nachrichten versendet werden. Der AP kann mit einer Nachricht einer Station Benutzerdaten zusenden, sie pollen und eine vorher empfangene Nachricht bestätigen. Dies geht selbst dann, wenn die Nachricht, die der AP bestätigen möchte, nicht von der Station stammt, die er unmittelbar danach pollt. Ebenso können Stationen, die eine Punkt-zu-Punkt-Nachricht an den AP senden, mit dieser Nachricht eine unmittelbar vorher vom AP empfangene Punkt-zu-Punkt-Nachricht bestätigen.

2.2.2.5 Der Wechsel von CFP und CP

Kommen sowohl die DCF als auch die PCF zum Einsatz, dann wechseln sich Phasen von kollisionsauflösender und kollisionsfreier Kommunikation, CPs und CFPs, auf dem Medium ab, wobei der AP über die Zeitpunkte der Phasenwechsel entscheidet.

Die CFP beginnt mit der Übertragung einer speziellen Nachricht, des sog. Beacons, durch den AP. Beacons werden vom AP in regelmäßigen Zeitabständen (den Beaconintervallen) gesendet, um Informationen an assoziierte und noch nicht assoziierte Stationen in seinem Sende-/Empfangsbereich zu übermitteln. Der Zeitabstand zwischen dem Start einer CFP und dem Start der nächsten CFP ist als ein vielfaches des Beaconintervalls festgelegt, so daß jede CFP mit der Übertragung eines speziellen Beacons beginnt. Der eingeplante Sendezeitpunkt für dieses Beacon (target beacon transmission time, TBTT) ist allen Stationen durch die Informationen in vorhergehenden Beacons bekannt. Jede Station setzt zur TBTT ihren NAV auf die Maximaldauer der CFP, die ebenfalls

aufgrund der vorhergehenden Beacons bekannt ist. So ist sichergestellt, daß nach der TBTT bis zum Ende der CFP keine Station, ohne gepollt zu werden, auf das Medium zugreift. Dennoch kann sich der Start der CFP verzögern, wenn die Übertragung des Beacons durch die Übertragung einer Nachricht, die vor der TBTT begonnen hat und noch andauert, aufgehalten wird.

Die Dauer der CFP ist variabel und befindet sich unter der Kontrolle des AP. Der AP kann die CFP zu beliebigen Zeitpunkten beenden, solange die Dauer unter einer vorgegebenen Obergrenze bleibt. Diese Obergrenze muß so gewählt sein, daß zwischen dem Ende einer CFP und dem Anfang der nächsten mindestens eine Nachricht in der CP übertragen werden kann. Der AP beendet die CFP, indem er eine spezielle Nachricht an die Stationen sendet. Weiterhin ist die verbleibende Dauer der CFP in jedem Beacon, das der AP während der CFP sendet (das Beacon mit der die CFP beginnt eingerechnet), enthalten.

2.2.2.6 Assoziation

Es wurde bereits erwähnt, daß jede Station, die am Verteilungssystem teilnehmen will, an einem AP assoziiert sein muß, damit bekannt ist, wo sie erreichbar ist. Ebenso muß auch eine Station, die an der CFP teilnehmen will, an einem AP assoziiert sein, damit der AP sie pollen kann.

Die Assoziation findet immer auf Initiative einer Station hin statt. Diese sendet eine Anfrage an den AP, bei dem sie sich anmelden möchte. Wird die Anfrage dort korrekt empfangen und kann die Station sich authentisieren, dann sendet der AP ihr einen positiven Bescheid zu ihrer Anfrage, ansonsten sendet der AP eine Ablehnung. Sobald der AP von der Station eine Bestätigung für seinen positiven Bescheid erhalten hat, nimmt er sie in die Menge der assoziierten Stationen auf.

Reassoziation erlaubt es einer Station, den AP, an den sie assoziiert ist, zu wechseln, wenn sie sich von einem BSS in ein anderes bewegt. Außerdem können durch die Reassoziation die Merkmale einer bestehenden Assoziation verändert werden (z. B., ob eine Station gepollt werden will oder nicht). Die Reassoziation entspricht in ihrem Ablauf der Assoziation.

Die Disassoziation sollte von Stationen genutzt werden, die das Netzwerk verlassen, um eine bestehende Verbindung mit einem AP aufzulösen. Ebenso kann aber auch der AP die Disassoziation initiieren, wenn er z. B. als AP nicht länger zur Verfügung steht.

Da jede Assoziation von der Station initiiert wird, die sich assoziieren möchte, müssen Stationen zunächst in der Lage sein, einen AP zu finden, an den sie sich assoziieren können. Dazu werden im Standard zwei Verfahren vorgestellt: aktives und passives Scannen. Beim aktiven Scannen sendet eine Station sog. Probe-Frames aus. Stationen, die eine solche Nachricht empfangen und zu dem als Ziel angegebenen BSS gehören,

senden eine Antwort an die Station. Beim passiven Scannen wartet die Station darauf, Beacon-Frames von einem AP zu empfangen.

Die Assoziationskonzepte beinhalten zwar einen Begriff von einer Gruppe (alle Stationen, die mit dem AP assoziiert sind), stellen aber nicht die vom Teilnehmerprotokoll geforderten Eigenschaften (s. Paragraph 2.1.3) sicher. Stationen werden nur ausgeschlossen, wenn sie wissen, daß sie die Gruppe verlassen, und sich rechtzeitig disassozieren. Das Erkennen von Ausfällen von Stationen oder von Stationen, die die Gruppe verlassen haben, ist nicht möglich. Ebensowenig ist für eine konsistente Sicht der Gruppe unter den Stationen gesorgt. Die einzige Station, die Kenntnis von der aktuellen Gruppe hat, ist der AP.

2.2.3 Die Systemannahmen

In diesem Paragraphen soll das beim Entwurf des Gruppenkommunikationsprotokolls zugrunde gelegte System eingehend beschrieben werden.

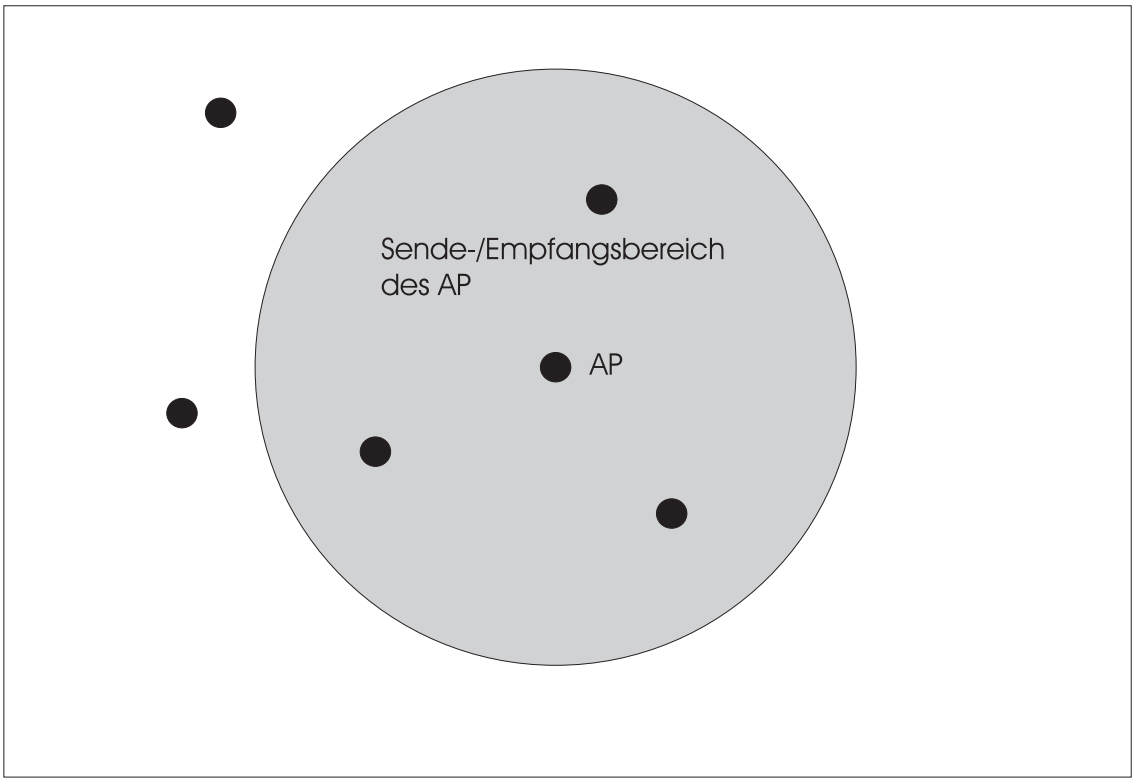


Abbildung 2.4 Alle Stationen, die sich im Sende-/Empfangsbereich des AP befinden, bilden eine Gruppe.

Die betrachtete Gruppe von Stationen wird aus den Stationen eines BSS eines Infrastrukturnetzwerkes nach dem IEEE 802.11 Standard gebildet. Dieses BSS besteht aus

einem AP, der i. A. an einer bestimmten Stelle, z. B. einer Kreuzung oder Autobahnauffahrt, positioniert ist und einer Gruppe von Stationen, die

- sich im Sende-/Empfangsbereich dieses AP befinden;
- sich mit dem AP assoziiert haben;
- nicht ausgefallen sind (s. Abb. 2.4).

Diese Definition der Gruppe ist notwendig, um die Konzepte, die in der CFP den kollisionsfreien Zugriff auf das Medium erlauben, einsetzen zu können. Diese Definition ist aber auch sinnvoll:

- Welche Stationen zusammen eine Gruppe bilden, ist immer eindeutig bestimmt, zumindest solange die APs so positioniert sind, daß sich ihre Sende-/Empfangsbereiche nicht überschneiden. Die Eindeutigkeit ist beispielsweise nicht sichergestellt, wenn per Definition alle Stationen, die sich wechselseitig direkt erreichen können, eine Gruppe bilden, denn nach dieser Definition kann es zu einer Topologie mehrere mögliche Gruppen geben.
- Der AP kann ohne großen Aufwand feststellen, welche Stationen der Gruppe angehören.
- Die Definition beinhaltet eine Invariante bzgl. der Netzwerktopologie, nämlich, daß jede Station der Gruppe sich im Sende-/Empfangsbereich des AP befindet. Hierdurch wird der Einsatz einfacher, zeitlich überschaubarer Routing-Algorithmen möglich.
- Gruppen bilden sich an bestimmten Orten, an denen ein AP positioniert ist. Dies ist in Szenarien wie dem RoboCup oder kooperativem Verhalten von Fahrzeugen im Straßenverkehr sehr sinnvoll.

In dem BSS, das die Gruppe bildet, kommen beide Zugriffsverfahren, DCF und PCF, im zeitlichen Wechsel zum Einsatz. Da jede Station vor ihrer Anmeldung, in einem Beacon-Frame oder in der Antwort auf einen Probe-Frame, die notwendigen Daten zum Bestimmen der TBTT und der maximalen Dauer der CFP erhält, wird davon ausgegangen, daß alle Stationen des BSS ihren NAV entsprechend setzen und so die CFP spätestens nach einer bekannten maximalen Verzögerung nach der TBTT anfangen kann.

Durch den zeitlichen Wechsel der beiden Zugriffsverfahren kann das in [MN99] beschriebene Verfahren der Kanalüberlagerung realisiert werden. Auf das Funknetz angewendet bedeutet dies, daß die CP ein von allen Stationen gemeinsam genutztes Zeitintervall darstellt, das genutzt werden kann, um in effizienter Weise Nachrichten ohne harte Echtzeitanforderungen zu übertragen. In der CFP findet der Zugriff auf das Medium hingegen exklusiv statt. Da die Möglichkeit des exklusiven Zugriffs eine wesentliche Voraussetzung für die Realisierung rechtzeitiger und zuverlässiger Kommunikation

ist, wird das Gruppenkommunikationsprotokoll vollständig in der CFP realisiert. Dabei wird davon ausgegangen, daß zumindest ein Teil der CFP allein dem Gruppenkommunikationsprotokoll zur Verfügung steht.

Da der AP Stationen nicht pollen kann, ohne von deren Existenz zuvor erfahren zu haben, assoziieren sich Stationen, die neu zu einem BSS hinzukommen, in der CP nach dem im Standard dafür vorgesehen Verfahren. In der CP können sich beliebige, a priori unbekannte Stationen am AP anmelden; die maximale Größe der Gruppe ist aber begrenzt (die maximale Gruppengröße wird im Folgenden mit n bezeichnet). Es wird davon ausgegangen, daß am Anfang der CFP alle Stationen eine konsistente Sicht der Gruppe haben. Im ungünstigsten Fall muß der AP dies sicherstellen, indem er am Anfang der CFP die aktuelle Mitgliedschaft allen Stationen zuverlässig bekannt macht, wenn sich in der CP die Gruppe geändert hat.

Wie in Paragraph 2.2.1 gezeigt wurde, besteht eine wesentliche Voraussetzung für die Realisierbarkeit zuverlässiger und rechtzeitiger Kommunikation in der zeitlichen Spezifikation der beteiligten Komponenten. Daher wird das zeitliche Verhalten der Stationen und des Netzwerkes wie folgt spezifiziert:

- Wenn eine Station, nachdem sie vom AP eine Polling-Nachricht erhalten hat, eine Nachricht sendet, dann wird diese von jedem Adressaten entweder nach t_m Zeiteinheiten oder gar nicht empfangen. Ebenso gilt: Wenn der AP eine Nachricht sendet, dann wird diese von jedem Adressaten entweder nach t_m Zeiteinheiten empfangen oder gar nicht.

- Die Bearbeitungszeit in den Instanzen des Gruppenkommunikationsprotokolls der Stationen ist vernachlässigbar klein.

Die maximale Verzögerung t_m beinhaltet alle Verzögerungen, die die Nachricht zwischen dem Aufruf von *send* beim Sender und dem Aufruf von *receive* beim Empfänger erfährt. Die Annahme, daß diese Verzögerung begrenzt ist, gründet sich auf die begrenzte Bitzeit und den exklusiven Zugriff, der sicherstellt, daß der Beginn der Übertragung nicht immer wieder durch andere Stationen hinausgezögert werden kann. Wenn zu der Verzögerung auch lokale Scheduling-Verzögerungen am Sender und Empfänger gehören, dann muß durch den Einsatz eines Echtzeitbetriebssystems deren Begrenztheit sichergestellt werden. Alle Stationen haben Uhren, deren Rate nicht von der der realen Zeit abweicht.

Neben diesen Zeitannahmen werden die folgenden Fehlerannahmen zugrunde gelegt:

1. Das Netzwerk kann Auslassungsfehler erleiden, d. h. es kann beim Übertragen einer Nachricht zum Verlust dieser Nachricht kommen. Durch die im Standard spezifizierte Prüfsumme können Veränderungen von Nachrichten erkannt und auf Nachrichtenverluste abgebildet werden. Nachrichten werden nicht länger als t_m Zeiteinheiten verzögert. Die Anzahl der Nachrichtenverluste ist durch den Omission-Degree OD nach oben begrenzt. Dies bedeutet für die Broadcast-Nachrichten:

a) Wenn die gleiche Nachricht (SDU) in *OD+1* Broadcast-Nachrichten (PDU) übertragen wird, dann hat jede korrekte Station mindestens einmal die Nachricht empfangen. Dies bedeutet nicht, daß eine der Übertragungen von allen Stationen empfangen werden muß.

b) Von *OD+1* aufeinanderfolgenden Broadcast-Nachrichten (PDU) des selben Senders empfängt jede korrekte Station mindestens eine. Auch hier wird nicht verlangt, daß dies bei allen Stationen die gleiche Broadcast-Nachricht ist.

Für Punkt-zu-Punkt-Nachrichten wird angenommen, daß der Omission-Degree sich auf Paare von Polling- und zugehörigen Antwort-Nachrichten bezieht. Das heißt, wenn der AP eine Station *OD+1*-mal pollt und jedes Mal eine Antwort der Station erwartet, dann wird mindestens eines dieser Paare erfolgreich durchgeführt, so daß die Station die Polling-Nachricht und der AP die zugehörige Antwort-Nachricht emfpängt.

2. Stationen können einen Totalausfall erleiden.

3. Stationen können die Gruppe dauerhaft verlassen ohne auszufallen. Dies entspricht einem Totalausfall der Verbindung zwischen dem AP und der betreffenden Station. Da der AP eine ausgefallene Station nicht von einer unterscheiden kann, die die Gruppe verlassen hat, werden nur solche Stationen als korrekt bezeichnet, die nicht ausgefallen sind und die Gruppe nicht verlassen haben.

4. Der AP ist stabil, d. h. der AP fällt nicht aus.

Die Annahme einer begrenzten Anzahl von Auslassungsfehlern des Netzwerkes stellt zusammen mit der begrenzten Verzögerung t_m der Nachrichten sicher, daß durch ein geeignetes Protokoll die Zuverlässigkeit und Rechtzeitigkeit der Kommunikation sichergestellt werden kann. Ohne diese Annahmen wäre dieses nicht möglich. Um sicherzustellen, daß die Annahmen auch plausibel sind, müssen t_m und *OD* hinreichend groß gewählt werden.

Die Notwendigkeit, die Stabilität des AP anzunehmen, stammt von der Kommunikationsstruktur der CFP, bei der der AP als zentraler Koordinator das Zugriffsrecht für das Medium vergibt. Ein Ausfall des AP führt daher dazu, daß keine Station mehr auf das Medium zugreifen kann. Unter der Annahme, daß der AP u. U. ausfällt, könnte also keinerlei Garantie vergeben werden. Die Plausibilität der Annahme kann durch Verfahren der Fehlertoleranz, z. B. durch den Einsatz eines sekundären AP, der beim Ausfall des primären AP dessen Aufgaben übernimmt, sichergestellt werden. Hierzu werden allerdings im IEEE Standard keine Aussagen gemacht und auch diese Diplomarbeit wird sich mit diesem Problem nicht beschäftigen.

Für das Funknetzwerk sei weiterhin angenommen, daß Nachrichten des gleichen Senders in der Reihenfolge empfangen werden, in der sie gesendet wurden.

3 Verwandte Arbeiten

In diesem Kapitel werden Arbeiten dargestellt, die sich mit der Gruppenkommunikation befaßt haben. Wegen der vielen Vorteile, die das Vorhandensein eines Gruppenkommunikationsprotokolls in einem verteilten (Echtzeit-)System bietet, gibt es eine große Fülle von Arbeiten zu dieser Thematik. Daher erhebt die folgende Darstellung der verwandten Arbeiten nicht den Anspruch, jede dieser Arbeiten zu betrachten; dennoch werden einige der bekanntesten Arbeiten so beschrieben, daß die grundlegenden Konzepte, vor allem für das Realisieren von Rechtzeitigkeit und Zuverlässigkeit, deutlich werden.
Wie im vorhergehenden Kapitel zwischen den Eigenschaften von Broadcastprotokollen und von Teilnehmerprotokollen unterschieden wurde, so sollen auch bei den verwandten Arbeiten Broadcastprotokolle und Teilnehmerprotokolle gesondert dargestellt werden. Eine weitere Unterteilung, sowohl der Broadcast- als auch der Teilnehmerprotokolle, wird danach getroffen, ob die Protokolle für den Einsatz in Echtzeitsystemen entworfen wurden und daher zeitlich vorhersagbar sind oder ob sie ohne die Anforderung der zeitlichen Vorhersagbarkeit entwickelt wurden.

3.1 Broadcastprotokolle

3.1.1 Nicht-Echtzeitfähige Broadcastprotokolle

Da keines der im Folgenden betrachteten Protokolle der Eigenschaft der Rechtzeitigkeit genügt, erfüllt keines von ihnen vollständig alle geforderten Eigenschaften. Dennoch ist es sinnvoll, diese Protokolle zu betrachten, da sie meist so entworfen sind, daß sie Nachrichtenverluste tolerieren können. Daher liegt das besondere Augenmerk bei der Betrachtung der Protokolle darauf, welche Mechanismen zum Tolerieren von Nachrichtenverlusten zum Einsatz kommen.

Beim Entwurf von Broadcastprotokollen, die die Rechtzeitigkeit der Nachrichtenübertragung nicht garantieren müssen, werden i. A. nur schwache Annahmen über das zugrunde liegende System getroffen; häufig werden fast vollständig asynchrone Systeme zugrunde gelegt. Zwar sind die Annahmen, die beim Entwurf der Protokolle getroffen wurden, nicht immer die gleichen, die meisten dieser Systeme stimmen aber dennoch in den folgenden Annahmen überein:

- Zeitschranken für die Prozessorgeschwindigkeiten sowie für die Nachrichtenverzögerung sind entweder nicht bekannt, oder sie sind bekannt und es kann zu Zeitfehlern kommen.

- Prozesse können Crash-Fehler erleiden.

- Nachrichten können verloren gehen, d. h. auf dem Medium sind Auslassungsfehler möglich.

- Timeouts können als Mechanismen zur unzuverlässigen Fehlererkennung eingesetzt werden.

Die Gliederung der folgenden Darstellung der Broadcastprotokolle erfolgt nach dem zum Ordnen der Nachrichten eingesetzten Verfahren in symmetrische und zentralisierte Ansätze. Beim zentralisierten Ansatz gibt es zu jedem Zeitpunkt im System nur eine einzige Station[10], die das Recht hat, Nachrichten zu ordnen. Im Gegensatz dazu gibt es bei den symmetrischen Ansätzen eine solche zentrale Station nicht. Hier ordnet jede Station die Nachrichten aufgrund der ihr verfügbaren Informationen, die sie aus den erhaltenen Nachrichten ableitet, selbst.

Symmetrischer Ansatz

Symmetrische Broadcastprotokolle (Psync [PBS89], Trans/Total [MMA90], [MMA93], Transis/ToTo [ADKM92b], [DKM92]) sind in zwei Schichten aufgeteilt. Auf der unteren Schicht kommen Protokolle zum Einsatz, die zuverlässige, kausal geordnete Broadcasts realisieren, darüber Protokolle, die darauf aufbauend eine totale Ordnung generieren.

Die Protokolle für zuverlässige, kausal geordnete Broadcasts (Psync [PBS89], Trans [MMA90], Transis [ADKM92b]) garantieren die Eigenschaften Integrität, Validität, Einigung und kausale Ordnung. Die Eigenschaft der kausalen Ordnung von Broadcast-Nachrichten besagt, daß keine Station eine Broadcast-Nachricht ausliefert, bevor sie nicht alle potentiellen kausalen Vorgänger dieser Nachricht ausgeliefert hat. Ein Ereignis *e* (Broadcasten oder Ausliefern einer Nachricht) ist potentieller kausaler Vorgänger eines Ereignisses *f*, wenn ([Lam78], [HT93]):

1. eine Station zuerst *e* und dann *f* ausführt oder

2. *e* das Broadcasten einer Nachricht und *f* das Ausliefern dieser Nachricht ist oder

3. wenn es ein Ereignis *h* gibt, so daß *e* potentieller kausaler Vorgänger von *h* und *h* potentieller kausaler Vorgänger von *f* ist.

Durch die kausale Ordnung der Ereignisse wird eine partielle Ordnung über den Broadcast-Nachrichten definiert. Daher kann der Strom der Broadcast-Nachrichten auch als azyklischer, gerichteter Graph aufgefaßt werden, der für die Prozesse nach und nach sichtbar wird. Dieser Graph, der sog. Kontextgraph, ist die Grundlage der symmetrischen Algorithmen.

---

[10] Auch in diesem Kapitel wird der Begriff der Station beibehalten, auch wenn in der Literatur meistens von „Stelle“, „Prozessor“ oder „Prozeß“ gesprochen wird.

Die Informationen, die notwendig sind, um die kausal geordnete Auslieferung der Nachrichten in allen Prozessen sicherzustellen, können ohne zusätzliche Nachrichten zwischen den Stationen übermittelt werden. Dazu genügt es beispielsweise, daß jede Station in jeder Broadcast-Nachricht alle diejenigen Broadcast-Nachrichten bestätigt, die sie nach dem Senden der letzten Broadcast-Nachricht empfangen hat (s. Abb. 3.1). Um dies zu ermöglichen, werden alle Nachrichten eindeutig gekennzeichnet (z. B. durch Paare <Adresse der Station, lokale Sequenznummer>).

Wenn eine Station s_1 eine Nachricht m von einer Station s_2 empfängt und eine Nachricht m' ist Vorgänger von m im Kontextgraph, aber s_1 hat m' noch nicht empfangen, dann kann Station s_1 dies feststellen und die Nachricht m' von s_2 anfordern. Unter den Annahmen, daß jede Station hinreichend viele Nachfolger jeder Nachricht empfängt und jede Nachricht beliebig oft erneut übertragen werden kann, wird durch dieses Verfahren die Zuverlässigkeit der Nachrichtenübertragung gewährleistet. Solange jeder Prozeß alle Nachrichten speichert, die er erhalten hat, stellt das Verfahren auch die Einigung unter den Stationen sicher. Allerdings kann bei diesem Verfahren der Verlust einer Nachricht m von einer Station erst dann erkannt werden, wenn diese eine Nachfolgenachricht von m empfängt. Daher ist die Zeit, die benötigt wird, bis eine Nachricht bei allen korrekten Prozessen ausgeliefert wird, von dem Verkehr auf dem Medium abhängig und schwer vorhersagbar. Wenn eine Nachricht nur wenige kausale Nachfolger hat, kann es vorkommen, daß eine Station keinen dieser Nachfolger empfängt, und es können Nachrichtenverluste auftreten.

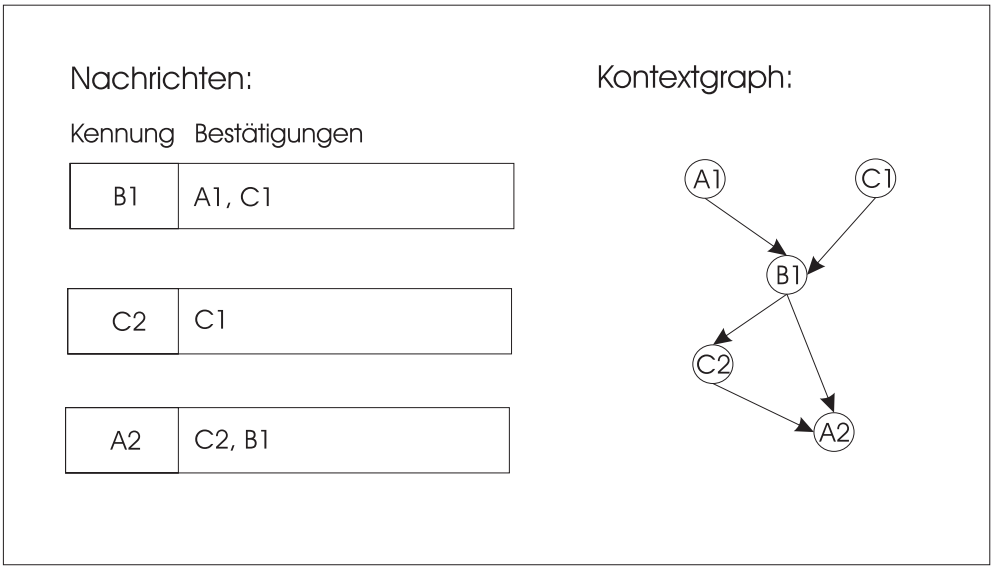


Abbildung 3.1 Der Kontextgraph kann aufgrund der Bestätigungen gebildet werden.

Protokolle wie Total [MMA93] und ToTo [DKM92] erzeugen aus einer kausal, also partiell geordneten Folge von Nachrichten eine total geordnete, wobei sie die kausale Ordnung der Nachrichten aufrecht erhalten. Die Aufgabe dieser Protokolle ist das Ordnen der nebenläufigen, d. h. der durch die kausale Ordnung nicht vergleichbaren, Nach-

richten. Um dies zu erreichen, einigen sich alle Stationen auf eine Teilmenge der nebenläufigen Nachrichten, die im nächsten Schritt ausgeliefert werden soll. Innerhalb dieser Teilmenge kann eine beliebige deterministische Ordnung unter den Nachrichten gewählt werden.

In einem asynchronen System kann es keinen fehlertoleranten, deterministischen Algorithmus geben, der aus der kausalen Ordnung von Nachrichten eine totale Ordnung erzeugt. Dies folgt unmittelbar aus der Unmöglichkeit atomarer Broadcasts in asynchronen Systemen und der Möglichkeit, kausale Ordnung in asynchronen Systemen zu realisieren. Daher müssen die Algorithmen entweder Annahmen treffen, die über zeitlose asynchrone Systeme hinausgehen, oder die erzielten Eigenschaften einschränken.

Das Protokoll Total entscheidet über die Teilmenge der Nachrichten, die als nächstes ausgeliefert werden sollen, aufgrund eines Voting-Mechanismus, bei dem sich eine Mehrheit der Stationen auf eine Teilmenge von Nachrichten einigen muß. Diese Einigung wird in jedem Prozeß nach dem gleichen Verfahren allein aus den Informationen des Kontext-Graphen erzielt. Der Total Algorithmus kann auch dann mit dem Ordnen der Nachrichten fortfahren, wenn es zu Stationsausfällen kommt, ohne diese Ausfälle erkennen zu müssen. Da dem Algorithmus keine zusätzlichen Annahmen zugrunde gelegt werden, kann er aber nur eine probalisitische Terminierung sicherstellen, d. h. das Protokoll garantiert nicht, daß eine Nachricht, die es als Input erhält, nach endlicher Zeit ausgeliefert wird. Daher garantiert es die Eigenschaft der Validität nicht, weshalb die Atomarität der Nachrichten nur probabilistisch sichergestellt ist.

In [PBS89] wird ein einfaches Verfahren zum Erzielen einer totalen Ordnung vorgestellt, das eine Menge von nebenläufigen Nachrichten immer dann ordnet und ausliefert, wenn mindestens eine der Nachrichten stabil, d. h. von allen Stationen empfangen worden ist. Um die Stabilität einer Nachricht auch beim Auftreten von Stationsausfällen feststellen zu können, bedarf das Protokoll allerdings eines Teilnehmerprotokolls, das Stationsausfälle erkennt. Dieses wiederum setzt die Möglichkeit voraus, Stationsausfälle (unzuverlässig) erkennen zu können. Auch das „ToTo“ Protokoll [DKM92] verläßt sich auf die Existenz eines Teilnehmerprotokolls, um die Terminierung zu garantieren. ToTo führt (ähnlich wie Total) aufgrund der Informationen des Kontextgraphen Mehrheitsentscheidungen von Stationen über die Menge der Nachrichten herbei, die als nächstes geordnet und ausgeliefert werden sollen.

Symmetrische Protokolle haben den Vorteil, daß es sich um wirklich verteilte Protokolle handelt, bei denen es keine Station gibt, die sich von den anderen unterscheidet. Daher ist auch die Last auf allen Prozessen etwa gleich und es treten keine Engpässe auf. Zum Erreichen von Validität, Einigung, kausaler und totaler Ordnung werden zusätzliche Nachrichten nur dann benötigt, wenn Nachrichtenverluste auftreten.

Der hauptsächliche Nachteil der Protokolle besteht darin, daß die Entscheidungen von der Existenz von Nachfolgenachrichten abhängig sind. Nachrichtenverluste oder die Stabilität können erst erkannt werden, wenn kausale Nachfolger von Nachrichten von

den Stationen empfangen werden. Da die Zeitpunkte, wann dies der Fall sein wird, nicht bestimmbar sind, sind die Protokolle in ihrem Zeitverhalten nicht vorhersagbar. Abhilfe kann nur durch das Einfügen von zusätzlichen Nachrichten, die in bestimmten Zeitabständen gesendet werden, geschaffen werden. Dies würde aber auch den Vorteil des geringen Nachrichtenaufwandes wieder einschränken.

Zentralisierter Ansatz

Beim zentralisierten Ansatz ist zu jedem Zeitpunkt höchstens eine Station im System für das Ordnen der Broadcast-Nachrichten verantwortlich. Das Recht, Broadcast-Nachrichten zu Ordnen, wird als *Token,* und die Station, die dieses Recht besitzt, mithin als *Tokenstation* bezeichnet. Die Kommunikation wird daher bei den in [CM84] und [KT91] dargestellten Protokollen sowie bei RMP [JKN96] in zwei Phasen unterteilt: Die Kommunikation zwischen der Station, die eine Broadcast-Nachricht übertragen möchte (dem Urheber der Nachricht) und der Tokenstation, und die Kommunikation zwischen der Tokenstation und allen Stationen der Gruppe.

Eine Station, die eine Nachricht *m* atomar broadcastet, sendet diese als Broadcast-Nachricht an die Gruppe. Die Tokenstation sendet eine Bestätigung für *m* an die Gruppe, in der sie der Nachricht eine eindeutige Sequenznummer zuordnet. Die Zuordnung zwischen einer Nachricht und einer Bestätigung erfolgt durch eine eindeutige Bezeichnung der Nachrichten. Die Tokenstation inkrementiert vor dem Versenden jeder Bestätigung, die Sequenznummer, so daß die Nachrichten eindeutige, aufsteigende Sequenznummern erhalten. Bei einer anderen Variante des selben Verfahrens, die in [KT91] und [JKN96] beschrieben wird, sendet eine Station, die eine Nachricht *m* Broadcasten möchte, *m* an die Tokenstation, welche *m* eine globale Sequenznummer zuordnet und *m* anschließend als Broadcast-Nachricht sendet. Im Gegensatz zur ersten Variante erzeugt dieses Verfahren mehr Last auf dem Netzwerk, dafür aber weniger Interrupts an den Stationen. In den Stationen werden die Nachrichten aufsteigend nach ihren Sequenznummern ausgeliefert; eine Station liefert eine Nachricht mit Sequenznummer s erst dann aus, wenn sie alle Nachrichten mit Sequenznummern $s' < s$ zuvor ausgeliefert hat. So ist die totale Ordnung der ausgelieferten Nachrichten sichergestellt.

Zum Tolerieren von Nachrichtenverlusten werden in beiden Protokollphasen unterschiedliche Verfahren eingesetzt. Die Kommunikation zwischen dem Urheber der Nachricht und der Tokenstation wird durch ein Verfahren mit positiven Bestätigungen gesichert: Der Urheber sendet die Nachricht solange erneut, bis er eine Bestätigung von der Tokenstation empfängt. Bei der zweiten Variante des Verfahrens besteht diese Bestätigung im Broadcast der Nachricht durch die Tokenstation.

Die Kommunikation zwischen der Tokenstation und den Mitgliedern der Gruppe wird durch ein Verfahren mit negativen Bestätigungen gegen Nachrichtenverluste gesichert: Stationen, die Nachrichten oder Bestätigungen verpaßt haben, können dies anhand von Lücken in der Folge der Sequenznummern feststellen. Sie senden, wenn dieser Fall eintritt, negative Bestätigungen an die Tokenstation, die das Neusenden der verlorenen

Nachrichten anfordern. Solange die Tokenstation immer wieder neue Nachrichten broadcastet, wird auf diese Weise jeder Nachrichtenverlust erkannt und behoben.

Der Einsatz von negativen Bestätigungen hat zwar den Vorteil, daß zusätzliche Nachrichten nur dann gesendet werden müssen, wenn es tatsächlich zu Nachrichtenverlusten kommt, aber auch den Nachteil, daß die Tokenstation jede Nachricht unendlich lange aufheben muß. Um dies zu vermeiden, wird in [CM84] und [JKN96] die Verantwortlichkeit der Tokenstation innerhalb der Gruppe in einem logischen Ring weitergegeben. Dies geschieht implizit, wenn die bisherige Tokenstation eine Bestätigung bzw. eine Broadcast-Nachricht sendet. Die neue Tokenstation darf den Token nur dann annehmen, wenn sie alle Nachrichten bis einschließlich der zuletzt bestätigten erhalten hat. Wenn umgekehrt die neue Tokenstation den Erhalt des Tokens bestätigt, dann bestätigt sie damit implizit auch den Empfang aller Nachrichten und Bestätigungen bis zur höchsten Sequenznummer. Dies hat zur Folge, daß nach einer vollständigen Tokenrotation, von Station *s* zu Station *s*, alle Stationen implizit die von Station *s* zuletzt geordnete Nachricht bestätigt haben. Diese Nachricht ist stabil und wird von keiner Station mehr angefordert werden.

Kommt es in RMP oder in dem in [CM84] dargestellten Protokoll zu Stationsausfällen, so ist der logische Ring unterbrochen und das Versenden und Ordnen von Broadcast-Nachrichten kann erst dann fortgesetzt werden, wenn durch ein Teilnehmerprotokoll eine neue Gruppe hergestellt wurde. Dieser Vorgang wird als Reformation des Ringes bezeichnet. Wenn bei der Reformation zwischenzeitlich korrekte Stationen aus der Gruppe ausgeschlossen werden, dann kann dies dazu führen, daß die Einigung nicht mehr gewährleistet ist. Um dies zu vermeiden, darf eine Station eine Nachricht nur dann ausliefern, wenn sie aufgrund der Tokenrotation festgestellt hat, daß die Nachricht von mindestens *k* Stationen außer dem Sender empfangen wurde, d. h. *k*-stabil ist. Unter der Annahme, daß höchstens *k* Stationen ausfallen, kann das Teilnehmerprotokoll dann sicherstellen, daß in jeder neuen Gruppe mindestens eine Station enthalten ist, die alle bereits ausgelieferten Nachrichten empfangen hat, so daß keine Nachricht, die von einer Station bereits ausgeliefert wurde, verloren gehen kann. Durch diese Erweiterung erfüllen die Protokolle die Eigenschaft der uniformen Einigung.

Wie häufig der Token rotiert, d. h. nach wie vielen Bestätigungen der Token zur nächsten Station wechselt, ist in [CM84] nicht festgelegt, sondern als Protokollparameter spezifiziert, so daß es sich eigentlich um eine Protokollfamilie handelt. Eine Instanz dieser Familie ist das in [KT91] dargestellte Protokoll, bei dem der Token nicht in einem logischen Ring rotiert, sondern fest an eine Station, den sog. Sequenzer, gebunden ist. Um auch ohne ein rotierendes Token die Stabilität von Nachrichten feststellen zu können, übertragen die Stationen in den Punkt-zu-Punkt-Nachrichten einen Parameter *s* an den Sequenzer. Durch diesen Parameter bestätigt der Urheber der Punkt-zu-Punkt-Nachricht, daß er alle Broadcast-Nachrichten mit Sequenznummern kleiner oder gleich *s* empfangen hat. Der Sequenzer leitet diese Information in den Broadcast-Nachrichten an

die Stationen weiter, so daß jede Station s feststellen kann, bis zu welcher Sequenznummer jede andere Station s' die Broadcast-Nachrichten empfangen hat ($s.sg[s']$[11]). Der Wert $s.k = \min\{s.sg[s'] \mid s' \text{ ist Gruppenmitglied}\}$ besagt, daß jede Station der Gruppe alle Broadcast-Nachrichten mit Sequenznummern $s \leq s.k$ erhalten hat und daß diese Nachrichten daher nicht länger gespeichert werden müssen. RMP setzt dieses Verfahren ebenfalls ein, um dadurch das Erkennen der Stabilität von Nachrichten und damit ihre Auslieferung zu beschleunigen.

Auch Tanenbaum et al. [KT91] nutzen das Konzept der k-Stabilität von Nachrichten, um uniforme Einigung zu erreichen bzw. um Einigung zu erreichen, wenn sich Gruppen bilden, in denen nicht alle korrekten Prozesse enthalten sind. Da auch hierzu nicht die Tokenrotation verwendet werden kann, wie es in den bisher besprochenen Protokollen der Fall ist, wird das Protokoll wie folgt abgewandelt: Eine Station, die eine Nachricht atomar broadcastet, sendet diese als Broadcast-Nachricht an alle Stationen. Der Sequenzer speichert die Nachricht und wartet, bis er Bestätigung von mindestens k Stationen empfangen hat. Erst dann sendet er eine *accept*-Nachricht, in der er auch die Sequenznummer der Nachricht bekannt gibt. Wenn es zu Ausfällen kommt, kann sichergestellt werden, daß in jeder neuen Gruppe immer mindestens ein korrekter Prozeß enthalten ist, der die Nachricht gespeichert hat. Ob und für welchen Parameter k die k-Stabilität sichergestellt werden soll, kann bei der Konfiguration der Gruppe entschieden werden.

Auch das Protokoll Totem ([MMABL96], [AMMAC95]) ist ein zentralisiertes Verfahren, bei dem ein Token in einem logischen Tokenring kreist. Allerdings erfolgt bei Totem die Tokenweitergabe nicht implizit mit dem Senden der Broadcast-Nachrichten, sondern in Form einer expliziten Tokennachricht. Diese Tokennachricht enthält neben anderen Feldern auch die aktuelle Sequenznummer. Eine Station, welche die Tokennachricht empfängt, darf neue Broadcast-Nachrichten senden. Dazu inkrementiert sie vor dem Senden jeder neuen Broadcast-Nachricht die Sequenznummer aus der Tokennachricht und ordnet diese der Broadcast-Nachricht zu. Auch bei Totem werden Nachrichtenverluste anhand von Lücken in der Folge der Sequenznummern festgestellt. Ein Prozeß, der eine verlorene Nachricht feststellt, hängt die Sequenznummer dieser Nachricht an die "retransmission-request-list", die ebenfalls Teil des Tokens ist. Jeder Prozeß, der die Tokennachricht erhält, prüft, ob er Nachrichten aus der retransmission-request-list empfangen hat, und überträgt sie ggf. neu.

Die Stabilität der Nachrichten wird anhand des *aru*-Feldes (all received up to) der Tokennachricht festgestellt. In dem *aru*-Feld kann ein Prozeß, der den Token weitergibt, angeben, bis zu welcher Sequenznummer die Nachrichten von allen Prozessen seiner Ansicht nach empfangen sein könnten. Wenn ein Prozeß zweimal in Folge die Tokennachricht mit einem *aru*-Wert $\geq k$ weitergibt, dann kann er sicher sein, daß alle Nach-

---

[11] Bei Variablen, die von allen Stationen verwaltet werden, wird durch die Notation <*Station*>.<*Variable*> angegeben, welche Variable gemeint ist.

richten mit Sequenznummern kleiner oder gleich *k* von allen Prozessen empfangen worden sind.

In ISIS ([BJ87], [BSS91]) ist die Grundlage des Protokolls für atomare Broadcasts (ABCAST) ein Protokoll, das zuverlässige kausale Broadcasts realisiert (CBCAST). Das CBCAST-Protokoll geht davon aus, daß die zuverlässige Übertragung der Nachrichten durch Protokolle auf unteren Schichten sichergestellt ist, weshalb in beiden Protokollen, CBCAST und ABCAST, keine Mechanismen zum Tolerieren von Nachrichtenverlusten vorhanden sind. Der ABCAST ermöglicht aufbauend auf dem CBCAST die total geordnete Auslieferung der Nachrichten. Dazu gibt es in der Gruppe eine Tokenstation, die für das Ordnen der Nachrichten verantwortlich ist. Die Tokenstation merkt sich die Reihenfolge, in der sie die Nachrichten ausliefert, und gibt diese von Zeit zu Zeit den anderen Prozessen in Form von kausalen Broadcasts bekannt.

Der zentralisierte Ansatz ist in seinem zeitlichen Verhalten überschaubarer als die symmetrischen Ansätze. Weiterhin kann das Erkennen von Nachrichtenverlusten und damit die Realisierung der Zuverlässigkeit beim Auftreten von Nachrichtenverlusten unter geringem zusätzlichem Aufwand an Nachrichten mit Hilfe von positiven und negativen Bestätigungen durchgeführt werden. Der Einsatz eines rotierenden Tokens erlaubt es weiterhin, daß auch die Stabilität von Nachrichten ohne zusätzlichen Aufwand festgestellt werden kann.

Der Einsatz eines rotierenden Tokens hat aber den Nachteil, daß der Zeitaufwand für die Reformation erheblich und zeitlich schwer vorhersagbar ist. Dies ist nicht problematisch in Systemen, bei denen zeitliche Vorhersagbarkeit von untergeordneter Bedeutung ist und in denen Tokenverluste und Reformationen nur sehr selten auftreten. In der Zielumgebung dieser Diplomarbeit ist zeitliche Vorhersagbarkeit aber von äußerster Wichtigkeit. Da weiterhin aufgrund der dynamischen und unbekannten Topologie des Netzwerkes Tokenverluste zwischen zwei Stationen eines logischen Ringes häufig auftreten werden, sind diese Protokolle für die betrachtete Umgebung ungeeignet.

Das Verfahren, das in [KT91] zum Erreichen von stabilen Nachrichten eingesetzt wird, hat einen erheblich höheren Nachrichtenaufwand als die Verfahren mit rotierendem Token. Für jede *k*-stabile Nachricht müssen *k* explizite Bestätigungsnachrichten gesendet werden. Das Teilnehmerprotokoll, das bei Stationsausfällen zum Einsatz kommt, ist ähnlich komplex und schwer vorhersagbar wie die Verfahren bei rotierendem Token. Der große Vorteil des Sequenzer-Protokolls im hier betrachteten Kontext liegt darin, daß die Kommunikationsstruktur sehr gut mit der des Infrastrukturnetzwerkes übereinstimmt. Der Nachrichtenaustausch findet nur zwischen den Stationen und dem Sequenzer statt, die Stationen untereinander tauschen keine Nachrichten aus. Würde man beim Infrastrukturnetzwerk den AP zum Sequenzer machen, dann fände der Nachrichtenaustausch nur zwischen dem AP und den Stationen statt. Da der Token nicht zwischen Stationen, die sich u. U. nicht erreichen können, explizit oder implizit weitergegeben werden muß, ist die Wahrscheinlichkeit, daß es zu Tokenverlusten und Reformationen kommt, bei einem Ansatz mit Sequenzer erheblich geringer.

Keines der hier vorgestellten Protokolle kommt direkt für den Einsatz in der Zielumgebung in Frage, da die Protokolle alle nicht für den Einsatz in Echtzeitsystemen konzipiert wurden. Daher wird von keinem der Protokolle die zeitliche Vorhersagbarkeit gewährleistet. Dies gilt natürlich insbesondere, wenn der Einsatz auf einem Funknetz erfolgt. Dennoch sind einige der vorgestellten Konzepte auch im Rahmen dieser Diplomarbeit einsetzbar, beispielsweise die Verwendung einer zentralen Station für das Versenden und Ordnen von Broadcasts oder die Verwendung von impliziten positiven Bestätigungen für die Punkt-zu-Punkt Kommunikation.

3.1.2 Echtzeitfähige Broadcastprotokolle

3.1.2.1 In Synchronen Systemen

In Echtzeitsystemen müssen Broadcast-Protokolle neben der Atomarität der Broadcast-Nachrichten auch ihre rechtzeitige Auslieferung garantieren. Daher werden Broadcast-Protokolle für Echtzeitsysteme häufig auf Grundlage von synchronen Systemen, d. h. unter der Annahme, daß die Möglichkeit zu rechtzeitiger und zuverlässiger Kommunikation gegeben ist, entworfen.

Das TTP (Time Triggered Protocol [Kop97], [KG93]) setzt auf einem Broadcastmedium auf, auf das die angeschlossenen Stationen nach einem statischen TDMA-Verfahren zugreifen, Zur Compilezeit des Systems werden jeder Station feste Zeitabschnitte (Slots) zugewiesen, in denen sie auf das Medium zugreifen darf. Eine globale Zeit mit bekannter Genauigkeit stellt die dazu notwendige globale Zeitbasis dar. Um die zuverlässige Übertragung der Broadcast-Nachrichten sicherzustellen, sendet jede Station jede Nachricht auf zwei redundanten Übertragungskanälen. Mehrere redundante Stationen können zu einer FTU (fault tolerant unit) zusammengefaßt werden. Da alle Stationen einer FTU die gleichen Nachrichten senden, werden alle Nachrichten auf jedem Medium mehrmals gesendet.

Aufbauend auf dieser Möglichkeit der rechtzeitigen und zuverlässigen Übertragung ist auch die totale Ordnung der Nachrichten sichergestellt, da die Nachrichten einfach in der Reihenfolge ausgeliefert werden, in der sie auf dem Medium übertragen werden. Da es nicht zu Neuübertragungen von verlorenen Nachrichten kommt, wird an allen Stationen die gleiche Reihenfolge beobachtet. Kommt es zu asymmetrischen Nachrichtenverlusten, die auf Auslassungsfehler der Stationen zurückgeführt werden, dann wird durch das Teilnehmerprotokoll (s. Paragraph 3.2.2) sichergestellt, daß sich die Mehrheitsansicht durchsetzt: Nur solche Stationen bleiben aktiv, die gleicher Ansicht über die empfangenen Nachrichten sind und die gemeinsam eine Mehrheit aller Stationen bilden.

Das beschriebene Verfahren realisiert rechtzeitige, atomare Broadcasts, ist aber in seiner Struktur sehr statisch, da es auf vorgeplanten Zeitplänen beruht. Für die Vorplanung müssen zum Planungszeitpunkt alle Stationen, die an der Gruppenkommunikation teil-

nehmen, bekannt sein; die Aufnahme neuer Stationen ist nicht möglich. Der Einsatz statischer Zeitredundanz, d. h. das mehrfache versenden jeder Nachricht, verursacht einen konstanten, von der tatsächlichen Anzahl von Nachrichtenverlusten unabhängigen Aufwand, der sich nach der maximalen Anzahl von Nachrichtenverlusten richtet. Daher ist dieses Verfahren sehr ungünstig, wenn die maximale Anzahl von Nachrichtenverlusten groß ist.

In [CASD85] wird eine Folge von Protokollen vorgestellt, die atomare, rechtzeitige Broadcasts unter der Annahme verschiedener Fehlermodelle für Stationsausfälle realisieren. Auch diese Protokolle gehen von der Möglichkeit der zuverlässigen und rechtzeitigen Kommunikation aus.

Alle Protokolle basieren auf dem Prinzip der Diffusion, bei dem jede Station, die eine Nachricht zum erstenmal empfängt, diese an alle anderen Stationen, mit denen sie verbunden ist, weiterleitet. Dieses Verfahren stellt unter der Annahme, daß jede korrekte Station mit jeder anderen korrekten Station, u. U. indirekt, verbunden ist, sicher, daß:

1. jede korrekte Station jede andere korrekte Station erreicht, da jede Nachricht über alle möglichen Wege gesendet wird;

2. jede Nachricht, die von einer korrekten Station ausgeliefert wurde, auch von jeder anderen korrekten Station ausgeliefert wird, da jede Station, die eine Nachricht ausliefert, diese vorher selbst an alle ihre Nachbarn weitergeleitet hat und die Nachricht so von allen korrekten Prozessen empfangen wird (s. 1.), selbst dann, wenn der Urheber der Nachricht ausfällt;

3. es eine Zeitschranke Δ gibt, so daß jede Nachricht von allen korrekten Stationen nach Δ Zeiteinheiten ausgeliefert wird oder gar nicht.

Das Diffusions-Verfahren realisiert also unter den gegeben Annahmen zuverlässige, rechtzeitige Broadcasts.

Unter den Nachrichten wird eine zeitliche Ordnung hergestellt. Hierzu wird davon ausgegangen, daß die Uhren aller korrekten Stationen mit bekannter Genauigkeit synchronisiert sind. Jede Station, die eine Nachricht sendet, versieht die Nachricht vorher mit einem Zeitstempel; alle Stationen liefern die Nachrichten in der Reihenfolge ihrer Zeitstempel aus. Damit dieses Verfahren funktioniert, ist es wichtig, daß eine Station eine Nachricht erst dann ausliefert, wenn sie sicher sein kann, daß sie keine Nachricht mehr empfangen wird, die einen kleineren Zeitstempel hat. Dies kann aufgrund der bekannten Zeitschranke Δ sichergestellt werden, indem jede Station eine Nachricht mit Zeitstempel t zum Zeitpunkt $t + \Delta$ ausliefert.

Das Diffusionsverfahren ist mit einem sehr hohen Nachrichtenaufwand verbunden. Zum Beispiel würden auf einem Broadcast-Medium mit n angeschlossenen Stationen $n \times (n-1)$ Punkt-zu-Punkt oder n Broadcast-Nachrichten gesendet werden. Weiterhin muß beim

Berechnen von Δ der ungünstigste Fall zugrunde gelegt werden, um sicherzustellen, daß nach dem Zeitpunkt $t + \Delta$ auf der Uhr einer Station definitiv keine Nachrichten mit Zeitstempel $t' < t$ mehr an dieser Station ankommen. So wird jede Nachricht mit der maximalen Verzögerung Δ ausgeliefert.

3.1.2.2 In Partiell Synchronen Systemen

Wie oben erläutert, ist die Realisierung synchroner Systeme mit erheblichen Kosten verbunden. In [RV92] wird mit xAMP daher ein Protokoll vorgestellt, daß auch unter schwächeren Annahmen über das zugrunde gelegte System die zeitliche Vorhersagbarkeit für das Versenden von atomaren Broadcasts erreicht.

Die dem xAMP zugrunde gelegten Annahmen über das Kommunikationssystem sind dahingehend gelockert, daß Nachrichtenverluste zugelassen werden (s. Paragraph 2.2.1). Um dennoch die Vorhersagbarkeit der Kommunikation sicherstellen zu können, werden die Eigenschaften eines abstrakten Netzwerkes definiert, die hinreichend sind, um mit geeigneten Protokollen rechtzeitige und zuverlässige Kommunikation zu ermöglichen. In dieser Hinsicht besteht der wesentliche Unterschied zu TTP darin, daß die Behandlung von Nachrichtenverlusten nicht statisch und unabhängig vom Broadcastprotokoll erfolgt, sondern dynamisch und innerhalb des Broadcastprotokolls. Die wesentlichen Annahmen, auf denen die Protokolleigenschaften realisiert werden, sind (eine detaillierte Beschreibung der Annahmen findet sich in [VM90]):

- Begrenzte Nachrichtenverzögerungen auf dem Medium (das schließt die Medienzugangszeit ein).

- Begrenzte Anzahl von Auslassungsfehlern des Mediums. Das bedeutet, daß in jedem Zeitintervall einer vorgegebenen Länge nur eine begrenzte Anzahl von Nachrichtenverlusten auftritt.

Der Ablauf eines Broadcasts in xAMP kann in zwei Phasen unterteilt werden, die unter der Kontrolle des Urhebers des Broadcasts stehen. In der ersten Phase (Übertragungs-Phase) broadcastet der Urheber die Nachricht und erwartet daraufhin von allen adressierten Stationen eine Bestätigungsnachricht. Der Urheber wechselt in die Entscheidungsphase, wenn er entweder von allen Adressaten eine Bestätigung empfangen hat oder ein Timeout, das er vor dem Senden der Nachricht gesetzt hat, abgelaufen ist. Im ersten Falle weiß der Urheber, daß alle Stationen die Broadcast-Nachricht empfangen haben, und er versendet eine *accept*-Nachricht, die die Stationen anweist, die Nachricht auszuliefern. Im zweiten Falle geht der Urheber davon aus, daß ein Auslassungsfehler aufgetreten ist; er versendet eine *reject*-Nachricht, die alle Stationen anweist, die empfangene Broadcast-Nachricht zu verwerfen, und wechselt erneut in die Übertragungsphase. Auch in der Entscheidungsphase werden Bestätigungen verwendet, um Nachrichtenverluste zu tolerieren und sicherzustellen, daß alle Adressaten die Nachricht ausliefern.

Stationen, die eine Broadcast-Nachricht empfangen, liefern diese erst aus, wenn sie eine *accept*-Nachricht empfangen haben. Die Broadcast-Nachrichten werden von allen Stationen in der Reihenfolge ausgeliefert, in der sie empfangen wurden. Da das Protokoll sicherstellt, daß alle Stationen sich beim Ausliefern auf den selben Sendeversuch der Nachricht beziehen, ist unter der Annahme, daß die Nachrichten auf dem Medium total geordnet wahrgenommen werden, die total geordnete Auslieferung der Nachrichten sichergestellt.

Die Annahmen über das abstrakte Netzwerk, das z. B. durch einen Tokenring realisiert werden kann, stellen sicher, daß sowohl die Übertragungs- als auch die Entscheidungsphase terminieren. Fällt der Urheber einer Nachricht während einer der Phasen aus, so stellt ein Terminierungsprotokoll sicher, daß die verbleibenden Stationen sich einigen, ob die Nachricht ausgeliefert wird oder nicht.

Beim xAMP ist die maximale Abweichung zwischen den Verzögerungen der Broadcast-Nachrichten (Gleichmäßigkeit, steadiness) sowie die maximale Abweichung zwischen den Auslieferungszeitpunkten der selben Broadcast-Nachricht an unterschiedlichen Stationen (Laufzeitvarianz, tightness) erheblich größer als in TTP und den von Cristian et. al. vorgestellten Protokollen. Dies liegt daran, daß bei den letztgenannten Protokollen die Broadcast-Nachrichten immer zu bestimmten Zeitpunkten auf einer globalen Zeitachse ausgeliefert werden. Diese Zeitpunkte werden anhand der maximalen Verzögerung aus dem Sendezeitpunkt einer Nachricht bestimmt. Durch den Verzicht auf kleine Werte für Gleichmäßigkeit und Laufzeitvarianz, können beim xAMP Nachrichten ausgeliefert werden, sobald ihre Stabilität sichergestellt ist. Daher kann beim xAMP die Auslieferung einer Nachricht mit einer Verzögerung erfolgen, die u. U. deutlich kleiner ist als die maximale. Weiterhin hat das xAMP den Vorteil, daß die Nachrichtenverluste explizit im Rahmen des Broadcastprotokolls behandelt werden und nicht durch Zuverlässigkeitsmechanismen auf unteren Ebenen. Vor allem baut der Zuverlässigkeitsmechanismus nicht auf statische sondern auf dynamische Redundanz, was besonders dann von Vorteil ist, wenn eine große Anzahl von Nachrichtenverlusten auftreten kann und erhebliche Abweichungen zwischen der maximalen und durchschnittlichen Anzahl von Verlusten bestehen. Die Verbindung der Tolerierung von Nachrichtenverlusten mit Rechtzeitigkeit machen xAMP besonders interessant für die Entwicklung eines Echtzeit-Broadcastprotokolls auf einem Funknetz.

Das xAMP setzt zum Erkennen von Nachrichtenverlusten explizite Bestätigungsnachrichten ein. Da jede Broadcast-Nachricht von jedem Empfänger durch eine gesonderte Nachricht bestätigt werden muß, führt dieser Mechanismus zu einem hohen Nachrichtenaufkommen (2 Broadcast-Nachrichten und $2n$ Bestätigungen im günstigsten Falle). Treten Nachrichtenverluste auf, dann kann der Aufwand sogar noch erheblich größer werden. Besonders ungünstig ist der Einsatz expliziter Bestätigungsnachrichten, wenn die begrenzte Zugriffszeit wie nach dem IEEE Standard durch Polling erreicht wird. In diesem Falle kommt nämlich zu jeder Nachricht noch eine Polling-Nachricht hinzu, so daß sich insgesamt $4+4n$ Nachrichten pro Broadcast ergeben.

In [AV95] beschreiben Almeida und Veríssimo, wie auch nach einer weiteren Abschwächung der Eigenschaften des Mediums noch zeitliche Vorhersagbarkeit erreicht werden kann. Sie gehen davon aus, daß eine Möglichkeit zum zuverlässigen und rechtzeitigen Versenden von Nachrichten vorhanden ist, daß die maximale Verzögerung der Broadcast-Nachrichten Δ_{BC} aber nicht genau bekannt oder so groß ist, daß für die betrachtete Anwendung eine kleinere gewählt werden muß. Daher gehen sie davon aus, daß während der Laufzeit des Protokolls Zeitfehler auftreten können. Um die zeitliche Vorhersagbarkeit trotz der Möglichkeit von Zeitfehlern sicherzustellen, gehen sie von der Existenz eines synchronen Kanals mit kleiner Bandbreite aus, der z. B. durch die am höchsten priorisierten Nachrichten realisiert werden kann. Diesen Kanal nutzen sie, um einen Fehlererkennungsdienst zu implementieren. Dazu tauschen die Stationen auf dem synchronen Kanal periodisch Nachrichten aus, die außer einem impliziten "I am up" auch Informationen über gesendete und empfangene Nachrichten enthalten. Diese Informationen werden genutzt, um in allen Stationen Zeitfehler zu erkennen und eine entsprechende Fehlerbehandlung durchzuführen.

Durch die Abschwächung der Systemannahmen kann die rechtzeitige Auslieferung aller Nachrichten nicht mehr gewährleistet werden, da z. B. in Überlastsituationen Zeitfehler auftreten können. Dennoch erlaubt die Verwendung des Fehlererkennungsdienstes, diese Fehler rechtzeitig zu erkennen und so die rechtzeitige Einigung zwischen den Stationen sicherzustellen sowie die totale zeitliche Ordnung der Nachricht aufrechtzuerhalten. Durch die Wahl angemessener Abschätzungen für die Verzögerung von Broadcastnachrichten wird also deren Zuverlässigkeit eingeschränkt, aber die Zeitgarantien, die u. U. erheblich wichtiger sind, verbessert.

3.1.3 Broadcastprotokolle für drahtlose Medien

Da die Kommunikation in lokalen Funknetzwerken erst seit relativ kurzer Zeit ein Gegenstand der Forschung ist, wurden bisher über zuverlässige Gruppenkommunikation auf lokalen Funknetzen nur sehr wenige Arbeiten veröffentlicht. Die beiden betrachteten Arbeiten könnten, da sie nicht für den Einsatz in Echtzeitsystemen konzipiert wurden, auch in Paragraph 3.1.1 dargestellt werden. Wegen ihrer größeren Nähe zur vorliegenden Diplomarbeit, werden sie aber dennoch in einem gesonderten Paragraphen vorgestellt.

Inoue et al. [IITM98] haben ein Protokoll entwickelt, das die Zuverlässigkeit von Broadcast-Nachrichten auf drahtlosen Medien erhöht und dabei mit einer geringen Zahl von Bestätigungen auskommt. Sie betrachten Gruppen von Stationen, die einer speziellen Station, der sog. *Base Station*, zugeordnet sind, die eine ähnliche Funktion erfüllt wie der AP im Infrastrukturnetzwerk. Die Base Station ist für das Versenden von Broadcast-Nachrichten in der Gruppe verantwortlich. Die Zuverlässigkeit der Übertragung von Broadcast-Nachrichten wird durch den Einsatz von positiven und negativen Bestätigungen und Neuübertragungen von Broadcast-Nachrichten erhöht. Das spezielle Ziel der Arbeit von Inoue et al. liegt darin, die Anzahl der benötigten Bestätigungen

möglichst klein zu halten. Dazu fassen sie die Stationen einer Gruppe zu Untergruppen zusammen. In jeder Untergruppe gibt es eine Station, die als Repräsentant der Untergruppe fungiert. Nach dem Versenden einer Broadcast-Nachricht pollt die Base Station den Repräsentanten jeder Untergruppe und erwartet eine positive oder negative Bestätigung für die letzte Broadcast-Nachricht. Empfängt die Base Station von einem der Repräsentanten eine negative Bestätigung (oder keine positive Bestätigung), dann überträgt sie die Broadcast-Nachricht erneut. Die Stationen einer Untergruppe beobachten das Verhalten ihres Repräsentanten: Wenn der Repräsentant den Empfang einer Broadcast-Nachricht bestätigt, eine Station seiner Untergruppe diese aber nicht empfangen hat, dann sendet sie eine negative Bestätigung an die Base Station, die daraufhin die Broadcast-Nachricht erneut überträgt.

Inoue et al. erreichen durch den beschriebenen Mechanismus eine erhöhte Zuverlässigkeit von Broadcast-Nachrichten. Weiterhin führt ihr Ansatz zu einer Reduzierung der Bestätigungen, was sich vor allem bei größeren Gruppen auswirkt. Allerdings realisiert das von ihnen beschriebene Protokoll nicht atomare Broadcasts im oben definierten Sinne. Der Empfang einer Broadcast-Nachricht ist z. B. nicht sichergestellt, wenn eine Station sowohl die Broadcast-Nachricht als auch die anschließende Bestätigung durch den Repräsentanten nicht empfängt. Einigung zwischen den korrekten Stationen und total geordnete Auslieferung der Nachrichten sind nicht gewährleistet.

Rodrigues et al. beschreiben in [RFV95] einen hybriden Ansatz zum Ordnen von Nachrichten in gemischten, drahtlosen und drahtgebundenen, Systemen, der sowohl das symmetrische als auch das zentralisierte Ordnungsverfahren zum Einsatz bringt. Jede Station des Systems kann entweder einen aktiven oder passiven Betriebsmodus einnehmen. Stationen im passiven Betriebsmodus sind eindeutig einer aktiven Station zugeordnet, die für sie die Nachrichten nach dem in [KT91] beschriebenen Verfahren ordnet. Die aktiven Stationen ordnen untereinander die Nachrichten nach einem symmetrischen Verfahren. Jede mobile Station nimmt grundsätzlich den passiven Modus ein, so daß bei dem beschriebenen Ansatz Gruppen von mobilen Stationen einer ortsfesten, d. h. an ein drahtgebundenes Netz angeschlossenen Station, zugeordnet sind, die für sie als Sequenzer fungiert. Mobile Stationen, die ihre Position verändern, können zu einer anderen aktiven Station wechseln.

In [RFV95] werden keine grundsätzlich neuen Protokolle vorgestellt, sondern es wird dargelegt, wie man vorhandene Verfahren kombinieren kann, um den unterschiedlichen Anforderungen von drahtlosen und drahtgebundenen Netzen Rechnung zu tragen und insgesamt ein gutes Verhalten zu erreichen. Da weder der verwendetete symmetrische noch der token-basierte Ansatz die Rechtzeitigkeit sicherstellen, gilt dies auch für den vorgestellten hybriden Ansatz.

Atomare Broadcast sind ein Konzept, das schon lange und häufig in verteilten Systemen eingesetzt wird, um die Programmierung von verteilten und zuverlässigen Anwendungen erheblich zu erleichtern.

Vor allem für asynchrone Systeme wurde eine große Zahl von Broadcastprotokollen entwickelt und Arbeiten zu diesem Thema veröffentlicht. Alle diese Protokolle haben gemeinsam, daß sie nicht in der Lage sind, die Rechtzeitigkeit der Nachrichtenauslieferung sicherzustellen, weshalb sie für den betrachteten Kontext nicht angewendet werden können. Dennoch ist es lohnend, die zur Behebung von Nachrichtenverlusten eingesetzten Verfahren zu betrachten.

Andere Protokolle stellen neben der Atomarität auch die rechtzeitige Auslieferung der Nachrichten sicher. Dabei gehen Kopetz et al. und Cristian et al. auf der Ebene der Broadcastprotokolle von der Verfügbarkeit zeitgerechter und zuverlässiger Broadcast-Kommunikation aus, sie stellen also die Zuverlässigkeit der Übertragung auf unteren Schichten sicher, ein Vorgehen, das nur bei der Verfügbarkeit von relativ zuverlässigen Medien effizient ist. Beim xAMP werden die Nachrichtenverluste innerhalb des Broadcastprotokolls behandelt. Das verwendete Verfahren benötigt aber eine große Anzahl von Bestätigungsnachrichten und würde sich unter der Kommunikationsstruktur des IEEE Standards daher ineffizient Verhalten.

Über atomare Broadcasts auf Funkmedien wurden bisher nur sehr wenige Arbeiten vorgestellt, von denen meines Wissen keine rechtzeitige, atomare Broadcasts sicherstellt und der Kommunikationstruktur des Standards Rechnung trägt.

3.2 Teilnehmerprotokolle

Die folgende Darstellung verwandter Arbeiten im Bereich der Teilnehmerprotokolle ist wie das Unterkapitel über Broadcastprotokolle nach dem Gesichtspunkt unterteilt, ob die dargestellten Protokolle für den Einsatz in Echtzeitsystemen entwickelt wurden, also zeitlich vorhersagbar sind, oder ob dies nicht der Fall ist.

Da die Aufnahme neuer Stationen nicht in dem in der Diplomarbeit entwickelten Protokoll, sondern in der CP erfolgt, wird auch bei der Betrachtung der verwandten Arbeiten auf die Darstellung von Verfahren zur Aufnahme von Stationen in die Gruppe verzichtet. Dennoch muß man darauf achten, ob die betrachteten Protokolle in der Lage sind, mit a priori unbekannten Stationen zu arbeiten, oder ob sie die a priori Kenntnis der Gruppe voraussetzen.

3.2.1 Nicht-Echtzeitfähige Teilnehmerprotokolle

In verteilten Systemen kann das Verhalten von Stationen nur aufgrund der von ihnen gesendeten Nachrichten beobachtet werden. Daher kann auch der Ausfall von Stationen nur aufgrund von gesendeten bzw. ausbleibenden Nachrichten festgestellt werden. Teilnehmerprotokolle, die nicht in Echtzeitsystemen eingesetzt werden sollen, werden i. A. in asynchronen Systemen entwickelt. Weil in asynchronen Systemen das Verhalten der

Stationen und des Netzwerkes nicht zeitlich spezifiziert ist, kann in asynchronen Systemen generell nicht zwischen:

- einer ausgefallenen Station,
- einer langsamen Station,
- verlorenen Nachrichten und
- stark verzögerten Nachrichten

unterschiedenen werden. Daher ist es in asynchronen Systemen nicht möglich, Teilnehmerprotokolle zu entwickeln, die sowohl lebendig als auch genau sind. In den meisten asynchronen Systemen werden Timeouts zum unzuverlässigen Erkennen von Stationsausfällen verwendet, und damit zugunsten der Lebendigkeit auf die Genauigkeit verzichtet.

Ein Ansatz, der in mehreren Protokollen zum Bestimmen der Gruppenmitgliedschaft eingesetzt wird, entstammt dem von Garcia-Molina [Gar82] vorgestellten *Invitation*-Protokoll zum Ermitteln eines eindeutigen Koordinators. Die Protokolle arbeiten in mehreren Phasen. Eine Station, die eine Gruppenänderung vermutet, sendet in der ersten Phase an alle anderen Stationen eine Einladung, eine Gruppe zu bilden, und schlägt sich damit selbst als Koordinator vor. Jede Station antwortet mit einer Bestätigung, es sei denn, sie hat bereits eine Einladung für eine andere Gruppe erhalten oder versucht selbst, als Koordinator eine Gruppe zu bilden. In den beiden letzten Fällen entscheidet die Station aufgrund eines deterministischen Kriteriums, z. B. aufgrund von eindeutigen Gruppenbezeichnern, ob sie ihre eigene Reformation beendet und die Einladung annimmt oder ob sie ablehnt und fortfährt.

Nach dem Versenden der Einladung setzt der Koordinator einen Timeout und wartet auf die Antworten der anderen Stationen. Empfängt er vor Ablauf des Timeouts eine Einladung von einem anderen Koordinator mit höherer Priorität (z. B. höherem Gruppenbezeichner), dann bricht er die Gruppenbildung ab und schließt sich der Gruppe des anderen Koordinators an. Nach Ablauf des Timeouts bildet der Koordinator eine neue Gruppe aus allen Stationen, die auf seine Einladung geantwortet haben, und versendet die so gebildete Gruppe in einer Broadcast-Nachricht (*Join*-Message). Jede Station, die die Einladung bestätigt und sich zwischenzeitlich keiner neuen Gruppe angeschlossen hat, schließt sich beim Empfang der Broadcast-Nachricht der neuen Gruppe an.

Mehrere Protokolle ([CM84], [KT91], [CS95], [JKN96]) verwenden dieses Verfahren in abgewandelten und erweiterten Versionen, die teilweise auch weitere Phasen in den Protokollablauf einfügen. In [CM84], [KT91] und [JKN96], wo das Protokoll im Rahmen eines Broadcastprotokolls zum Einsatz gelangt, wurden solche Erweiterungen z. B. durchgeführt, um die virtuelle Synchronität sicherzustellen oder um zu gewährleisten, daß in jeder neuen Gruppe mindestens eine Station enthalten ist, die alle k-Stabilen Nachrichten erhalten hat. Weiterhin wurden Erweiterungen eingeführt, um zu vermei-

den, daß sich mehrere Gruppen gleichzeitig bilden oder um anderenfalls zu gewährleisten, daß sich alle Stationen auf die Folge von Gruppen, die vor der aktuellen Gruppe existierten, einigen.

Problematisch an dem beschriebenen Verfahren ist, daß sein Zeitverhalten nicht gut vorhersagbar ist. Dies liegt daran, das am Anfang des Protokolls mehrere Koordinatoren in Konkurrenz treten können, so daß Reformationen abgebrochen und neu gestartet werden müssen. Die Wahrscheinlichkeit für solche Erscheinungen kann zwar durch die Verwendung von zufälligen Timeouts oder eines designierten Koordinators ([JKN96]) reduziert, ihr Auftreten aber nicht gänzlich verhindert werden. Weiterhin können auch verzögerte oder verlorene Nachrichten dazu führen, daß die Reformation neu begonnen werden muß.

In [ADKM92a] wird ein anderes Verfahren vorgestellt, das auf einem Protokoll für zuverlässige, kausal geordnete Broadcast-Nachrichten (Transis, s. o.) basiert. Im Gegensatz zum Invitation-Protokoll wird hier kein Koordinator bestimmt, vielmehr ist das Protokoll vollständig symmetrisch, d. h. alle Stationen verfahren nach dem selben Algorithmus. Jede Station verwaltet den sog. *f_set*, die Menge von Stationen, die sie für ausgefallen hält. Immer wenn sich dieser *f_set* ändert, broadcastet die Station ihn kausal an alle Stationen. Stationen, die eine solche Nachricht erhalten, ändern ihren *f_set* entsprechend, broadcasten u. U. selbst eine Änderungsnachricht aber liefern die Änderungen noch nicht an den Benutzer aus. Erst wenn eine Station von jeder Station, die sie für korrekt hält, eine Änderungsnachricht empfangen hat, deren *f_set* mit ihrem eigenen übereinstimmt, liefert sie die Änderungen aus. In diesem Falle ist die Einigung unter den Stationen gewährleistet und weiterhin durch die kausale Ordnung die virtuelle Synchronität sichergestellt.

Da der *f_set* immer nur wächst, kann unter der Annahme, daß die Menge aller möglichen Stationen beschränkt ist, die Terminierung des Protokolls sichergestellt werden. Unter der Annahme bestimmter Timeouts für das Erkennen von Stationsausfällen können sogar Zeitschranken hergeleitet werden. Dennoch ist das Protokoll für den Einsatz in Echtzeitsystemen nicht geeignet, weil es

1. auf Transis aufbaut, einem Kommunikationsprotkoll, das selbst nicht für den Einsatz in Echtzeitsystemen konzipiert ist (s. Parapraph 3.1.1), und weil

2. auch bei diesem Protokoll der Ablauf und damit der zeitliche Aufwand nur schwer vorherzusagen sind, so daß eine evtl. Zeitschranke wahrscheinlich sehr groß ausfallen müßte.

Das Teilnehmerprotokoll, das im Rahmen von Totem für die Reformation des Ringes bei Stationsausfällen eingesetzt wird ([AMMAC95], [MMABL96]), arbeitet in mehreren Phasen. Die erste Phase, in der unter den Stationen Einigung über die aktuelle Gruppe erzielt wird, entspricht weitgehend dem oben vorgestellten Teilnehmerprotokoll von

Transis. Allerdings baut das in Totem eingesetzte Protokoll nicht auf einer kausalen Ordnung auf.

Die erste Phase des Protokolls beginnt, sobald eine Station einen Ausfall bemerkt. Die Station sendet in diesem Falle eine *JOIN*-Nachricht an alle Stationen, die die Menge der ihr bekannten Stationen (*proc_set*) und die Menge der nach ihrer Meinung fehlerhaften Stationen (*fail_set*) enthält. Stationen, die die Nachricht erhalten, ändern ihre Ansicht bzgl. *proc_set* und *fail_set* entsprechend und senden ihrerseits eine *JOIN*-Message an alle Stationen. Eine Station, die sich in der ersten Phase befindet, stellt fest, daß sich ein Konsens über die neue Gruppe gebildet hat, wenn sie von jeder Station aus ihrer aktuellen Gruppensicht (*proc_set* / *join_set*) eine *JOIN*-Message erhalten hat, bei der *proc_set* und *join_set* mit ihrer eigenen Ansicht übereinstimmen, oder wenn sie eine spezielle Nachricht, den *commit_token* (s. u.), erhält. Mit dem Feststellen des Konsenses ist für diese Station die erste Phase beendet. Wenn eine Station, die eine *JOIN*-Message versendet hat, bis zum Ablauf eines bestimmten Timeouts nicht in die zweite Phase gewechselt ist, dann entfernt sie alle Stationen, mit denen sie nicht im Konsens ist, aus ihrer Gruppensicht (fügt sie in *f_set* ein) und beginnt erneut die erste Phase mit dem neuen *f_set*. Jede Station, die in der ersten Phase einen Konsens festgestellt hat, prüft, ob sie die Station mit der kleinsten Id in der neuen Gruppe ist. In diesem Falle ist sie der Koordinator der zweiten Phase und sendet eine spezielle Nachricht, den *commit-token*, an die nächste Station in der neuen Gruppe. Der *commit-token* kreist zweimal um den neu formierten Ring: Die erste Rotation stellt sicher, daß alle Stationen der neuen Gruppe im Konsens sind und sammelt Informationen über die im alten Ring ausgetauschten Nachrichten; die zweite Runde dient dazu, die gesammelten Informationen in der Gruppe zu verteilen. Eine Station, die das *commit_token* zum zweiten mal erhält, wechselt in die Recovery-Phase. In dieser Phase wird der neu formierte Ring genutzt, um Nachrichten, die im alten Ring nicht von allen Stationen empfangen wurden, unter den Stationen zu verteilen. Erst wenn dies geschehen ist, kann die neue Gruppe endgültig installiert werden. So stellt das Protokoll die virtuell synchrone Auslieferung der Gruppenänderungen sicher.

Ein anderer Ansatz wird von Moser et. al. ([MMA94]) beschrieben. Sie bauen ihr Teilnehmerprotokoll auf einem Protokoll für atomare Broadcasts auf. Dieses fehlertolerante Broadcastprotokoll (Total, s. Paragraph 3.1.1) ist in der Lage, Broadcast-Nachrichten zu ordnen, solange eine bestimmte Anzahl von Stationsausfällen nicht überschritten wird. Da aber auch das Broadcastprotokoll auf Informationen über die aktuelle Gruppenmitgliedschaft angewiesen ist, werden beide Protokolle im Wechsel ausgeführt: Das Broadcastprotokoll ordnet Broadcast-Nachrichten auf der Grundlage der aktuellen Gruppenmitgliedschaft; daraufhin wird das Teilnehmerprotokoll aufgerufen, um festzustellen, ob sich aufgrund der letzten Nachricht eine Gruppenänderung ergeben hat; danach wird wieder eine Broadcast-Nachricht geordnet u. s. f.

Aufbauend auf den Sequenznummern der total geordneten Broadcast-Nachrichten werden zwei Kriterien für den Ausfall einer Station (*s*) definiert:

1. Unter den letzten *d* Broadcast-Nachrichten war keine von *s*, d. h. *s* hat sich „lange nicht mehr gemeldet“.

2. *s* sendet eine Nachricht *m* mindestens *d* Broadcasts nach einer Nachricht *m'* von einer anderen Station, aber *m* bestätigt nicht *m'*, d. h. *s* bestätigt einen Broadcast nicht, „der schon vor langer Zeit gesendet wurde“.

Aufgrund der angegebenen Kriterien kann jede Station, die Broadcast-Nachrichten empfängt und ordnet, selbständig den Ausfall anderer Stationen feststellen. Durch die total geordnete Auslieferung der Nachrichten ist sichergestellt, daß alle Stationen die gleichen Ausfälle feststellen und daß die Ausfälle bei allen Stationen an der gleichen Stelle im geordneten Nachrichtenstrom festgestellt werden. Um Ausfälle festzustellen, Einigung und virtuelle Synchronität zu erreichen, werden bei diesem vollständig symmetrischen Verfahren also keine zusätzlichen Nachrichten benötigt. Das Protokoll ist in seinem Ablauf gut überschaubar. Da aber das zu Grunde gelegte Broadcastprotokoll nicht der Eigenschaft der Rechtzeitigkeit genügt und sogar nur stochastisch terminiert, ist auch das vorgestellte Teilnehmerprotokoll nicht vorhersagbar.

Für alle vorgestellten Protokolle gilt, daß:

- sie entweder nur eine Terminierung in endlicher Zeit, ohne Angabe von Zeitschranken, gewährleisten (oder sogar nur eine stochastische Terminierung);

- oder, wenn Zeitschranken angegeben werden können, der komplexe, schwer vorhersagbare Ablauf des Protokolls nahelegt, daß diese Zeitschranken für den Einsatz in harten Echtzeitsystemen zu groß sein werden.

Es müssen statt dessen Protokolle gefunden werden, die in ihrem Verhalten weniger komplex sind und von vornherein für den Einsatz in Echtzeitsystemen entwickelt wurden.

3.2.2 Echtzeitfähige Teilnehmerprotokolle

Teilnehmerprotokolle, die in Echtzeitsystemen eingesetzt werden, müssen sicherstellen, daß die Auslieferung der Änderungsnachrichten mit einer bekannten maximalen Verzögerung nach dem Stationsausfall, d. h. rechtzeitig (s. Paragraph 2.1.3), erfolgt. Echtzeitfähige Teilnehmerprotokolle wurden daher auf der Basis von synchronen Systemen erstellt. Da in synchronen Systemen das Verhalten der Stationen zeitlich spezifiziert ist und die Kommunikation zuverlässig und rechtzeitig erfolgt, können Teilnehmerprotokolle in synchronen Systemen sowohl genau als auch lebendig sein.

Von großer Bedeutung in der Arbeit von Cristian et al. ([Cri91], [CS95], [Cri96]) sind die Definitionen der Eigenschaften von Teilnehmerprotokollen in synchronen Systemen. In [Cri91] werden aber auch mehrere Protokolle vorgestellt, welche die definierten Eigenschaften realisieren. Die Protokolle unterscheiden sich im wesentlichen in Dauer und

Nachrichtenaufwand der Fehlererkennung. Das Herbeiführen der Einigung nach der Fehlererkennung baut in allen Protokollen auf das Vorhandensein einer globalen Zeit und eines Protokolls für rechtzeitige, atomare Broadcasts mit maximaler Verzögerung Δ (z. B. [CASD85], s. Paragraph 3.1.2) auf.

Der einfachste Ansatz besteht darin, periodisch zwischen allen Stationen kurze *present*-Nachrichten als atomare Broadcasts auszutauschen. Zuverlässigkeit und Rechtzeitigkeit der atomaren Broadcasts stellen sicher, daß Δ Zeiteinheiten nach dem Versenden der Nachrichten, jede korrekte Station von jeder anderen korrekten Station eine *present*-Nachricht erhalten hat. Aufgrund dieser Nachrichten können die korrekten Stationen ihre neue Mitgliedschaft bestimmen und die entsprechenden Änderungsnachrichten ausliefern.

Da das oben beschriebene Protokoll den periodischen Austausch von n atomaren Broadcast-Nachrichten verlangt, selbst dann, wenn gar keine Stationsausfälle auftreten, schlägt Cristian zum Erkennen von Stationsausfällen zwei andere Protokolle vor, die mit n unzuverlässigen Punkt-zu-Punkt-Nachrichten auskommen. Für beide Protokolle werden die Stationen der aktuellen Gruppe in einem logischen Ring formiert. Beim *attendance-list*-Protokoll sendet eine der Stationen periodisch eine Tokennachricht, die einmal um den logischen Ring kreisen muß. Das Ausbleiben der Tokennachricht kann daher als Zeichen für einen Stationsausfall gedeutet werden. Beim *neighbour-surveillance* Protokoll sendet jede Station periodisch eine Nachricht an ihre Nachbarstation. Ein Ausbleiben der Nachricht kann von der Nachbarstation als Ausfall interpretiert werden. Nach dem Feststellen eines Stationsausfalls findet die Formierung einer neuen Gruppe wie im ersten Protokoll durch das Versenden von *present*-Nachrichten statt.

Die beschriebenen Protokolle stellen sicher, daß nach dem Auftreten von Stationsausfällen alle korrekten Stationen mit einer bekannten maximalen Verzögerung Einigung über alle ausgefallenen Stationen erzielen und die entsprechenden Änderungsnachrichten ausliefern können. Weiterhin kann die virtuell synchrone Auslieferung der Gruppenänderungen zwischen den atomaren Broadcasts erreicht werden, da das Teilnehmerprotokoll ebenfalls atomare Broadcast-Nachrichten verwendet. Andererseits sind die Verfahren durch den Einsatz von n atomaren Broacast-Nachrichten aufwendig, vor allem können Gruppenänderungen immer erst nach der maximalen Verzögerung der atomaren Broadcasts (Δ) durchgeführt werden; diese Zeit kann ganz erheblich über den normalen Nachrichtenverzögerungen liegen. Schließlich könnte durch die Integration des Teilnehmerprotokolls mit z. B. Broadcast-Protokollen der Aufwand an zusätzlichen Nachrichten für das Feststellen von Ausfällen noch weiter reduziert werden, wie dies zum Beispiel in [Kop97] der Fall ist.

In dem in [Kop97] und [KG93] vorgestellten Teilnehmerprotokoll werden a priori Informationen über die Kommunikationsstruktur genutzt, um Stationsausfälle zu erkennen. Die Kommunikation findet in TTP (time triggered protocol) nach dem TDMA-Verfahren statt (s. Paragraph 3.1.2), wobei innerhalb einer TDMA-Runde jede Station genau einmal eine Nachricht über die redundanten Kanäle versendet. Der Zeitpunkt, an

dem eine Station Zugriff auf das Medium hat, wird von den anderen Stationen als sog. Mitgliedschaftszeitpunkt aufgefaßt: Da sie wissen, daß die betreffende Station zu diesem Zeitpunkt eine Nachricht senden muß, können sie aus dem Ausbleiben dieser Nachricht schließen, daß die Station ausgefallen ist. Daher hat jede Station in jeder Runde genau einen Mitgliedschaftszeitpunkt und die maximale Dauer für das Erkennen von Stationsausfällen beträgt eine Runde.

Wenn eine Station s an einem Mitgliedschaftszeitpunkt von Station s' keine der redundanten Nachrichten von s' empfängt, dann kann das aufgrund der angenommenen Zuverlässigkeit der Nachrichtenübertragung nur zwei Gründe haben:

1. Der Sender (s') ist ausgefallen (Total- oder Auslassungsausfall) und keine Station hat die Nachricht erhalten

2. Der Empfänger (s) ist ausgefallen (Total- oder Auslassungsausfall). Andere Stationen haben die Nachricht von s' u. U. erhalten.

Ändern Stationen ihre Mitgliedschaft aufgrund von Senderausfällen, dann ist die Konsistenz der Änderungen sichergestellt, weil alle Stationen das Ausbleiben der Nachrichten feststellen und ihre Mitgliedschaft entsprechend ändern. Kommt es hingegen aufgrund von Empfängerausfällen zu einer Veränderung der Mitgliedschaft, dann kann dies zur Folge haben, daß sich Teilgruppen mit unterschiedlichen Ansichten über die Gruppenmitgliedschaft bilden: Die korrekten Stationen, die die Nachricht empfangen haben, auf der einen Seite und die Stationen, die die Nachricht nicht empfangen haben, auf der anderen Seite.

Da sich ein Teil der Stationen offenbar in einem fehlerhaften Zustand befindet, muß die Kommunikation zwischen den beiden Teilgruppen unterbunden werden, um eine Fehlerpropagierung von den ausgefallenen Stationen auf die korrekten Stationen zu vermeiden. Dies wird wie folgt erreicht: Jede Station s hat einen Mitgliedschaftsvektor ($s.MSV[]$), der für jede Station der Gruppe genau ein Bit enthält. $s.MSV[s']$ nimmt den Wert *true* an, wenn Station s Station s' für korrekt hält, sonst den Wert *false*. Jede Nachricht, die versendet wird, wird durch eine Prüfsumme gesichert, um Veränderungen der Nachricht zu erkennen. Die Prüfsumme wird aber nicht nur über den Inhalt der versendeten Nachricht gebildet, sondern über die Konkatenation von Nachrichteninhalt und Zustand des Urhebers. Entsprechend bildet der Empfänger die Prüfsumme über den Inhalt der Nachricht konkateniert mit seinem eigenen Zustand. Da der Zustand der Stationen auch ihren Mitgliedschaftsvektor enthält und eine Station eine Nachricht nur dann ausliefert, wenn die Prüfsumme stimmt, werden keine Nachrichten zwischen zwei Stationen mit unterschiedlicher Ansicht über die Gruppenmitgliedschaft ausgetauscht. Dies hat zur Folge, daß sich ab dem Zeitpunkt, von dem an die Ansichten über die Gruppenmitgliedschaft voneinander abweichen, zwei Gruppen bilden und Kommunikation nur noch innerhalb dieser Gruppen stattfinden kann. Um die dauerhafte Existenz solcher Teilgruppen zu vermeiden und die ausgefallenen Stationen einer Fehlerbehebung zu unterziehen, stellt jede Station, bevor sie eine Nachricht sendet, fest, ob sie in der letzten Runde mehr Nachrichten aufgrund einer falschen Prüfsumme verwerfen mußte, als sie

korrekt ausgeliefert hat. Ist dies der Fall, dann geht sie davon aus, daß ihre Ansicht der Mitgliedschaft von der Mehrheitsmeinung abweicht, und sie verläßt die Gruppe.

Das TTP hat für den Einsatz in Echtzeitsystemen einige entscheidende Vorteile: Da Stationsausfälle aufgrund von a priori Wissen über die Kommunikationsstruktur festgestellt werden können, müssen keine zusätzlichen Nachrichten für das Erkennen von Stationsausfällen ausgetauscht werden. Jede Station kann unabhängig von den anderen Stationen Ausfälle erkennen und ihre Mitgliedschaft anpassen, ohne die Einigung unter den Stationen zu gefährden. Es handelt sich also um einen vollständig symmetrischen Ansatz, der keinerlei zusätzliche Nachrichten benötigt. Allerdings wird durch die Ausdehnung der Prüfsumme auf den Zustand der Stationen die durch die Prüfsumme eingefügte Redundanz reduziert. Das Protokoll hat eine einfache Struktur und ist zeitlich sehr gut vorhersagbar. Stationsausfälle können nach nur einer TDMA-Runde erkannt werden. Das virtuell synchrone Erkennen von Stationsausfällen ist sichergestellt; die Stationsausfälle werde sogar fast gleichzeitig, d. h. tatsächlich synchron erkannt.

Die erreichten Vorteile gründen sich allerdings auf zwei Annahmen, die im Rahmen der Funkkommunikation sicherlich nicht gewährleistet sind:

1. Die zuverlässige Übertragung der Nachrichten kann durch ein annehmbares Maß an statischer Redundanz sichergestellt werden, so daß das Teilnehmerprotokoll aufgrund von ausbleibenden Nachrichten unmittelbar auf den Ausfall einer Station schließen kann, ohne explizit die Nachrichtenverluste zu berücksichtigen.

2. Die Rate, mit der asymmetrische Nachrichtenverluste aufgrund von Empfängerausfällen auftreten, ist sehr gering. Treten mehrere asymmetrische Nachrichtenverluste in einer Runde auf, dann kann es dazu kommen, daß sich mehr als zwei Gruppen bilden, von denen keine eine Mehrheit erreicht, was zu einem Totalausfall des Systems führen würde.

Das von Ezhilchelvan et al. in [EL90] beschriebene Protokoll arbeitet ähnlich wie das TTP. Ezhilchelvan et al. setzten ebenfalls auf einem Broadcast-Medium auf, auf das die Stationen nach dem TDMA-Verfahren zugreifen. Es wird vorausgesetzt, daß die Kommunikation zuverlässig und rechtzeitig ist; Stationen können Total- und Auslassungsausfälle erleiden. Die Kommunikation findet in Runden statt und die Stationen erkennen Stationsausfälle anhand von eingeplanten, aber nicht empfangenen Nachrichten.

Jede Station verwaltet einen Mitgliedschaftsvektor (MSV), in dem sie für jede Station eine der drei Statusinformationen:

- die Station ist nicht in der Gruppe

- die Station ist in der Gruppe, aber ihre letzte Nachricht wurde nicht empfangen

- die Station ist in der Gruppe und ihre letzte Nachricht wurde empfangen

verwaltet.

Die MSVs werden in jeder Runde zwischen allen Stationen ausgetauscht. Am Ende der Runde nutzt jede Station die MSVs und ihre Informationen über die empfangenen Nachrichten der letzten Runde, um ihren neuen MSV zu berechnen. Da alle korrekten Stationen alle Nachrichten empfangen, ist sichergestellt, daß die MSVs aller korrekten Stationen immer gleich sind. Die Änderungen der MSVs werden von allen korrekten Stationen gleichzeitig, am Ende der gleichen Runde, durchgeführt. Das Protokoll stellt weiterhin sicher, daß der Ausfall einer Station spätestens am Ende der übernächsten Runde von allen korrekten Stationen erkannt wird, sowie daß jede ausgefallene Station, die keinen Totalausfall erleidet, spätestens am Ende der nächsten Runde von ihrem Fehler erfährt.

Das Protokoll von Ezhilchelvan et al. zeichnet sich wie das TTP durch seine gute zeitliche Vorhersagbarkeit aus. Das Erkennen von Ausfällen und das Erreichen einer neuen konsistenten Sicht der Gruppe erfolgt garantiert innerhalb kurzer Fristen. Da zum Erkennen von Ausfällen a priori Informationen genutzt werden und das Versenden der MSVs via Piggy-Backing erfolgen kann, sind keine zusätzlichen Nachrichten zum Erreichen einer konsistenten und aktuellen Gruppensicht erforderlich.

Wie das TTP basiert das Protokoll auf einem statischen (a priori geplanten) TDMA-Verfahren. Dies impliziert, daß die Gruppe aus einer a priori bekannten Menge von Stationen besteht, die sich nur durch Stationsausfälle und die Fehlerbehebung von ausgefallenen Stationen verändert; die Gruppenbildung zwischen a priori unbekannten Stationen ist nicht möglich.

Für asynchrone Systeme sind eine Reihe von Teilnehmerprotokollen entwickelt worden, die aber alle den Nachteil einer schlechten zeitlichen Vorhersagbarkeit haben. Auch für synchrone Systeme wurden Teilnehmerprotokolle entwickelt. Diese sind in der Lage, die zeitliche Vorhersagbarkeit sicherzustellen. Dabei zeichnen sich vor allem das TTP und das Protokoll von Ezhilchelvan et al. durch einen geringen Overhead aus, ein Vorteil, der aber auf der Verwendung statisch vorgeplanter, kalenderbasierter Kommunikation beruht und daher die Aufnahme neuer Stationen ausschließt. Die synchronen Protokolle haben den gemeinsamen Nachteil, daß sie die Möglichkeit der zuverlässigen Kommunikation voraussetzen und Nachrichtenverluste daher nicht explizit in den Protokollen berücksichtigen.

Keines der Protokolle ist der Kommunikationsstruktur des Standards angemessen, die erfordert, daß eine zentrale Station, der AP, über die Mitgliedschaft der Gruppe entscheidet. Vielmehr gehen alle Protokolle davon aus, daß sich eine Gruppe mit vollständiger wechselseitiger Erreichbarkeit bildet, weil entweder alle Stationen an einem gemeinsamen Broadcast-Kanal angeschlossen sind oder transparente Routing-Mechanismen die Erreichbarkeit sicherstellen.

4 Das Echtzeit-Gruppenkommunikationsprotokoll

Im vorliegenden Kapitel wird das Gruppenkommunikationsprotokoll, das im Rahmen der Diplomarbeit entwickelt wurde, beschrieben. Dazu werden in Unterkapitel 4.1 zunächst einige grundlegende Entwurfsentscheidungen erläutert. Unterkapitel 4.2 beschreibt, wie das Protokoll Nachrichtenverluste erkennt und durch erneutes Übertragen der betroffenen Nachricht behebt. In Unterkapitel 4.3 wird das Konzept der wählbaren Resiliency von Nachrichten eingeführt, das es erlaubt, die Zuverlässigkeit der Nachrichtenübertragung einzuschränken, um so die maximale Verzögerung der Nachrichten zu verkürzen. Ebenso wird in diesem Kapitel der synchrone Kanal eingeführt, mit dem das Protokoll die rechtzeitige Einigung unter den Stationen sicherstellt, wenn eine Nachricht nicht an alle Stationen übertragen werden kann. In Unterkapitel 4.4 wird erläutert, wie das Protokoll die totale Ordnung der ausgelieferten Nachrichten sicherstellt. Das Teilnehmerprotokoll wird in Unterkapitel 4.5 beschrieben.

4.1 Grundlegende Konzepte

Einsatz dynamischer Redundanz

Das Tolerieren von Auslassungsausfällen des Funknetzes, d. h. das Tolerieren von Nachrichtenverlusten, ist angesichts der erheblichen Unzuverlässigkeit von Funknetzwerken einer der Hauptaspekte beim Entwurf des Protokolls. Ausfälle von Komponenten, d. h. Fehlerursachen im System, werden durch den Einsatz von Redundanz toleriert. Da bei Funknetzen der Einsatz von struktureller Redundanz, in diesem Falle also der Einsatz mehrerer Funk- oder anderer drahtloser Medien, als wenig sinnvoll erscheint, wird Zeitredundanz eingesetzt, um Nachrichtenverluste zu tolerieren. Dies bedeutet, daß die selbe Nachricht nicht nur einmal, sondern u. U. mehrmals übertragen wird. Die Annahme einer begrenzten Anzahl von Auslassungsausfällen (*OD*, s. Unterkapitel 2.2) ermöglicht dabei die Garantie begrenzter Verzögerungen, da bei einer unbegrenzten Anzahl von möglichen Verlusten Zuverlässigkeit und Rechtzeitigkeit nicht zugleich gewährleistet werden können.

Fußend auf der Annahme eines Omission-Degree *OD* gibt es prinzipiell zwei Möglichkeiten, die zuverlässige und rechtzeitige Nachrichtenübertragung zu gewährleisten (vgl. [VRR91]):

1. Durch den Einsatz *statischer Redundanz*. Dies bedeutet, daß jede Nachricht *OD+1*-mal übertragen wird, auch dann, wenn tatsächlich keine Nachrichtenverluste aufgetreten sind. Diese Art der Kommunikation, bei der Nachrichten ohne den Einsatz von Bestätigungen versendet werden, wird als Diffusion bezeichnet. (s. beispielsweise [CASD85] und [Kop97]).

2. Durch den Einsatz *dynamischer Redundanz*. Dies bedeutet, daß Nachrichten nur dann erneut übertragen werden, wenn tatsächlich Nachrichtenverluste aufgetreten

sind. Da dieser Ansatz auf dem Erkennen und Beheben von Fehlern beruht, ist ein Mechanismus zum Erkennen von Nachrichtenverlusten erforderlich. Solche Erkennungsmechanismen werden durch Rückmeldungen von den Empfängern zu den Sendern in Form von positiven und negativen Bestätigungen realisiert.

Beim Eintreten des ungünstigsten Falles wird die beste Leistungsfähigkeit von diffusionsbasierten Verfahren erreicht: Wenn *OD* Nachrichtenverluste auftreten, dann ist das *OD+1*-malige, aufeinanderfolgende Senden der Nachricht der schnellste und am wenigsten aufwendige Weg, diese zuverlässig an den Empfänger zu übertragen. Im Gegensatz dazu haben Verfahren, die dynamische Redundanz einsetzen, im ungünstigsten Falle einen größeren Aufwand, da außer dem *OD+1*-maligen Übertragen der eigentlichen Nachricht auch noch die Bestätigungen übertragen werden müssen. Geht man allerdings davon aus, daß i. A. erheblich weniger als *OD* Nachrichtenverluste auftreten, d. h. davon, daß die durchschnittliche Anzahl von Nachrichtenverlusten deutlich unter *OD* liegt, dann sind Verfahren, die auf dynamische Redundanz bauen, leistungsfähiger: Während der Einsatz statischer Redundanz dazu führt, daß immer so viel Bandbreite und Zeit benötigt wird wie im ungünstigsten Falle, benötigen Verfahren dynamischer Redundanz im Gros der Fälle nur erheblich weniger Bandbreite und Zeit. Bei Verfahren, die auf dynamische Redundanz setzen, ist der Aufwand abhängig von der tatsächlichen Anzahl von Fehlern, bei Verfahren statischer Redundanz ist der Aufwand von der maximalen Anzahl von Fehlern abhängig.

Wenn sich Stationen aus ihrem gegenseitigen Empfangsbereich entfernen oder wenn externe Störungen auftreten, dann können auf einem Funknetz lange Folgen von Fehlern (Fehlerbursts) entstehen. Daher muß man, um die Zuverlässigkeit der Übertragung sicherstellen zu können, von einer großen maximalen Anzahl von Nachrichtenverlusten ausgehen. Die tatsächliche Anzahl wird aber häufig deutlich hinter der Obergrenze zurückbleiben. Daher ist gerade auf Funknetzen der Einsatz von dynamischer Redundanz der Verwendung statischer Redundanz vorzuziehen. Der Einsatz statischer Redundanz würde in Verbindung mit einem großen *OD* und geringer Bandbreite effiziente Kommunikation nahezu unmöglich machen. Auch beim Einsatz dynamischer Redundanz muß zum Berechnen der maximalen Verzögerung Δ_{BC} der Omission-Degree *OD* herangezogen werden. Trotzdem hat es erhebliche Vorteile, die tatsächliche Verzögerung der Broadcasts und damit auch die tatsächliche Dauer der CFP zu reduzieren, da die ganze Zeit (und entsprechend Bandbreite), die für die CFP eingeplant, aber nicht verwendet wird, aufgrund der variablen Länge der Phasen der CP zugeschlagen und somit für den Austausch von weichen Echtzeit- und Nicht-Echtzeit-Nachrichten verwendet werden kann.

Ein kritischer Punkt bei Verfahren dynamischer Redundanz besteht in der Realisierung des Erkennungsmechanismus: Wenn der durch diesen Mechanismus erzeugte Aufwand zu groß wird, dann werden die gegenüber der statischen Redundanz erzielten Vorteile wieder hinfällig. Dies gilt insbesondere für Funknetze mit ihrer relativ geringen Bandbreite. Die Konzeption einer effizienten Fehlererkennung ist daher beim Entwurf des Protokolls von großer Bedeutung.

Implizite Informationen und Piggy-Backing

Beim Entwurf des Protokolls muß in besonderem Maße darauf geachtet werden, daß der Mehraufwand (Overhead), der durch das Protokoll entsteht, möglichst gering bleibt. Zum einen ist die zur Verfügung stehende Rohbandbreite mit 1 MBit/sek. bis 2 MBit/sek. relativ gering und der Verlust weiterer Bandbreite durch Protokollnachrichten daher besonders ungünstig. Zum anderen dürfen Stationen in der CFP nur dann Nachrichten senden, wenn sie zuvor durch den AP gepollt wurden. Daher wird durch jede weitere Protokollnachricht, die von einer Station gesendet werden muß, auch eine weitere Polling-Nachricht notwendig.

Mehraufwand für die Protokolle kann zum einen beim Erkennen von Nachrichtenverlusten durch Bestätigungen entstehen. Zum anderen können Nachrichten, die zum Gewinnen und Versenden von Informationen über die Gruppenmitgliedschaft benötigt werden, einen nicht unbeträchtlichen Aufwand verursachen.

Daher sollen im Rahmen des Protokolls nach Möglichkeit implizite Informationen verwendet werden. Dies bedeutet, daß Stationen allein aus dem Erhalt oder Ausbleiben von Nachrichten, die ohnehin gesendet werden müssen, Informationen gewinnen können, ohne daß zusätzliche Nachrichten nötig werden. Beispiele für die Verwendung solcher Informationen finden sich bei Broadcastprotokollen nach dem zentralisierten Ansatz (s. Paragraph 3.1.1), bei dem der Broadcast einer Nachricht durch die Tokenstation als implizite Bestätigung für den Erhalt der entsprechenden Punkt-zu-Punkt-Nachricht gewertet werden kann. Ein anderes Beispiel findet sich in den Teilnehmerprotokollen von Kopetz et al. (TTP) und Ezhilchelvan et al. (s. Paragraph 3.2.2), die das Ausbleiben von eingeplanten Nachrichten als Indikation für den Ausfall einer Station benutzen.

Eine weitere Möglichkeit, die Anzahl von Protokollnachrichten zu reduzieren, besteht im *Piggy-Backing*. Hierbei werden Protokollinformationen in Nachrichten übermittelt, die zum Transport von Nutzdaten ohnehin hätten versendet werden müssen. Dies reduziert zwar nicht die Menge der übermittelten Protokollinformationen, spart aber dennoch Zeit und Bandbreite ein, da weniger Nachrichten

- weniger Header von Protokollen auf unteren Ebenen und weniger Bearbeitungszeit in Protokollen auf unteren Ebenen (z. B. MAC-Header nach dem Standard) und

- weniger Zeit, die das Medium zwischen dem Übertragen von Nachrichten ungenutzt bleibt, und

- weniger Polling-Nachrichten

bedeuten.

Integration von Teilnehmerprotokoll und Broadcastprotokoll

Um die Effizienz der Gruppenkommunikation zu steigern, werden Teilnehmer- und Broadcastprotokoll zu einem Protokoll integriert. Dies bedeutet, daß nicht jedes Proto-

koll seine eigenen Nachrichten verwendet, die dem anderen Protokoll weitgehend oder völlig verborgen bleiben. Vielmehr werden Teilnehmer- und Broadcastprotokoll in einem gemeinsamen Protokollautomaten realisiert und verarbeiten einen gemeinsamen Strom von eingehenden Nachrichten.

Diese Integration erlaubt es, Nachrichtenverluste, die im Broadcastprotokoll auftreten, im Teilnehmerprotokoll zum Erkennen von Stationsausfällen zu verwenden. Das Teilnehmerprotokoll benötigt auf diese Weise keine eigenen „Are you up“ oder „I am alive“ Nachrichten, um festzustellen, ob eine Station ausgefallen ist. Des weiteren kann das Teilnehmerprotokoll Nachrichten des Broadcastprotokolls verwenden, um via Piggy-Backing Informationen über die Gruppe an die Stationen zu verteilen. So können insgesamt deutliche Effizienzsteigerungen erzielt werden.

Optimierung für maximale Last

Das für die CFP zu entwickelnde Broadcastprotokoll ist allein für das Versenden von Echtzeit-Nachrichten gedacht. Sporadische Nachrichten, weiche Echtzeit- und Nicht-Echtzeit-Nachrichten werden in der CP versendet. Dabei soll die der CP zur Verfügung stehende Zeit bei gegebener Anzahl von Echtzeit-Nachrichten möglichst lang sein, um möglichst viele der eben genannten Nachrichten transportieren zu können. Da also alle Stationen die CFP nutzen, um in möglichst kurzer Zeit alle ihre Echtzeit-Nachrichten zu übermitteln, wird beim Entwurf des Protokolls die Annahme zugrunde gelegt, daß in der CFP i. A. alle Stationen Echtzeit-Nachrichten zu übermitteln haben und daß diese weiterhin mit relativ kurzen Zwischenankunftszeiten bei den Stationen für die Übertragung anstehen.

Daher steht bei der Entwicklung des Protokolls nicht die Minimierung der Verzögerung einzelner Nachrichten im Vordergrund. Vielmehr soll die Zeit minimiert werden, die dazu nötig ist, mehrere Broadcasts von jeder Station zu übertragen. Wenn man die Bearbeitungsdauer einzelner Nachrichten minimieren will, dann ist es am günstigsten, wenn nach dem Versenden einer Nachricht alle Stationen möglichst schnell eine Bestätigungsnachricht senden. So kann mit der Bearbeitung der selben Nachricht fortgefahren und diese ggf. erneut gesendet werden. Da aber in der Zeit, in der diese eine Nachricht bearbeitet wird, keine andere Station eine Nachricht versenden kann, ist dieses Verfahren nicht dazu geeignet, die Dauer für die Übertragung aller Nachrichten zu reduzieren. Da weiterhin auch die Zeit, die eine Station warten muß, bis sie mit dem Übertragen ihrer Nachricht beginnen kann, neben der eigentlichen Bearbeitungszeit zu der Verzögerung der Nachricht gerechnet werden muß, wird ein Ansatz, der berücksichtigt, daß u. U. alle Stationen gleichzeitig Nachrichten übertragen wollen, auch die Verzögerung der einzelnen Nachrichten reduzieren.

Minimale Verzögerung statt Laufzeitvarianz und Gleichmäßigkeit

Kleine Werte für Laufzeitvarianz und Gleichmäßigkeit können realisiert werden, indem die Broadcast-Nachrichten von allen Stationen gleichzeitig, an bestimmten Zeitpunkten auf einer globalen Zeitachse ausgeliefert werden. Bei der Bestimmung der Zeitpunkte müssen Annahmen über den ungünstigsten Fall zugrunde gelegt werden, da nur so si-

chergestellt ist, daß alle Stationen die Nachrichten zu den betreffenden Zeitpunkten auch ausliefern können. Meistens wird aber der Fall eintreten, daß die Broadcast-Nachrichten deutlich vor den berechneten Zeitpunkten ausgeliefert werden können. Dies kann beispielsweise vorkommen, wenn weniger Nachrichtenverluste als angenommen auftreten. Bei der Entwicklung des Protokolls wird die frühzeitige Auslieferung der Nachrichten Laufzeitvarianz und Gleichmäßigkeit vorgezogen. Dies ist zum einen von Vorteil, weil hierdurch die Länge der CFP zugunsten der CP verkürzt werden kann. Zum anderen ist nicht nur die maximale Verzögerung von Nachrichten von Bedeutung, sondern auch die tatsächliche. Wenn z. B. Broadcasts eingesetzt werden, um Wechsel zwischen verschiedenen Team-Strategien oder -Verhalten zu realisieren, dann kann es von Vorteil sein, diese Wechsel so schnell wie möglich durchzuführen.

Tradeoffs werden durch den Benutzer entschieden

Bei der Entwicklung von Protokollen wird man vor verschiedene Tradeoff-Entscheidungen gestellt. Diese Tradeoffs resultieren daraus, daß die Realisierung weitergehender Quality-of-Service Eigenschaften, wie z. B. Ordnung oder Zuverlässigkeit, i. A. auch mit größeren Kosten in Form von Zeit- und Nachrichtenaufwand verbunden ist. Solche Tradeoff-Entscheidungen können durch den Anwender viel besser getroffen werden als durch den Protokollentwickler, da nur der Anwender entscheiden kann, was ihm die Realisierung bestimmter Eigenschaften wert ist. In diesem Sinne bieten verschiedene existierende Systeme dem Benutzer die Wahl zwischen verschieden starken Service-Eigenschaften, die mit unterschiedlichen Kosten verbunden sind: In ISIS ([BJ87], [BSS91]) gibt es z. B. kausale und atomare Broadcasts; xAMP ([RV92]) bietet verschiedene Ordnungs-, Einigungs- und Zeiteigenschaften an.

Durch die hohe Unzuverlässigkeit des Funknetzes ist für die vorliegende Arbeit die Tradeoff Entscheidung zwischen Zuverlässigkeit und maximaler Verzögerung von besonderer Bedeutung. Wenn angesichts einer großen maximalen Anzahl von Nachrichtenverlusten die Zuverlässigkeit dennoch gewährleistet sein soll, muß auch eine große Anzahl von Neuübertragungen eingeplant werden, was dazu führt, daß die maximale Verzögerung der Nachrichten sehr groß wird und damit die Anzahl der pro Zeiteinheit einplanbaren Nachrichten (Echtzeitdurchsatz) sehr klein. Da es andererseits Anwendungen gibt, die u. U. gelegentliche Nachrichtenverluste tolerieren können, ist es sinnvoll, die Entscheidung über das notwendige Maß an Zuverlässigkeit in die Hände des Anwendungsentwicklers zu legen.

Zentrale Koordinierung der Broadcasts durch den AP

Nach dem IEEE 802.11 Standard kommt der AP als zentraler Koordinator für die CFP zum Einsatz. Dabei kann der AP alle Stationen koordinieren, die sich in seinem Sende-/-Empfangsbereich befinden. Der daraus für den Standard resultierende und in der Diplomarbeit übernommene Gruppenbegriff legt es nahe, den AP als Router für die Nachrichten einzusetzen. Da jedes Gruppenmitglied nach Definition im Sendebereich des AP liegt, ist durch den Einsatz des AP als Router sichergestellt, daß jede Station alle Broadcast-Nachrichten empfängt, solange keine Auslassungsausfälle des Funknetzes auftreten. In Situationen, in denen vollständige Erreichbarkeit zwischen alle Gruppenmitglie-

dern besteht, impliziert dieser Ansatz einen Mehraufwand. Da sich die Erreichbarkeit zwischen den Gruppenmitgliedern aufgrund der Mobilität allerdings äußerst dynamisch gestaltet, würde ein dynamisches Routing-Verfahren, das die Erreichbarkeit immer wieder neu feststellen muß, einen größeren Mehraufwand mit sich bringen.

Da der AP nach dem Standard als zentraler Koordinator der CFP zum Einsatz kommt, kann die Zuverlässigkeit der Kommunikation in der CFP nur sichergestellt werden, solange der AP nicht ausfällt. Daher impliziert die Kommunikationsstruktur der CFP die Annahme eines stabilen AP, d. h. die Annahme, daß der AP nie ausfällt (s. Unterkapitel 2.2). Wenn man diese Annahme trifft, dann ist es sinnvoll, aus der Stabilität des AP den größtmöglichen Nutzen zu ziehen. Dies erreicht man ebenfalls, indem man den AP als zentralen Koordinator verwendet: Hierdurch sind alle Nachrichten, die der AP empfangen hat, vom Empfangszeitpunkt an stabil, d. h. sie können nicht mehr verloren gehen. Weiterhin kommt der AP auch als zentrale Station für das Ordnen der Nachrichten zum Einsatz, so daß gewährleistet ist, daß das Ordnen der Nachrichten nie unterbrochen werden muß und daß immer eine Station weiß, wie weit das Ordnen der Nachrichten fortgeschritten ist.

Zentralisiertes Teilnehmerprotokoll

Der Gruppenbegriff des Gruppenkommunikationsprotokolls muß mit dem des Standards übereinstimmen, weil es nicht sinnvoll ist, Stationen in der Gruppe zu haben, die der AP nicht pollen kann. Die Anmeldung erfolgt, wie oben erläutert, in der CP zentral nach dem im Standard festgelegten Verfahren. Daher soll auch in der CFP die Entscheidung über die Gruppenmitgliedschaft zentral durch den AP getroffen werden. Dies hat den weiteren Vorteil, daß mit dem AP der Koordinator für Gruppenänderungen immer zur Verfügung steht und nicht erst unter den korrekten Stationen ermittelt werden muß. Vor allem hat der AP, weil er auch die Broadcast-Nachrichten koordiniert, die aktuellsten Informationen über die Broadcast-Nachrichten, so daß diese Informationen beim Auftreten von Ausfällen nicht mehr von den übrigen Stationen eingesammelt werden müssen.

Geringe Komplexität des Teilnehmerprotokolls

Das Teilnehmerprotokoll soll das Ordnen und Versenden der Broadcast-Nachrichten nicht unterbrechen. Vielmehr sollten beide Protokolle weitgehend nebenläufig arbeiten, damit die maximale Verzögerung der Broadcast-Nachrichten nicht durch die Verzögerungen des Teilnehmerprotokolls erhöht werden. Der Ablauf des Teilnehmerprotokolls sollte möglichst überschaubar sein, um kurze Verzögerungen beim Erkennen von Stationsausfällen zu garantieren und den Einfluß des Teilnehmerprotokolls auf das Broadcastprotokoll bestimmen zu können. Damit Stationsausfälle mit kleinen Verzögerungen erkannt werden können, ist es notwendig, daß die Stationen in kurzen Abständen Lebenszeichen in Form von Nachrichten an den AP senden. Bei der Entwicklung des Teilnehmerprotokolls ist darauf zu achten, daß das Versenden dieser Lebenszeichen nur wenig Bandbreite benötigt.

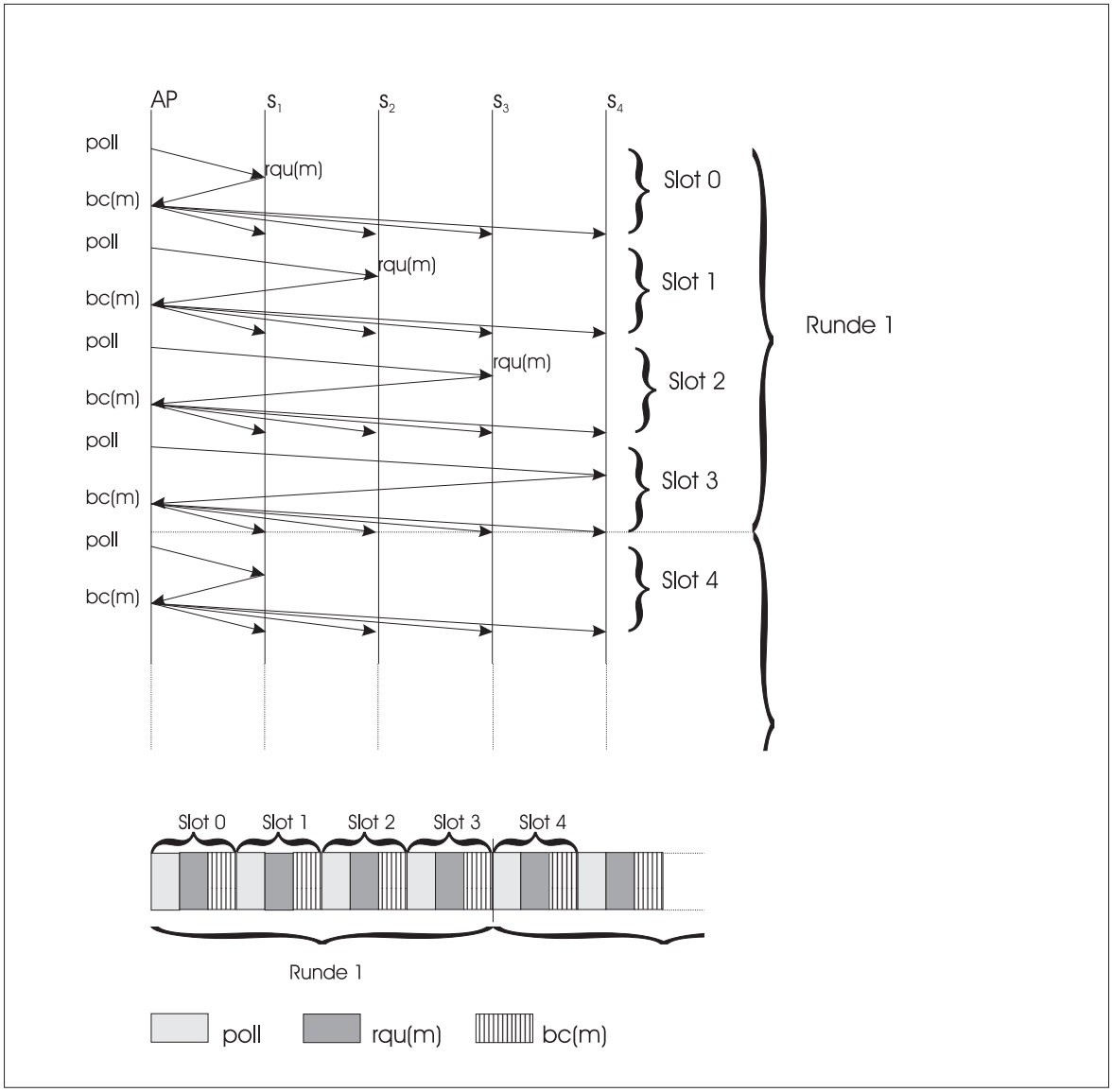


Abbildung 4.1 Struktur der Runde (Der zeitliche Abstand zwischen dem Senden einer Broadcast-Nachricht und der folgenden Polling-Nachrichten wurde aus Gründen der Übersichtlichkeit vergrößert).

4.2 Grundlegender Ablauf des Protokolls

In diesem Unterkapitel wird erläutert, wie das Protokoll die Zuverlässigkeit der Nachrichtenübertragung angesichts von Nachrichtenverlusten sicherstellt. Wie Ordnung realisiert und Stationsausfälle gehandhabt werden, wird erst in den folgenden Unterkapiteln erläutert. Daher sei für dieses Unterkapitel angenommen, daß keine Stationsausfälle auftreten und die Ordnung der Broadcast-Nachrichten nicht von Bedeutung ist.

4.2.1 Die Struktur des zeitlichen Ablaufs

Das grundlegende Problem bei der Konzeption eines Echtzeit-Gruppenkommunikationsprotokolls für ein lokales Funknetz besteht darin,

1. die Zuverlässigkeit angesichts einer großen Anzahl von Nachrichtenverlusten zu garantieren,

2. gleichzeitig die Rechtzeitigkeit der Übertragung sicherzustellen und dabei

3. möglichst effizient die geringe Bandbreite und die Kommunikationsstruktur des Standards zu nutzen.

Im vorhergehenden Unterkapitel wurde erläutert, daß sich dieses Problem durch den Einsatz von dynamischer Redundanz lösen läßt, wobei die Annahme einer begrenzten Anzahl von Nachrichtenverlusten sicherstellt, daß eine Nachricht höchstens $OD+1$-mal übertragen werden muß, so daß auch die Rechtzeitigkeit gewährleistet bleibt. Es wurde aber auch darauf hingewiesen, daß besonderer Wert auf den Entwurf des Fehlererkennungsmechanismus gelegt werden muß, damit die Vorteile dynamischer Redundanz nicht durch ein hohes Aufkommen an Bestätigungen gefährdet werden.

Implizite Bestätigungen und Piggy-Backing sind Methoden, die es erlauben, Bestätigungsnachrichten einzusparen und damit die Fehlererkennung effizient zu implementieren. Weitere Effizienzgewinne können erzielt werden, indem Nachrichten nicht nur Bestätigungen für einzelne sondern mehrere Nachrichten beinhalten.

Um den Einsatz dieser Konzepte möglichst weitgehend zu ermöglichen, ist die Kommunikation in der CFP in Runden gegliedert: In jeder Runde pollt der AP jede Station genau einmal, wobei die Reihenfolge, in der die Stationen gepollt werden, in jeder Runde die gleiche ist. Diese Struktur stellt eine spezielle Realisation der Polling-Liste dar, die im Standard absichtlich nur grob spezifiziert ist. Wenn man die Gruppe als ein Feld von Stationen auffaßt, dann hat jede Station (s) in der Gruppe (g) eine eindeutige Position, die im Folgenden als *id* der Station bezeichnet wird und diese eindeutig kennzeichnet: $g[s.id]^{12} = s$ (wobei $g[i]$ diejenige Station bezeichnet, die sich an Position i der Gruppe g befindet, beginnend mit Position *0*). Innerhalb einer Runde werden die Stationen in der Reihenfolge ihrer Ids gepollt, d. h. der Nachfolger von Station s ist die Station $g[(s.id + 1) \bmod g.size]$, wobei *g.size* die Anzahl der Stationen in Gruppe g bezeichnet. Der AP numeriert die Runden fortlaufend durch: Er beginnt in Runde *1* und inkrementiert den Rundenzähler (*round*) jedesmal, wenn er alle Stationen der Gruppe einmal gepollt hat.

---

[12] Die Darstellung *s.id* besagt, daß *id* ein Attribut von Station *s* ist, sie soll nicht darauf hindeuten, daß *id* ein Feld in einer Datenstruktur *s* ist.

Abbildung 4.1 stellt die Struktur einer Runde in einem Raum-Zeit-Diagramm dar. In diesem Diagramm bezeichnen die senkrechten Linien die einzelnen Stationen, wobei die Zeit von oben nach unten fortschreitet. Nachrichten werden durch Pfeile dargestellt, deren Anfangspunkt Sender und Sendezeitpunkt und deren Spitze Empfänger und Empfangszeitpunkt kennzeichnet. Gehen mehrere Pfeile von einem Punkt aus, so bezeichnet dies eine Broadcast-Nachricht.

Wie in Abbildung 4.1 dargestellt, gestaltet sich der Verlauf einer Runde wie folgt: Der AP pollt die erste Station (Id 0), woraufhin diese Station eine Request-Nachricht an den AP überträgt. Nach dem Erhalt der Request-Nachricht broadcastet der AP eine Nachricht der zuvor gepollten Station. Diese Folge von drei Nachrichten wird im Folgenden als Slot bezeichnet, wobei die Station, die der AP pollt und von der er in dem Slot ggf. eine Nachricht broadcastet, als Eigentümer dieses Slots bezeichnet wird. Nach dem Broadcast der Nachricht pollt der AP die nächste Station der Gruppe. Auch diese Station überträgt eine Request-Nachricht an den AP und der AP broadcastet daraufhin eine Nachricht dieser Station. Nach diesem Slot pollt der AP die nächste Station u. s. f. Wenn der Slot der letzten Station in der Gruppe beendet ist, dann fängt der AP mit dem Pollen wieder bei der ersten Station der Gruppe an.

In Abbildung 4.1 ist noch eine weitere Darstellung der Rundenstruktur zu finden. Diese benötigt weniger Raum und beschreibt die Struktur der Kommunikation intuitiver, so daß diese Art der Darstellung im Folgenden häufig Verwendung finden wird. In dieser Darstellung werden die Nachrichten durch Rechtecke dargestellt. Die unterschiedlichen Arten von Nachrichten (Polling-, Request- und Broadcast-Nachrichten) werden durch die Füllungen der Rechtecke unterschieden. Das Ende einer Runde ist durch eine vertikale Linie gekennzeichnet.

Nun soll die Struktur eines einzelnen Slots näher betrachtet werden: Zunächst sendet der AP eine Polling-Nachricht an Station *s* (*poll(r)*, s. Abb. 4.2). Nachdem *s* die Polling-Nachricht empfangen hat, sendet *s* eine Request-Nachricht an den AP (*rqu(ack*[]*, sl, res, sdu)*, s. Abb. 4.2). Mit dieser Request-Nachricht kann die Station eine Nachricht *m*, die der Benutzer an das Protokoll übergeben hat, an den AP übertragen (wie in Kapitel 2.1 gesagt, werden solche Nachrichten durch das *broadcast*-Primitiv an das Protokoll übergeben und im Unterschied zu den Protokoll-Nachrichten (PDU) als SDU bezeichnet). Damit stellt die Station die Anfrage an den AP, die SDU *m* an die Gruppe zu broadcasten und dabei die geforderten Eigenschaften atomarer, rechtzeitiger Broadcasts sicherzustellen. Wenn die Station keine SDU zum Übertragen an den AP hat, sendet sie eine leere Request-Nachricht an den AP, die nur das *ack*-Feld enthält. Nach dem Erhalt der Request-Nachricht broadcastet der AP eine SDU, die er von *s* empfangen hat (*bc(sg, syncCh*[]*, sl, m'),* s. Abb. 4.2). Dieses muß nicht die SDU sein, die er in der unmittelbar vorhergehenden Request-Nachricht empfangen hat. Der AP bezeichnet jede Broadcast-Nachricht eindeutig mit einer globalen Sequenznummer (*sg*). Der erste Broadcast des AP trägt die Sequenznummer 0 und der AP erhöht die globale Sequenznummer vor jedem Broadcast, so daß die Broadcast-Nachrichten des AP fortlaufend durchnumeriert

sind. Da in jedem Slot genau eine Broadcast-Nachricht gesendet wird, können die globalen Sequenznummern auch als fortlaufende Numerierung der Slots aufgefaßt werden.

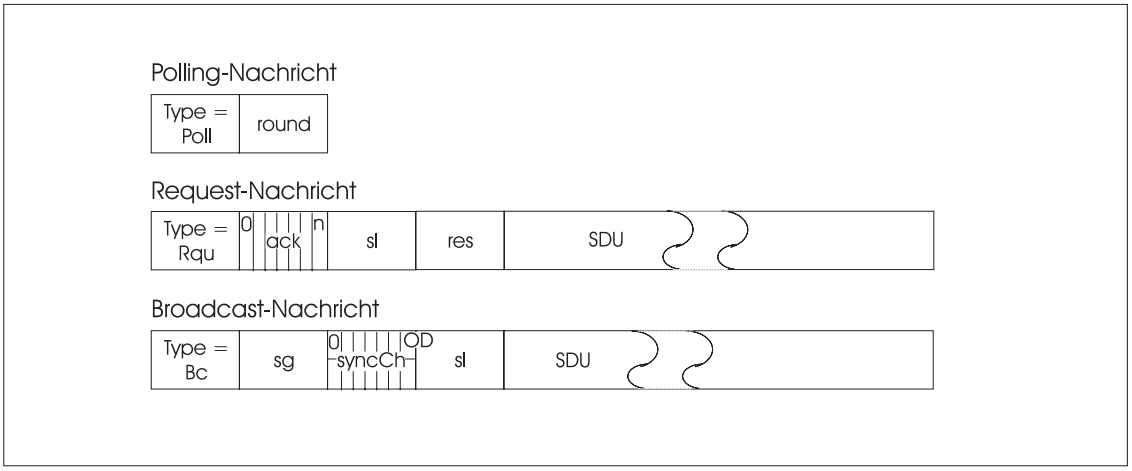


Abbildung 4.2 Aufbau der Nachrichten.

Da auf dem Funknetzwerk Nachrichtenverluste möglich sind, können Polling-, Request- und Broadcast-Nachrichten verloren gehen. Diese Nachrichtenverluste dürfen aber nicht dazu führen, daß das Protokoll verklemmt und nicht mehr weiter arbeitet. Aufgrund der angenommenen Obergrenze für die Verzögerung einer Nachricht (t_m) und der Annahme, daß die lokalen Bearbeitungszeiten vernachlässigbar sind, weiß der AP, daß spätestens $2 \times t_m$ Zeiteinheiten nach dem Übertragen der Polling-Nachricht die Request-Nachricht empfangen worden sein muß. Daher setzt der AP nach dem Übertragen der Polling-Nachricht ein Timeout der Länge $2 \times t_m$. Wenn dieses Timeout abgelaufen ist, dann geht der AP davon aus, daß entweder die Polling- oder die Request-Nachricht verloren gegangen ist. Er überträgt dann eine alte SDU von der gepollten Station (wann dies der Fall sein kann wird unten beschrieben) oder führt einen leeren Broadcast durch, der nur die Protokoll-Informationen *sg* und *syncCh* enthält. So ist sichergestellt, daß der AP spätestens $2 \times t_m$ Zeiteinheiten nach dem Übertragen der Polling-Nachricht mit dem Übertragen des Broadcasts beginnt, unabhängig davon, ob eine Nachricht verloren gegangen ist oder nicht. Der Verlust von Broadcast-Nachrichten ist in Hinsicht auf den Stillstand des Protokolls unkritisch; der AP kümmert sich zunächst nicht darum, ob der Broadcast an allen Stationen angekommen ist, und pollt die nächste Station.

Durch die bisher eingeführte Kommunikationsstruktur wurden die folgenden Ziele erreicht:

- Da nur der AP SDUs als Broadcast-Nachrichten sendet, kommt der AP für alle SDUs als Router zum Einsatz. Da der AP alle Stationen erreichen kann, ist damit die Erreichbarkeit unter den Stationen für das Protokoll unerheblich.

- Jede SDU, die von einer Station empfangen wurde, wurde vom AP gebroadcastet. Da die SDU im AP gespeichert wird und der AP nicht ausfällt, kann der AP sicher-

stellen, daß auch alle anderen Stationen die Nachricht erhalten, solange sie nicht ausfallen.

- Da jede Station sich regelmäßig in Form einer Request-Nachricht beim AP melden muß, können Ausfälle von Stationen erkannt werden, ohne zusätzliche Nachrichten einzuführen.

- Die Struktur erlaubt es, daß Bestätigungen implizit und via Piggy-Backing versendet werden können. Wie dies geschieht, wird im Folgenden erläutert.

Die Übertragung einer SDU *m* von einer Station *s* an die Gruppe kann in zwei Abschnitte unterteilt werden:

1. Die Übertragung der SDU *m* von *s* an den AP. Dies geschieht in den Request-Nachrichten.

2. Die Übertragung der SDU *m* vom AP an die Gruppe. Dies geschieht in den Broadcast-Nachrichten.

Auf beiden Abschnitten können Nachrichtenverluste auftreten und Neuübertragungen notwendig machen. Auf dem ersten Abschnitt können Polling- und Request-Nachrichten verloren gehen; im zweiten Abschnitt kann es zum Verlust von Broadcast-Nachrichten kommen. Daher müssen auf beiden Abschnitten Bestätigungen eingesetzt werden, um Nachrichtenverluste zu erkennen.

4.2.2 Die Übertragung der Benutzernachrichten (SDUs) an den AP

Auf dem ersten Abschnitt werden Nachrichtenverluste durch implizite Bestätigungen erkannt. Genauer ausgedrückt wird mit Hilfe der Bestätigungen erkannt, daß eine Request-Nachricht für eine SDU *m* am AP angekommen ist. Solange diese Bestätigung ausbleibt, vermutet der Urheber von *m* (*m.orig*), daß die Request-Nachricht verloren gegangen ist. Als implizite Bestätigung dafür, daß der AP die SDU *m* empfangen hat, dient dem Urheber von *m* der Empfang einer Broadcast-Nachricht, die *m* enthält. Dies bedeutet: Wenn der Urheber von *m* nach dem Senden einer Request-Nachricht mit Inhalt *m* keine Broadcast-Nachricht mit Inhalt *m* empfängt, dann geht er davon aus, daß die Request-Nachricht verloren gegangen ist und wird *m* erneut in einer Request-Nachricht senden, wenn er das nächste Mal gepollt wird. So kann die Zuverlässigkeit bei der Übertragung der Request-Nachrichten gesichert werden, ohne daß explizite Bestätigungsnachrichten notwendig sind.

Damit der Urheber einer SDU *m* feststellen kann, daß diese SDU Inhalt einer Broadcast-Nachricht ist, ordnet jede Station den SDUs lokal eindeutige, aufsteigende Sequenznummern zu (*m.sl*). Diese Sequenznummern werden sowohl in den Broadcast- wie auch in den Request-Nachrichten zusammen mit *m* übertragen (s. Abb. 4.2). Die lokale Sequenznummer einer SDU ergibt zusammen mit ihrem Urheber eine eindeutige Kenn-

zeichnung dieser SDU ($m.sl = m'.sl \wedge m.orig = m'.orig \Leftrightarrow m = m'$). Wenn eine Station eine Broadcast-Nachricht *bc(sg, syncCh*[*], sl, m)* empfängt, dann gibt *sl* die lokale Sequenznummer von *m* an. Aber auch den Urheber von *m* kann der Empfänger der Broadcast-Nachricht einfach bestimmen. Aufgrund der zeitlichen Struktur der Kommunikation ist der Urheber von *m* die Station, die Eigentümer des Slots ist, in dem *m* empfangen wird. Der Eigentümer eines Slots kann auf einfache Weise durch den Empfänger der Broadcast-Nachricht dieses Slots bestimmt werden, wie in Unterkapitel 4.5 erläutert wird. Solange keine Stationsausfälle auftreten, kann dies z. B. durch *g*[*sg mod g.size*] geschehen.

Wenn der Urheber einer SDU *m* nach dem Senden der Request-Nachricht keine Broadcast-Nachricht empfängt, die *m* enthält, so wird er *m* in der nächsten Runde erneut an den AP übertragen. Er wird dies aber in maximal *OD+1* Runden tun, weil danach zum einen gewährleistet ist, daß der AP die SDU erhalten hat, und zum anderen andauerndes Übertragen der Nachricht durch ihren Urheber dazu führt, daß eine maximale Verzögerung für *m* nicht gewährleistet werden kann.

Damit die Stationen feststellen können, ab wann sie eine SDU nicht an den AP übertragen dürfen, teilt der AP den Stationen in den Polling-Nachrichten die aktuelle Rundennummer mit. Dazu setzt er das *round*-Feld der Polling-Nachricht (s. Abb. 4.2) immer auf den aktuellen Stand des Rundenzählers *round*. Die Stationen bestimmen die Runde, in der eine SDU zum letzten Mal gesendet werden darf (*m.lrts*), direkt, wenn diese ihnen vom Benutzer übergeben wird. Dazu addieren die Stationen die aktuelle Rundennummer, d. h. den Wert von *round* aus der letzten Polling-Nachricht, und *OD+1*. Eine Station, die gepollt wird, sendet eine SDU *m* nur dann an den AP, wenn *m.lrts* kleiner gleich der aktuellen Rundennummer ist. So ist sichergestellt, daß die SDU zuverlässig, aber nicht zu spät, an den AP übertragen wird.

Damit gibt es für eine Station zwei Möglichkeiten, auf die Polling-Nachricht eines AP mit einer Request-Nachricht zu antworten:

1. Die Station hat eine SDU, die sie noch nicht in einer Broadcast-Nachricht des AP empfangen hat und die nochmals übertragen werden darf: dann sendet sie diese an den AP.

2. Die Station überträgt eine leere Request-Nachricht, die nur das *ack*-Feld enthält.

4.2.3 Die Übertragung der Benutzernachrichten (SDUs) an die Gruppe

Das bis hierher vorgestellte Verfahren stellt sicher, daß SDUs zuverlässig und rechtzeitig an den AP übertragen werden können. Dabei werden keine zusätzlichen Nachrichten für die Bestätigungen benötigt. Im Folgenden soll erläutert werden, wie der AP die SDUs, die er empfangen hat, zuverlässig und rechtzeitig an alle Stationen überträgt.

Wenn der AP eine SDU *m* an die Gruppe übertragen möchte, dann sendet er eine Broadcast-Nachricht, die *m* enthält, an alle Stationen. Danach muß er feststellen, welche Stationen diese Broadcast-Nachricht empfangen haben, um ggf. *m* in der nächsten Runde erneut in einer Broadcast-Nachricht zu übertragen. Um dies zu ermöglichen, benötigt der AP von jeder Station eine Bestätigung für die betreffende Broadcast-Nachricht. Die Stationen nutzen ihre Request-Nachrichten, um jede Broadcast-Nachricht der Vorrunde positiv oder negativ zu bestätigen. Auf diese Weise wird der Aufwand vermieden, der verursacht würde, wenn jede Station jede Broadcast-Nachricht mit einer speziellen Nachricht bestätigen würde. In den Request-Nachrichten ist ein Bitfeld der Länge *n* enthalten (das *ack*-Feld, s. Abb 4.2), das es den Stationen erlaubt, für jede der vorhergehenden *g.size* Broadcast-Nachrichten eine positive bzw. negative Bestätigung abzugeben, indem sie das entsprechende Bit im *ack*-Feld auf *true* bzw. *false* setzen. So kann der AP eine Runde nach dem Versenden einer Broadcast-Nachricht feststellen, ob diese von allen Stationen empfangen wurde, und ggf. den Inhalt der Broadcast-Nachricht erneut übertragen, ohne daß explizite Bestätigungen notwendig wären. Vielmehr werden Nachrichten, die ohnehin an den AP hätten gesendet werden müssen, lediglich um ein Bitfeld erweitert.

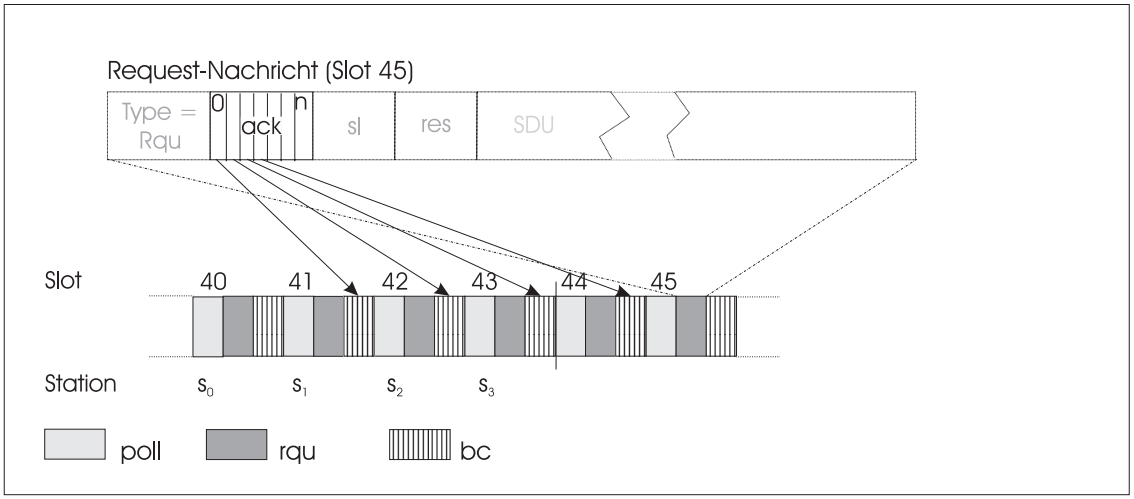


Abbildung 4.3 Zuordnung der Bits des *ack*-Feldes zu Broadcast-Nachrichten.

Die Zuordnung der Komponenten des *ack*-Feldes einer Request-Nachricht von Station *s* zu den Broadcast-Nachrichten geschieht wie folgt (s. Abb. 4.3): Das Bit an Position 0 des *ack*-Feldes bestätigt die erste der letzten *g.size* Broadcast-Nachrichten, d. h. die Broadcast-Nachricht, die der AP in der letzten Runde nach dem Pollen von *s* gesendet hat; das Bit an Position 1 bestätigt den nächsten Broadcast des AP u. s. f. Das Bit an Position *g.size - 1*, schließlich bestätigt die Broadcast-Nachricht, die der AP in dieser Runde vor dem Pollen von *s* gesendet hat. Dies macht deutlich: Wenn *s.lastPolled* die Sequenznummer des Slots beschreibt, in dem Station *s* das letzte Mal gepollt wurde, und wenn *s* eine Broadcast-Nachricht mit Sequenznummer *sg* bestätigen will, dann muß Station *s* dazu das Bit *sg – s.lastPolled* in ihrem *ack*-Feld auf *true* setzen.

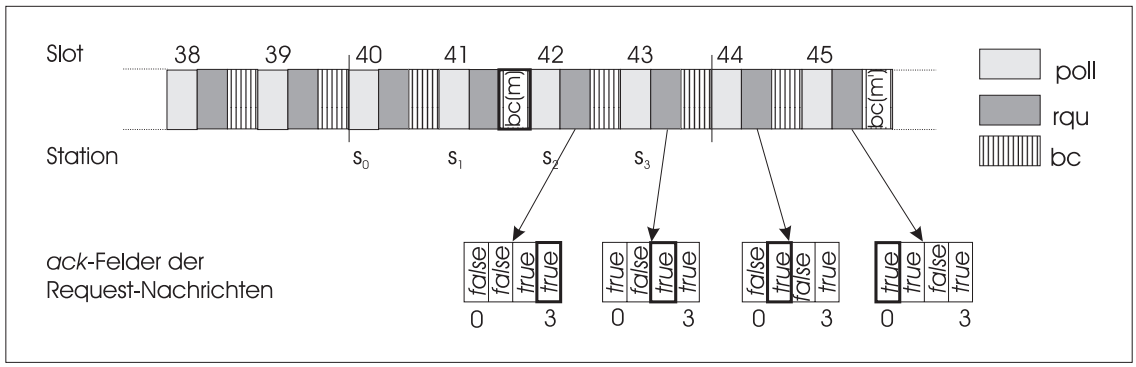


Abbildung 4.4 Mit den *ack*-Feldern der Request-Nachrichten aus den Slots 42 bis 45 bestätigen die Stationen s_0 bis s_3 die Broadcast-Nachricht aus Slot 41. In Slot 45 kann der AP daher die SDU *m'* übertragen.

In Abbildung 4.4 wird verdeutlicht, wie die Stationen den beschriebenen Bestätigungsmechanismus nutzen, um die Broadcast-Nachricht mit Sequenznummer 41, die der AP für Station s_1 gesendet hat, zu bestätigen: Der Nachfolger von s_1 benutzt das Bit $ack[41-38]$ seiner Request-Nachricht, um die Broadcast-Nachricht zu bestätigen. Dessen Nachfolger benutzt das Bit $ack[41-39]$, u. s. f., bis zu Station s_1 selbst, die das Bit $ack[41-41]$ ihrer Request-Nachricht benutzt, um die Broadcast-Nachricht der Vorrunde zu bestätigen. Wenn der AP eine der Request-Nachrichten nicht empfängt, dann wertet er dies so, als hätten alle Bits des *ack*-Feldes den Wert *false*, da er davon ausgehen muß, daß der Sender der Request-Nachricht, die entsprechenden Broadcast-Nachrichten nicht empfangen hat.

Durch den beschriebenen Mechanismus kann der AP eine Runde nachdem er eine Broadcast-Nachricht gesendet hat feststellen, ob diese von allen Stationen empfangen wurde. Ergibt sich aufgrund der Bestätigungen, die der AP für Broadcast-Nachrichten, die eine SDU *m* von Station *s* enthielten, erhalten hat, daß *m* von allen Stationen empfangen wurde, dann kann der AP das nächste Mal, wenn er für *s* eine Broadcast-Nachricht sendet, die nächste SDU von *s* übertragen (s. Abb. 4.4). Ansonsten überträgt er die SDU *m* erneut (s. Abb. 4.5). Jede SDU wird in maximal $OD+1$ Broadcast-Nachrichten übertragen; danach ist sicher, daß jede Station diese SDU in einer der Broadcast-Nachrichten empfangen hat, selbst dann, wenn der AP dieses aufgrund verlorener Request-Nachrichten u. U. nicht feststellen konnte. Weiterhin ist die Begrenzung der Anzahl der Übertragungen notwendig, um die Rechtzeitigkeit der Übertragung von SDUs vom AP zu den Stationen sicherzustellen. Ohne eine Einschränkung der Anzahl der Übertragungen könnten die Stationen eine SDU *m* beliebig lange nach dem ersten Broadcast von *m* empfangen.

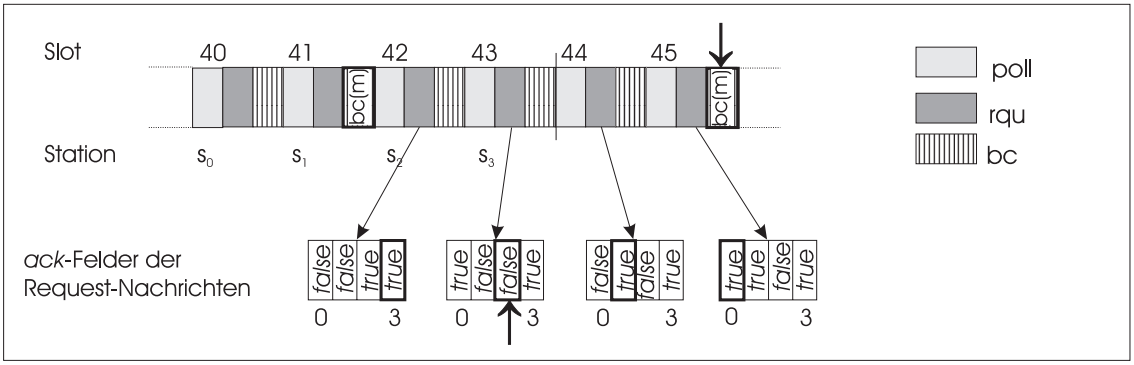


Abbildung 4.5 Da Station s_3 die Broadcast-Nachricht mit Sequenznummer 41 negativ bestätigt, überträgt der AP SDU *m* in Slot 45 erneut.

Um die Bestätigungen, die er von den Stationen erhält, auszuwerten, verwaltet der AP ein zweidimensionales Bitfeld *global_ack*[·,·]. In diesem Bitfeld ist *global_ack*[*originator.id, receiver.id*] = *true* genau dann, wenn die Station *receiver* eine Broadcast-Nachricht bestätigt hat, welche die aktuelle SDU von Urheber *originator* enthielt. Dabei ist die aktuelle SDU einer Station *s* diejenige SDU mit Urheber *s*, die der AP in der letzten Runde in einer Broadcast-Nachricht übertragen hat. Wenn der AP beispielsweise eine Request-Nachricht von Station *s*, mit *s.id = 1*, empfängt, in der *ack*[*0*]=*true*, dann setzt der AP *global_ack*[*1,1*] := *global_ack*[*1,1*] ∨ *true*, weil Station *s* mit diesem Bit den Empfang der letzten Broadcast-Nachricht bestätigt, die der AP für *s* gesendet hat (s. Abb. 4.3), und damit auch den Erhalt der in dieser Broadcast-Nachricht enthaltenen SDU. Entsprechend setzt der AP für jede Station *g*[*i*] der Gruppe *global_ack*[*i,1*] := *global_ack*[*i,1*] ∨ *ack*[*g*[*i*].*lastPolled* – *s.lastPolled*[13]]. Allgemein verändert der AP das Feld *global_ack* beim Empfang einer Request-Nachricht von Station *receiver* wie folgt:

$$\forall i \in [0,\ g.size-1]:\ global\_ack[i,\ receiver.id] :=$$
$$global\_ack[i,\ receiver.id] \vee ack[g[i].lastPolled - receiver.lastPolled].$$

Der AP kann also sicher sein, daß jede Station die aktuelle SDU von Station *s* erhalten hat, wenn er sie entweder in *OD+1* Broadcast-Nachrichten versendet hat oder wenn $\forall i \in [0,g.size]$: *global_ack*[*s.id, i*] = *true.* In diesem Falle braucht der AP die SDU nicht länger zu speichern, da sie von allen Stationen empfangen wurde und keine weiteren Übertragungen mehr stattfinden müssen. Weiterhin kann der AP die nächste SDU von *s* zur aktuellen SDU von *s* machen und mit dem Übertragen dieser SDU beginnen. Wenn der AP auf diese Art die Übertragung einer SDU abgeschlossen hat, ohne zwischenzeit-

[13] Es wird davon ausgegangen, daß *s.lastPolled* zu dem Zeitpunkt, an dem der AP die Bestätigungen auswertet, noch nicht auf die Sequenznummer des aktuellen Slots gesetzt ist. Das heißt, wenn der AP in Abbildung 4.3 die Bestätigungen von Station s_1 auswertetet, dann hat s_1.*lastPolled* noch den Wert 41.

lich eine neue SDU vom gleichen Urheber empfangen zu haben, sendet er eine leere Broadcast-Nachricht, die keine SDU enthält. Warum dies notwendig ist, wird in Paragraph 4.3.2 deutlich.

Wenn nicht direkt die erste Broadcast-Nachricht, die eine SDU *m* enthält, von alle Stationen bestätigt wird, dann wird *m* in mehreren Broadcast-Nachrichten übertragen. Der AP kann sicher sein, daß jede Station die SDU *m* erhalten hat, sobald jede Station mindestens eine Broadcast-Nachricht, die *m* enthält, bestätigt hat. Darum werden die Bits im *global_ack*-Feld nicht nach jedem Broadcast zurückgesetzt, sondern nur dann, wenn der AP mit dem Übertragen einer neuen SDU beginnt. Dies hat den Vorteil, daß die Anzahl der Stationen, die eine SDU noch bestätigen müssen, mit jeder Übertragung der SDU fällt und die Wahrscheinlichkeit, daß die Übertragung der SDU abgeschlossen werden kann, damit steigt. Daher werden weniger Übertragungen benötigt, um eine SDU an alle Stationen zu verteilen, als dies der Fall wäre, wenn alle Stationen den selben Broadcast einer SDU erhalten müßten.

Läßt man die Notwendigkeit der total geordneten Auslieferung der SDUs zunächst unberücksichtigt, dann könnte nach dem bisher beschriebenen Verfahren jede Station eine SDU ausliefern, sobald sie diese zum erstenmal erhalten hat. Da diese SDU offensichtlich als aktuelle SDU im AP gespeichert ist und der AP die aktuelle SDU solange überträgt, bis jede korrekte Station sie ebenfalls erhalten hat (und damit auch ausliefert), ist die Einigung unter den korrekten Stationen somit sichergestellt. Um die Auslieferung von Duplikaten zu verhindern, muß jede Station darauf achten, daß sie SDUs tatsächlich nur dann ausliefert, wenn sie sie zum erstenmal erhält. Dies ist aufgrund der eindeutigen Kennzeichnung der SDUs durch lokale Sequenznummer und Urheber problemlos zu realisieren.

4.2.4 Der Einfluß von Nachrichtenverlusten

In den beiden vorhergehenden Paragraphen wurde der Einfluß von Nachrichtenverlusten dahingehend untersucht, daß das Protokoll sicherstellt, daß der Verlust von Request- und Broadcast-Nachrichten nicht dazu führt, daß auch die darin enthaltenen SDUs nicht zuverlässig übertragen werden. Dieser Paragraph geht darauf ein, wie Nachrichtenverluste noch weitergehend den Ablauf des Protokolls beeinflussen können und wie die daraus erwachsenden Probleme gelöst werden.

Die erste Situation, die in diesem Zusammenhang untersucht werden soll, besteht darin, daß die Übertragung der Nachrichten von der Station an den AP schneller verläuft, als die Übertragung der Nachrichten vom AP an die Gruppe. Konkret ist dabei an folgende Situation gedacht: Eine Station sendet eine SDU *m* an den AP, die dieser noch in der gleichen Runde broadcastet. Der Urheber von *m* empfängt diese Broadcast-Nachricht und sendet daraufhin in der folgenden Runde eine neue SDU *m'* an den AP. Allerdings hat der AP aufgrund von Nachrichtenverlusten noch nicht von allen Stationen Bestätigungen für den Broadcast von *m* erhalten. Er muß daher die SDU *m* erneut übertragen

und kann m' nicht als aktuelle SDU speichern. Um die Nachricht m' dennoch nicht verwerfen zu müssen, hat der AP für jede Station einen Speicherplatz, in dem er solche ausstehenden Request-Nachrichten abspeichern kann. Würde diese Situation allerdings sehr häufig unmittelbar hintereinander auftreten, dann könnte dies dazu führen, daß der Nachrichtenspeicher des AP überläuft.

Der vorgestellte implizite Bestätigungsmechanismus schützt aber den AP vor dem Überlaufen des Nachrichtenspeichers. In der eben geschilderten Situation broadcastet der AP die SDU m, obwohl er die SDU m' schon erhalten hat. Daher erhält auch der Urheber von m' nicht die notwendige Bestätigung für m' durch den AP, und er wird somit in der nächsten Runde erneut die Nachricht m' an den AP übertragen, obwohl m' schon beim AP angekommen ist. Diese Situation ändert sich erst dann, wenn der AP das Broadcasten der Nachricht m abgeschlossen hat und zum erstenmal die Nachricht m' broadcastet. Nach dem Erhalt der Nachricht m' kann der Urheber mit dem Übertragen den nächsten SDU m'' beginnen. Dann hat aber auch der AP m' als aktuelle SDU übernommen und der Speicherplatz für eine ausstehende SDU des Urhebers ist wieder frei.

In der beschriebenen Situation kommt also das implizite Bestätigungsschema auch zur Flußkontrolle für die Übertragung zwischen Stationen und AP zum Einsatz und stellt sicher, daß es am AP nicht zu Nachrichtenüberläufen kommen kann. Daher braucht der AP im ungünstigsten Falle $2 \times n$ Nachrichtenspeicherplätze, um alle notwendigen SDUs zu speichern.

Eine weitere Situation, die durch das Auftreten von Nachrichtenverlusten eintreten kann, besteht darin, daß eine Station die Nachrichten einer ganzen Runde oder sogar mehrerer Runden verpaßt und direkt anschließend eine Polling-Nachricht empfängt. Wenn die Station diese Situation nicht feststellen könnte, dann würde sie veraltete Bestätigungen für Nachrichten aus vorhergehenden Runden an den AP übertragen. Hier schafft aber die Rundennummer, die in allen Polling-Nachrichten mit übertragen wird, Abhilfe. Durch sie kann die Station feststellen, daß mehr als eine Runde vergangen ist, nachdem sie das letzte mal gepollt wurde. Sie setzt in diesem Falle alle Bits des *ack*-Feldes auf *false* und überträgt sie in ihrer Request-Nachricht an den AP. Da sie tatsächlich alle Nachrichten der Vorrunde verpaßt hat, überträgt sie auf diese Weise die richtigen Informationen an den AP.

4.2.5 Zeitanalyse

In den vorhergehenden beiden Paragraphen wurde dargelegt, wie das Protokoll in der Lage ist, die Validität und Einigung beim Übertragen von SDUs sicherzustellen. Weiterhin wurde in Paragraph 4.2.3 erläutert, daß, solange die totale Ordnung der SDUs nicht gefordert wird, die Stationen die SDUs ausliefern können, sobald sie sie zum erstenmal erhalten. Unter diesen Annahmen soll nun untersucht werden, welche maximale Verzögerung zwischen dem Aufruf des *broadcast(m)*-Primitivs beim Urheber von m und dem Ausliefern von m an der letzten Station liegt.

Entsprechend den in Unterkapitel 2.2 formulierten Annahmen beträgt die maximale Dauer für das Übertragen einer einzelnen Nachricht auf dem Funknetz t_m Zeiteinheiten. Weiterhin wird davon ausgegangen, daß diese Zeit auch schon die Verzögerungen, die durch das lokale Scheduling entstehen, beinhaltet. Die Bearbeitungszeiten in den Protokollinstanzen der einzelnen Stationen und des AP werden als vernachlässigbar angenommen. Daher ergibt sich, daß die maximale Dauer eines Slots als

$$\Delta_{Slot} := 3 \times t_m$$

berechnet werden kann. Jede Runde besteht aus maximal n Slots, so daß sich für die Maximaldauer einer einzelnen Runde

$$\Delta_{Round} := n \times \Delta_{Slot}$$

ergibt.

Da beim Berechnen der Größe Δ_{Round} die maximale Anzahl von Stationen in einer Gruppe zugrunde gelegt wird, gelten die berechneten Zeitschranken a priori und daher auch über mehrere CFPs hinweg, unabhängig von der aktuellen Gruppengröße. Wenn die maximale Verzögerung jeweils nur dynamisch für die Dauer einer CFP garantiert werden soll, dann kann auch die aktuelle Gruppengröße als Grundlage genommen werden, was die berechnete Verzögerung u. U. erheblich verkleinert.

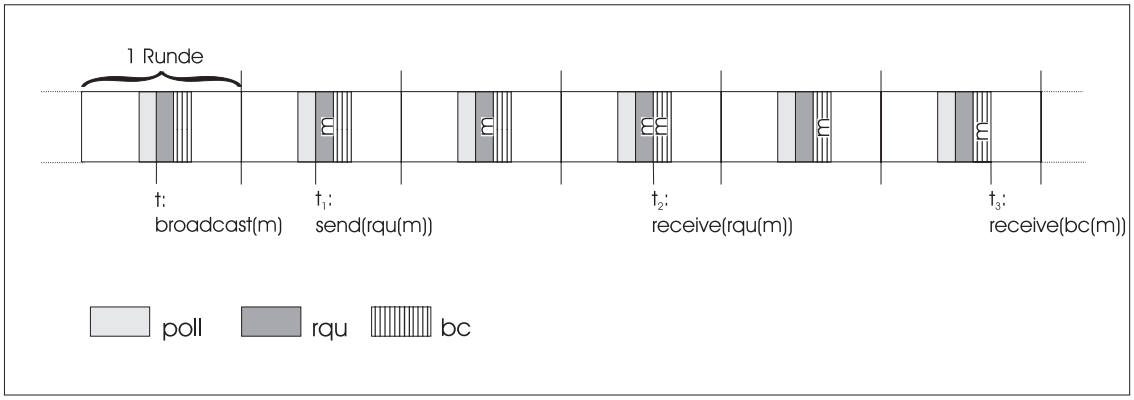


Abbildung 4.6 In der dargestellten Situation hat die Übertragung von *m* die maximale Verzögerung (Es sind nur die Slots einer einzelnen Station dargestellt; *OD = 2*).

Zunächst soll die Situation beschrieben werden, die bei der Berechnung der maximalen Verzögerung zugrunde gelegt werden muß, d. h. die Situation, in der die Übertragung einer SDU *m* an die Gruppe maximal verzögert wird (s. Abb. 4.6). Dieser Fall tritt ein, wenn der Aufruf von *broadcast(m)* erfolgt, unmittelbar nachdem der Urheber von *m* gepollt wurde. In diesem Falle kann die SDU *m* zum ersten Mal in der nächsten Runde an den AP übertragen werden. Im ungünstigsten Falle erhält der AP *m* weitere *OD* Run-

den später und er broadcastet *m* noch in der selben Runde. Der AP kann *m* direkt zur aktuellen SDU machen, weil zu dem Zeitpunkt, an dem der AP *m* spätestens empfängt, seit dem Empfang der vorhergehenden SDU des selben Urhebers mindestens *OD+1* Runden vergangen sind. Das Übertragen einer vorhergehenden SDU ist also zum spätesten Empfangszeitpunkt von *m* auf jeden Fall abgeschlossen. Spätestens *OD* Runden nachdem der AP *m* empfangen und zum ersten Mal gebroadcastet hat, wird die letzte Station *m* empfangen und ausliefern.

Für diese Situation können die folgenden Werte für die in Abbildung 4.6 bezeichneten Zeitpunkte berechnet werden:

$$t_1 = t + \Delta_{Round} ,$$
$$t_2 = t_1 + OD \times \Delta_{Round} + t_m ,$$
$$t_3 = t_2 + OD \times \Delta_{Round} + t_m .$$

Daraus ergibt sich die maximale Verzögerung eines zuverlässigen Broadcasts als

$$\Delta_{BC} = t_3 - t = (2 \times OD + 1) \times \Delta_{Round} + 2 \times t_m .$$

Möchte man nun aber die Zeit bestimmen, die für das Übertragen mehrerer SDUs von mehreren Stationen notwendig ist, so bestimmt sich diese nicht einfach, indem man Δ_{BC} mit der Anzahl der zu übertragenden Nachrichten multipliziert. Vielmehr wird die benötigte Zeit aufgrund der Parallelität des Protokolls deutlich unter einem solchermaßen berechneten Wert liegen.

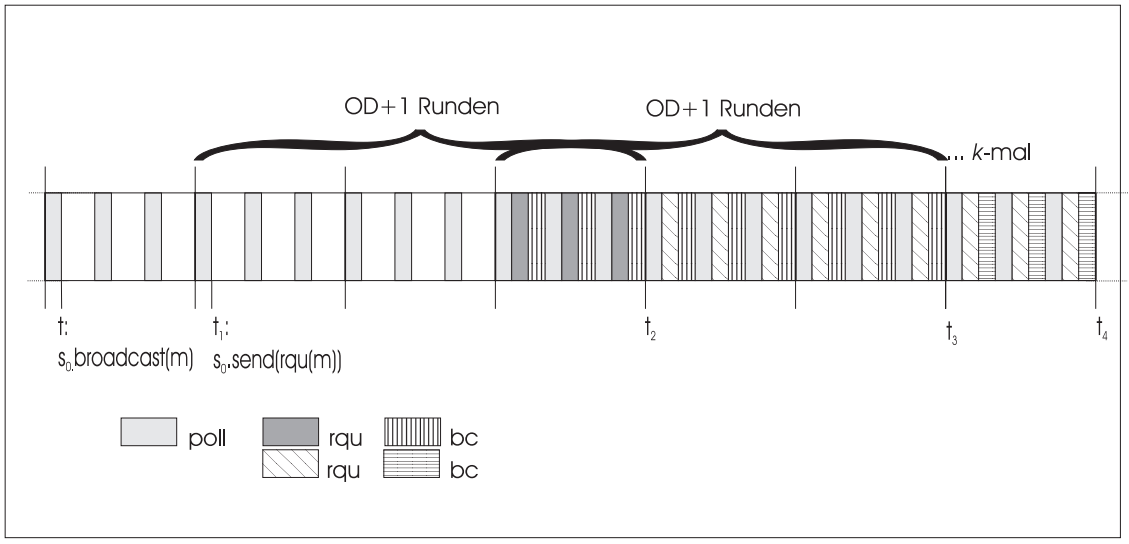


Abbildung 4.7 Jede Station überträgt *k* SDUs. Für jede einzelne SDU tritt der ungünstigste Fall ein (*OD=2*).

Dieses kann man sich verdeutlichen, indem man sich in Abbildung 4.6 alle Slots als vollständig mit Nachrichten ausgefüllt denkt. Man kann sehen, daß in diesem Falle in $2 \times OD + 2$ Runden n Nachrichten übertragen werden können. Weil die Übertragung der SDUs aller Stationen weitgehend parallel verlaufen kann, können also in nur wenig mehr Zeit n-mal so viele SDUs übertragen werden.

Aber auch die Übertragung von zwei SDUs der selben Station kann überlappt werden, da der erste Übertragungsabschnitt von SDU m' schon durchgeführt werden kann, während der zweite Abschnitt der vorhergehenden SDU m noch im Gange ist. Dies wird in Abbildung 4.7 deutlich. Diese zeigt, wie n Stationen jeweils k SDUs übertragen und dabei die mögliche Parallelität optimal ausnutzen. Für jede einzelne Nachricht tritt der ungünstigste Fall ein.

Zum Zeitpunkt t erfolgt bei Station s_0 der Aufruf von *broadcast(m)*, unmittelbar nachdem diese gepollt worden ist. Während der folgenden Runde treffen auch an den anderen Stationen die ersten SDUs ein, ebenfalls unmittelbar nach dem Empfang der Polling-Nachrichten. Daher übertragen in der darauffolgenden Runde alle Stationen erstmals die betreffenden Nachrichten an den AP (angefangen bei t_1). $OD + 1$ Runden später (Zeitpunkt t_2) hat der AP daher alle SDUs erhalten und zum erstenmal gebroadcastet. Weitere OD Runden später (Zeitpunkt t_3) haben alle Stationen die SDUs erhalten und ausgeliefert sowie OD Übertragungsversuche für die nächste SDU unternommen. Daher wird der AP eine weitere Runde später diese SDUs empfangen und alle zum erstenmal gebroadcastet haben (Zeitpunkt t_4). OD Runden später sind diese SDUs von allen Stationen ausgeliefert und OD Übertragungsversuche für die nächsten SDUs haben stattgefunden. Dieser Ablauf wiederholt sich bis die letzte SDU jeder Station von allen Stationen ausgeliefert ist.

Die für die Übertragung der Nachrichten benötigte Zeit kann wie folgt berechnet werden: Eine Runde nach der Ankunft der ersten Nachricht bei Station s_0 beginnt s_0 mit dem Übertragen der Request-Nachrichten. $k \times (OD + 1)$ Runden später (weniger eine Polling-Nachricht) hat der AP von jeder Station die letzte SDU erhalten und zum erstenmal gebroadcastet. OD Runden später werden diese letzten SDUs von allen Stationen ausgeliefert, so daß nach insgesamt

$$\begin{aligned} \Delta_{BC,k} &= [(OD+1) \times k + OD + 1] \times \Delta_{Round} - t_m \\ &\leq [(OD + 1) \times (k + 1)] \times \Delta_{Round} \end{aligned}$$

Zeiteinheiten alle $n \times k$ Nachrichten übertragen worden sind.

So können erheblich mehr Nachrichten pro Zeiteinheit eingeplant werden (Echtzeit-Durchsatz). Statt

$$1 \text{ Broadcast pro } (2 \times OD + 1) \times \Delta_{Round} + 2 \times t_m \text{ Zeiteinheiten}$$

ergeben sich

$$n \text{ Broadcasts pro } (2 \times OD + 1) \times \Delta_{Round} + \Delta_{Round} - t_m \text{ Zeiteinheiten}$$
$$\approx 1 \text{ Broadcast pro } 2 \times (OD + 1) \times \Delta_{Slot} \text{ Zeiteinheiten},$$

da die Broadcasts aller Stationen parallel ausgeführt werden und sogar

$$n \times k \text{ Broadcasts pro } [(OD+1) \times (k + 1)] \times \Delta_{Round} \text{ Zeiteinheiten}$$
$$\approx 1 \text{ Broadcast pro } (OD + 1) \times \Delta_{Slot} \text{ Zeiteinheiten},$$

wenn sichergestellt ist, daß sich die Kommunikationsabschnitte der einzelnen Stationen überlagern.

In diesem Unterkapitel wurde dargelegt, wie auf einem unzuverlässigen Medium die zuverlässige und rechtzeitige Übertragung von Broadcast-Nachrichten erreicht werden kann. Es zeigt sich allerdings, daß die dabei garantierbare maximale Verzögerung sehr groß ist. Dieses Problem ergibt sich zwangsläufig, wenn man auf einem unzuverlässigen Medium, auf dem potentiell sehr viele Nachrichtenverluste auftreten, die zuverlässige Übertragung der Nachrichten garantieren möchte. Eine Möglichkeit, dieses Problem zu behandeln, wird im folgenden Unterkapitel vorgestellt.

4.3 Wählbare Zeit- und Zuverlässigkeitseigenschaften

4.3.1 Das Konzept

Die im vorhergehenden Unterkapitel hergeleiteten großen maximalen Verzögerungen haben den Nachteil, daß in einer CFP nur die Übertragung relativ weniger Nachrichten garantiert werden kann. Weiterhin sind die garantierbaren Verzögerungen u. U. für bestimmte Anwendungen zu groß. In diesem Unterkapitel wird ein Konzept vorgestellt, das es erlaubt, die maximalen Verzögerungen zu verkürzen. Diesem Konzept liegt der Gedanke zugrunde, daß es Anwendungen gibt, vor allem in dynamischen Systemen, die gelegentliche Nachrichtenverluste tolerieren können (eine ähnliche Annahme findet sich auch in [AV95]). Ein Team im RoboCup beispielsweise, das Broadcast-Nachrichten verwendet, um von einer Verteidigungs- zu einer Angriffsstrategie zu wechseln, wenn sich der Ball vor dem gegnerischen Tor befindet, kann gelegentliche Nachrichtenverluste tolerieren. Ebenso können Roboter, die Broadcasts nutzen, um sich beim gemeinsamen Anheben eines Bauteils zu koordinieren, u. U. gelegentliche Nachrichtenverluste tolerieren.

Häufig muß aber gewährleistet sein, daß Nachrichtenverluste konsistent wahrgenommen werden (s. Paragraph 2.2.1), damit die Anwendungen diese tolerieren können. Dies bedeutet für atomare Broadcasts, daß trotz der eingeschränkten Zuverlässigkeit

deutet für atomare Broadcasts, daß trotz der eingeschränkten Zuverlässigkeit (Validität), die Einigung gewährleistet sein muß. Durch die garantierte Einigung wird sichergestellt, daß das Verhalten der Gruppe in sich immer konsistent bleibt. So soll es beispielsweise in den oben geschilderten Beispielen nicht dazu kommen, daß einige der Roboter denken, es liege eine Verteidigungssituation vor, und andere, es liege eine Angriffssituation vor, oder daß einer der Roboter versucht, das Bauteil anzuheben, der andere aber nicht.

Anwendungen, wie sie gerade beschrieben wurden, sollen die Möglichkeit haben, auf die volle Zuverlässigkeit der Übertragung zu verzichten, um dadurch kürzere maximale Verzögerungen zu erreichen. Diese Möglichkeit wird den Anwendungen in dem hier beschriebenen Protokoll dadurch gegeben, daß sie für jede Nachricht spezifizieren können, wie oft diese maximal neu übertragen werden soll. Diese Größe, die im Folgenden als *Resiliency* (*m.res*) einer Nachricht *m* bezeichnet wird, erlaubt es dem Anwender, selbst die Tradeoff-Entscheidung zwischen Zuverlässigkeit und maximaler Verzögerung zu treffen: Durch das Erhöhen der Resiliency einer Nachricht wird die Wahrscheinlichkeit, daß diese Nachricht von alle Stationen ausgeliefert wird, erhöht; gleichzeitig wird aber auch die garantierte maximale Verzögerung größer.

Entsprechend der im vorhergehenden Unterkapitel vorgestellten Kommunikationsstruktur bezieht sich die Resiliency einer Nachricht auf beide Kommunikationsabschnitte, d. h., sowohl bei der Übertragung von den Stationen zum AP wird die SDU *m* in maximal $m.res + 1$ vielen Runden übertragen, wie auch bei der Übertragung vom AP zu den Stationen. Die Angabe einer Resiliency von *OD* garantiert weiterhin die zuverlässige Übertragung der SDU.

Dieses Konzept wirft aber folgendes Problem auf: Durch die Einschränkung der Zuverlässigkeit der Übertragung ist auch die Einigung nicht mehr sichergestellt. Diese war bisher dadurch gegeben, daß der AP gewährleistet hat, daß eine Nachricht, die von einer Station empfangen wurde, auch von allen anderen korrekten Stationen empfangen und ausgeliefert wird. Aufgrund der eingeschränkten Zuverlässigkeit ist dies aber nicht mehr gewährleistet: Eine Nachricht, die von einer Station empfangen wurde, muß deshalb durchaus nicht von allen korrekten Stationen ebenfalls empfangen werden.

Daher wurde das Protokoll so gestaltet, daß die Stationen SDUs nicht unmittelbar ausliefern, nachdem sie sie empfangen haben, sondern daß sie diese zunächst speichern. Erst wenn der AP den Stationen bestätigt, daß eine SDU von alle Stationen empfangen wurde, wird diese von den Stationen ausgeliefert. Dazu hat jede Station für jedes Gruppenmitglied *s* genau einen Speicherplatz, um SDUs von *s*, die sie empfangen hat, aber noch nicht ausliefern kann, zwischenzuspeichern. Wenn der AP feststellt, daß er eine SDU, die er gebroadcastet hat, nicht in der angegebenen Anzahl von Übertragungsversuchen an alle Stationen verteilen kann, dann teilt er dies den Stationen mit, so daß diejenigen, die die Nachricht schon empfangen haben, den Speicher freigeben können.

Der Ablauf des Broadcasts einer einzelnen SDU ist also wie folgt: Die Urheberstation versucht die SDU *m* an den AP zu übertragen, wobei sie höchstens in $m.res + 1$ Runden

die Übertragung versucht. Empfängt der AP die SDU, dann broadcastet er sie an die Gruppe. Auch dies geschieht in maximal *m.res + 1* Runden. Bis hierher wird also das im vorhergehenden Unterkapitel vorgestellte Verfahren verwendet, wobei *OD* durch *m.res* ersetzt werden muß. Wenn der AP von allen Stationen eine Bestätigung für die SDU erhalten hat oder diese *OD+1*-mal gebroadcastet hat, dann entscheidet er, daß die SDU ausgeliefert werden kann (*accept*) und teilt dieses den Stationen mit. Wenn der AP die SDU *m.res+1*-mal übertragen hat, ohne von allen Stationen eine Bestätigung zu erhalten, und wenn *m.res* < *OD* ist, dann entscheidet er, daß die SDU verworfen werden muß (*reject*) und teilt dieses den Stationen mit.

Um sicherzustellen, daß die Stationen sich rechtzeitig darüber einigen, ob sie eine SDU ausliefern oder nicht, muß gewährleistet sein, daß die Entscheidungen des AP zuverlässig und rechtzeitig, mit einer relativ kleinen maximalen Verzögerung, an die Stationen übertragen werden können. Zum Übertragen der Entscheidungen wird also ein synchroner Kanal benötigt. Dieser synchrone Kanal braucht allerdings nur eine sehr geringe Bandbreite zur Verfügung zu stellen, da nur sehr kleine Informationseinheiten, *accept* oder *reject*, versendet werden. Weiterhin findet die Kommunikation im synchronen Kanal nur unidirektional, vom AP zu den Stationen, statt.

Aufgrund der geforderten Eigenschaften wird der synchrone Kanal mit Hilfe des Diffusionsverfahrens realisiert. Der AP sendet seine Entscheidungen an die Stationen, ohne von diesen Bestätigungen zu erwarten; jede Entscheidung *OD+1*-mal. Dabei werden die Entscheidungen aber nicht in speziellen Nachrichten versendet, sondern der AP nutzt die Broadcast-Nachrichten, die er versendet, um via Piggy-Backing seine Entscheidungen zu übertragen. Jede Entscheidung wird in genau *OD + 1* aufeinanderfolgenden Broadcast-Nachrichten übertragen. So kann die zuverlässige und rechtzeitige Übertragung der Entscheidungen realisiert werden, ohne daß i. A. zusätzliche Nachrichten notwendig sind.

4.3.2 Der synchrone Kanal aus Sicht des AP

Um die Übertragung der Entscheidungen des AP im synchronen Kanal mit Hilfe von Piggy-Backing zu realisieren, befindet sich in jeder Broadcast-Nachricht ein Feld von genau *OD + 1* Bit-Tupeln (*synchCh*[], s. Abb. 4.2). Die Bit-Tupel haben die folgende Bedeutung: Hat das erste Bit den Wert *true*, so beinhaltet dieser Tupel eine Entscheidung des AP bzgl. einer SDU. In diesem Falle besagt das zweite Bit, ob es sich um eine *accept*-, der Wert des zweiten Bits ist *true*, oder eine *reject*-Entscheidung, der Wert des zweiten Bits ist *false*, handelt. Hat das erste Bit den Wert *false*, so handelt es sich nicht um eine Entscheidung des AP bzgl. einer SDU. Damit ergeben sich die folgenden Bedeutungen für die Tupel:

(true, true): Eine *accept*-Entscheidung des AP

(true, false): Eine *reject*-Entscheidung des AP

(false,false): Der AP hat weder *accept* noch *reject* entschieden, die SDU wird nochmals übertragen.

(false,true): Dieses Tupel wird zum Übermitteln von Mitgliedschaftsinformationen verwendet.

Jedesmal bevor der AP im Slot einer Station *s* eine Broadcast-Nachricht versendet, prüft er, ob er bzgl. der aktuellen SDU von *s* eine Entscheidung treffen kann. Ist dies der Fall, dann fügt er den entsprechenden *accept*- oder *reject*-Tupel an der Position 0 in das *synchCh*-Feld ein. Kann er keine Entscheidung treffen und muß er daher die SDU in dieser Runde nochmals übertragen, dann fügt er den *(false, false)*-Tupel in *synchCh*[*0*] ein.

Der AP fügt also vor jedem Broadcast genau einen Bit-Tupel in den synchronen Kanal ein. Bevor er dies tut, verschiebt er alle Tupel, die sich bereits im synchronen Kanal befinden, um eine Position nach rechts, wodurch sich die Position jedes Tupels um eins erhöht. Daher ist sichergestellt, daß der AP in *synchCh*[*0*] nie einen anderen Bit-Tupel überschreibt. Das Rotieren der Bit-Tupel stellt weiterhin sicher, daß jeder Bit-Tupel genau *OD+1*-mal übertragen wird, da er an Position 0 eingefügt und dann *OD*-mal im synchronen Kanal verschoben wird, bevor er diesen bei der *OD+1*sten Verschiebung verläßt.

Ein Tupel, den der AP zum Zeitpunkt t in den synchronen Kanal einfügt, wird mithin spätestens zum Zeitpunkt $t + OD \times \Delta_{Slot} + t_m$ von jeder korrekten Station empfangen, d. h., jede Station empfängt bis zu spätestens diesem Zeitpunkt eine Broadcast-Nachricht, die den Tupel enthält. Diese maximale Verzögerung ist deutlich kleiner, als die für Broadcast-Nachrichten erreichte (vgl. Unterkapitel 4.2).

Wenn eine Station eine Broadcast-Nachricht empfängt, die einen bestimmten Bit-Tupel enthält, so ist sichergestellt, daß auch jede andere korrekte Station eine Broadcast-Nachricht empfängt, die diesen Bit-Tupel enthält.

4.3.3 Der synchrone Kanal aus Sicht der Stationen

Da jeder Tupel, den der AP an die Stationen überträgt, in mehreren Broadcast-Nachrichten enthalten ist, muß das Protokoll dafür sorgen, daß die Stationen jeden der Tupel nur genau einmal bearbeiten. Solange eine Station keine Broadcast-Nachricht verpaßt, muß sie, um dies zu erreichen, immer nur den Bit-Tupel an Position 0 im synchronen Kanal bearbeiten, da dieser der einzige ist, den sie noch nicht in der vorhergehenden Nachricht empfangen hat. Wenn eine Station eine oder mehrere Broadcast-Nachrichten nicht empfangen hat, dann kann sie durch die globale Sequenznummer feststellen, wie viele Nachrichten sie verloren hat, und damit auch, wie viele Bit-Tupel sie aus dem synchronen Kanal noch bearbeiten muß. Wenn eine Station eine Broadcast-Nachricht mit Sequenznummer sg_1 empfängt und anschließend eine weitere mit Se-

quenznummer sg_2, dann weiß sie, daß sie genau sg_2 - sg_1 Tupel aus dem synchronen Kanal bearbeiten muß. Die Annahme eines begrenzten Omission-Degree stellt dabei sicher, daß sg_2 - $sg_1 \leq OD + 1$, so daß eine Station nie mehr Tupel aus dem synchronen Kanal bearbeiten muß als in der gerade empfangenen Broadcast-Nachricht enthalten sind.

Wenn eine Station feststellt, daß sie genau *x* Tupel aus dem synchronen Kanal bearbeiten muß, dann bearbeitet sie die Tupel in absteigender Reihenfolge. Dies bedeutet, sie bearbeitet zunächst den Tupel an Position *x - 1*, dann den an Position *x - 2*, u. s. f., bis sie als letztes den Tupel an Position 0 bearbeitet. Auf diese Weise bearbeiten alle Stationen die Tupel in der Reihenfolge, in der sie vom AP übertragen worden sind: Der jüngste Tupel an Position 0 wird zuletzt bearbeitet. Dies ist vor allem dann von Bedeutung, wenn sich mehrere Tupel auf die selbe SDU beziehen, z. B. zuerst ein *(false,false)*-Tupel, der anzeigt, daß der AP eine SDU nochmals überträgt und dann ein *(true,true)*-Tupel als *accept*-Entscheidung. In anderer Reihenfolge ausgewertet könnten diese Tupel zu einem Fehlverhalten der Stationen führen. Der synchrone Kanal stellt also sicher, daß jede korrekte Station jede Entscheidung des AP empfängt und weiterhin, daß jede Station, bevor sie eine Entscheidung des AP empfängt, alle vorhergehenden Entscheidungen des AP bereits bearbeitet hat. Damit ist auch sichergestellt, daß alle Stationen die Entscheidungen in der gleichen Reihenfolge bearbeiten.

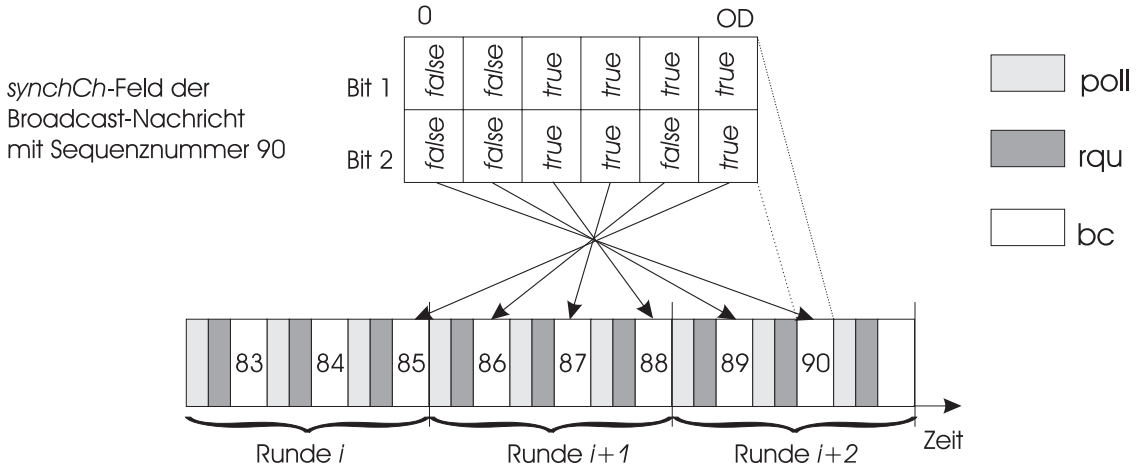


Abbildung 4.8 Die Zuordnung von Bit-Tupeln zu den Broadcast-Nachrichten, in denen sie zum erstenmal übertragen wurde.

Schließlich wird durch die Weise, in der der AP die Tupel überträgt, erreicht, daß jede Station feststellen kann, in welcher Broadcast-Nachricht ein Bit-Tupel zum ersten Mal enthalten war (s. Abb. 4.8). Wenn eine Station eine Broadcast-Nachricht mit globaler Sequenznummer 90 empfängt, dann war die erste Broadcast-Nachricht, in welcher der Tupel an Position 0 enthalten war, die Broadcast-Nachricht mit Sequenznummer 90 (kurz: Broadcast-Nachricht 90); der Tupel an Position 1, wurde zum erstenmal in Broadcast-Nachricht 89 versendet u. s. f. Der Tupel an Position *i* in *synchCh*[] einer

Broadcast-Nachricht *bc(sg, synchCh, sl, m)* wurde also zum erstenmal in Broadcast-Nachricht *sg - i* versendet.

Durch die solchermaßen bestimmten Sequenznummer können die Stationen feststellen, auf welche SDU sich die Entscheidungen des AP beziehen. Die Station, die in Abbildung 4.8 die Broadcast-Nachricht 90 empfängt, weiß, daß der *accept*-Tupel an Position 2 zum erstenmal in der Broadcast-Nachricht 88 gesendet wurde. Sie weiß weiterhin (s. Unterkapitel 4.5), daß Station s_3 Eigentümer des Slot 88 ist und daß sich daher die *accept*-Entscheidung auf die aktuelle SDU von s_3 bezieht.

Wenn eine Station eine *reject*-Entscheidung für eine SDU erhält, dann kann es sein, daß sie diese SDU nicht gespeichert hat. Wenn eine Station aber eine *accept*-Entscheidung für eine SDU *m* erhält, dann kann dies nur zwei Gründe haben:

1. Der AP hat *m OD+1*-mal gebroadcastet oder

2. der AP hat von jeder Station eine Bestätigung für *m* erhalten.

In beiden Fällen ist sichergestellt, daß *m* bei jeder Station gespeichert ist, die die *accept*-Entscheidung erhält, und somit auch ausgeliefert werden kann.

4.3.4 Zeitanalyse

In Abbildung 4.9 ist die Situation dargestellt, bei der die maximale Verzögerung zwischen dem Aufruf des *broadcast*-Primitivs und dem Aufruf des *deliver*-Primitivs an der letzten Station auftritt. Diese Situation, die sich weitgehend wie die in Paragraph 4.2.5 analysierte gestaltet, wird der folgenden Analyse zugrunde gelegt.

Im ungünstigsten Falle erfolgt der Aufruf des *broadcast*-Primitivs unmittelbar nachdem die Station gepollt worden ist (Zeitpunkt *t*), so daß eine Runde verstreicht, bis die Station die SDU zum erstenmal an den AP sendet (Zeitpunkt t_1). Aufgrund der in Unterkapitel 4.2 vorgestellten Berechnung von *m.lrts* wird die SDU zum letzten Mal *m.res* Runden später an den AP übertragen. Wenn auch diese Übertragung mißlingt, erhält der AP *m* nicht und keine der Stationen wird *m* ausliefern. Wenn der AP *m* empfängt, geschieht dies spätestens zum Zeitpunkt t_2. Zum Zeitpunkt t_3, *m.res* Runden später, unternimmt der AP den *m.res+1*-ten Übertragungsversuch und trifft spätestens eine Runde später (Zeipunkt t_4) eine Entscheidung bzgl. der SDU *m*, die er in der folgenden Broadcast-Nachricht zum ersten Mal im synchronen Kanal überträgt. $OD \times \Delta_{Slot} + t_m$ Zeiteinheiten später hat daher jede Station die Entscheidung empfangen und *m* entweder ausgeliefert oder verworfen.

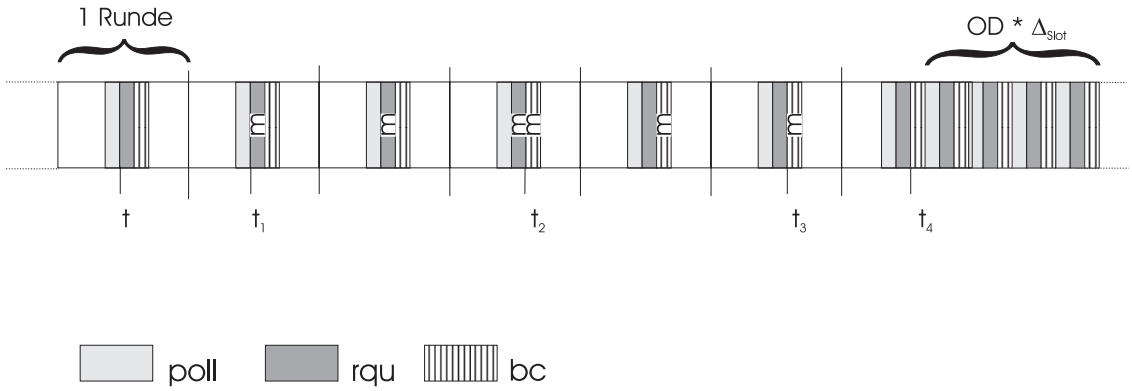


Abbildung 4.9 In der dargestellten Situation hat die Übertragung von *m* die maximale Verzögerung (Es sind nur die Slots einer einzelnen Station dargestellt; *m.res = 2, OD = 4*).

Für die Zeitpunkte t_1 bis t_5 ergeben sich die folgenden Werte:

$$t_1 = t + \Delta_{Round} ,$$
$$t_2 = t_1 + m.res \times \Delta_{Round} + t_m ,$$
$$t_3 = t_2 + m.res \times \Delta_{Round} ,$$
$$t_4 = t_3 + \Delta_{Round} ,$$
$$t_5 = t_4 + OD \times \Delta_{Slot} + t_m .$$

Daher beträgt die maximale Verzögerung einer Broadcast-Nachricht:

$$\Delta_{BC} = t_5 - t = 2 \times (m.res + 1) \times \Delta_{Round} + OD \times \Delta_{Slot} + 2 \times t_m$$
$$\leq 2 \times (m.res + 1) \times \Delta_{Round} + (OD + 1) \times \Delta_{Slot} .$$

Bei der Herleitung der maximalen Verzögerung Δ_{BC} wurde vorausgesetzt, daß zum spätesten Ankunftszeitpunkt einer SDU am AP (t_2) der AP sofort mit der Übertragung der SDU beginnen kann. Solange die Resiliencies aufeinanderfolgender SDUs gleich sind, z. B. alle SDUs werden mit Resiliency *OD* übertragen, kann der Urheber von SDU *m* mit dem Übertragen der nachfolgenden SDU *m'* beginnen, sobald er die Übertragung von *m* an den AP abgeschlossen hat, d. h. sobald er *m* in einer Broadcast-Nachricht empfangen oder *m* in *m.res+1* Runden übertragen hat. Da *m* und *m'* die gleiche Resiliency haben ist sichergestellt, daß der AP spätestens zum spätesten Ankunftszeitpunkt von *m'* eine Entscheidung bzgl. *m* getroffen hat und daher mit der Übertragung von *m'* beginnen kann. Wenn aber aufeinanderfolgende SDUs unterschiedliche Resiliencies haben, dann ist dies nicht mehr gewährleistet: Wenn z. B. eine Station eine SDU *m* mit Resiliency *m.res=5* an den AP überträgt und direkt im Anschluß den Broadcast von *m* empfängt, dann kann sie in der darauffolgenden Runde die nächste SDU *m'* an den AP übertragen. Wenn aber nun die Resiliency von *m'* Eins ist und *m* fünfmal gebroadcastet werden muß, bis der AP von allen Stationen eine Bestätigung erhalten hat, dann kann

der AP nicht rechtzeitig mit dem Übertragen von *m'* beginnen. Wenn also aufeinanderfolgende Nachrichten des selben Urhebers eine unterschiedliche Resiliency haben können, darf eine Station mit dem Übertragen der nachfolgenden Nachricht erst dann beginnen, wenn der AP die Übertragung der vorhergehenden Nachricht abgeschlossen hat. Dies kann der Urheber einer Nachricht anhand der Entscheidungen des AP feststellen: Wenn er eine *accept*- oder *reject*-Entscheidung des AP empfängt, hat dieser die Übertragung der betreffenden Nachricht abgeschlossen; wenn der AP nicht bis spätestens in Runde $m.lrts + m.res + 1$ eine Entscheidung getroffen hat, hat er die SDU *m* nicht empfangen.

Da das Übertragen der Entscheidungen im synchronen Kanal parallel zur Durchführung anderer SDUs stattfinden kann, kann das in Paragraph 4.2.5 vorgestellte Beispiel der parallelen Übertragung von jeweils *k* Broadcasts von *n* Station völlig analog übernommen werden (unter der Annahme, daß alle Nachrichten die gleiche Resiliency haben). Die einzige Änderung besteht darin, daß nach dem letzten Broadcast der letzten SDU eine weitere Runde zum ermitteln der Bestätigungen und danach noch *OD* zusätzliche Broadcasts stattfinden müssen, um die Entscheidung für die letzte SDU zu übertragen. Daher ergibt sich die Zeit für das zuverlässige Übertragen von *k* Nachrichten von *n* Stationen als:

$$\begin{aligned}\Delta_{BC,k} &= [(m.res+1) \times k + m.res + 1] \times \Delta_{Round} - t_m + \Delta_{Round} + OD \times \Delta_{Slot} \\ &= [(m.res + 1) \times (k + 1) + 1]\ \Delta_{Round} + OD \times \Delta_{Slot} - t_m\end{aligned}$$

Im Rahmen der Zeitanalyse wird ein weiterer Vorteil des Einsatzes des synchronen Kanals deutlich. Eine Station kann eine SDU frühestens ausliefern, nachdem sie die erste Broadcast-Nachricht des AP empfangen hat, die eine *accept*-Entscheidung für diese SDU erhält (Zeitpunkt $t_4 + t_m$ in Abb. 4.9). Die letzte Station wird die SDU spätestens *OD*-Slots später ausliefern, so daß sich für die Broadcast-Nachrichten eine Laufzeitvarianz von

$$\tau_{BC} = OD \times \Delta_{Slot}$$

ergibt. Diese ist deutlich kleiner als die Laufzeitvarianz, die ohne den synchronen Kanal erreicht worden wäre ($OD \times \Delta_{Round}$).

4.4 Ordnung

Weil nach dem in Unterkapitel 4.2 beschriebenen Verfahren die selbe SDU u. U. in mehreren Broadcast-Nachrichten übertragen wird, können die Stationen die SDUs in unterschiedlicher Reihenfolge erhalten. In Abbildung 4.10 erhält z. B. Station s_0 SDU m_0 vor SDU m_1, während Station s_1 die SDUs in umgekehrter Reihenfolge erhält. Daher

darf auch aus Gründen der totalen Ordnung die Auslieferung einer SDU nicht unmittelbar nach dem Erhalt erfolgen.

Wie im vorhergehenden Unterkapitel beschrieben, liefern Stationen eine SDU immer erst dann aus, wenn sie für diese eine *accept*-Entscheidung erhalten. Die eindeutige Zuordnung der Entscheidungen des AP zu den SDUs ist durch die globalen Sequenznummern und die Positionen der Entscheidungen im *syncCh*-Feld möglich (s. Paragraph 4.3.3) Da der AP für jede SDU höchstens eine *accept*-Entscheidung trifft und alle Stationen die Entscheidungen des AP in der gleichen Reihenfolge bearbeiten, ist sichergestellt, daß alle Stationen die SDUs in der gleichen Reihenfolge ausliefern: Wenn zwei Stationen zwei SDUs ausliefern, dann legen beide Stationen die gleichen *accept*-Entscheidungen zugrunde; da die Stationen diese Entscheidungen in der gleichen Reihenfolge bearbeiten, liefern sie auch die SDUs in der gleichen Reihenfolge aus.

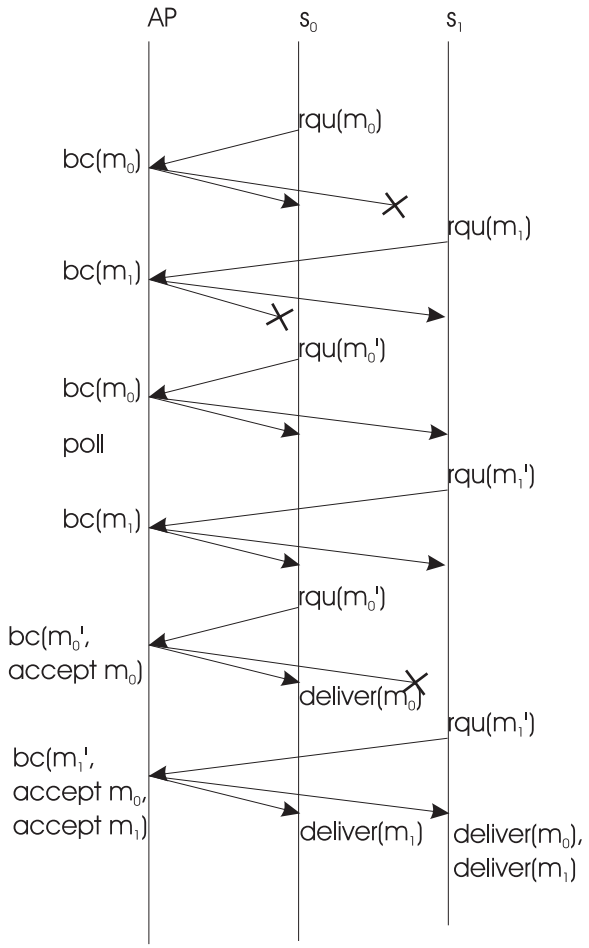


Abbildung 4.10 Das Ausliefern der SDUs m_0 und m_1 wird durch den synchronen Kanal geordnet (Aus Platzgründen wurden die Polling-Nachrichten nicht dargestellt).

In Abbildung 4.10 ist zu erkennen, daß der AP zunächst die *accept*-Entscheidung für die SDU m_0 und dann die *accept*-Entscheidung für die SDU m_1 trifft. Entsprechend liefern

beide Stationen zunächst die SDU m_0 und erst dann die SDU m_1 aus. Die Abbildung soll weiterhin nochmals verdeutlichen, daß die totale Ordnung der Entscheidungen und damit auch der Auslieferung der SDUs, selbst dann erhalten bleibt, wenn es zum Verlust von Broadcast-Nachrichten kommt.

Bei dem hier vorgestellten Verfahren ordnet der AP die SDUs durch seine *accept*-Entscheidungen. Das Ordnen einer SDU findet also erst statt, *nachdem* diese SDU von allen Stationen empfangen wurde. Bei anderen Verfahren, z. B. in den zentralisierten Ansätzen (s. Paragraph 3.1.1), findet das Ordnen der SDUs statt, bevor diese zum erstenmal gebroadcastet werden. Das nachträgliche Ordnen der SDUs hat den Vorteil, daß „schnelle“ SDUs „langsame“ SDUs überholen können. Wenn beispielsweise der AP zunächst SDU m_0 und dann SDU m_1 broadcastet und weiterhin m_1 schon nach der ersten Runde von allen Stationen bestätigt wurde, m_0 aber mehrmals übertragen werden muß, dann kann der AP die *accept*-Entscheidung für m_1 vor der *accept*-Entscheidung für m_0 treffen und alle Stationen können m_1 vor m_0 ausliefern. Hätte der AP vor dem Broadcast der SDUs festgelegt, daß m_0 vor m_1 auszuliefern ist, dann hätte die Verzögerung von m_0 auch die Auslieferung von m_1 an allen Stationen verzögert.

4.5 Gruppenmitgliedschaft

Nachdem in den vorangegangenen Unterkapiteln erläutert wurde, wie die verschiedenen Eigenschaften rechtzeitiger, atomarer Broadcasts durch das Protokoll erreicht werden, wird in diesem Unterkapitel darauf eingegangen, wie das Protokoll Stationsausfälle erkennt und die Informationen über den aktuellen Status der Gruppe an die korrekten Stationen verteilt. Neben der Darstellung, wie das Protokoll die in 2.1.3 an ein Teilnehmerprotokoll gestellten Anforderungen erfüllt, wird in diesem Unterkapitel auch darauf eingegangen, wie sich der Ausfall von Stationen und die damit verbundene dynamische Änderung der Gruppe auf die Realisierung der atomaren Broadcasts auswirken.

Die Aufgabe des AP im Rahmen des Teilnehmerprotokolls kann in zwei Teilaufgaben gegliedert werden:

1. Das Erkennen von Stationsausfällen.

2. Das Verteilen der durch die Ausfallerkennung gewonnenen Information an die korrekten Stationen derart, daß das Ausliefern der Änderungsnachrichten an den Stationen den geforderten Eigenschaften genügt.

Das vorliegende Unterkapitel ist entsprechend dieser Zweiteilung gegliedert, wobei zunächst auf das Erkennen von Stationsausfällen eingegangen wird.

4.5.1 Das Erkennen von Stationsausfällen

Aufgrund der über das System getroffen Annahmen (s. Unterkapitel 2.2) ist es möglich, einen Mechanismus zum Erkennen von Stationsausfällen zu realisieren, der sowohl lebendig als auch genau ist, d. h. der sowohl ausgefallene Stationen nach endlicher Zeit als solche erkennt als auch keine korrekten Stationen als ausgefallen betrachtet. Es sei an dieser Stelle nochmals darauf hingewiesen, daß auch eine Station, die sich aus dem Empfangsbereich des AP entfernt hat, als ausgefallen betrachtet wird, weil der AP diese Situation nicht von einem Ausfall unterscheiden kann.

In verteilten Systemen kann das Verhalten einer Station nur aufgrund der von dieser Station gesendeten Nachrichten beobachtet und beurteilt werden. Dementsprechend kann auch der Totalausfall einer Station nur aufgrund empfangener bzw. nichtempfangener Nachrichten diagnostiziert werden. Um feststellen zu können, daß eine Station Nachrichten nicht gesendet hat, muß bekannt sein, daß und wann diese Station bestimmte Nachrichten senden muß. Das Ausbleiben dieser Nachrichten kann dann erkannt und als Indiz für einen Ausfall gewertet werden. Das hier beschriebene Teilnehmerprotokoll nutzt das Wissen des AP, daß eine Station, die er gepollt hat, innerhalb von $2 \times t_m$ Zeiteinheiten mit einer Request-Nachricht geantwortet haben muß, um den Ausfall von Stationen festzustellen. Wenn der AP nach dem Pollen einer Station nicht die erwartete Request-Nachricht empfängt, kann dies außer dem Ausfall der Station auch einen Auslassungsausfall des Funknetzes zur Ursache haben, so daß der AP nicht unmittelbar auf den Ausfall der Station schließen kann. Die getroffenen Annahmen stellen aber sicher, daß der AP von einer korrekten Station spätestens dann eine Request-Nachricht empfangen muß, wenn er diese $OD+1$-mal gepollt hat. So kann der AP sicher auf den Ausfall einer Station schließen, wenn er diese $OD+1$-mal gepollt, aber für keine dieser Polling-Nachrichten eine Request-Nachricht empfangen hat. Da der AP die Request-Nachrichten nutzt, um den Ausfall von Stationen festzustellen, und da die Stationen diese Request-Nachrichten ohnehin an den AP senden müssen, benötigt das Teilnehmerprotokoll keine zusätzlichen Nachrichten zum Erkennen der Stationsausfälle.

Konkret funktioniert das Protokoll wie folgt (s. Abb 4.11): Bevor der AP eine Station s pollt, setzt er ein Timeout der Länge $2 \times t_m$. Wenn er nach dem Ablauf dieses Timeouts keine Request-Nachricht empfangen hat, inkrementiert er den Zähler der nichtempfangenen Request-Nachrichten dieser Station ($s.l$). Wenn er eine Request-Nachricht empfängt, dann setzt er den Zähler auf Null zurück. Erreicht der Zähler den Wert $OD+1$, dann weiß der AP, daß die betreffende Station ausgefallen ist.

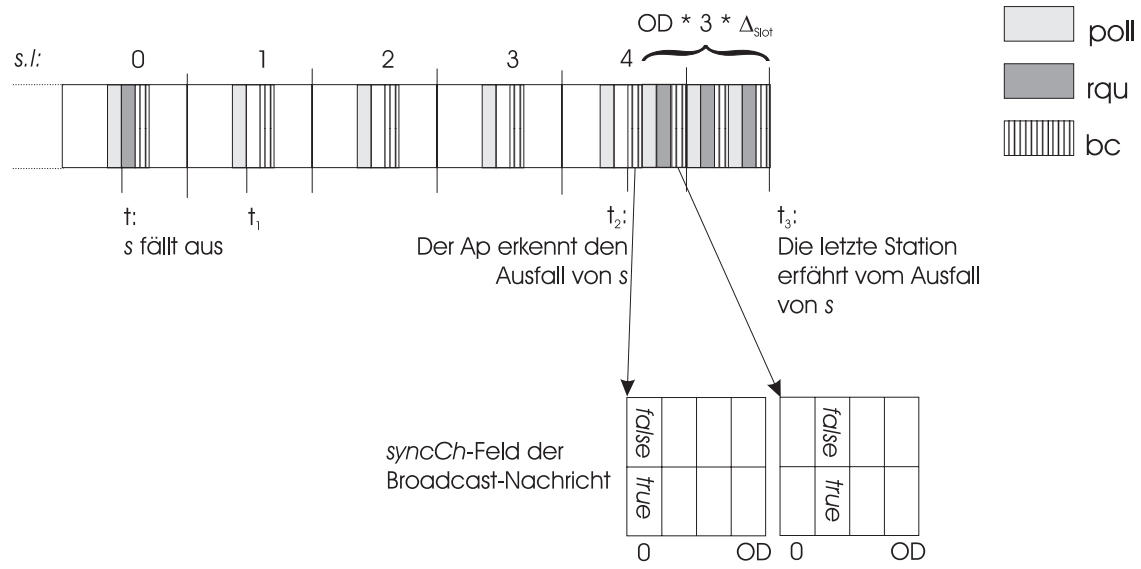


Abbildung 4.11 Ausschluß einer Station aus der Gruppe (*OD=3*).

In Abbildung 4.11 ist die Situation dargestellt, in der das Erkennen eines Ausfalls die maximale Zeit in Anspruch nimmt. In dieser Situation fällt die Station *s* aus, unmittelbar nachdem sie eine Request-Nachricht erfolgreich an den AP übertragen hat (Zeitpunkt *t* in Abb. 4.11). Eine Runde später wird der AP die Station erneut pollen (Zeitpunkt t_1 in Abb. 4.11) und nach dem Ablauf des Timeouts *s.l* inkrementieren. Weitere *OD* Runden später (Zeitpunkt t_2 in Abb. 4.11) zählt der AP den *OD+1*sten Verlust und stellt damit den Ausfall von *s* fest. Zwischen dem Ausfall der Station und dem Zeitpunkt, an dem der AP diesen feststellt, vergehen folglich maximal $(OD + 1) \times \Delta_{Round} + 2 \times t_m$ Zeiteinheiten. Das Protokoll stellt demnach sicher, daß der AP alle Stationsausfälle mit einer bekannten maximalen Verzögerung erkennt. Aufgrund der Systemannahmen aus Kapitel 2 wird mindestens eine von *OD+1* aufeinanderfolgenden Polling-Nachrichten des AP an die selbe Station beantwortet, sofern die betreffende Station nicht ausgefallen ist. Da der AP eine Station nur dann aus der Gruppe ausschließt, wenn diese keine von *OD+1* aufeinanderfolgenden Polling-Nachrichten beantwortet hat, schließt der AP also nur ausgefallene Stationen aus der Gruppe aus.

4.5.2 Das Verteilen der Änderungsinformationen

Der AP benutzt den synchronen Kanal, um die in der Gruppe verbleibenden Stationen über den Ausfall einer Station zu informieren. Dies hat drei wesentliche Vorteile:

1. Die starken Eigenschaften des synchronen Kanals bzgl. Zuverlässigkeit, Einigung, Ordnung und Rechtzeitigkeit können für das Teilnehmerprotokoll verwendet werden.

2. Da der synchrone Kanal für das Broadcast-Protokoll bereits realisiert wurde und eine der vier in diesem Kanal möglichen Bit-Kombinationen vom Broadcast-Protokoll nicht genutzt wird, kann der synchrone Kanal ohne jeglichen Mehraufwand verwendet werden.

3. Da die Tupel im synchronen Kanal total geordnet übertragen werden und der synchrone Kanal von Broadcast- und Teilnehmerprotokoll gemeinsam genutzt wird, läßt sich die virtuelle Synchronität gut realisieren.

Wenn der AP den Ausfall einer Station *s* feststellt, so geschieht dies immer in einem Slot, in dem der AP die Station *s* gepollt hat und unmittelbar bevor der AP eine Broadcast-Nachricht versendet. Der AP fügt in das *synchCh*-Feld dieser Broadcast-Nachricht an die Position 0 den Tupel *(false, true)* (Ausschluß-Entscheidung) ein und teilt den Stationen auf diese Weise mit, daß der Eigentümer dieses Slots ausgefallen ist (s. Abb. 4.11). Diese Entscheidung wird wie die anderen Entscheidungen in genau *OD+1* Broadcast-Nachrichten übertragen und daher von jeder korrekten Station empfangen.

Wenn eine Station *s* eine Broadcast-Nachricht empfängt, die eine Ausschluß-Entscheidung enthält, dann kann sie, wie in Paragraph 4.3.3 beschrieben, feststellen, in welcher Broadcast-Nachricht die Ausschluß-Entscheidung zum ersten Mal versendet wurde, und daraufhin, welche Station Eigentümer des betreffenden Slots war. Die so ermittelte Station *s′* ist die Station, auf die sich die Ausschluß-Entscheidung des AP bezieht. Daher liefert die Station *s* eine Änderungsnachricht aus, die dem Benutzer den Ausfall von *s′* anzeigt. Weiterhin wird Station *s* Station *s′* aus ihrer aktuellen Mitgliedschaft entfernen, wobei alle Stationen mit einer größeren Id als *s′* eine Position nach vorne rücken.

Bisher wurde davon ausgegangen, daß eine Station den Eigentümer eines Slots *sg* ermitteln kann. Wie in Paragraph 4.2.2 erwähnt, kann der Eigentümer von Slot *sg* durch *g*[*sg mod g.size*] bestimmt werden, solange keine Änderungen der Gruppe auftreten. Aber auch unter der Annahme einer dynamischen Gruppe können die Stationen die Eigentümer von Slots ohne großen Aufwand bestimmen. Dies liegt daran, daß die Stationen, wenn sie die Entscheidungen des AP im synchronen Kanal bearbeiten, nachvollziehen können, welche Stationen der AP gepollt hat und welche Stationen der AP aus der Gruppe ausgeschlossen hat. Nach der Initialisierung kennen alle Stationen die aktuelle Gruppe g_0 und sie wissen, daß der Eigentümer von Slot 0 die Station $g_0[0]$ ist. Wenn nun eine Station *s* eine Entscheidung bearbeitet, die der AP in Slot *sg* getroffen hat, und sie weiß, daß zu diesem Zeitpunkt die Mitgliedschaft des AP g_{sg} war und daß der Eigentümer von Slot *sg* die Station *s′* ist, dann gibt es nur zwei Möglichkeiten:

1. Bei der Entscheidung handelt es sich um eine Ausschluß-Entscheidung: Folglich ist die Station *s′* ausgefallen und der AP hat sie aus der Gruppe ausgeschlossen. Entsprechend kann auch Station *s* Station *s′* aus der Mitgliedschaft ausschließen und feststellen, welche Station der AP in Slot *sg+1* pollt.

2. Es handelt sich nicht um ein Ausschluß-Entscheidung. Die Gruppe bleibt unverändert und *s* kann feststellen, welche Station der AP in Slot *sg+1* pollt.

Man sieht also, daß jede Station beginnend bei der Ausgangsgruppe alle Entscheidungen des AP nachvollziehen und für jeden Slot, für den sie eine Entscheidung des AP bearbeitet, den Eigentümer bestimmen kann. Da jede Station, die eine Broadcast-Nachricht des AP mit Sequenznummer *sg* empfängt und die Tupel im *syncCh*-Feld dieser Broadcast-Nachricht bearbeitet, alle Entscheidungen des AP von Slot 0 bis Slot *sg* bearbeitet hat, kennt sie auch den Eigentümer von Slot *sg*. Sie weiß aber auch, da sie für jeden der Slots 0 bis *sg* den Eigentümer bestimmt hat, wann sie zuletzt gepollt wurde und welches die aktuelle Rundennummer ist.

Der AP kann jeden Stationsausfall höchstens $(OD + 1) \times \Delta_{Round} + 2 \times t_m$ Zeiteinheiten nach dem Ausfall erkennen. Der AP erkennt den Ausfall, nachdem er die ausgefallene Station gepollt hat und unmittelbar bevor er eine Broadcast-Nachricht versendet (Zeitpunkt t_2 in Abb. 4.11). Daher kann er die Information über den Ausfall der Station direkt in den synchronen Kanal einfügen und broadcasten. Einen Broadcast und *OD* Slots später (Zeitpunkt t_3 in Abb. 4.11) hat der AP diese Entscheidung zum *OD+1*sten mal übertragen. Sie wird daher spätestens zu diesem Zeitpunkt von der letzten Station empfangen. Insgesamt stellt der AP also sicher, daß spätestens

$$(OD + 1) \times \Delta_{Round} + (OD + 1) \times \Delta_{Slot}$$

Zeiteinheiten nach einem Stationsausfall jede korrekte Station eine Ausschluß-Entscheidung erhalten hat. Wie oben erläutert, werden alle Stationen die Station, auf die sich diese Entscheidung bezieht, richtig bestimmen und eine entsprechende Änderungsnachricht ausliefern.

Da alle korrekten Stationen alle Entscheidungen des AP im synchronen Kanal in der gleichen Reihenfolge empfangen und bearbeiten, liefern alle Stationen die Änderungsnachrichten in der gleichen Reihenfolge aus. Da weiterhin die Ausschluß-Nachrichten im synchronen Kanal zwischen den *accept*-Entscheidungen des AP eingeordnet sind, impliziert die total geordnete Bearbeitung der Entscheidungen des AP auch, daß alle Stationen Gruppenänderungen an der gleichen Stelle zwischen den *accept*-Entscheidungen bearbeiten. Daher werden Änderungsnachrichten bei allen Stationen zwischen den gleichen SDUs ausgeliefert. So ist durch die totale Ordnung der Entscheidungen im synchronen Kanal auch die virtuelle Synchronität der Gruppenänderungen sichergestellt.

Stationen, die der AP aus der Gruppe ausgeschlossen hat, werden vom AP nicht mehr gepollt. Sie haben also keine Möglichkeit, weitere Nachrichten in der Gruppe zu versenden, ehe sie sich nicht in der nächsten CFP wieder in der Gruppe angemeldet haben. Nach dem Ausschluß einer Station können also alle Stationen sicher sein, daß keine weiteren Nachrichten der ausgefallenen Station zu erwarten sind. Wenn der AP eine Station aus der Gruppe ausschließt, dann ist diese entweder ausgefallen oder sie hat sich

aus dem Sende-/Empfangsbereich des AP entfernt. Diese Station kann folglich weder die Entscheidung des AP, die ihren Ausfall anzeigt, noch eine der folgenden Entscheidungen bearbeiten. Dies bedeutet für die in der Gruppe verbleibenden Stationen, daß jede Nachricht, die nach dem Ausschluß einer Station *s* ausgeliefert wird, definitiv nicht von *s* ausgeliefert wird.

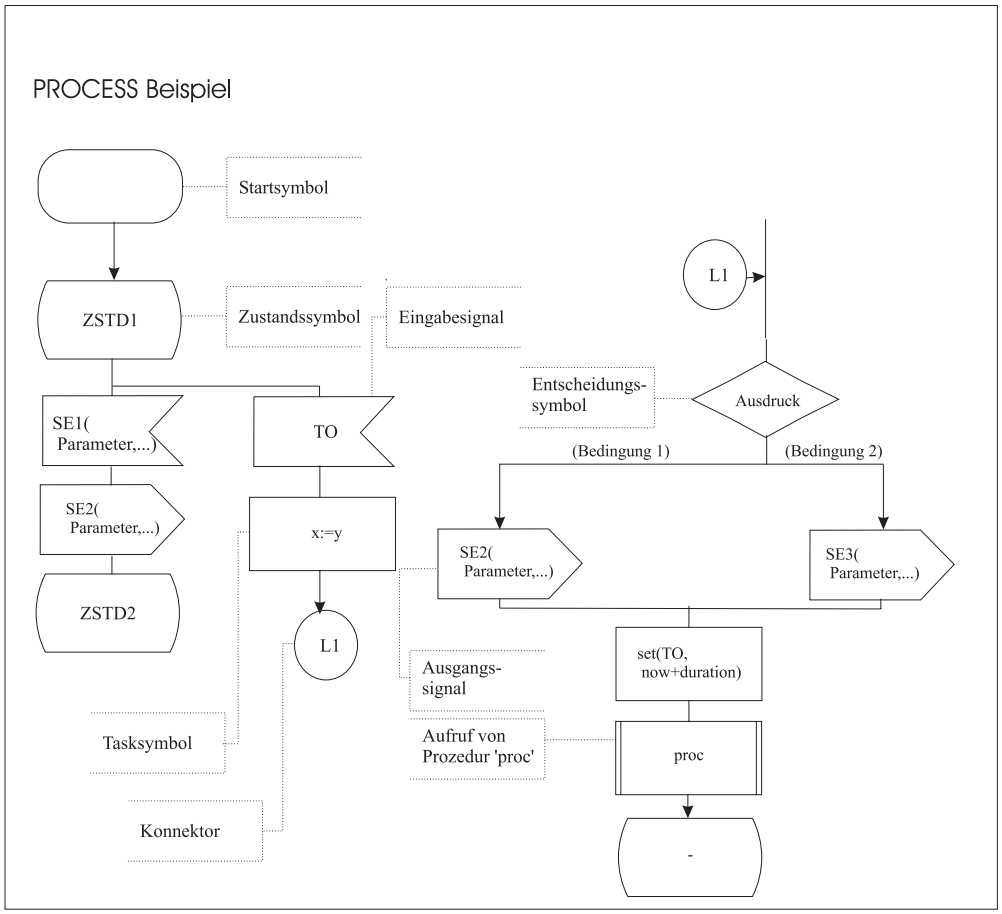
5 Formale Darstellung des Protokolls

In diesem Kapitel wird das in Kapitel 4 vorgestellte Protokoll in Form eines endlichen Automaten formal dargestellt. Die graphischen Darstellungsmittel, mit denen die Protokollautomaten beschrieben werden, sind der Spezifikationssprache SDL entnommen und werden im ersten Unterkapitel kurz erläutert. Unter Verwendung dieser Darstellungsmittel werden in den Unterkapiteln 5.3 und 5.4 der AP- sowie der Stationsautomat formal beschrieben. Aufbauend auf der formalen Beschreibung der Automaten wird im Kapitel 5.5 gezeigt, daß das Protokoll die geforderten Eigenschaften erfüllt.

5.1 Verwendete Darstellungsmittel

Das Protokoll stellt ein System dar, das mit seiner Umgebung, den Benutzern, Nachrichten (bzw. Signale) austauscht, beispielsweise dann, wenn ein Benutzer dem Protokoll eine SDU zum Übertragen übergibt. Das Verhalten bzw. der Service des Protokolls kann anhand dieser Nachrichten beschrieben werden (Unterkapitel 2.1). Intern kann das System in Komponenten gegliedert werden, die selbst wieder Systeme sind (vgl. Paragraph 2.2.1). Für das vorgestellte Protokoll bestehen diese Komponenten aus dem Funknetzwerk und den Instanzen des Gruppenkommunikationsprotokolls in den einzelnen Stationen. Jede Protokollinstanz kann aus einem oder mehreren Prozessen bestehen. Das Verhalten des Systems Funknetzwerk wurde in Paragraph 2.2.3 (und wird nochmals in Unterkapitel 5.4) beschrieben. Das Verhalten der Protokollinstanzen ergibt sich aus dem Verhalten der Prozesse, die diese Instanzen bilden. Das Verhalten dieser Prozesse kann in Form von erweiterten endlichen Automaten beschrieben werden.

Ein erweiterter endlicher Automat verfügt über eine endliche Menge von Zuständen, die er einnehmen, einer endlichen Menge von Eingabesignalen, die er bearbeiten, und einer endlichen Menge von Ausgabesignalen, die er erzeugen kann. Zu jedem Zeitpunkt ist der Automat entweder in einem bestimmten Zutand, oder er befindet sich im Übergang zwischen zwei Zuständen. Solche Zustandsübergänge werden durch die Eingabesignale ausgelöst. Während eines Zustandsübergangs kann der AP Ausgabesignale erzeugen. Im Gegensatz zum herkömmlichen endlichen Automaten verfügt der erweiterte endliche Automat auch über Variablen, die einen unendlichen Wertebereich haben können. Der Gesamtzustandsraum des Automaten, der sich als Kreuzprodukt aus der Menge der Zustände und den Wertebereichen der Variablen ergibt, ist daher nicht mehr endlich. Zustandsübergänge im erweiterten endlichen Automaten können nicht nur vom aktuellen Zustand und Eingabesignal, sondern auch von Bedingungen bzgl. der Variablen abhängig gemacht werden. Während des Zustandsüberganges können die Variablen ihre Werte verändern.



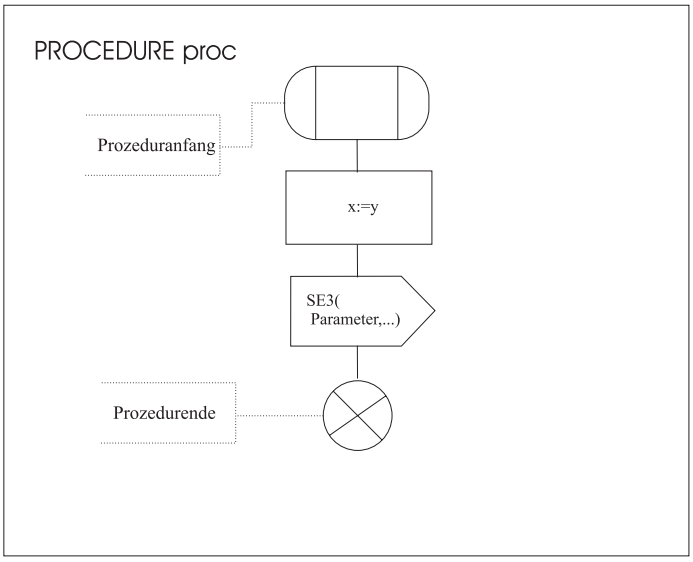


Abbildung 5.1 Symbole in SDL/GR (Die gestrichelten Symbole stellen Kommentare dar).

SDL ist eine standardisierte Spezifikationssprache für Kommunikationsprotokolle (ITU-T Recommendation Z.100), die auf der Beschreibung von Protokollen als kommunizierenden Prozessen und von Prozessen als erweiterten endlichen Automaten basiert. Die Darstellungsmittel von SDL/GR (s. z. B. [Mar97], [IEEE97]), der graphischen Syntax von SDL, sind für die graphische Darstellung von erweiterten endlichen Automaten gut geeignet. Sie werden daher für die formale Beschreibung des Protokolls verwendet. Dies bedeutetet aber nicht, daß die folgende formale Beschreibung eine vollständige und standardkonforme Spezifikation des Protokolls in SDL/GR darstellt.

Im Folgenden sollen die Darstellungsmittel von SDL/GR, die für die Beschreibung des Protokolls verwendet werden, kurz vorgestellt werden.

Systeme, Blöcke, Prozesse und Signalkanäle

In SDL wird ein System in Blöcke und Blöcke in weitere Blöcke oder schließlich in Prozesse gegliedert. Die Interaktion von Blöcken erfolgt über Signalkanäle, über die die Blöcke Signale austauschen können. Die Interaktion von Prozessen erfolgt über Signalwege. Signale können Parameter haben, die vom Sender des Signals an den Empfänger übertragen werden.

In Abbildung 5.2 ist auf diese Weise die Struktur des Protokolls dargestellt.

Zustände, Eingangs- und Ausgangssignale

Das grundlegende Mittel zum Darstellen von endlichen Automaten bilden Symbole für Zustände, Eingabesignale und Ausgabesignale. Durch die Verbindung dieser Symbole können die Zustandsübergänge endlicher Automaten beschrieben werden. In Abbildung 5.1 bezeichnet z. B. der linke Ast den Zustandsübergang, den der Automat durchführt, wenn er in Zustand *ZSTD1* ist und das Eingabesignal *SE1(...)* empfängt. In diesem Falle gibt er das Signal *SE2* aus und wechselt in den Zustand *ZSTD2*. Durch das Anfügen weiterer Symbole für Eingangssignale an Zustand *ZSTD1* können weitere Zustandsübergänge beschrieben werden. Das Symbol für Ausgabesignale muß nicht in jeder Transition vorhanden sein. Ebenso müssen nicht nach jedem Zustand alle möglichen Eingabesignale angegeben werden. Empfängt der Automat ein Signal, für das kein Übergang explizit definiert ist, so geht der Automat unmittelbar in den Ausgangszustand zurück, d. h. das Signal wird ignoriert.

Das Symbol vor Zustand *ZSTD1* ist das Startsymbol. Der Übergang vom Startsymbol zu Zustand *ZSTD1* wird unmittelbar nach dem Start des Prozesses durchgeführt. Aus Gründen der Eindeutigkeit darf es daher in jedem Prozeß nur genau ein Startsymbol geben. Ein weiteres spezielles Symbol ist das Zustandssymbol mit der Zustandsbezeichnung ‚–'. Wird dieses Symbol als Folgezustand verwendet, so bedeutet dies, daß der Automat in den Ausgangszustand der Transition zurückkehrt.

Alternativen

In erweiterten endlichen Automaten können auch Variablen verwendet und Zustandsübergänge von Bedingungen abhängig gemacht werden. In SDL können Variablentypen definiert und Variablen deklariert werden. In der vorliegenden

definiert und Variablen deklariert werden. In der vorliegenden Beschreibung wird auf eine formale Deklaration der Variablen verzichtet. Statt dessen wird die Struktur und Bedeutung der verschiedenen Variablen erläutert.

Um Zustandsübergänge von Bedingungen bzgl. der Variablen abhängig zu machen, wird das in Abbildung 5.1 dargestellte Entscheidungssymbol verwendet. Hinter dem Entscheidungssymbol verzweigt sich die Symbolfolge, wobei jeder Ast mit einer Bedingung bzgl. des Wertes von ‚Ausdruck' bezeichnet wird. Die Bedingungen müssen so gewählt werden, daß immer genau ein Ast gewählt werden kann, der den Fortgang des Zustandsüberganges beschreibt.

Aktionen

Im Rahmen eines Zustandsüberganges können neben der Ausgabe von Signalen auch Wertzuweisungen an die Variablen stattfinden. Diese werden durch die ‚:='-Notation im Tasksymbol (das einfache quadratische Symbol in Abb. 5.1) dargestellt.

Spezielle Variablen sind Timeouts. Sie können durch das Ausführen der Aktion *set(TO, value)* (ebenfalls in einem Tasksymbol) auf den absoluten Zeitpunkt *value* gesetzt werden. Um auch relative Zeitwerte verwenden zu können, kann auf die aktuelle Zeit durch die Variable *now* zugegriffen werden. Wenn der Zeitpunkt *value* erreicht ist, wird ein Signal *TO* erzeugt. In Abbildung 5.1 kann in Zustand *ZSTD1* das Timeoutsignal *TO* verarbeitet werden, das erzeugt wird, wenn Timeout *TO* gesetzt wird und abläuft. Durch die Aktion *reset(TO)* wird ein noch nicht abgelaufener Timeout gestoppt bzw. ein noch nicht konsumiertes *TO*-Signal konsumiert.

Aktionen können zu Unterprogrammen zusammengefaßt werden. Der Aufruf eines Unterprogramms erfolgt durch das in Abbildung 5.1 dargestellte Symbol für den Unterprogrammaufruf. Zur Definition des Unterprogramms können die Ausgabe-, Entscheidungs- und Tasksymbole verwendet werden. Die Definition beginnt und endet mit einem speziellen Symbol (‚Prozeduranfang' und ‚Prozedurende', s. Abb. 5.1).

Konnektorsymbole können verwendet werden, um Verbindungen zwischen Punkten herzustellen, ohne eine Verbindungslinie einzuzeichnen. Beispielsweise stellen die 'L1' benannten Konnektorsymbole in Abbildung 5.1 eine Verbindung zwischen dem linken und dem rechten Teil des Automaten her.

5.2 Struktur des Protokolls

Das System ist in drei Blöcke eingeteilt (s. Abb. 5.2), die jeweils ein unterschiedliches Verhalten beschreiben: Der Block ‚Access Point' beschreibt das Verhalten des AP und entsprechend die Blöcke ‚Station' und ‚Medium' das Verhalten der Stationen und des Mediums. Jeder der beiden Blöcke ‚Access Point' und ‚Station' ist durch genau einen Prozeß realisiert, der beim Start des Systems erzeugt und nie beendet wird. Das Verhalten der beiden Prozesse wird in den folgenden Unterkapiteln durch erweiterte Zustands-

automaten beschrieben. Der Block ‚Medium' wird als Blackbox betrachtet. Sein Service wird in diesem Unterkapitel beschrieben.

Zwischen den Blöcken ‚Access Point', ‚Medium' und ‚Station' sowie zwischen dem Block ‚Station' und der Umgebung des Systems werden über Signalkanäle Signale ausgetauscht. Da alle verwendeten Signalkanäle bidirektional sind, sind sie in Abbildung 5.2 als Doppelpfeile dargestellt. Die Annotation eines Signals an einer bestimmten Pfeilspitze gibt an, in welche Richtung dieses Signal transportiert wird. Das Protokoll verwendet die folgenden Signale:

A-BC(sdu, res): Mit diesem Signal beauftragt die Anwendung die lokale Protokollinstanz mit der Übertragung der SDU *sdu* an die Gruppe, wobei für *sdu* eine Resiliency von *res* gelten soll.

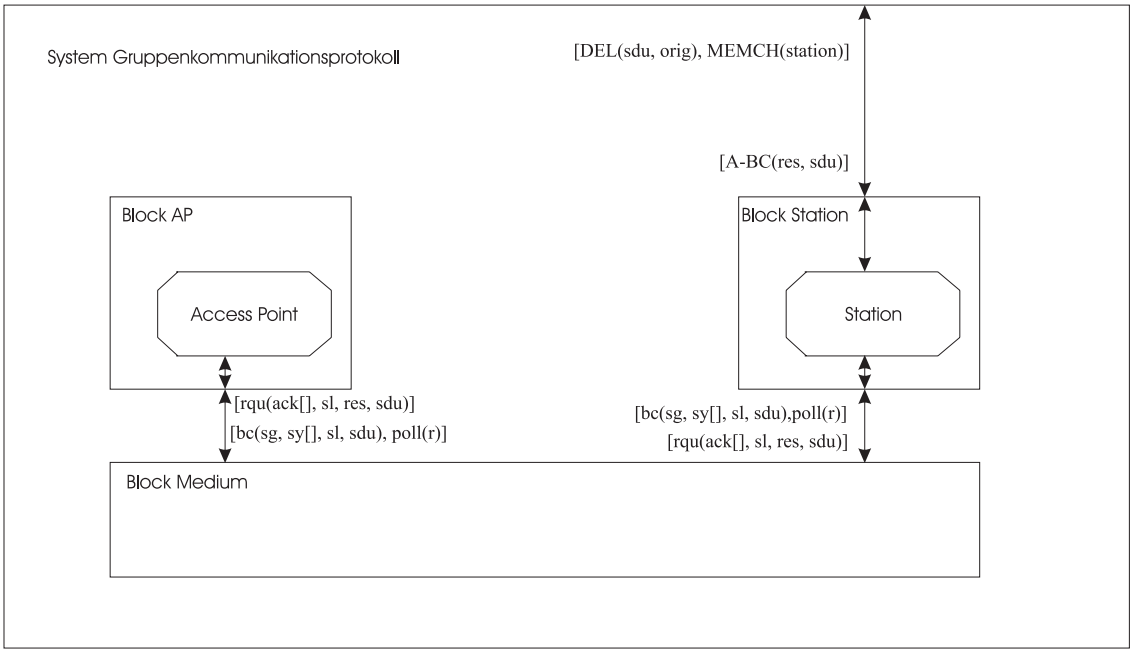


Abbildung 5.2 Struktur des Protokolls

DEL(sdu, orig): Mit diesem Signal liefern die Instanzen des Stationsautomaten eine SDU *sdu* von Station *orig* aus.

MEMCH(station): Mit diesem Signal liefern die Instanzen des Stationsautomaten Gruppenänderungen an die jeweilige Anwendung aus und geben damit bekannt, daß Station *station* ausgefallen ist.

rqu(ack[]*, sl, res, sdu)*: Mit diesem Signal übertragen die Stationen Request-Nachrichten an den AP. Dabei ist:

ack[]: Ein Vektor der Länge *n* von Booleans (Werte *true* und *false*)

sl: Die lokale Sequenznummer, die der Urheber für *sdu* vergeben hat.

res: Die Resiliency der SDU.

sdu: Die SDU.

poll(r): Das Signal, mit dem der AP die Stationen pollt. *r* ist die aktuelle Rundennummer.

bc(sg, sy[]*, sl, sdu)*: Das Signal, das der AP als Broadcast an alle Stationen sendet. Dabei ist:

sg: Die globale Sequenznummer, die der AP fortlaufend für jede Broadcast-Nachricht vergibt.

sy[]: Ein Vektor der Länge *OD+1* von Entscheidungen. Jede Entscheidung kann einen der vier Werte *no decision*, *accept*, *reject*, *exclude* annehmen.

sl: Die lokale Sequenznummer von *sdu*.

sdu: Die Nachricht, die an die Gruppe übertragen werden soll.

Wenn solche Signale versendet werden, dann wird die Adressierung da, wo sie aus dem Kontext eindeutig hervorgeht, weggelassen. Dies bezieht sich auf die Signale *DEL* und *MEMCH*, die immer an die lokale Anwendung adressiert sind, *rqu*, das immer an den AP-Automaten adressiert ist, sowie *bc*, das immer an alle Stationsautomaten adressiert ist. Folglich muß nur für das Signal *poll* die Adresse der gepollten Station angegeben werden.

Der Dienst, den der Block ‚Medium' für die Übertragung der Signale *poll*, *rqu* und *bc* zur Verfügung stellt, genügt den folgenden Eigenschaften (Die Signale, die über das Medium ausgetauscht werden, werden im folgenden als Nachrichten bezeichnet).

(Medium 1): *Integrität:* Für alle Stationen *s* und jede Nachricht *m* gilt: *s* empfängt *m* höchstens einmal und nur dann, wenn *m* auch von einer Station gesendet wurde.

(Medium 2): *FIFO-Ordnung:* Wenn eine Station *s* zuerst die Nachricht m_1 und dann die Nachricht m_2 sendet, dann empfängt keine Station Nachricht m_2 vor Nachricht m_1.

(Medium 3): *Begrenzter Omission Degree:* Die Anzahl der Auslassungsausfälle des Netzwerkes ist begrenzt. Dies bedeutet für die Broadcast-Nachrichten (*bc*-Signale):

- Wenn der AP *OD + 1* Broadcast-Nachrichten sendet, die die gleiche SDU enthalten, dann empfängt jede korrekte Station mindestens eines dieser Signale.

- Von *OD+1* aufeinanderfolgenden Broadcast-Nachrichten des AP empfängt jede korrekte Station mindestens eine.

- Das Protokoll stellt sicher, daß jede korrekte Station eine Polling-Nachricht des AP mit genau einer Request-Nachricht beantwortet (s. unten). Für diese Paare von Polling- und Request-Nachrichten gilt:

  Von *OD+1* aufeinanderfolgenden Paaren von Polling- und Request-Nachrichten zwischen dem AP und einer Station *s* ist mindestens eines erfolgreich, d. h., sowohl die Polling-Nachricht des AP wird von der Station *s* als auch die Antwort von Station *s* vom AP empfangen.

(Medium 4): *Begrenzte Verzögerung:* Wenn eine Station eine Nachricht *m* zum Zeitpunkt *t* sendet, dann empfängt keine Station *m* nach dem Zeitpunkt $t + t_m$.

Die Signalkanäle, die die einzelnen Blöcke verbinden, sind zuverlässig und der Transport der Signale durch diese Kanäle benötigt keine Zeit.

5.3 Der AP-Automat

5.3.1 Datenstrukturen und Operationen

Der AP verwendet die folgenden Datenstrukturen:

sg: Die globale Sequenznummer. Der Startwert ist *0*

g[]: Ein Vektor von Stationen, in dem der AP die aktuelle Gruppe speichert. Die Operation $g := g/g[i]$ entfernt die *i*-te[14] Komponente aus dem Vektor und rückt alle folgenden um eine Position nach vorne. *g.size* bezeichnet die Anzahl der in *g* gespeicherten Stationen.

Zu jeder Station *s* verwaltet der AP die folgenden Attribute:

s.l: Die Anzahl aufeinanderfolgender nicht erfolgreicher Polling-Request-Paare zwischen dem AP und *s*.

s.id: Die Position von *s* in der aktuellen Gruppe, so daß $g[s.id]=s$.

s.aSDU: Die aktuelle SDU von *s*, d. i. die SDU von *s*, die der AP schon einmal oder mehrmals gesendet hat, für die er aber noch keine Entscheidung getroffen hat. Der Startwert ist ε (die leere SDU).

---

[14] Die Numerierung der Elemente von Feldern und der Zeilen und Spalten von Matrizen beginne im Folgenden bei 0.

s.aSDU.sdu: Die eigentliche SDU.

s.aSDU.res: Resiliency dieser SDU.

s.aSDU.sl: Lokale Sequenznummer dieser SDU.

s.tr: Gibt an wie oft der AP die aktuelle SDU bereits gebroadcastet hat. Der Startwert ist *0*.

s.nSDU: Die SDU von *s* die der AP schon empfangen hat, aber noch nicht überträgt. Der Startwert ist *ε*. Die Attribute *sdu*, *res* und *sl* entsprechen *s.aSdu*.

s.sl: Die lokale Sequenznummer der zuletzt von *s* empfangenen SDU. Der Startwert ist *0*.

s.lp: Die Sequenznummer des Slots, in dem der AP *s* zum letzen mal gepollt hat. Der Startwert ist *0*.

o: Die Id des Eigentümers des aktuellen Slots. Der Startwert ist *-1*.

r: Die aktuelle Rundennummer. Der Startwert ist *1*.

d: Die aktuelle Entscheidung des AP. *d* kann einen der Werte *accept*, *reject*, *exclude* oder *no decision* annehmen.

sy[]: Ein Vektor der Länge *OD+1* von Entscheidungen. *RotR(sy*[]*, 1)* bewegt jede Entscheidung um eine Position nach rechts und entfernt die Entscheidung an Position *OD* aus dem Vektor. Der Startwert ist *no decision* in allen Komponenten.

gacks[][]: Eine $n \times n$-Matrix, in der der AP die Bestätigungen speichert. *gacks*[*i*][*j*] nimmt den Wert *true* an, wenn die aktuelle SDU von Station *g*[*i*] von Station *g*[*j*] bestätigt wurde. *gacks*[*i*] bezeichnet den *i*-ten Zeilenvektor. $(gacks)_{i,i}$ die Matrix, die aus *gacks* durch das Entfernen der *i*-ten Zeile und der *i*-ten Spalte hervorgeht. Der Startwert ist *false* in allen Komponenten.

TP: Ein Timeout zum Erkennen von nicht erfolgreichen Polling-Request-Paaren.

Wenn einem Vektor ein Wert zugewiesen wird oder wenn ein Vektor mit einem Wert verglichen wird, so soll sich dies auf jedes seiner Elemente beziehen. Bei expliziten oder impliziten Zuweisungen von Größen mit mehreren Attributen werden die gleichnamigen Attribute zugewiesen; z. B. wird beim Senden von *bc(sg, sy*[]*, g*[*o*]*.aSDU)* der Parameter sdu von *bc* auf *g*[*o*]*.aSDU.sdu* und der Parameter *sl* von *bc* auf *g*[*o*]*.aSDU.sl* gesetzt.

5.3.2 Darstellung des AP-Automaten

Nach dem Start pollt der AP die erste Station der Gruppe (*g*[*0*], Prozedur ‚PollNextStation', s. Abb. 5.5). Anschließend wechselt der AP in den Zustand *Normal*, seinen einzigen Zustand. Aus diesem Zustand führt er Übergänge entweder aufgrund einer eintreffenden Request-Nachricht (Signal *rqu(pdu)*) oder des Timeoutsignals *TP* durch (s. Abb. 5.3). Dieses Timeout setzt er immer unmittelbar vor dem Senden der Polling-Nachricht (s. Abb. 5.5). Beide Zustandsübergänge enden in dem mit *l1* bezeichneten Ast (s. Abb.5.4). In diesem Ast trifft der AP eine Entscheidung bzgl. der aktuellen SDU der zuvor gepollten Station (Makro *EVAL_DEC*) und fügt diese in den synchronen Kanal ein (*sy*[*0*]:=*d*). Anschließend sendet der AP eine Broadcast-Nachricht an alle Stationen (Signal *bc*), inkrementiert die globale Sequenznummer, pollt die nächste Station und wechselt zurück in den Zustand *Normal*. Diese Aktionen werden vom AP während der gesamten Laufzeit des Protokolls wiederholt. Dabei wird die Zeitspanne zwischen dem Ausführen der Zuweisung *sg*:=*sg+1* und dem erneuten Ausführen dieser Zuweisung als ein Slot bezeichnet. Aufgrund der Struktur des Protokolls, kann jedem Slot und jeder Broadcast-Nachricht des AP eine eindeutige globale Sequenznummer *sg* zugeordnet werden.

Wenn der AP nach dem Pollen einer Station von dieser eine Request-Nachricht empfängt, dann wertet er die darin enthaltenen Bestätigungen für die Broadcast-Nachrichten der vorhergehenden Runde aus. Eine Station übermittelt Bestätigungen für jede Broadcast-Nachricht, die der AP gesendet hat, nachdem er die Station zum letzten Mal gepollt hat. Der AP merkt sich für jede Station den Slot, in dem er diese zuletzt gepollt hat (Attribut *lp*). Die Auswertung der Bestätigungen geschieht durch das Makro *UPGACKS*, das die folgende Zuweisung durchführt:

$\forall\ i \in [0,\ g.size\text{-}1]$: *gacks*[*i, o*] := *gacks*[*i, o*] `or` *ack*[*g*[*i*].*lp* – *g*[*o*].*lp*] .

In einem weiteren Makro *EVAL_DEC* bestimmt der AP seine Entscheidung bzgl. der aktuellen SDU der soeben gepollten Station (*g*[*o*].*aSDU*). *EVAL_DEC* nimmt den folgenden Wert an:

`if` *g*[*o*].*l*=*OD+1* `then` *exclude*
`else if` *aSDU*≠ε `and` (*gack*[*o*]=*true* `or` *g*[*o*].*tr*=*OD+1*) `then` *accept*
`else if` *aSDU*≠ε `and` *g*[*o*].*tr* < *OD+1* `and` *g*[*o*].*tr* ≥ *g*[*o*].*aSDU.res+1* `then` *reject*
`else` *no decision* `fi fi fi`

Wenn der AP bzgl. einer SDU eine Entscheidung außer *no decision* getroffen hat, wird er diese nie mehr in einer Broadcast-Nachricht versenden und er setzt daher *g*[*o*].*aSDU*:=ε. Bevor der AP anschließend eine Broadcast-Nachricht versendet, prüft er zunächst, ob er von der aktuellen Station *g*[*o*] eine SDU zu übertragen hat (*g*[*o*].*aSDU*≠ε oder *g*[*o*].*nSDU*≠ε). Nur wenn dieses nicht der Fall ist, überträgt er in der Broadcast-Nachricht eine leere SDU.

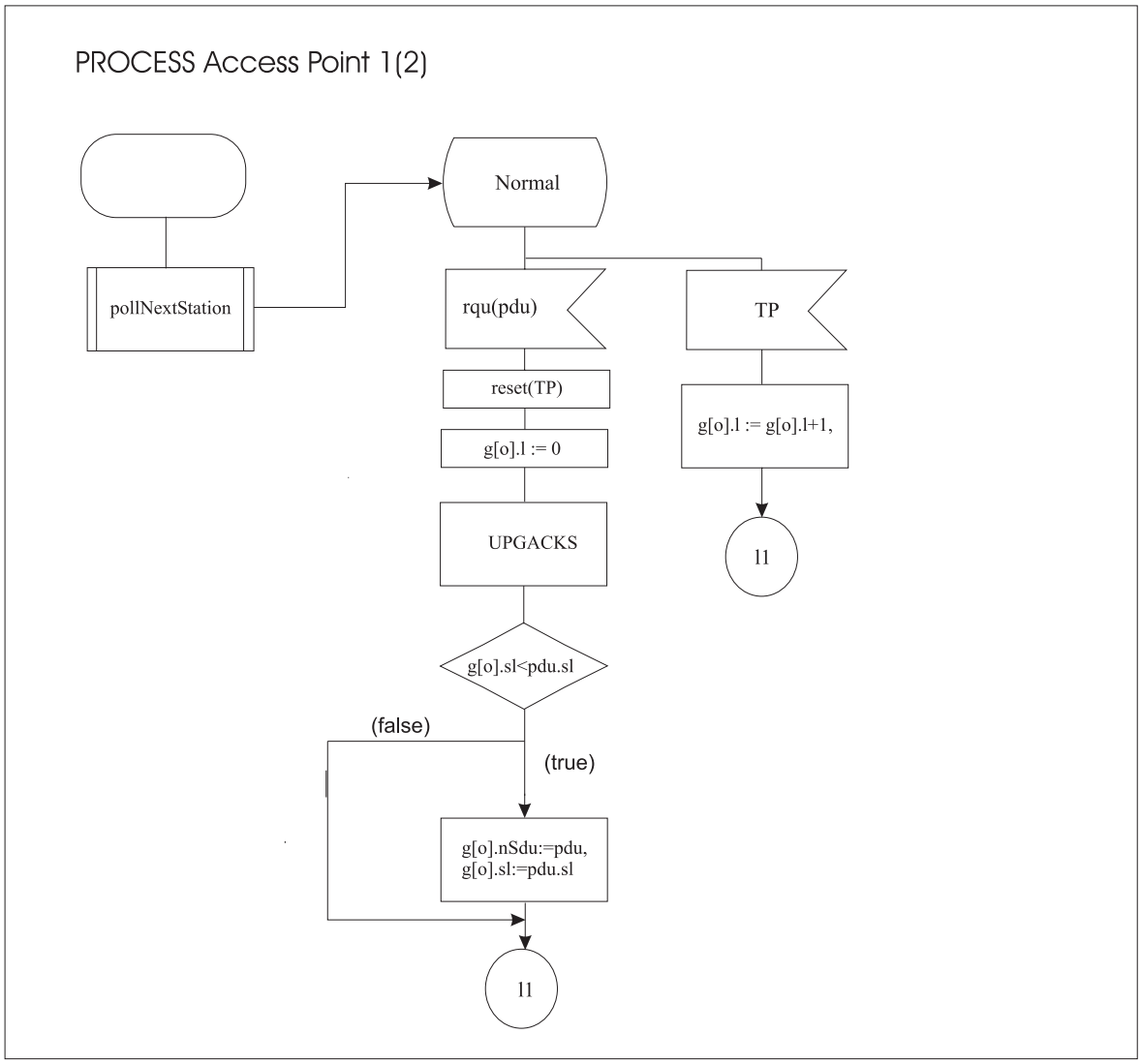


Abbildung 5.3 Darstellung des AP-Automaten (1. Teil)

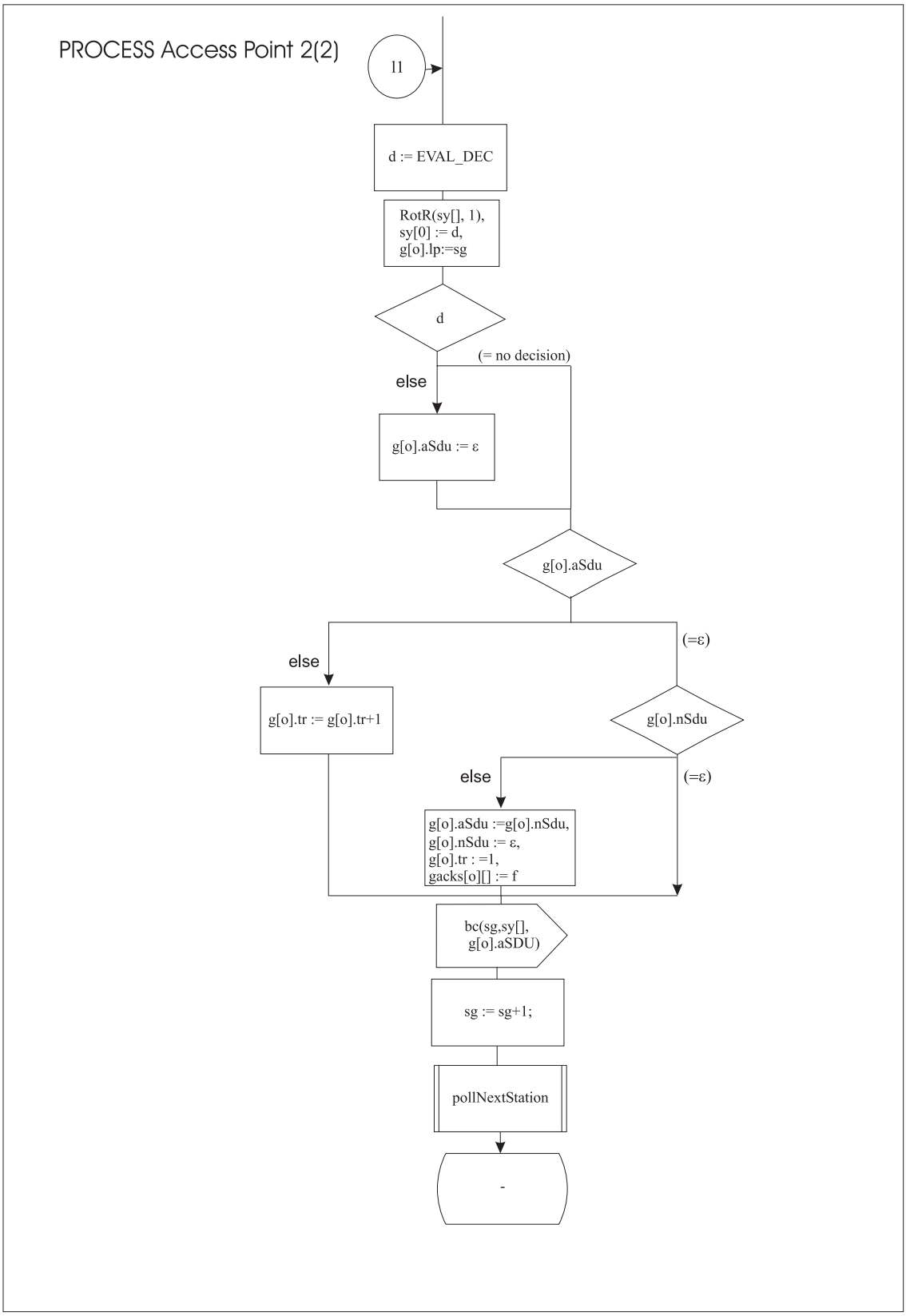


Abbildung 5.4 Darstellung des AP-Automaten (2. Teil)

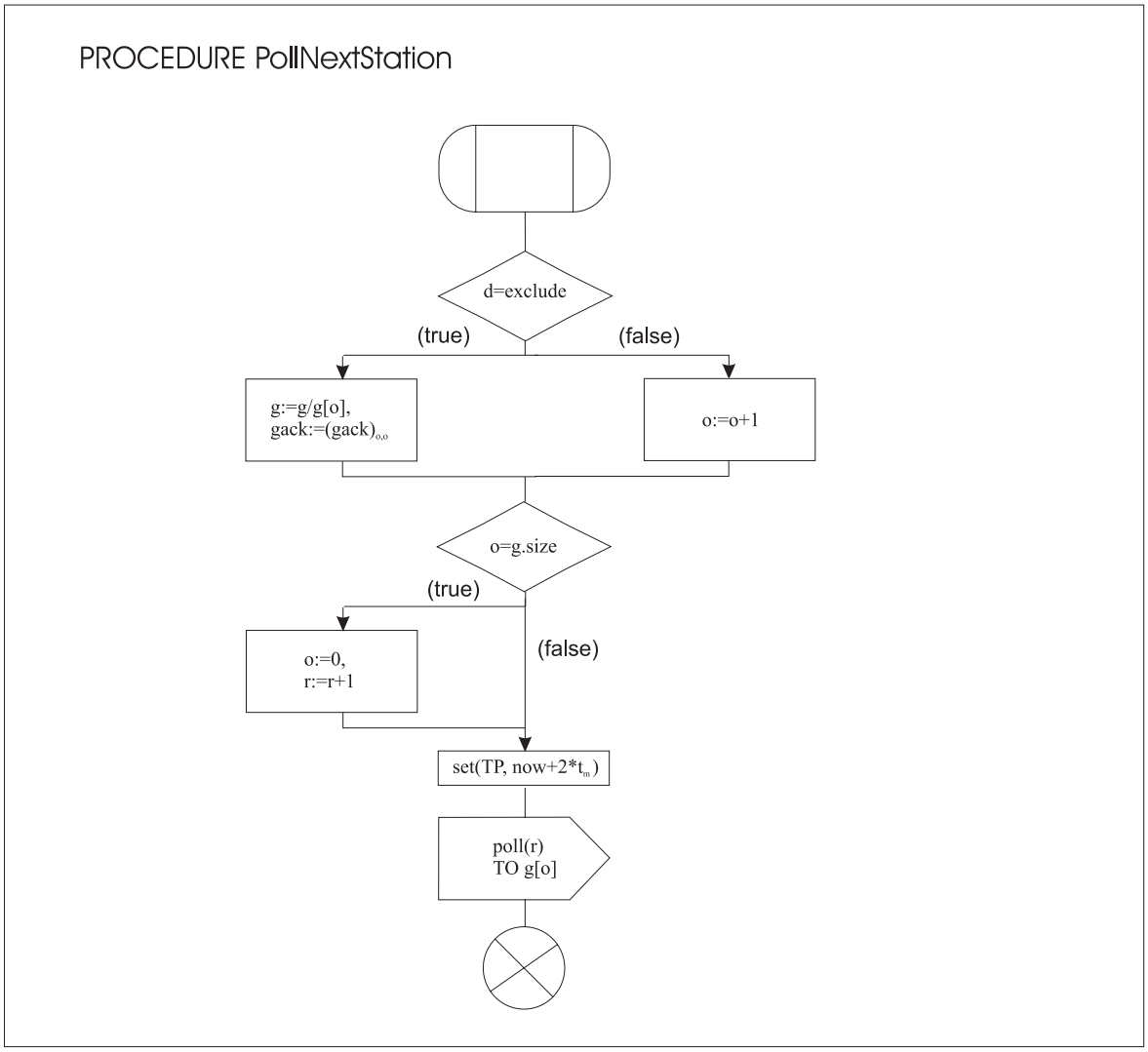


Abbildung 5.5 Darstellung des Unterprogramms ‚PollNextStation'

5.4 Der Stationsautomat

5.4.1 Datenstrukturen und Operationen

sg: In *EvDecs*: Sequenznummer der Broadcast-Nachricht, in der die gerade bearbeitete Entscheidung zum erstenmal enthalten war. Nach *EvDecs*: Sequenznummer der als nächstes erwarteten Broadcast-Nachricht. Der Startwert ist 0.

lp: Die Sequenznummer des Slots, in dem die Station zuletzt gepollt wurde. Der Startwert ist 0.

o: Id des Eigentümers des Slots *sg*. Der Startwert ist 0.

g[]: Ein Vektor von Stationen, in dem die Station die aktuelle Gruppe speichert. *g.size* ist die Anzahl der Stationen in diesem Vektor.

Jede Station verwaltet zu jedem Gruppenmitglied *s* das Attribut *s.bc*, die zuletzt in einer Broadcast-Nachricht empfangene SDU mit Urheber *s*.

nR: Nummer der Runde, in der die Station die nächste Polling-Nachricht erwartet. Der Startwert ist 1.

ack[]: Vektor der Länge *n* von Booleans, in dem die Station die Bestätigungen für empfangene Broadcast-Nachrichten speichert.

decide: Booleanvariable, die anzeigt, ob die Station eine Entscheidung bzgl. einer ihrer eigenen SDUs empfangen hat.

aSdu: Sdu, die sich in Bearbeitung befindet. Dabei ist

aSdu.sdu: Die SDU, die übertragen werden soll

aSdu.res: Deren Resiliency

sl: Die fortlaufende lokale Sequenznummer.

r: Wert des Parameters *r* der zuletzt empfangenen Polling-Nachricht.

lrts: Letzte Runde, in der die aktuelle SDU an den AP übertragen wird.

5.4.2 Darstellung des Stationsautomaten

Der Stationsautomat kann drei Zustände einnehmen: *Idle*, *Busy* und *Wait*. Im Zustand *Idle* kann der Stationsautomat das Signal *A-BC(m, res)* verarbeiten, d. h., er kann vom Benutzer Broadcast-Nachrichten zur Übertragung entgegen nehmen. Wenn der Automat ein solches Signal empfängt, wechselt er in den Zustand *Busy,* in dem er versucht, die SDU an den AP zu übertragen. Das heißt, im Zustand *Busy* beantwortet die Station Polling-Nachrichten des AP, indem sie Request-Nachrichten sendet, welche die aktuelle SDU enthalten. Wenn der Stationsautomat feststellt, daß die SDU am AP angekommen ist oder wenn die Station die SDU nicht mehr an den AP übertragen kann, weil die spezifizierte maximale Anzahl von Übertragungsversuchen *aSdu.res* erreicht ist, wechselt sie in den Zustand *Wait*. Im Zustand *Wait* verbleibt die Station solange, bis sie sicher ist, daß der AP die Bearbeitung ihrer aktuellen SDU abgeschlossen hat, was durch den Boolean *decide* angezeigt wird. Erst wenn dies gewährleistet ist, wechselt der Automat wieder in den Zustand *Idle*, in welchem er die nächste Broadcast-Nachricht des Benutzer erwartet.

In jedem der drei Zustände des Automaten löst der Empfang einer Polling-Nachricht (*poll(r)*) einen Zustandsübergang aus. Bei jedem dieser Zustandsübergänge sendet der Automat genau eine Request-Nachricht an den AP. Diese Request-Nachrichten haben drei Aufgaben: Sie dienen als Lebenszeichen der Stationen, sie enthalten die Bestätigungen für die Broadcast-Nachrichten der letzten Runde und sie transportieren die SDUs von den Stationen zum AP. Wenn der Automat eine SDU zum Übertragen hat (Zustand *Busy*), überträgt er diese in der Request-Nachricht; wenn nicht, sendet er eine Request-Nachricht, die nur eine leere SDU enthält.

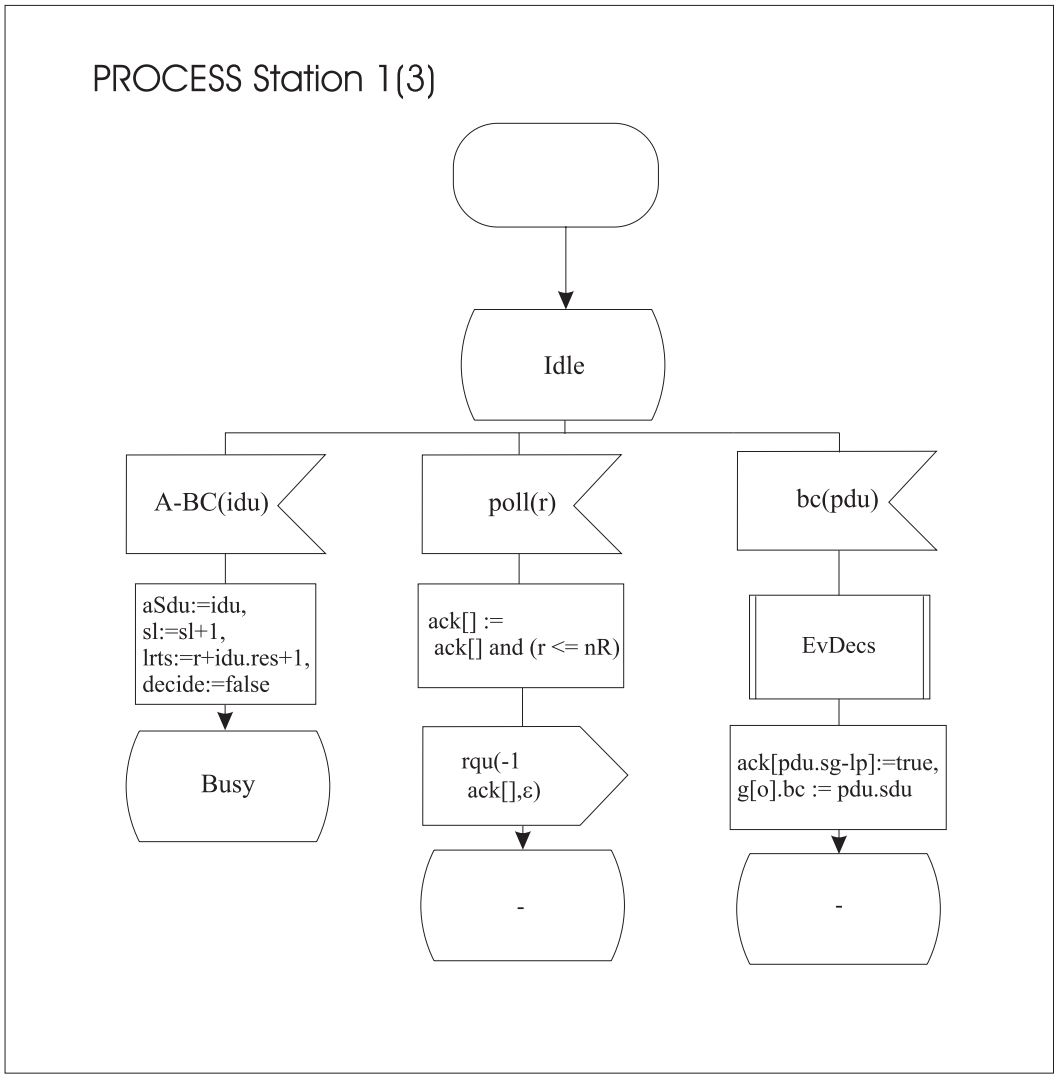


Abbildung 5.6 Darstellung des Stationsautomaten (Teil 1, Zustand *Idle*)

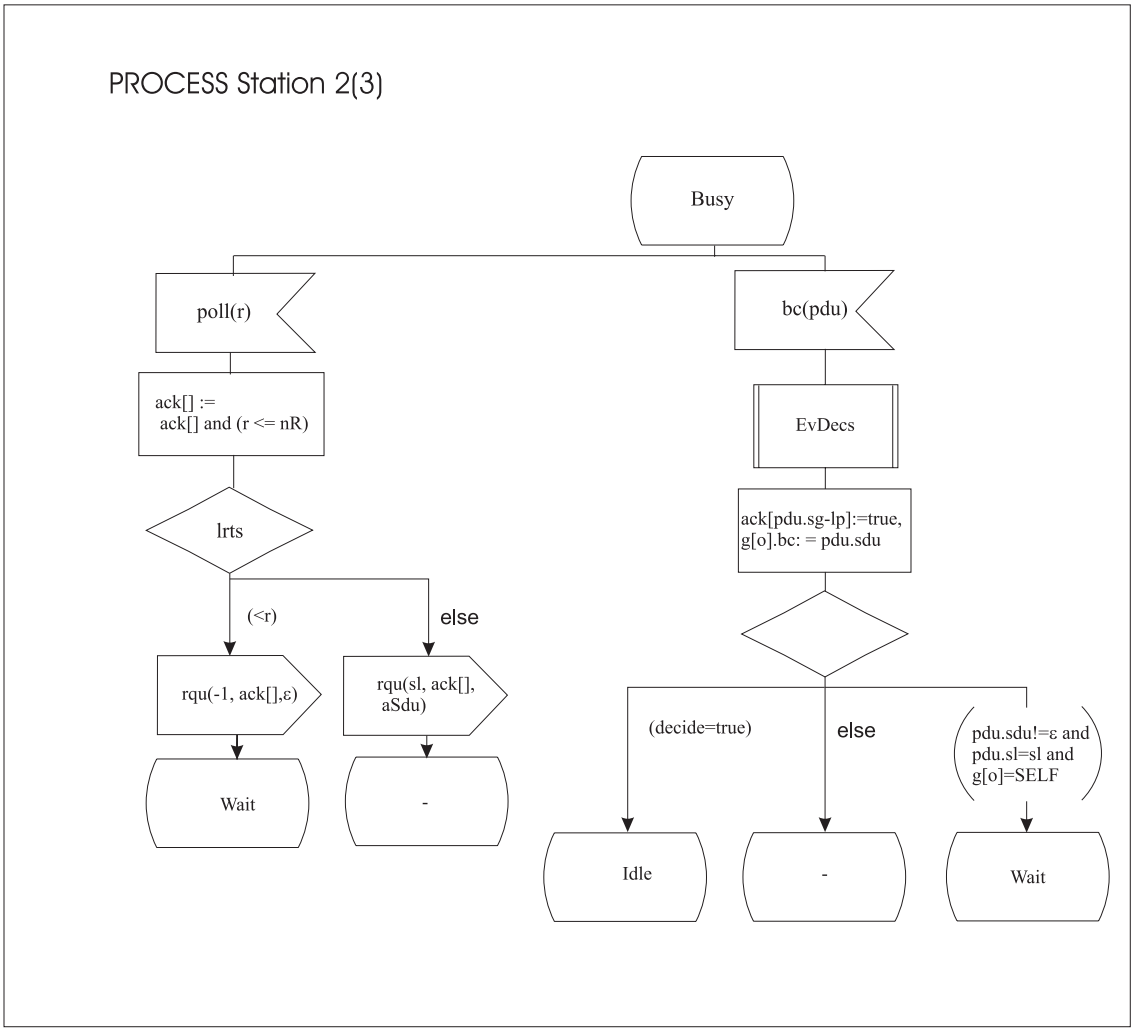


Abbildung 5.7 Darstellung des Stationsautomaten (Teil 2, Zustand *Busy*)

Auch der Empfang einer Broadcast-Nachricht löst in allen drei Zuständen einen Zustandsübergang aus. Bei diesen Übergängen bearbeitet der Stationsautomat immer die Entscheidungen des synchronen Kanals (*EvDecs*, Abb. 5.9), setzt das Bestätigungsbit für die empfangene Broadcast-Nachricht in *ack*[] und speichert die in der Broadcast-Nachricht enthaltene SDU als Attribut ihres Urhebers.

In der Prozedur *EvDecs* (s. Abb. 5.9) bearbeitet die Station die Entscheidungen des AP, die sie in den Broadcast-Nachrichten empfängt. Sie bearbeitet immer die Entscheidungen, die der AP seit der letzten Broadcast-Nachricht, die sie erhalten hat, in den synchronen Kanal eingefügt hat, in der Reihenfolge, in der sie der AP eingefügt hat. Die Station kann feststellen, in welchem Slot der AP eine Entscheidung getroffen hat (*sg*) und welche Station der AP in diesem Slot gepollt hat (*g*[*o*]), da es der Station aufgrund des synchronen Kanals möglich ist, alle Gruppenänderungen, die der AP durchgeführt hat, nachzuvollziehen. Eine *accept* Entscheidung, die der Automat bearbeitet, bezieht sich auf die aktuelle SDU der Station *g*[*o*] und der Automat liefert daher diese SDU an

den Benutzer aus. Bearbeitet der Automat eine *exclude* Entscheidung, so bezieht sich diese ebenfalls auf die Station *g*[*o*] weshalb der Automat *MEMCH(g*[*o*]*)* ausliefert.

Wenn der Automat die Entscheidungen des synchronen Kanals bearbeitet hat, kennt er die aktuelle Gruppe des AP, er weiß, welche Broadcast-Nachrichten der AP gesendet hat und welche Stationen die Eigentümer der betreffenden Slots waren. Daher weiß der Automat auch, in welchem Slot der AP zuletzt eine Polling-Nachricht an ihn gerichtet hat, und er kann das Bestätigungsboolean in seinem *ack*-Vektor entsprechend setzten. Weiterhin weiß der Automat, welche Station der Urheber der in der Broadcast-Nachricht enthaltenen SDU ist und er kann die SDU als Attribut der Urheberstation abspeichern.

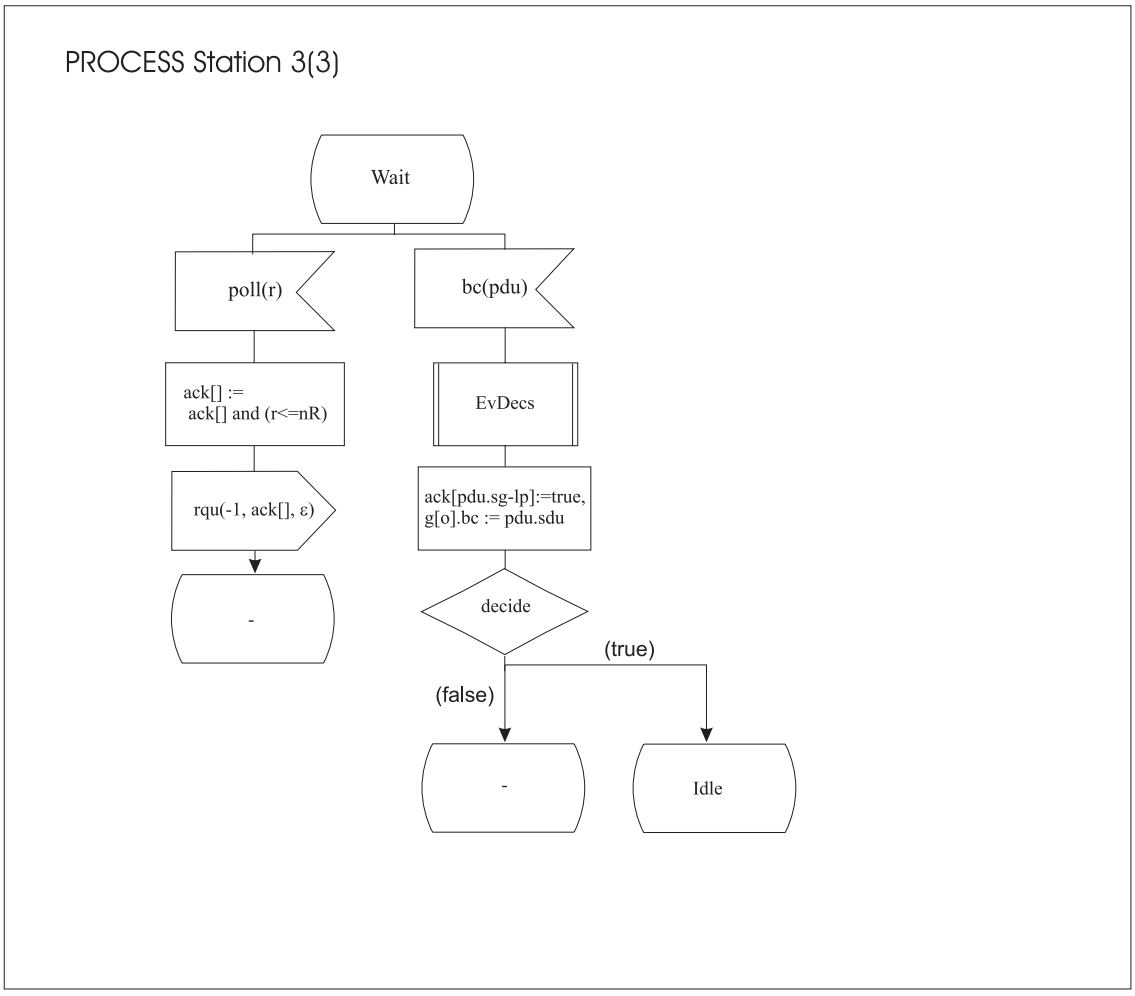


Abbildung 5.8 Darstellung des Stationsautomaten (Teil 3, Zustand *Wait*)

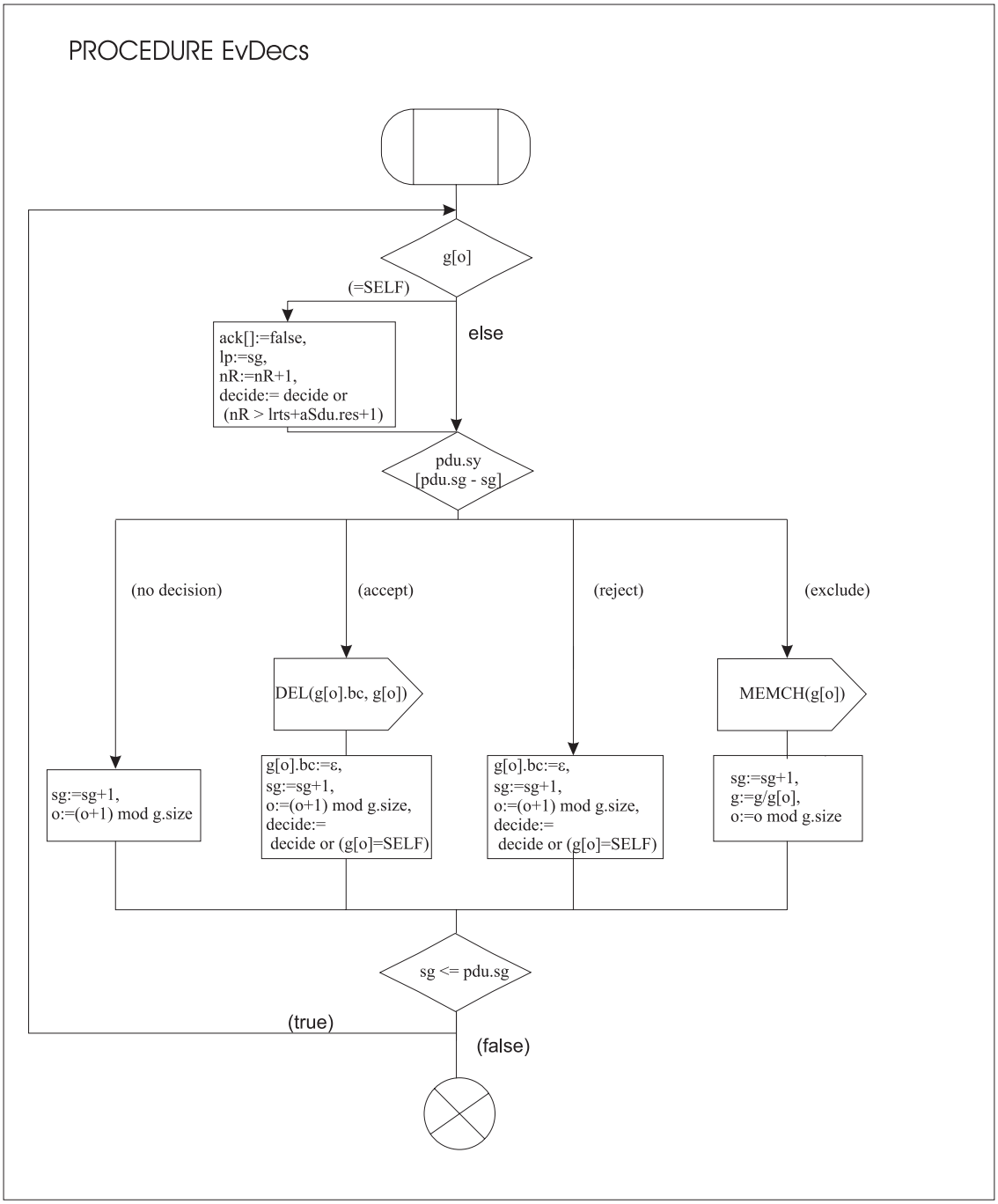


Abbildung 5.9 Darstellung des Unterprogramms ‚EvDecs'

Wenn der Stationsautomat eine Entscheidung bearbeitet, die sich auf seine eigene aktuelle SDU bezieht, dann weiß er, daß der AP die Bearbeitung dieser SDU abgeschlossen hat und daß er bereit für die Bearbeitung der nächsten SDU ist. Ebenso weiß der Automat, daß der AP die Bearbeitung seiner SDU abgeschlossen oder diese gar nicht empfangen hat, wenn die Rundennummer des AP den Wert *lrts + aSdu.res + 1* erreicht hat.

In beiden Fällen kann der Automat also in den Zustand *Idle* wechseln und mit der Bearbeitung der nächsten SDU beginnen.

5.5 Beweise der geforderten Eigenschaften

Im folgenden Unterkapitel werden die in Unterkapitel 2.1 geforderten Eigenschaften des Gruppenkommunikationsprotkolls in Form von Sätzen formuliert (Sätze 5.5 bis 5.15) und bewiesen. Hierbei werden die in den vorhergehenden Unterkapiteln erläuterten Variablen, sowie die Konstanten *OD* für den Omisssion-Degree und t_m für die maximale Verzögerung eines Broadcasts verwendet.

Im Folgenden wird das Senden eines Ausgangssignals durch *send(<Signal>, <Adressat>,<Zeitpunkt>)* und das Empfangen eines Eingabesignals durch *receive(<Signal>, <Absender>, <Zeitpunkt>)* bezeichnet. Absender, Adressat, Zeitpunkt und die Parameter von Signalen werden nicht angegeben, wo sie nicht von belang sind. Wie oben bereits erläutert kann jedem Slot und damit jeder Aktion, die der AP in diesem Slots durchführt, eine eindeutiger Sequenznummer *sg* zugeordnet werden. Auf diese Art können verschiedene Ausführungen des selben Statements unterschieden werden. Da diese Indizierung auch für die Durchführung von Zuweisungen möglich ist, kann die Entwicklung der Werte von Variablen ebenfalls durch Indizes bezeichnet werden. Hierzu bezeichne v_{sg} den Wert, den die Variable *v* nach der letzten Zuweisung im Slot *sg* annimmt. Zum Beispiel bezeichnet g_4 den Wert, den *g* in Slot *4* zugewiesen bekommt. Steht eine Variable *v* in einem Slot *sg* nicht auf der linken Seite einer Zuweisung, dann gilt $v_{sg} = v_{sg-1}$.

Aus der obigen Beschreibung des AP-Automaten ist ersichtlich, daß der Stationsautomat in allen Zuständen in der Lage, Polling-Nachrichten und Broadcast-Nachrichten zu empfangen und zu verarbeiten. Jede Station, die eine Polling-Nachricht des AP empfängt, sendet genau eine Request-Nachricht an den AP, diese wird als Antwort der Station auf die Polling-Nachricht bezeichnet.

Lemma 1 (Lebendigkeit der Stationen)
Sei *S* die Menge der Station und $K \subseteq S$ die Menge der korrekten Stationen. Für jede Station $s \in K$ gilt:

1. Wenn der AP eine Broadcast-Nachricht sendet und auf dem Netzwerk treten keine Fehler auf, dann wird *s* diese Broadcast-Nachricht empfangen und *EvDecs* aufrufen.
2. Wenn der AP eine Polling-Nachricht an *s* sendet und auf dem Netzwerk treten keine Fehler auf, dann wird *s* die Polling-Nachricht empfangen und genau eine Request-Nachricht an den AP senden.

■

Im folgenden bezeichne $poll_{sg}$ die Polling-Nachricht, die der AP in Slot *sg* sendet ($ap.send_{sg}(poll_{sg},s)$) und rqu_{sg} die Antwort von *s* auf die Polling-Nachricht, wenn *s* $poll_{sg}$ empfängt.

Bevor der AP eine Polling-Nachricht sendet, setzt er das Timeout *TP*. Wenn der AP also zum Zeitpunkt *t* eine Station *s* pollt, ist sichergestellt, daß er spätestens zum Zeitpunkt $t + 2 \times t_m$ eines der beiden Signale *rqu* oder *TP* empfängt. Da er in beiden Fällen eine Broadcast-Nachricht, deren Übertragungsdauer $\tau_m < t_m$ ist, und anschließend eine Polling-Nachricht sendet, gilt:

Lemma 2 (Lebendigkeit des AP)

$$send_{sg}\,(poll, s, t) \Rightarrow send_{sg+1}(poll, s', t') \wedge t' \in]t,\, t + \Delta_{Slot}] \text{ und}$$

$$send_{sg}\,(bc, t) \Rightarrow send_{sg+1}\,(bc, t') \wedge t' \in]t,\, t + \Delta_{Slot}]\,,$$

wobei $\Delta_{Slot} := 3 \times t_m$.

■

Durch Induktion kann man einfach zeigen, daß $send_{sg}(bc, t) \Rightarrow send_{sg+k}(bc, t') \wedge t' \in]t\,,\, t + k \times \Delta_{Slot}]$.

Lemma 3 (Rundenstruktur)

1. Die Id einer Station kann immer nur kleiner werden: Gilt $s = g_{sg}[i]$, $s = g_{sg'}[i']$ und $sg < sg'$, dann ist $i \geq i'$.

2. Sei $POLL(s) := \{poll_{sg}(r) \mid ap.send_{sg}(poll_{sg}(r), s)\}$ die Menge aller Polling-Nachrichten, die der AP an die Station *s* gesendet hat. Der AP pollt jede Stationen *s* mit streng monoton steigenden Rundennummern:

   $$poll_{sg}(r_{sg}),\, poll_{sg'}(r_{sg'}) \in POLL(s) \wedge sg < sg' \Rightarrow r_{sg} < r_{sg'}$$

   und der AP pollt die Stationen mit fortlaufenden Rundennummern:

$$poll_{sg}(r^c),\, poll_{sg'}(r^c + x) \in POLL(s) \wedge x > 1 \Rightarrow \exists\, sg'' \in]sg, sg'[: poll_{sg''}(r^c + 1) \in POLL(s)\,.$$

■

Beweis
zu 1.
Eine Station kann ihre Id nur ändern, wenn der AP eine andere Station (mit Id *i*) aus der Gruppe ausschließt ($g := g/g[o]$, Abb. 5.5). In diesem Falle bleibt aber die Id aller Stationen, deren Id kleiner als *i* ist unverändert, die Id von allen Stationen, deren Id größer als *i* ist, wird um eins kleiner.

zu 2.

Aufgrund von $sg < sg'$, $g[o_{sg}] = s = g[o_{sg'}]$ und 1. gilt $o_{sg'} \leq o_{sg}$. Es gibt zwei Möglichkeiten: Entweder der AP schließt in Slot *sg+1* die Station $g_{sg}[o_{sg}] = s$ aus der Gruppe aus, oder er inkrementiert *o*. Im ersten Falle kann der AP die Station *s* in Slot *sg'* nicht pollen, im Gegensatz zur Annahme $poll_{sg'} \in Poll(s)$. Im zweiten Falle gilt $o_{sg'} \leq o_{sg} < o_{sg} + 1$, weshalb der AP vor dem Senden von $poll_{sg'}$ mindestens einmal $o := 0$ und daher auch $r := r+1$ ausgeführt haben muß. Darum ist $r_{sg} < r_{sg'}$.

Da der AP in jedem Slot nur eine Polling-Nachricht sendet, gilt, daß $sg \neq sg'$, und da r_{sg} monoton steigend ist, daß $sg' > sg$. Da *r* nur inkrementiert wird und die Werte r^c und r^c+x mit $x > 1$ annimmt, muß *r* auch die Werte r^c+1 und r^c+2 annehmen. Nehme *r* den Wert r^c+1 in Slot sg^+ und den Wert r^c+2 in Slot sg^* an. Es gilt $sg < sg^+ < sg^* \leq sg'$. Da *r* in Slot sg^+ den Wert r^c+1 angenommen hat, gilt $o_{sg^+} = 0$. Bevor *r* in Slot sg^* den Wert r^c+2 angenommen hat, war der Wert von *o* gleich $g.size$. Da der AP nie gleichzeitig *o* inkrementiert und eine Station aus der Gruppe ausschließt (s. Abb. 5.5), muß es einen Slot $sg'' \in [sg^+, sg^*[$ gegeben haben, in dem $g_{sg''}[o_{sg''}] = s$. Daher hat der AP Station *s* auch mit $r = r^c + 1$ gepollt.

■

Lemma 4
Sei $succ_{sg} := (g_{sg}[o_{sg}], g_{sg}[(o_{sg}+1) \bmod g_{sg}], \ldots, g_{sg}[(o_{sg} + g_{sg}.size-1) \bmod g_{sg}.size])$, dann gilt:

$$succ_{sg+1} = (succ_{sg}[1], succ_{sg}[2], \ldots, succ_{sg}[g_{sg}.size-1], succ[0]) \text{ oder}$$

$$succ_{sg+1} = (succ_{sg}[1], succ_{sg}[2], \ldots, succ_{sg}[g_{sg}.size-1])$$

■

Beweis
Fallunterscheidung

1. *Fall:* In Slot *sg+1* wird die Station $g_{sg}[o_{sg}]$ ausgeschlossen. In diesem Falle ist $g_{sg+1} = (g_{sg}[0], \ldots, g_{sg}[o_{sg}-1], g_{sg}[o_{sg}+1], \ldots, g_{sg}[g_{sg}.size - 1])$ und $o_{sg+1} = o_{sg} \bmod g_{sg+1}.size$. Daher ist $succ_{sg+1} = (g_{sg}[o_{sg}+1], \ldots, g_{sg}[g_{sg}.size - 1], g_{sg}[0], \ldots, g_{sg}[o_{sg}-1]) = (succ_{sg}[1], \ldots, succ_{sg}[g_{sg}.size-1])$.

2. *Fall:* In Slot *sg+1* wurde keine Station ausgeschlossen. In diesem Falle ist $g_{sg+1} = g_{sg}$ und $o_{sg+1} = (o_{sg} + 1) \bmod g_{sg}.size$. Daher ist $succ_{sg+1} = (g_{sg}[o_{sg}+1], \ldots, g_{sg}[g_{sg}.size-1], g_{sg}[0], \ldots, g_{sg}[o_{sg}]) = (succ_{sg}[1], \ldots, succ_{sg}[g_{sg}.size-1], succ_{sg}[0])$.

■

Aus Lemma 4 kann man durch Induktion einfach folgern, daß für alle $i \in [0, g_{sg}.size-1]$ $succ_{sg+i}[0] = succ_{sg}[i]$. Da $succ_{sg}[0] = g_{sg}[o_{sg}]$ immer genau diejenige Station bezeichnet, die der AP in Slot *sg* pollt, pollt der AP in den Slots $sg, \ldots, sg + g_{sg}.size-1$ genau die Station in $succ_{sg}$ gemäß ihrer Reihenfolge. Da weiterhin $succ_{sg}$ und g_{sg} die gleichen Sta-

tionen enthalten folgt, daß der AP in den Slots *sg, ..., sg* + $g_{sg}.size-1$ jede Station aus g_{sg} genau einmal pollt.

Lemma 5
Wenn der AP die Station *s* in Slot *sg* zum Zeitpunkt *t* mit Rundennummer r^c pollt ($send_{sg}(poll(r^c), s, t)$), tritt genau eine der beiden folgenden Möglichkeiten ein:

(a) Der AP schließt *s* in Slot *sg+1* aus der Gruppe aus: $g_{sg+1} = g_{sg} / s$

(b) Der AP pollt *s* mit Rundennummer r^c+1 in Slot $sg' := sg + g_{sg}.size$ und damit spätestens zum Zeitpunkt $t + \Delta_{Round}$, wobei $\Delta_{Round} := n \times \Delta_{Slot}$ und $\Delta_{Slot} := 3 \times t_m$.

■

Beweis
Wenn der AP die Station *s* in Slot $sg' > sg$ pollt, dann ist offensichtlich $s \in g_{sg'}$ und damit kann der AP *s* in Slot *sg* nicht aus der Gruppe entfernt haben.

Wenn der AP *s* in Slot *sg* nicht aus der Gruppe entfernt, dann ist nach Lemma 4 $succ_{sg+1}[g_{sg}.size-1] = s$ und daher $succ_{sg+g_{sg}.size}[0] = s$. Der AP wird *s* also in Slot $sg' = sg + g_{sg}.size$ erneut pollen. Da der AP *s* zwischen *sg* und *sg'* nicht pollt folgt aus Lemma 3, daß er *s* in Slot *sg'* mit $r = r^c + 1$ pollt. Aufgrund von Lemma 2 gilt für den Zeitpunkt *t'*, an dem der AP in Slot *sg'* die Polling-Nachricht an *s* sendet: $t' \le t + g_{sg}.size \times \Delta_{Slot} \le t + n \times \Delta_{Slot}$.

■

Im Folgenden werden einige wichtige Eigenschaften des synchronen Kanals bewiesen.

Sei

$$DEC(sg) = \{bc(sg') \mid ap.send(bc(sg')) \wedge sg' - sg \in [0, OD]\}$$

die Menge der *OD+1* aufeinanderfolgenden Broadcast-Nachrichten des AP, beginnend mit der globalen Sequenznummer *sg*. d_{sg} ist die Entscheidung, die der AP in Slot *sg* trifft, an Position 0 des *sy*-Vektors einträgt ($sy_{sg}[0] := d_{sg}$, s. Abb. 5.4) und in der Broadcast-Nachricht dieses Slots versendet ($ap.send_{sg}(bc(sg, sy_{sg}, m))$).

Lemma 6

$$bc(sg', syncCh') \in DEC(sg) \Rightarrow syncCh'[sg' - sg] = d_{sg}.$$

■

Beweis
Der AP hat nach dem Senden von *bc(sg ,syncCh)* und vor dem Senden von *bc(sg',syncCh')* die Task *sg*:=*sg+1* (*sg'-sg*)-mal ausgeführt und damit auch *RotR(sy* [*], 1)* (s. Abb. 5.4). Wegen $sg' - sg \in [0, OD]$ folgt die Behauptung. ■

Lemma 7
Für jede korrekte Station *s* und jeden Slot *sg* gilt:

$$\exists\, bc(sg', synchCh') \in DEC(sg): s.receive(bc(sg',synchCh'))$$

■

Beweis
Folgt weil die Menge *DEC* aus *OD+1* aufeinanderfolgenden Nachrichten des selben Senders besteht und aus **(Medium 3).** ■

Lemma 8
Wenn eine Station *s* eine Broadcast-Nachricht empfängt, ist *s.sg-1* die Sequenznummer, der letzten Broadcast-Nachricht, die *s* empfangen hat (bzw. –1 bei der ersten Broadcast-Nachricht, die *s* empfängt.

■

Beweis
Durch Induktion über die Anzahl der empfangenen Broadcast-Nachrichten *k*.

k = 1:
s.sg = 0 aufgrund der Initialisierung.

Die Annahme gelte für $k \le k^c$.

Zu dem Zeitpunkt, an dem *s* die k^c-te Broadcast-Nachricht $bc(sg_0)$ empfängt, gilt nach Induktionsannahme $s.sg \le sg_0$. Da *s.sg* in *EvDecs* (s. Abb. 5.9) bei jedem Schleifendurchlauf inkrementiert wird, folgt aus der Bedingung $sg \le pdu.sg$, daß beim Verlassen von *EvDecs* $s.sg = pdu.sg+1 = sg_0+1$. Da *s.sg* erst nach dem Empfang der (k^c+1)ersten Broadcast-Nachricht wieder inkrementiert wird, folgt die Aussage.

■

Satz 5.1 (Synchrone Zuverlässigkeit)
Für jeden Slot *sg* und jede korrekte Station *s* gilt:

$$\exists\, i \in [0,OD]: s.receive(bc(sg+i, syncCh) \wedge synchCh\,[i] = ap.d_{sg}) \qquad \textbf{(1)}$$

Sei $bc(sg+i')$ die erste dieser Nachrichten, dann gilt zum Empfangszeitpunkt von $bc(sg+i')$:

$$s.sg \le sg \qquad \textbf{(2)}$$

Wenn *t* der Zeitpunkt ist, an dem der AP $bc(sg)$ sendet, und *t'* der Zeitpunkt ist, an dem die Station $bc(sg+i')$ empfängt, dann ist

$$t' - t \le OD \times \Delta_{Slot} + t_m\,, \qquad \textbf{(3)}$$

wobei $\Delta_{Slot} := 3 \times t_m$

■

Beweis
zu 1.
Folgt zusammen aus Lemma 6 und Lemma 7.

zu 2.
Da *bc(sg + i')* die erste Broadcast-Nachricht aus *DEC(sg)* ist, die *s* empfängt, hat *s* keine Broadcast-Nachricht *bc(sg')* mit $sg' \in [sg, sg+i[$ empfangen. Daher gilt für die letzte Broadcast-Nachricht *bc(sg'')*, die der AP vor *bc(sg + i')* empfangen hat, $sg'' < sg$ und daher folgt aus Lemma 8, daß $s.sg = sg''+1 \leq sg$.

zu 3.
Da $i' \leq OD$ folgt aus Lemma 2, daß der AP *bc(sg + i')* spätestens zum Zeitpunkt $t + OD \times \Delta_{Slot}$ sendet und daher wegen **(Medium 4)**, daß *s bc(sg + i')* spätestens zum Zeitpunkt $t + OD \times \Delta_{Slot} + t_m$ empfängt.

■

Satz 5.1 besagt: Wenn der AP in Slot *sg* eine Entscheidung *d* trifft, dann empfängt jede korrekte Station eine Broadcast-Nachricht *bc(sg')*, die *d* an der Position *sg'-sg* im synchronen Kanal enthält. Die Station wird daraufhin *EvDecs* aufrufen. Da $pdu.sg\text{-}s.sg = sg'\text{-}s.sg \geq sg'\text{-}sg$ und da in *EvDecs* jede Position des synchronen Kanals von *pdu.sg-s.sg* bis *0* bearbeitet wird, wird die Entscheidung des AP nach dem Empfang auch bearbeitet. Wenn die Station *s* d_{sg} bearbeitet, dann ist $pdu.sg - s.sg = sg' - sg$ und daher $s.sg=sg$. D. h. wenn der AP in Slot *sg* zum Zeitpunkt *t* die Entscheidung d_{sg} trifft, dann nimmt bei jeder korrekten Station *s* spätestens zum Zeitpunkt $t + OD \times \Delta_{Slot} + t_m$ die Variable *s.sg* den Wert *sg* an und *s* bearbeitet die Entscheidung $pdu.sy[pdu.sg - sg] = d_{sg}$.

Satz 5.2 (Synchrone Uniforme Integrität)
Für jede Station *s* gilt:

$$s.sg = sg' \wedge s.pdu.sy[s.pdu.sg - s.sg] = d \Rightarrow ap.d_{sg'} = d$$

■

Beweis
s hat die Broadcast-Nachricht *bc(sg'', syncCh)* empfangen mit $syncCh[sg'' - sg'] = d$. Aus Lemma 8 und **(Medium 3)** folgt, daß $pdu.sg - s.sg \in [0, OD]$. Aufgrund von **(Medium 1)** hat der AP *bc(sg'', syncCh)* gesendet und da $sg'' - sg' \in [0, OD]$, ist $bc(sg'', syncCh) \in DEC(sg')$. Damit gilt aufgrund von Lemma 6, daß $ap.d_{sg'}=d$. ■

Satz 5.3 (Synchrone Uniforme Einigung)
Wenn eine Station *s* eine Entscheidung des AP bearbeitet, d. h. $s.sg = sg_0$ und $s.pdu.sy[s.pdu.sg - s.sg] = d$, dann bearbeitet jede korrekte Station *s'* ebenfalls diese Entscheidung, d. h. $s'.sg = sg_0 \wedge s'.pdu.sy[s'.pdu.sg - s'.sg] = d$. ■

Beweis
Folgt aus Satz 5.1 und Satz 5.2.

■

Satz 5.4 (synchrone FIFO-Ordnung)
Wenn eine Station s die Entscheidung $d_{sg'}$ bearbeitet ($s.sg = sg' \wedge s.pdu.sy[s.pdu.sg - s.sg] = d_{sg'}$), dann hat sie vorher jede Entscheidung $d_{sg''}$ des AP mit $sg'' \in [0,sg'[$ bearbeitet.

■

Beweis
Da $s.sg$ mit 0 initialisiert und immer nur inkrementiert wird, folgt aus $s.sg = sg'$, daß $s.sg$ alle Werte aus $[0, sg']$ angenommen haben muß. Da vor dem Inkrementieren von $s.sg$ immer eine Entscheidung bearbeitet wird, folgt mit Satz 5.2 die Aussage.

■

Nach Satz 5.4 bearbeiten alle Stationen die Entscheidungen des AP in der Reihenfolge, in der sie von diesem in den synchronen Kanal eingefügt werden. Folglich werden die Entscheidungen auch von allen Stationen in der gleichen Reihenfolge bearbeitet.

Das folgende Lemma stellt einen wichtigen Zusammenhang zwischen dem synchronen Kanal und der Gruppenmitgliedschaft her, der grundlegend für alle späteren Beweise sein wird.

Lemma 9
Bezeichne $s.g_{sg}$ bzw. $s.o_{sg}$ den Wert $s.g$ bzw. $s.o_{sg}$ beim Bearbeiten d_{sg}, dann gilt für jede Station s, die d_{sg} bearbeitet:

$$s.g_{sg} = ap.g_{sg} \qquad \textbf{(1)}$$

und

$$s.o_{sg} = ap.o_{sg} \qquad \textbf{(2)}$$

■

Beweis
Durch Induktion über sg. Dies ist aufgrund von Satz 5.4 möglich, da wegen der FIFO-Ordnung des synchronen Kanals jede Station s, die d_{sg} bearbeitet, vorher jede Entscheidung $d_{sg'}$ mit $sg' \in [0,sg[$ bearbeitet hat.

$sg = 0$:

$ap.g_0 = s.g_0$ und $ap.o_0 = 0 = s.o_0$, aufgrund der Startwerte.

Die Behauptung gelte nun alle $sg' \in [0, sg[$.

Z. zg: die Behauptung gilt für sg

Fallunterscheidung:

1. Fall: $d_{sg-1} = exclude$

   $ap.g_{sg} = ap.g_{sg-1} / ap.g_{sg-1}[ap.o_{sg-1}]$. Aufgrund von Satz 5.4 hat auch s d_{sg-1} bearbeitet und $s.g_{sg} = s.g_{sg-1} / s.g_{sg-1}[s.o_{sg-1}]$ gesetzt (s. Abb. 5.9). Nach Induktionsannahme gilt $s.g_{sg-1} = ap.g_{sg-1}$ und $s.o_{sg-1} = ap.o_{sg-1}$ und damit $s.g_0 = ap.g_0$.

   $ap.o_{sg} := ap.o_{sg-1} \; mod \; ap.g_{sg}.size = s.o_{sg-1} \; mod \; s.g_{sg}.size =: s.o_{sg}$

2. Fall: $d_{sg-1} \neq exclude$

   Es gilt $ag.g_{sg} = ap.g_{sg-1} = s.g_{sg-1} = s.g_{sg}$ und
   $ap.o_{sg} := (ap.o_{sg-1} + 1) \; mod \; ap.g_{sg}.size = (s.o_{sg-1} + 1) \; mod \; s.g_{sg}.size =: s.o_{sg}$

■

Mit Hilfe von Lemma 9 kann folgende Eigenschaft des Teilnehmerprotokolls bewiesen werden:

Satz 5.5 (Teilnehmerprotokoll Einigung)
Wenn eine Station s eine Nachricht ausliefert, die den Aufall von Station s' anzeigt (*s.send(MEMCH(s'))*), dann liefert jede korrekte Station s'' ebenfalls eine Nachricht aus, die den Ausfall von s' anzeigt (*s''.send(MEMCH(s'))*).

■

Beweis
Aus *s.send(MEMCH(s'))* folgt, daß s' eine Entscheidung $d_{sg}=exclude$ bearbeitet mit $s.g_{sg}[s.o_{sg}] = s'$ ($s.g_{sg}$ und $s.o_{sg}$ seien wie in Lemma 9 definiert). Aufgrund von Satz 5.3 wird jede korrekte Station s'' ebenfalls die Entscheidung d_{sg} bearbeiten, wobei wegen Lemma 9 $s''.g_{sg}[s''.o_{sg}] = s.g_{sg}[s.o_{sg}] = s'$. Daher liefert jede korrekte Station ebenfalls *MEMCH(s')* aus.

■

Satz 5.6 (Teilnehmerprotokoll Totale Ordnung)
Wenn zwei Stationen s und s' die Änderungsnachrichten $MEMCH(s^2)$ und $MEMCH(s^3)$ ausliefern, dann liefern beide Stationen die Nachrichten in der gleichen Reihenfolge aus.

■

Beweis
Wenn Station s die Nachrichten $MEMCH(s^2)$ und $MEMCH(s^3)$ in dieser Reihenfolge ausliefert, dann hat s zunächst eine Entscheidung d_{sg} mit $s.g_{sg}[s.o_{sg}] = s^2$ ($s.g_{sg}$ und $s.o_{sg}$ seien wie in Lemma 9 definiert) und dann eine Entscheidung $d_{sg'}$ mit $s.g_{sg'}[s.o_{sg'}] = s^3$ bearbeitet. Da der AP vor dem Senden einer Ausschluß-Entscheidung die betreffende Station aus seiner Mitgliedschaft entfernt, kann es für jede Station s^+ höchstens eine Entscheidung $d_{sg}=exclude$ mit $ap.g_{sg}[ap.o_{sg}] = s^+$ geben. Daher muß Station s' die Änderungsnachrichten beim Bearbeiten der gleichen beiden Entscheidungen d_{sg} und $d_{sg'}$ ausgeliefert haben wie s. Aufgrund von Satz 5.4 hat s' die Entscheidungen in der glei-

chen Reihenfolge bearbeitet wie Station *s*. Daher hat *s'* die Nachrichten $MEMCH(s^2)$ und $MEMCH(s^3)$ in der gleichen Reihenfolge ausgeliefert wie *s*.

■

Aufgrund der Eigenschaft **(Medium4)** und Lemma 1 gilt: Wenn der AP eine Station *s* zum Zeitpunkt *t* pollt und *s* ist korrekt und auf dem Netzwerk treten keine Fehler auf, dann empfängt der AP spätestens zum Zeitpunkt $t' < t + 2 \times t_m$ eine Request-Nachricht von *s*. Wenn umgekehrt der AP zum Zeitpunkt *t* eine Polling-Nachricht an *s* sendet und bis zum Zeitpunkt $t + 2 \times t_m$ keine Request-Nachricht von *s* empfängt, dann ist entweder *s* ausgefallen oder es sind Nachrichtenverluste aufgetreten.

Lemma 10

1. Sei $POLL(r, s) := \{poll(r+i) \mid i \in [0, OD] \wedge ap.send(poll(r+i),s)\}$. Für alle korrekten Stationen *s* und alle Rundennummern *r* gilt:

$$|POLL(r,s)| = OD+1 \Rightarrow \exists\, i \in [0,OD],\ sg:\ ap.send_{sg}(poll(r+i),s,t)$$
$$\wedge\ s.receive(poll_{sg},\ ap) \wedge ap.receive(rqu_{sg},\ s,\ t') \wedge t' - t < 2 \times t_m$$

   Das heißt: Wenn der AP *OD+1*-mal in Folge die korrekte Station *s* pollt, dann empfängt *s* mindestens eine der Polling-Nachrichten und der AP weniger als $2 \times t_m$ Zeiteinheiten nach dem Senden der Polling-Nachricht die Antwort von *s* auf diese Polling-Nachricht.

2. Für alle korrekten Stationen *s* und alle Slots *sg*:

$$ap.s.l_{sg} < OD+1$$

■

Beweis

zu 1.

Nach Lemma 3 bedeutet $|POLL(r,s)| = OD+1$, daß der AP aufeinanderfolgend *OD+1* Polling-Nachrichten an *s* gesendet hat. **(Medium 3)** besagt dann, daß *s* mindestens eine dieser Nachrichten empfängt und (Lemma 1) eine Antwort sendet. Nach **(Medium 3)** muß für mindestens eine der erhaltenen Polling-Nachrichten auch die Antwort vom AP empfangen werden. Aus **(Medium 4)** folgt, daß zwischen dem Senden der Polling-Nachricht und dem Empfang der Antwort weniger als $2 \times t_m$ Zeiteinheiten vergehen.

zu 2.

Seien *s* korrekt, $g_{sg}[o_{sg}].l = OD+1$ und $sg_{(1)}, \ldots, sg_{(OD+1)} = sg$ die Sequenznummern der letzten *OD+1* Slots mit $g_{sg_{(i)}}[o_{sg_{(i)}}] = s$. Da der AP $g[o].l$ nur inkrementiert, ist $g_{sg_{(i)}}[o_{sg_{(i)}}].l = i$ für alle $i \in [1, OD+1]$. Da der AP $g[o].l := 0$ setzt, wenn er eine Request-Nachricht empfängt, hat er folglich in keinem der Slots $sg_{(1)}, \ldots, sg_{(OD+1)}$ eine Request-Nachricht empfangen. Da aber der AP *s* in jedem Slot $sg_{(i)}$ pollt, gilt, daß $|POLL(r_{sg(1)}, s)| = OD+1$

und damit nach 1., daß es ein $i' \in [1,OD+1]$ gibt, so daß der AP in Slot $sg_{(i')}$ eine Request-Nachricht von s empfängt. Widerspruch!

■

Satz 5.7 (Teilnehmerprotokoll Genauigkeit)
Wenn eine Station s eine Nachricht *MEMCH(s')* zum Zeitpunkt t ausliefert, dann ist die Station s' vor dem Zeitpunkt t ausgefallen.

■

Beweis
Wenn s *MEMCH(s')* ausliefert, dann hat s eine Entscheidung $d_{sg} = exclude$ bearbeitet mit $s' = s.g_{sg}[o_{sg}]$ ($s.g_{sg}$ und $s.o_{sg}$ seien wie in Lemma 9 definiert). Daraus folgt nach Satz 5.2, daß $ap.d_{sg} = exclude$. Es gilt also $ap.g_{sg}[o_{sg}].l = OD+1$ und nach Lemma 9, daß $ap.g_{sg}[o_{sg}]=s'$. Daher folgt aus Lemma 10, daß s' ausgefallen ist. Da sich Station s' bis zu ihrem Ausfall wie eine korrekte Station verhält, ist für jeden Zeitpunkt $t' < t$ nach Lemma 10 $ap.s'.l < OD$.

■

Da $ap.g$ am Anfang alle korrekten Stationen enthält und der AP nach Satz 5.7 nur ausgefallene Stationen aus der Gruppe ausschließt gilt für jede korrekte Station s, daß $s \in ap.g$. Dies impliziert zusammen mit Lemma 5, daß der AP jede korrekte Station mit jeder Rundennummer genau einmal pollt und daß zwischen zwei solchen Nachrichten weniger als Δ_{Round} ($= n \times \Delta_{Slot} = 3 \times t_m \times n$) Zeiteinheiten vergehen.

Lemma 11
Eine Request-Nachricht, die in Slot sg gesendet wird, wird entweder in Slot sg empfangen oder überhaupt nicht.

■

Beweis
Durch Induktion über sg.

$sg = 0$

> Wenn der AP $send_1(poll)$ durchführt, dann hat er entweder eine Request-Nachricht oder ein Timeout empfangen ($ap.receive_0\ (rqu)$ oder $ap.receive_0(TP)$). Im ersten Falle muß es sich um die Antwort auf $poll_0$ handeln ($rqu=rqu_0$). Daher kann der AP rqu_0 in keinem der Slots $sg' > 0$ nochmals empfangen (**(Medium 1)**). Im zweiten Falle sind seit dem Senden von $poll_0$ $2 \times t_m$ Zeiteinheiten vergangen, ohne daß der AP eine Request-Nachricht empfangen hat. Dies bedeutet, daß entweder $poll_0$ verloren gegangen ist oder der Adressat von $poll_0$ vor dem Senden von rqu_0 ausgefallen ist oder daß rqu_0 verloren gegangen ist. Auf jeden Fall wird der AP rqu_0 in keinem der Slots $sg' > 0$ empfangen.

Die Behauptung gelte für alle $sg' \in [0,sg]$

Zu zeigen: Der AP empfängt die Request-Nachricht aus Slot *sg+1* entweder in Slot *sg+1* oder gar nicht.

> Der Beweis kann analog zum Fall *sg=0* geführt werden, wobei die Induktionsannahme sicherstellt, daß es sich bei der Request-Nachricht, die der AP empfängt, um rqu_{sg+1} handelt.

■

Nach Lemma 11 gilt, daß $ap.receive_{sg}(rqu_{sg'}, s) \Rightarrow sg' = sg \wedge ap.g_{sg}[o_{sg}] = s$.

Satz 5.8 (Teilnehmerprotokoll Rechtzeitigkeit)
Wenn eine Station *s* zum Zeitpunkt *t* ausfällt, liefert jede korrekte Station *MEMCH(s)* spätestens zum Zeitpunkt $t^3 \leq t + (OD+1) \times \Delta_{Round} + (OD+1) \times \Delta_{Slot}$ aus. Hierbei sei $\Delta_{Round} := n \times \Delta_{Slot}$ und $\Delta_{Slot} := 3 \times t_m$.

■

Beweis
Wenn Station *s* zum Zeitpunkt *t* ausfällt (sei zu diesem Zeitpunkt $ap.sg=sg_{(A)}$), dann hat *s* sich entsprechend dem Fehlermodell bis zum Zeitpunkt *t* wie eine korrekte Station verhalten. Daher folgt aus $s \in ap.g_0$ und Lemma 10, daß $s \in ap.g_{sg_{(A)}}$. Folglich gibt es einen Zeitpunkt $t' \geq t$ (sei zu diesem Zeitpunkt $ap.sg = sg_{(0)}$), zu dem der AP *s* das erste Mal nach dem Ausfall pollt. Im ungünstigsten Falle ist $sg_{(0)} = sg_{(A)} + ap.\ g_{sg_{(A)}}.size$ (Lemma 5), so daß auf jeden Fall $t' \leq t + \Delta_{Round}$. Nach Lemma 5 gilt für jeden der Slots $sg_{(0)}$, $sg_{(1)} := sg_{(0)} + ap.g_{sg(0)}.size, \ldots, sg_{(OD)} := sg_{(OD-1)} + ap.g_{sg_{(OD-1)}}.size$, daß der AP *s* entweder in Slot $sg_{(i)}$+1 aus der Gruppe ausschließt oder *s* in Slot $sg_{(i+1)}$ erneut pollt. Dies bedeutet, daß der AP *s* in Slot $sg_{(OD)}$ schon aus der Gruppe ausgeschlossen hat oder *s* pollt. Damit hat der AP *s* in den Slots $sg_{(0)}$ bis $sg_{(OD)}$ *OD+1*-mal gepollt. Da *s* ausgefallen ist, hat der AP aufgrund von Lemma 11 in keinem der Slots $sg_{(0)}, \ldots, sg_{(OD)}$ eine Request-Nachricht empfangen. Daher empfängt der AP in jedem Slot $sg_{(i)}$ das Timeoutsignal *TP* und inkrementiert $ap.g_{sg_{(i)}}[o_{sg_{(i)}}].l$. Da also $ap.g_{sg_{(OD)}}[o_{sg_{(OD)}}].l \geq OD+1$, ist spätestens in Slot $sg_{(OD)}$ zum Zeitpunkt t'' $d_{sg_{(OD)}} = exclude$. Nach Lemma 5 ist $t'' \leq t' + OD \times \Delta_{Round} + 2 \times t_m$. Zusammen mit Satz 5.1 folgt, daß jede korrekte Station s' d_{sg} zum Zeitpunkt $t^3 \leq t'' + OD \times \Delta_{Slot} + t_m$ bearbeitet. Nach Lemma 9 ist dabei $s'.g_{sg_{(OD)}}[o_{sg_{(OD)}}] = ap.g_{sg_{(OD)}}[o_{sg_{(OD)}}] = s$ ($s.g_{sg}$ und $s.o_{sg}$ seien wie in Lemma 9 definiert). Jede korrekte Station wird also spätestens zum Zeitpunkt $t^3 \leq t + (OD+1) \times \Delta_{Round} + (OD+1) \times \Delta_{Slot}$ die Nachricht *MEMCH(s)* ausliefern.

■

Lemma 12
Jede SDU *m* ist eindeutig durch ihren Urheber und die lokale Sequenznummer bezeichnet:

$$ap.receive(rqu(sl,m), s) \wedge ap.receive(rqu(sl',m'), s) \Rightarrow (sl = sl' \Leftrightarrow m = m')$$

■

Beweis
1. $sl = sl' \Rightarrow m = m'$:

   Fallunterscheidung:

   1. Fall: $sl = sl' = -1$. Es folgt: $m = \varepsilon = m'$.

   2. Fall: $sl = sl' \neq -1$. Dann sind m und m' nicht leer und die Nachrichten wurden durch *send(rqu(sl, m)* und *send(rqu(sl', m'))* an den AP übertragen (o. B. d. A. auch in dieser Reihenfolge). Folglich hat s nach dem Senden von *rqu(sl, m)* das Statement $sl := sl+1$ nicht ausgeführt und daher auch nicht $aSDU = idu$. Da dies die einzige Stelle ist, an der sich *aSDU* ändern kann, folgt $m = m'$.

2. $sl \neq sl' \Rightarrow m \neq m'$

   Fallunterscheidung

   1. Fall: $sl = -1 \wedge sl' \geq 0$. Es folgt $m = \varepsilon \wedge m' \neq \varepsilon$.

   2. Fall: $sl' = -1 \wedge sl \geq 0$. Analog

   3. Fall: $sl \geq 0 \wedge sl' \geq 0$. Die Nachrichten wurden durch die Ausführungen *send(rqu (sl, m))* und *send(rqu(sl', m))* an den AP übertragen (o. B. d. A. auch in dieser Reihenfolge) Station s hat nach dem Senden von *rqu(sl, m)* mindestens einmal $sl := sl+1$ ausgeführt und daher auch *receive(A-BC(idu))* und $aSDU := idu$. Daraus folgt $m \neq m'$.

■

Da die Werte von *s.sl* nie kleiner werden können, haben aufeinanderfolgende SDU steigende lokale Sequenznummern.

Lemma 13
Es gilt:

$$ap.g[i].nSDU \neq \varepsilon \Rightarrow ap.g[i].nSDU.orig = ap.g[i] \quad \textbf{(1)}$$

$$ap.g[i].aSDU \neq \varepsilon \Rightarrow ap.g[i].aSDU.orig = ap.g[i] \quad \textbf{(2)}$$

$$s.g[i].bc \neq \varepsilon \Rightarrow s.g[i].bc.orig = s.g[i] \quad \textbf{(3)}$$

■

Beweis
(1) folgt aus Lemma 11 und der Zuweisung $g[o].nSDU := pdu$.

(2) folgt aus der Zuweisung $g[o].aSDU := g[o].nSDU$.

Wenn eine Station eine Broadcast-Nachricht *bc(sg, m)* empfängt, dann führt sie zunächst *EvDecs* aus. Die letzte Entscheidung, die in *EvDecs* bearbeitet wird, ist d_{sg} ($s.sg = pdu.sg$). Daher gilt beim Durchführen von $g[o].bc := pdu.sdu$, daß $g[o] = ap.g_{sg}[o_{sg}] = pdu.orig$ (Lemma 9 und **(2)**).

■

Lemma 14
Seien $s.lp_{sg}$ bzw. $s.nR_{sg}$ die Werte von *s.lp* bzw. *s.nR* beim bearbeiten von d_{sg}, dann gilt:

1. Wenn eine Station *s* eine Broadcast-Nachricht *bc(sg)* empfängt, gilt nach dem Bearbeiten von *EvDecs*

$$s.lp_{sg} = ap.s.lp_{sg}$$

2. Wenn *s* eine Entscheidung d_{sg_0} bearbeitet mit $s.g_{sg_0}[o_{sg_0}] = s$, gilt nach dem Inkrementieren von *s.nR*:

$$s.nR_{sg_0} = ap.r_{sg_0} + 1$$

   und nach dem Bearbeiten von *EvDecs* nach dem Empfang einer beliebigen Broadcast-Nachricht *bc(sg)*:

$$ap.r_{sg} \in [s.nR_{sg} - 1,\ s.nR_{sg}]$$

■

Beweis
zu 1)
Wenn *s bc(sg)* empfängt, dann bearbeitet *s* in *EvDecs* die Entscheidung d_{sg} des AP und nach Satz 5.4 damit auch die Entscheidung $d_{ap.s.lp_{sg}}$. Sei $Owner(sg) := ap.g_{sg}[o_{sg}]$. Da $Owner(ap.s.lp_{sg}) = s$ folgt aus Lemma 9, daß beim Bearbeiten dieser Entscheidung $s.g[o] = s$, weshalb *s* die Zuweisung $lp := sg$ ausführt. Beim Bearbeiten der folgenden Entscheidungen wird *s* die Zuweisung nicht mehr durchführen (Lemma 9), da $ap.s.lp_{sg}$ der letzte Slot war, in dem der AP *s* gepollt hat.

zu 2)
Wenn *s* die Entscheidung d_{sg_0} bearbeitet, dann hat *s* nach Satz 5.4 jede der Entscheidungen $d_0, \ldots, d_{sg_0}$ des AP bearbeitet und daher für jeden Slot $0, \ldots, sg_0$ den Eigentümer $ap.g_{sg_0}[o_{sg_0}]$ festgestellt. Für jeden Slot, von dem *s* selbst der Eigentümer war, hat *s nR* inkrementiert. Daher folgt, daß der AP *s* in den Slots $0, \ldots, sg_0$ genau $s.nR_{sg_0}-1$-mal gepollt hat. Da der AP *s* mit fortlaufenden Rundennummern pollt, folgt, daß die Polling-Nachricht, die der AP in Slot sg_0 an *s* gesendet hat die Rundennummer $s.nR_{sg_0}-1$ hatte.

Wenn die Station die Broadcast-Nachricht *bc(sg)* empfängt, dann bearbeitet sie in *Ev-Decs* die Entscheidung d_{sg} und sie hat daher jede Entscheidung $d_0, \ldots, d_{sg}$ des AP bearbeitet. Sei d_{sg_0} die letzte dieser Entscheidungen mit $s.g_{sg_0}[o_{sg_0}] = s$, dann folgt, daß der AP *s* in keinem der Slots aus $]sg_0, sg]$ gepollt hat. Folglich kann der AP nach Lemma 3 *ap.r* in diesen Slots höchstens einmal inkrementiert haben. Da beim Bearbeiten von d_{sg_0} in *s* galt $ap.r_{sg_0} = s.nR_{sg_0} - 1$, folgt $ap.r_{sg} \in [s.nR_{sg}-1, s.nR_{sg}]$.

■

Lemma 15
Zu jedem Zeitpunkt gilt:

$$\forall s_1, s_2 \in ap.g : gacks[s_1.id, s_2.id] = true \Rightarrow s_2.s_1.bc = ap.s_1.aSDU.$$

■

Beweis
Die Aussage gilt nach der Initialisierung, da für alle s_1 und s_2 $gacks[s_1.id, s_2.id] = false$.

Im Folgenden wird gezeigt, daß jede Task die der AP durchführt, die Satzaussage erhält. Dabei sind drei Tasks zu betrachten.

1. g := g/g[o]; ...
Durch die Veränderung der Gruppe können sich die Ids von s_1 und s_2 ändern. Da aber in der gleichen Task auch *gacks* entsprechend verändert wird, so daß $gacks_{sg}[s_1.id_{sg}, s_2.id_{sg}] = gacks_{sg+1}[s_1.id_{sg+1}, s_2.id_{sg+1}]$, bleibt die Aussage erhalten.

2. UPGACKS
Der AP führt $gacks[i,o] := gacks[i,o]$ *OR* $ack[g[i].lp - g[o].lp]$ aus mit $g[i]=s_1$ und $g[o]=s_2$ und $ack[g[i].lp - g[o].lp]=true$. Folglich (Lemma 11) hat der AP von s_2 in Slot sg_0 eine Request-Nachricht erhalten mit $ack[i_0] = true$ und $i_0 := ap.s_1.lp_{sg_0-1} - ap.s_2.lp_{sg_0-1}$, die s_2 auch gesendet haben muß **(Medium 1)**. Folglich hat s_2 in einem Slot $sg_1 < sg_0$ die Zuweisung $ack[pdu.sg - lp] := true$ ausgeführt mit $pdu.sg = sg_1$ und $sg_1 - s_2.lp_{sg_1} = i_0$ ($s_2.lp_{sg_1}$ sei der Wert von $s_2.lp$ beim Bearbeiten von d_{sg_1}). Sei sg_1 der letzte Slot vor sg_0, in dem s_2 diese Task mit $pdu.sg - lp = i_0$ ausführt, dann gilt, daß der AP s_2 in $]sg_1,sg_0[$ nicht gepollt hat (ansonsten hätte s_2 spätestens vor dem Senden von rqu_{sg_0} den *ack*-Vektor auf *false* gesetzt). Es folgt: $ap.s_2.lp_{sg_0-1} = ap.s_2.lp_{sg_1} = s_2.lp_{sg_1}$ (Lemma 14) und damit, daß $sg_1 = i_0 + s_2.lp_{sg_1} = ap.s_2.lp_{sg_0-1} + i_0 = ap.s_1.lp_{sg_0-1}$. Folglich war sg_1 der letzte Slot, in dem der AP $ap.s_1.aSDU$ gebroadcastet hat und es gilt zu dem Zeitpunkt, an dem der AP $gacks[s_1.id,s_2.id]$ auf *true* setzt, daß $s_2.s_1.bc = ap.s_1.aSDU$.

3. g[o].aSDU := g[o].nSDU
Da in der selben Task $gacks[o][] := false$ ausgeführt wird, gilt die Aussage.

Wenn sich $s_2.s_1.bc$ ändert, hat s_2 eine Broadcast-Nachricht empfangen, in der eine SDU m mit Urheber s_1 enthalten war. Da der AP diese Broadcast-Nachricht gesendet hat, gilt $ap.s_1.aSDU=m$ und damit $s_2.s_1.bc = ap.s_1.aSDU$.

■

Satz 5.9 (Uniforme Integrität)
Für alle Nachrichten m: Jede Station liefert m höchstens einmal aus und nur dann, wenn m gebroadcastet wurde.

■

Beweis
Wenn eine Station s_1 eine Nachricht m ausliefert, dann bearbeitet sie eine Entscheidung $d_{sg_0} = accept$ und $s_1.g_{sg_0}[o_{sg_0}].bc = m$. Sei $s_1.g_{sg_0}[o_{sg_0}] = s_2$. Da $m \neq \varepsilon$ hat s_1 eine Broadcast-Nachricht $bc(sg_1, m, sl_0)$ empfangen mit $sg_1 < sg_0$ und $s_1.g_{sg_1}[o_{sg_1}] = s_2$. Aufgrund der Integrität des Netzwerkes **(Medium 1)** muß der AP diese Nachricht gesendet haben und es folgt: $ap.g_{sg_1}[o_{sg_1}] = s2$ und $ap.s2.aSDU_{sg_1} = m$ und $ap.s_2.aSDU.sl_{sg_1} = sl_0$. Folglich gab es einen Slot $sg_2 \leq sg_1$, in dem der AP eine Request-Nachricht $rqu_{sg_2}(sl_0, m)$ empfangen hat und $ap.g_{sg_2}[o_{sg_2}] = s_2$. Nach Lemma 11 und **(Medium 1)** hat Station s_2 diese Nachricht gesendet. Da zu diesem Zeitpunkt $s_2.aSDU.sdu = m$, folgt, daß s_2 $A\text{-}BC(m, res)$ empfangen haben muß.

Wenn Station s_1 eine weitere Nachricht m' von Station s_2 ausliefert, dann folgt analog, daß es einen Slot sg_2' gegeben haben muß, in dem der AP die Request-Nachricht $rqu_{sg_2'}(sl', m')$ empfangen und $ap.s_2.nSDU$ zugewiesen haben muß. Sei o. B. d. A. $sg_2 < sg_2'$. Folglich muß aufgrund der Bedingung $g[o].sl < pdu.sl$ gelten, daß $sl' > sl$ und damit nach Lemma 12 $m \neq m'$.

■

Satz 5.10 (Uniforme Rechtzeitigkeit)
Wenn eine Station s eine Nachricht m zum Zeitpunkt t_0 broadcastet ($s.receive(A\text{-}BC(m,res))$), dann liefert keine Station s' die Nachricht m nach dem Zeitpunkt $t_0 + \Delta_{BC} = t_0 + 2 \times (m.res + 1) \times \Delta_{Round} + (OD + 1) \times \Delta_{Slot}$ aus. Hierbei sei $\Delta_{Round} := n \times \Delta_{Slot}$ und $\Delta_{Slot} := 3 \times t_m$.

■

Beweis
Sei o. B. d. A. zum Zeitpunkt t_0 $s.r = r_0$. Wenn die Nachricht m nicht ausgeliefert wird, dann ist für m die Aussage des Satzes erfüllt. Zu zeigen ist also: Wenn m von einer Station s' ausgeliefert wird, also sowohl von s an den AP als auch vom AP an s' übertragen wird, dann liefert s' m spätestens zum Zeitpunkt $t_0 + \Delta_{BC}$ aus. Wenn der AP m empfangen hat, dann folgt, daß s im Zustand *Busy* eine Polling-Nachricht $poll(r)$ empfangen und daraufhin eine Request-Nachricht $rqu(m)$ an den AP gesendet hat. Für jede dieser Polling-Nachrichten muß $r \leq s.lrts = r_0 + m.res + 1$ gewesen sein. Zusammen mit Lemma 3 folgt, daß s die Nachricht m spätestens zum Zeitpunkt $t_0 + (m.res + 1) \times$

Δ_{Round} zum letzten Mal an den AP überträgt und dieser die Nachricht spätestens zum Zeitpunkt $t_{AP} = t_0 + (m.res + 1) \times \Delta_{Round} + t_m$ empfängt.

Sei angenommen, daß zu diesem Zeitpunkt $ap.s.aSDU = \varepsilon$. Dann wird der AP m zu diesem, Zeitpunkt zum erstenmal broadcasten und vorher $ap.s.tr$ den Wert 1 zuweisen. In jeder der folgenden Runden, immer wenn $ap.g_{sg}[o_{sg}] = s$, gibt es zwei Möglichkeiten: Entweder $ap.s.aSDU=\varepsilon$ und m wird nie mehr gebroadcastet, oder m wird erneut gebroadcastet und $ap.s.tr$ wird inkrementiert. Der größte Wert, den $ap.s.tr$ auf diese Weise annehmen kann, ist $ap.s.aRes + 1$, weil der AP bei diesem Wert vor dem nächsten Inkrementieren eine Entscheidung $d \neq no\ decision$ trifft und daher $ap.s.aSDU:=\varepsilon$ setzt. Wenn r_1 die Runde ist, in der der AP m empfängt, dann wird er folglich in Runde $r_1 + m.res\ m$ zum letzten Mal übertragen und in Runde $r_1 + m.res + 1$, also spätestens zum Zeitpunkt $t_{AP} + (m.res + 1) \times \Delta_{Round}$, $d = accept$ entscheiden und an die Stationen übertragen. Nach Satz 5.1 werden alle Stationen, die bis zum Zeitpunkt

$$\begin{aligned} t_1 &= t_{AP} + (m.res + 1) \times \Delta_{Round} + OD \times \Delta_{Slot} + t_m \\ &= t_0 + 2 \times (m.res + 1) \times \Delta_{Round} + 2 \times t_m + OD \times \Delta_{Slot} \\ &\leq t_0 + \Delta_{BC} \end{aligned}$$

nicht ausgefallen sind, diese Entscheidung spätestens zu diesem Zeitpunkt ausliefern und dabei $s'.s.bc := \varepsilon$ setzen. Da keine Station m nach t_1 nochmals empfängt, kann keine Station m nach t_1 ausliefern.

Nun muß noch gezeigt werden, daß die im vorhergehenden Abschnitt getroffene Annahme, daß zum Ankunftszeitpunkt einer SDU m von Station s am AP $ap.s.aSDU=\varepsilon$ ist, gilt. Dazu soll die folgende Aussage bewiesen werden:

Hilfssatz 1
Wenn s im Zustand *Idle* ist, dann gilt $ap.s.aSDU = \varepsilon$. Weiterhin gilt: Wenn sg_0 der nächste Slot ist, in dem s eine Polling-Nachricht empfängt, und d_{sg_1} die nächste *accept*- oder *reject*-Entscheidung ist, die s bearbeitet, mit $s.g_{sg_1}[o_{sg_1}] = s$, dann ist $sg_1 > sg$.

Wenn s aus dem Zustand *Init* in den Zustand *Idle* übergeht, dann gilt die Aussage, weil s noch keine SDU an den AP übertragen hat und da der AP nur $d = accept$ bzw. $d = reject$ entscheidet, wenn $aSDU \neq \varepsilon$. Im Folgenden wird gezeigt, daß jede Folge von Übergängen, die vom Zustand *Idle* zurück in den Zustand *Idle* führt, die Aussage aufrecht erhält. Zwei solche Folgen gibt es:

1. *Idle* → *Idle*:
Da s eine leere Request-Nachrichten an den AP sendet, bleibt die Aussage erhalten.

2. *Idle* → *Busy* [→ *Wait*] → *Idle:*
s hat ein *A-BC(m, res)* Signal empfangen und ist in den Zustand *Busy* gewechselt. Es gibt zwei Möglichkeiten, die s in den Zustand *Idle* zurückgeführt haben können:

(a) *s* hat eine *accept*- oder *reject*-Entscheidung d_{sg} bearbeitet, für die $s.g_{sg}[o_{sg}]=s$. Aufgrund der Annahme muß der AP diese Entscheidung getroffen haben, nachdem er *m* empfangen hat. Folglich gilt nach dem Treffen der Entscheidung und bis zum Empfang der nächsten SDU von *s*, daß $ap.s.aSDU=\varepsilon$ ist und daher, daß der AP die nächste *accept*- oder *reject*-Entscheidung $d_{sg'}$ mit $ap.g_{sg'}[o_{sg'}]$=s erst treffen kann, wenn er wieder eine SDU von *s* empfangen hat.

(b) $nR_{sg} > lrts + aSdu.res + 1$. Folglich ist $ap.r_{sg} \geq lrts+aSdu.res + 1$ (Lemma 14). Die Polling-Nachricht $poll_{sg_0}(lrts)$ ist die letzte, die *s* mit *aSdu=m* beantwortet. Daher gibt es zwei Möglichkeiten:

- Der AP hat *m* nicht empfangen, dann wird er auch in den Slots $]sg_0, sg]$ keine *SDU* von *s* empfangen und es folgt die Aussage.

- der AP empfängt *m* spätestens in Runde *lrts* und broadcastet *m* aufgrund der Annahme unmittelbar. Daher wird er, wie oben bewiesen, spätestens in Slot sg_1, nachdem er *s* mit Rundennummer $lrts + m.res + 1$ gepollt hat, eine Entscheidung $d_{sg1} \neq no\ decision$ treffen. Da in Slot *sg* $ap.r_{sg} \geq lrts + aSdu.res + 1$ und $ap.g_{sg}[o_{sg}]=s$, folgt $sg \geq sg_1$ und damit die Aussage.

■

Satz 5.11 (Validität)
Wenn eine korrekte Station *s* eine Nachricht *m* mit Resiliency *OD* broadcastet (*s.receive(A-BC(OD, m))*), dann wird *m* von jeder korrekten Station ausgeliefert.

■

Beweis
Sei $s.r = r'$, zu dem Zeitpunkt an dem *receive(A-BC(m,OD))* erfolgt. Daher ist $s.lrts=r'+OD+1$. Aufgrund von Lemma 3 und Satz 5.7 wird der AP die Polling-Nachrichten $poll(r'+1), \ldots, poll(r'+OD+1)$ an *s* senden. Nach Lemma 10 wird *s* mindestens eine dieser Polling-Nachrichten empfangen, sie mit einer Request-Nachricht *rqu(sl, OD, m)* beantworten und der AP wird diese Request-Nachricht empfangen. Sei *poll(r'')* die erste Polling-Nachricht, für die dies der Fall ist und $rqu_{r''}$ die Antwort von *s* auf diese Polling-Nachricht. Beim Empfangen von *poll(r'')* ist *s* im Zustand *Busy*, da die folgenden vier Aussagen gelten:

- Da $r'' \leq r' + OD + 1 = s.lrts$, ist zum Empfangszeitpunkt von *poll(r'')* $ap.r = r'' \leq s.lrts$ und daher $s.nR \leq s.lrts+1 \leq s.lrts + aSdu.res + 1$ (Lemma 14).

- Da $rqu_{r''}$ die erste Request-Nachricht von *s* ist, die der AP empfängt, seit *s* den Zustand *Idle* verlassen hat, kann *s* nach dem Empfang von *A-BC(OD, m)* noch keine Entscheidung d_{sg} mit $ap.g_{sg}[o_{sg}] = s$ empfangen haben (Hilfssatz 1).

- Da $rqu_{r''}$ die erste Request-Nachricht von *s* ist, die der AP nach *A-BC(m)* empfängt, kann der AP noch keine Broadcast-Nachricht *bc(sg', m', sl')* gebroadcastet haben, mit $ap.g_{sg}[o_{sg}] = s$ und $sl' = sl$.

- Da die Polling-Nachrichten aufsteigend numeriert sind und $r'' \leq r' + OD + 1 = s.lrts$, hat s noch keine Polling-Nachricht *poll(r)* mit $r > s.lrts$ empfangen.

Der AP wird also m in der Request-Nachricht *rqu(sl, m, OD)* empfangen. Da $sl > ap.s.sl$ und $ap.s.aSDU = \varepsilon$ (nach Hilfssatz 1), wird der AP $ap.s.aSDU := m$ und $ap.s.tr := 1$ setzen.

Wenn r_1 die Runde ist, in der der AP die SDU m empfangen hat, dann wird er spätestens in Runde $r_1 + OD + 1$ eine Entscheidung d_{sg} treffen mit $ap.g_{sg}[o_{sg}]=s$, da in dieser Runde $g[o].tr$ in *EVAL_DEC* den Wert $OD+1$ hat. Da s korrekt ist und $ap.s.res = OD$, kann der AP nur eine *accept*-Entscheidung treffen (Lemma 10). Für diese *accept*-Entscheidung kann es zwei Ursachen geben:

1. $gacks_{sg}[s.id][]=true$. Nach Lemma 15 gilt für jede Station $s' \in ap.g_{sg}$: $s'.s.bc = ap.s.aSDU$. Weiterhin ist nach Satz 5.7 jede korrekte Station in $ap.g_{sg}$.

2. $ap.s.tr=OD+1$. Der AP hat $ap.s.aSDU$ $OD+1$-mal gebroadcastet. Nach **(Medium 3)** hat jede korrekte Station eine dieser Broadcast-Nachrichten empfangen. Daher gilt für jede korrekte Station s', daß $s'.s.bc = ap.s.aSDU$.

Nach Satz 5.1 wird jede korrekte Station s' die Entscheidung d_{sg} in einer Broadcast-Nachricht *bc(sg')* empfangen und bearbeiten. Da weiterhin *bc(sg')* die erste Broadcast-Nachricht mit einer Sequenznummer größer oder gleich sg ist, die s' empfängt, gilt beim Bearbeiten von d_{sg} $s'.s.bc = m$. Daher wird s' SDU m ausliefern.

■

Da der Stationsautomat *A-BC(res, m)* nur dann verarbeiten kann, wenn er sich im Zustand *Idle* befindet, ist die Garantie der Zuverlässigkeit natürlich nur auf solche Nachrichten beschränkt, die s broadcastet, wenn der Stationsautomat im Zustand *Idle* ist. Diese Einschränkung ist notwendig, da kein Protokoll Zuverlässigkeit und Rechtzeitigkeit der Übertragung sicherstellen kann, wenn die Anwendung Nachrichten mit einer beliebig hohen Rate an das Protokoll übergibt. Auf der anderen Seite kann die angegebene Einschränkung natürlich dazu führen, daß die Protokollgarantien trivial werden, nämlich dann, wenn der Automat nie, oder nur einmal, im Zustand *Idle* ist und daher nur die erste Benutzernachricht annimmt. Von Bedeutung ist daher die Frage, wie schnell die Anwendung, selbst beim Eintreten des ungünstigsten Falles, die Nachrichten an den Protokollautomaten übergeben kann. Diese Frage wird durch den folgenden Satz beantwortet.

Satz 5.12 (Echtzeit-Durchsatz)
Wenn Station s zum Zeitpunkt t_0 *A-BC(res, m)* empfängt, und daher von Zustand *Idle* nach Zustand *Busy* wechselt, dann befindet sich s spätestens zum Zeitpunkt $t_0 + 2 \times (res + 1) \times \Delta_{Round} + (OD + 1) \times \Delta_{Slot}$ wieder im Zustand *Idle*. Hierbei sei $\Delta_{Round} := n \times \Delta_{Slot}$ und $\Delta_{Slot} := 3 \times t_m$.

■

Beweis
Wenn Station s zum Zeitpunkt t_0 und $s.r = r^c$ das Signal *A-BC(res, m)* empfängt, dann wird der AP spätestens zum Zeitpunkt $t_0 + 2 \times (res + 1) \times \Delta_{Round}$ Station s mit Rundennummer $r^c + 2 \times (res + 1) = lrts + res + 1$ pollen (o. B. d. A. in Slot sg_0). Spätestens $(OD + 1) \times \Delta_{Slot}$ Zeiteinheiten später empfängt s eine Broadcast-Nachricht und wird d_{sg_0} bearbeiten. Beim Bearbeiten von d_{sg_0} ist $s.g_{sg_0}[o_{sg_0}] = s$ und nach Lemma 14 $s.nR_{sg_0} = ap.r_{sg_0} + 1 > lrts + res + 1$. Daher wird s *decide* auf *true* setzen und in den Zustand *Idle* wechseln.

■

Eine weitere Steigerung des Echtzeit-Durchsatzes kann erreicht werden, wenn alle Nachrichten einer Station die gleiche Resiliency haben. In diesem Falle kann eine Station bereits mit dem Übertragen der nächsten Request-Nachricht beginnen, wenn sicher ist, daß die vorhergehende Request-Nachricht beim AP angekommen ist (oder nie mehr ankommen wird). Spätestens zum ungünstigsten Ankunftszeitpunkt der Folgenachricht beim AP wird dieser eine Entscheidung über die Vorgängernachricht treffen und daher den Transport der Folgenachricht nicht über den ungünstigsten Zeitpunkt hinaus verzögern.

Satz 5.13 (Uniforme Einigung)
Liefert eine Station s die Nachricht m aus, dann liefert jede korrekte Station die Nachricht m aus. ■

Beweis
Liefert eine Station s die Nachricht m aus, dann hat sie eine *accept*-Entscheidung des AP bearbeitet und $s.g_{sg}[o_{sg}].bc = m$. Nach Satz 5.2 hat der AP diese Entscheidung gebroadcastet mit $ap.g_{sg}[o_{sg}] = s.g_{sg}[o]_{sg} = s'$ (Lemma 9). Wenn der AP eine *accept*-Entscheidung trifft, kann dies nur zwei Gründe haben, die beide implizieren, daß zu diesem Zeitpunkt für jede korrekte Station s'' $s''.s'.bc = ap.s'.aSDU$ gilt (s. Beweis von Satz 5.11). Da s die Entscheidung d_{sg} bearbeitet, hat sich nach Fehlerannahme s bis zum Slot sg wie eine korrekte Station verhalten und es gilt beim Bearbeiten von d_{sg}, daß $s.s'.bc = ap.s'.aSDU_{sg}$ und damit für jede korrekte Station s'' beim Bearbeiten von d_{sg}, daß $s''.s'.bc = s.s'.bc = m$. Da jede korrekte Station d_{sg} bearbeitet (Satz 5.1), liefert jede korrekte Station m aus.

■

Satz 5.14 (Totale Ordnung)
Liefern Stationen s_1 und s_2 die Nachrichten m_1 und m_2 aus, dann liefert Station s_1 Nachricht m_1 genau dann vor Nachricht m_2 aus, wenn s_2 m_1 vor m_2 ausliefert.

■

Beweis
Liefert s_1 Nachricht m_1 aus dann hat s_1 eine Entscheidung d_{sg_1} bearbeitet, mit $s_1.g_{sg_1}[o_{sg_1}].bc = m$ und $s_1.g_{sg_1}[o_{sg_1}]=m.orig$. Ebenso hat Station s_2, wenn sie m ausliefert eine Entscheidung d_{sg_2} bearbeitet, mit $s_2.g_{sg_2}[o_{sg_2}].bc = m$ und $s_2.g_{sg_2}[o_{sg_2}]=m.orig$. Sei

scheidung d_{sg_2} bearbeitet, mit $s_2.g_{sg_2}[o_{sg_2}].bc = m$ und $s_2.g_{sg_2}[o_{sg_2}]=m.orig$. Sei o. B. d. A. $sg_1 \leq sg_2$, dann hat s_2 sowohl d_{sg_1} als auch d_{sg_2} bearbeitet. Wie im Beweis der Einigung (Satz 5.13) folgt, daß s_2 beim Bearbeiten von d_{sg_1} ebenfalls m ausgeliefert hat. Da keine Station m zweimal ausliefert (Satz 5.9) folgt, daß $sg_2 = sg_1$, d. h. für jede Nachricht m, die ausgeliefert wird, gibt es eine eindeutige Entscheidung, die zum Ausliefern dieser Nachricht führt. Da alle Stationen die Entscheidungen in der gleichen Reihenfolge bearbeiten (Satz 5.4), folgt, daß alle Stationen die Nachrichten in der gleichen Reihenfolge ausliefern.

■

Satz 5.15 (Virtuelle Synchronität)
Wenn zwei Stationen s_1 und s_2 zwei Änderungsnachrichten *MEMCH*(s_3) und *MEMCH*(s_4) ausliefern, dann liefern beide Stationen zwischen diesen Nachrichten die gleiche Menge von Broadcast-Nachrichten aus.

■

Beweis
Wenn die Stationen s_1 und s_2 die Änderungsnachrichten *MEMCH*(s_3) und *MEMCH*(s_4) ausliefern, dann liefern beide Stationen die Nachrichten in der gleichen Reihenfolge aus (Satz 5.6), o. B. d. A. *MEMCH*(s_3) vor *MEMCH*(s_4) aufgrund der Entscheidungen d_{sg} und $d_{sg'}$, $sg < sg'$. Nach Satz 5.4 haben beide Stationen zwischen diesen Entscheidungen die gleichen *accept*-Entscheidungen des AP (d_{sg_0} …, d_{sg_n}) bearbeitet. Sei angenommen, daß s_1 beim Bearbeiten dieser Entscheidungen die SDUs m_1, …, m_n ausgeliefert hat, dann folgt wie im Beweis von Satz 5.13, daß s_2 beim Bearbeiten der *accept*-Entscheidungen ebenfalls die SDUs m_1, …, m_n ausgeliefert hat. ■

6 Implementierung und Messungen

6.1 Implementierung des Protokolls

Neben der Konzeption und formalen Beschreibung wurde auch eine Implementierung des Protokolls durchgeführt. Da zum Zeitpunkt der Implementierung die angestrebte Zielhardware noch nicht zur Verfügung stand, wurde als Basis der Implementierung eine Hardwarestruktur aus PCs und Laptops gewählt, die über Funknetzkarten nach dem IEEE 802.11 Standard der Firma Lucent Technologies (WaveLAN/IEEE) verbunden sind. Diese Funknetzkarten können als PCMCIA-Steckkarten in die PC bzw. Laptop-Hardware integriert werden. Als Betriebssystem wird Windows NT 4.0 verwendet.

Unter Verwendung der PCMCIA-Netzwerkkarten der Firma Lucent Technologies kann ein standardkonformes Ad-Hoc Netzwerk realisiert werden, auf das die Stationen nach der DCF zugreifen. Die fakultative Funktionalität der PCF steht allerdings noch nicht zur Verfügung, so daß diese im Rahmen der Implementierung durch eine spezielle Station emuliert werden muß.

Als Netzwerkschnittstelle für die Implementierung werden Sockets verwendet, die eine verbreitete Schnittstelle für den Zugriff auf die Netzwerkdienste eines Betriebssystems darstellen. Speziell kommen UDP-Sockets zum Einsatz, die den Transport von Nachrichten über den UDP/IP-Protokollstapel erlauben. UDP-Sockets stellen der Anwendung einen unzuverlässigen Datagrammdienst zur Verfügung, so daß sie die über das Netzwerk getroffenen Zuverlässigkeitsannahmen nicht durch weitere Services verfälschen. Auf eine Implementierung auf einer niedrigeren Ebene, z. B. in den Netzwerktreibern, wurde verzichtet, da eine grundsätzliche Realisierung des Protokollautomaten auch unabhängig von den Spezifika des gewählten Systemlevels und der konkreten Hardware durchgeführt werden kann. Eine Anpassung an solche speziellen Eigenschaften soll erst im Rahmen einer Portierung auf die Zielhardware durchgeführt werden.

Anhand der Implementierung sollte sich zum einen zeigen, ob die Zuverlässigkeit der Nachrichtenübertragung mit einer begrenzten Anzahl von Neuübertragungen zu erreichen ist, und zum anderen, ob das Protokoll in der Lage ist, durch den synchronen Kanal die Einigung unter den Stationen sicherzustellen, wenn es bei einer Resiliency unter dem Omission-Degree zu Nachrichtenverlusten kommt. Des weiteren sollten erste Messungen durchgeführt werden, die Aufschluß über den erreichbaren tatsächlichen Durchsatz und die tatsächlichen Verzögerungen - im Gegensatz zu den berechneten garantierbaren Größen - geben. Aufgrund der gewählten Architektur (NT 4.0 / UDP-Sockets) kann allerdings im Rahmen der Implementierung eine Überprüfung der Echtzeiteigenschaften sicherlich noch nicht durchgeführt werden, da hierzu das Verhalten des Schedulings und des Protokollstapels zu wenig vorhersagbar ist. Wenn sich aber anhand der Implementierung bestätigen läßt, daß durch eine begrenzte Anzahl von Neuübertragun-

gen die Zuverlässigkeit erreichbar ist, dann ist dies ein starkes Indiz dafür, daß die Rechtzeitigkeit in einer geeigneten Systemarchitektur sichergestellt werden kann.

Die Realisierung des Protokollautomaten erfolgte nicht in Form eines eigenständigen Prozesses, sondern in Form einer Library. Auf diese Weise teilen Protokoll und Anwendung einen gemeinsamen Adressraum, der zum Austausch der Informationen zwischen Protokoll und Anwendung genutzt werden kann. So kann lokal auf Interprozeßkommunikationskonzepte des Betriebssystems, die in ihrem zeitlichen Verhalten nicht vorhersagbar sind, verzichtet werden.

Der Implementierung des Protokolls wurde, wie der formalen Beschreibung, das Konzept des erweiterten endlichen Automaten zugrunde gelegt. Für jede Kombination von Automatenzustand und Eingabesignal wurde eine Prozedur implementiert, die die Aktionen des Automaten beim entsprechenden Zustandsübergang realisiert. Der Aufruf der Prozeduren erfolgt anhand einer Tabelle, in der zu jeder Kombination von Eingabesignal und Automatenzustand die Adresse des entsprechenden Unterprogramms gespeichert ist. Dieser Automat, sowie seine Umgebung, beispielsweise Puffer, wurden in der Programmiersprache C++ mit insgesamt ca. 4900 Zeilen Code programmiert.

Die Protokollautomaten bearbeiten zwei Arten von Eingabesignalen: Nachrichten, die sie über die Socket-Schnittstelle empfangen, und Timeouts. Die Reaktion auf die beiden Arten von Eingabesignalen erfolgt jeweils durch einen Thread, der auf den Eingang eines Signals wartet und bei Ankunft den entsprechenden Übergang durchführt. Eine Sperre stellt sicher, daß nie zwei Zustandsübergänge gleichzeitig erfolgen. Alle Zugriffe des Automaten auf das Netzwerk, d. h. das Senden und Empfangen von Nachrichten, erfolgen durch Methoden einer speziellen Klasse. Auf diese Weise wird eine spätere Portierung des Protokolls vereinfacht. Des weiteren bildet diese Klasse einen geeigneten Ort für die Realisierung eines einfachen Software-Fehlerinduktionsmechanismus.

Die Kommunikation zwischen Anwendung und Protokoll geschieht über Puffer im gemeinsamen Speicher. Dazu stehen ein Ein- und ein Ausgabepuffer bereit. Der Zugriff auf die Puffer durch den Automaten ist nicht blockierend und wird daher auch nicht durch eigene Threads durchgeführt.

Die Schnittstelle der Anwendung zum Protokoll ist durch ein Objekt mit den Methoden *start, stop, send, receive* und *getLastError* realisiert. Die Methoden haben die folgende Bedeutung:

bool start(): Der Eingabe- und der Ausgabepuffer sowie die Protokollthreads werden erzeugt. Nach dem Aufruf von *start* kann das Protokoll auf die Eingabesignale reagieren. Der Aufruf liefert *true*, wenn das Protokoll erfolgreich gestartet werden konnte, *false* sonst.

stop(): Die Protokollthreads und Puffer werden aufgelöst. Das Protokoll kann nicht mehr auf Eingabesignale reagieren.

*bool send(idu *)*: Fügt die Idu in den Eingabepuffer ein. Der Aufruf ist nicht blockierend und liefert *false*, wenn der Puffer voll ist.

*bool receive(idu *)*: Entnimmt eine SDU dem Puffer. Der Aufruf blockiert, wenn der Puffer leer ist. Der Aufruf liefert *false*, wenn das Protokoll beendet wird oder wenn ein Fehler auftritt.

int getLastError(): Falls ein Fehler auftritt, kann mit *getLastError()* der zugehörige Fehlercode ermittelt werden.

Die Implementierung erfolgte unter der Annahme, daß alle SDUs die gleiche Resiliency haben. Daher konnte das Protokoll dahingehend optimiert werden, daß eine Station mit der Übertragung einer SDU bereits dann beginnen kann, wenn sie sicher ist, daß die vorhergehende vom AP empfangen wurde. Hierdurch kann das Übertragen einer SDU von einer Station zum AP nebenläufig zum Übertragen der vorhergehenden SDU der selben Station vom AP zu den Stationen erfolgen.

Im Rahmen der Implementierung zeigte sich, daß beim Versenden von Punkt-zu-Punkt Nachrichten auf dem Medium, im Gegensatz zur Spezifikation des Standards, Duplikate von Nachrichten entstehen. Um diesen Fehler zu umgehen, wurden alle Nachrichten, auch die Punkt-zu-Punkt-Nachrichten, als Broadcast-Nachrichten versendet. Um beim Broadcast von Punkt-zu-Punkt-Nachrichten den Adressaten feststellen zu können, wurden die versendeten Nachrichten um ein Feld erweitert, das die IP-Adresse der Zielstation enthält. Durch die Notwendigkeit, den Fehler zu umgehen wurde die Effizienz des Protokolls natürlich eingeschränkt.

6.2 *Messergebnisse*

Das Ziel der im Folgenden geschilderten Messungen bestand einerseits darin, Aufschluß über bestimmte Eigenschaften des Mediums zu gewinnen; andererseits sollten erste Aussagen über die Leistungsfähigkeit des Gruppenkommunikationsprokolls getroffen werden.

6.2.1 Messungen zu Eigenschaften des Funkmediums

Um die Gültigkeit der über das Medium getroffenen Annahmen sicherzustellen, müssen Messungen durchgeführt werden, die eine geeignete Wahl des Omission Degree sowie der Verzögerung t_m des Mediums erlauben. Weiterhin unterstützen solche Messungen den Benutzer bei der Wahl einer geeigneten Resiliency.

Da die Kommunikation der Protokollautomaten über UDP-Sockets vonstatten geht, ist die relevante Größe zum Bestimmen der Verzögerung des Kommunikationsmediums für durchgeführte Implementierung die Verzögerung von UDP-Broadcasts. Zunächst

wurde daher die Ende-zu-Ende-Verzögerung für die Übertragung von UDP-Broadcasts zwischen Prozessen auf zwei unterschiedlichen Stationen in der folgenden Weise bestimmt: Eine Meßstation sendet Nachrichten, in die umittelbar vor der Übertragung Zeitstempel eingefügt werden, an eine zweite Station. Diese empfängt die Nachricht und sendet sie an die Meßstation zurück. Beim Empfang der Nachricht nimmt die Meßstation erneut einen Zeitstempel und berechnet die Differenz zwischen dem somit bestimmten Empfangszeitpunkt und der in der Nachricht enthaltenen Sendezeit. Auf diese Weise wird die Verzögerung, welche die Nachricht auf dem Hin- und Rückweg zusammen erfährt, bestimmt. Unter der Annahme, daß die Verzögerungen in beiden Richtungen etwa den gleichen Verteilungen unterliegen, wurde durch Halbieren der gemessenen Werte die Nachrichtenverzögerungen für eine Übertragungsstrecke bestimmt.

Die Messungen wurden mit den Paketgrößen 12 (Größe der Polling-Nachricht), 100, 250, 500, 750 und 1000 durchgeführt, jeweils mit einem Stichprobenumfang von 1000 Nachrichten (s. Abb. 6.1 und Tab. 6.1).

Die zu einer Nachrichtengröße gemessenen Verzögerungen liegen größtenteils sehr dicht beieinander, wie sich in den kleinen Streuungen und Quartilsabständen[15] zeigt. Die Meßwerte haben aber dennoch, trotz des relativ kleinen Stichprobenumfangs, eine große Spannweite[16]. Da Windows NT kein Echtzeitbetriebssystem ist muß sogar davon ausgegangen werden, daß selbst die gemessenen Maximalwerte noch keine obere Schranke für die Verzögerung der UDP-Broadcasts darstellen. Die Wahl eines geeigneten Wertes für den Parameter t_m ist daher in der Implementationsumgebung nicht möglich. Aus diesem Grunde wird auf eine empirische Überprüfung der Rechtzeitigkeit des Protokolls in der Implementationsumgebung verzichtet. Da die Übertragung der Nachrichten über UDP-Sockets erfolgt, also jede Nachricht im gesamten UDP/IP-Protokollstapel bearbeitet wird, ist davon auszugehen, daß ein nicht unerheblicher Teil der Verzögerungen in den lokalen Betriebssystemen erzeugt wird. Daher lassen sich die gemessenen Werte nicht auf das Funknetz an sich beziehen und sie stellen folglich die Existenz einer geeigneten Konstante t_m für das Funknetz nicht in Frage. Vielmehr verdeutlichen diese Werte, daß zur Realisierung der Echtzeiteigenschaften die Implementierung auf einer niedrigeren Ebene des Systems und unter Einsatz eines Echtzeitbetriebssystems erfolgen muß. Die empirische Überprüfung der Rechtzeitigkeit des Protokolls wird daher in der Zielumgebung erfolgen.

---

[15] Das *p-Quantil* x_p einer Meßreihe $x_1, \ldots, x_n$ ist für $0 < p < 1$ definiert durch $x_p = \begin{cases} x_{(np)} & \text{falls } np \text{ ganzzahlig} \\ x_{([np+1])} & \text{sonst} \end{cases}$, wenn $[a]$ die größte ganze Zahl, die nicht größer als a ist, bedeutet und $x_{(1)}, \ldots, x_{(n)}$ die zugehörige geordnete Meßreihe bezeichnet. Der Quartilsabstand ist die Differenz zwischen dem 0.75- und dem 0.25-Quantil [LW92].

[16] Die Spannweite ist die Differenz zwischen dem Maximum und dem Minimum der Meßreihe [LW92].

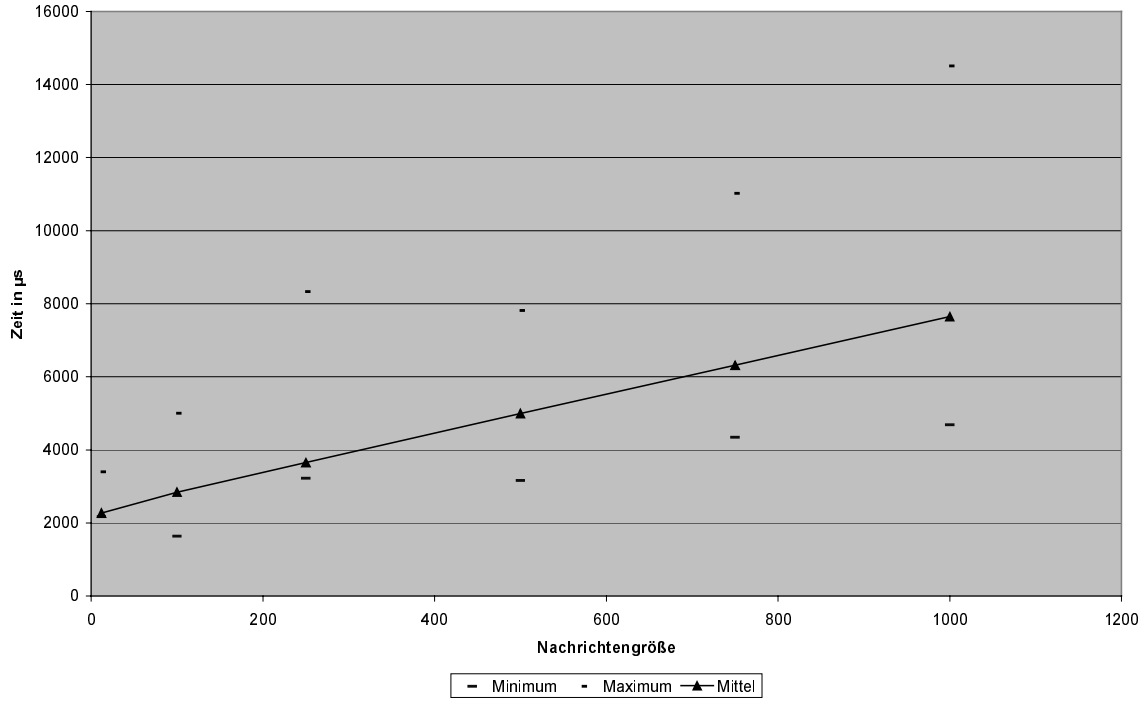


Abbildung 6.1 Verzögerung von UDP-Broadcasts

	Paketgröße					
	12	100	250	500	750	1000
Minimum	2171,5	1631,5	3223	3163	4347,5	4689
Maximum	3397	5003	8326,5	7811,5	11017,5	14505
Mittel	2271,05	2843	3651,55	4995,24	6314,63	7646,68
Streuung	51,83	96,08	163,89	398,26	252,27	395,91
Spannweite	1225,5	3371,5	5103,5	4648,5	6670	9816
Quartilsabst.	9,5	12,5	9	19,5	23,5	29,13

Tabelle 6.1 Kenngrößen der gemessenen Verzögerungen in µs

Betrachtet man zu jeder Nachrichtengröße den empirischen Mittelwert der Meßwerte, so zeigt sich ein linearer Zusammenhang zwischen Nachrichtengröße und Verzögerung (s. Abb. 6.1). Die Existenz eines Achsenabschnittes von ca. 2ms zeigt, daß beim Nachrichtentransport unabhängig von der Größe der transportierten Nachricht ein Overhead entsteht, der auch beim Versenden von sehr kleinen Nachrichten anfällt. Durch den Einsatz von Piggy-Backing müssen weniger solcher kleinen Nachrichten versendet werden, wodurch sich der insgesamt erzeugte Overhead deutlich reduziert.

Neben der Verzögerung besteht eine weitere wichtige Eigenschaft des Mediums in seiner Zuverlässigkeit. Messungen zur Zuverlässigkeit des Funkmediums bilden die Grundlage für eine angemessene Wahl des Omission Degrees und der Resiliency von Nachrichten. Allerdings ist speziell beim Funknetz die Zuverlässigkeit des Mediums abhängig von der Umgebung, in der das Netzwerk arbeitet. Daher sind die gemessenen Werte nicht auf jede andere Umgebung übertragbar. Bei diesen sowie auch bei den fol-

genden Messungen befanden sich alle Stationen in einem Raum, es herrschten also relativ günstige physische Bedingungen. Zur Bestimmung der Zuverlässigkeit des Mediums wurden zunächst die Verlustraten von Broadcast-Nachrichten und von Polling-Request-Paaren bestimmt. Die Messungen erfolgten, während das Protokoll Pakete 1000 Byte Nutzdaten transportierte.

Bei der Bestimmung der Zuverlässigkeit der Broadcast-Nachrichten wurde zum einen von jeder Station die lokale Verlustrate bestimmt, d. h. jede Station stellte fest, welcher Anteil der Broadcast-Nachrichten sie selbst nicht empfangen hat. Zum anderen wurde die globale Anzahl von Verlusten bestimmt, d. h. es wurde festgestellt, wie viele der Broadcast-Nachrichten von mindestens einer Station nicht empfangen wurden. Um die globalen Verluste bestimmen zu können, speicherte jede Station die Sequenznummer der nicht-empfangenen Broadcast-Nachrichten. Nach dem Ende der Messung konnte durch mischen der gespeicherten Sequenznummern und entfernen von mehrfach auftretenden Werten die Anzahl der Nachrichten bestimmt werden, die von mindestens einer Station nicht empfangen wurden. Es ergaben sich die folgenden Ergebnisse:

	Stationen			
	Direct3	Rhodos	Madeira	global
Nicht empfangene Broadcast-Nachrichten	1.280	2.783	2.549	4.695
Verlustraten (p. m.)	0,0958	0,2084	0,1908	0,3515

Tabelle 6.2 Verlustraten von Broadcast-Nachrichten. Insgesamt wurden 13.355.215 Broadcast-Nachrichten versendet.

Es zeigt sich, daß die lokalen Verlustraten sich deutlich unterscheiden, was höchstwahrscheinlich auf die räumlichen Positionen der Stationen zurückzuführen ist.

Die Verlustraten für Polling-Request-Nachrichten wurden durch den AP gemessen, wobei sich ergab, daß von 22.845.125 Polling-Nachrichten, die der AP gesendet hat, 13.798 nicht mit einer Request-Nachricht beantwortet wurden. Dies sind 0,6039 Promille.

	Burstlänge					gesendete
Station	0	1	2	3	12	Broadcasts
direct3	15.069.025	1194	3	1	0	15.070.228
madeira	10.622.397	1362	19	6	1	10.623.827
thassos	15.067.290	2040	16	2	0	15.069.368
kerkyra	15.065.741	2818	6	0	0	15.068.571

Tabelle 6.3 Burstlängen verlorener Broadcast-Nachrichten

Wichtiger als die Verlustwahrscheinlichkeiten einzelner Broadcast-Nachrichten sind aber die Burstlängen, d. i. die Anzahl aufeinanderfolgender verlorener Broadcast-Nachrichten. Empfängt z. B. eine Station zunächst die Broadcast-Nachricht mit Sequenznummer 25 und dann die mit Sequenznummer 30, so hat sie 4 aufeinanderfolgen-

de Broadcast-Nachrichten des AP verloren. Bei den Messungen ergab sich die folgende Verteilung der Burstlängen für Broadcast-Nachrichten.

Auch diese Messung verdeutlicht, daß die Zuverlässigkeit der Kommunikation zwischen dem AP und einem Empfänger nicht für alle Stationen gleich ist. Es ergibt sich aber dennoch für alle Stationen gemeinsam, daß Bursts einer Länge größer als drei fast nie auftreten. Da aber der Omission Degree eine kritische Größe ist, müssen für seine Bestimmung die gemessenen Maximalwerte, hier ein Wert von 12, zugrunde gelegt werden.

Die Messungen belegen, daß ein großer Omission-Degree zugrunde gelegt werden muß, um eine gute Überdeckung der Annahmen sicher zu stellen; in der betrachteten Umgebung wurde $OD = 15$ gewählt. Andererseits sind Bursts einer Länge größer drei sehr selten. Zusammen mit den Verlustraten, die sich im Promille-Bereich bewegen, zeigt dies, daß der Einsatz statischer Redundanz im weitaus größten Teil der Fälle einen unnötigen Aufwand erzeugen würde.

6.2.2 Messungen zum Gruppenkommunikationsprotokoll

Durch Messungen an der Implementierung sollte zunächst getestet werden, ob mit einer beschränkten Anzahl von Neuübertragungen die Zuverlässigkeit der Nachrichtenübertragung sichergestellt werden kann. Hierzu wurden über einen längeren Zeitraum von drei Stationen Nachrichten mit einer Resiliency von $OD = 15$ übertragen. Bei diesem Test wurden von den Stationen in 16,43 Std. 3.418.912 Nachrichten übertragen, ohne daß Nachrichtenverluste auftraten.

Das Ziel weiterer Messungen bestand darin zu untersuchen, welchen Einfluß die Wahl der Resiliency auf die Zuverlässigkeit der Nachrichtenübertragung hat. Dazu wurden Nachrichten mit einer Nutzlast von 1000 Byte und unterschiedlicher Resiliency von den Stationen übertragen. Die Ergebnisse (s. Abb. 6.2) zeigen, daß bei einer Steigerung der Resiliency die Anzahl der Nachrichtenverluste stark abnimmt, bis schließlich bei einer Resiliency von 3 keine Nachrichtenverluste mehr auftreten. Es zeigt sich also, daß die Wahl einer kleinen Resiliency nicht nur zu einer Verkürzung der maximalen Verzögerung führt, sondern daß i. A. eine kleine Resiliency eine wirklichkeitsnähere Schätzung der Burstlängen darstellt, als der sehr große Omission Degree, der sich am selten eintretenden ungünstigsten Fall orientieren muß.

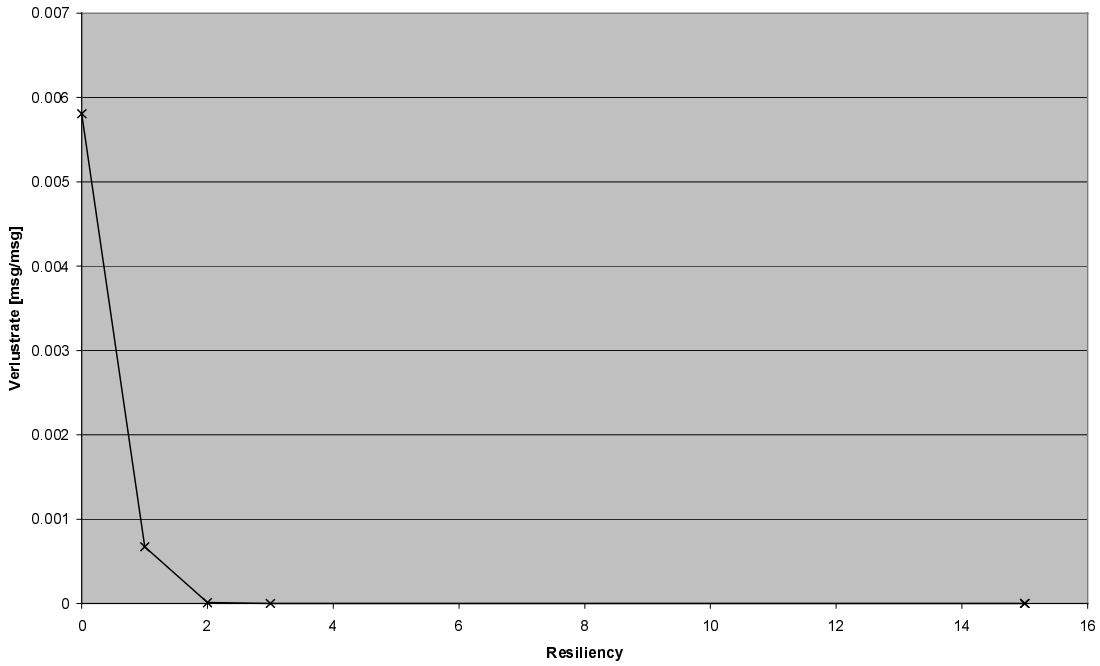


Abbildung 6.2 Die Verlustraten, gemessen in verlorenen Nachrichten pro gebroadcasteten Nachrichten, in Abhängigkeit von der Resiliency.

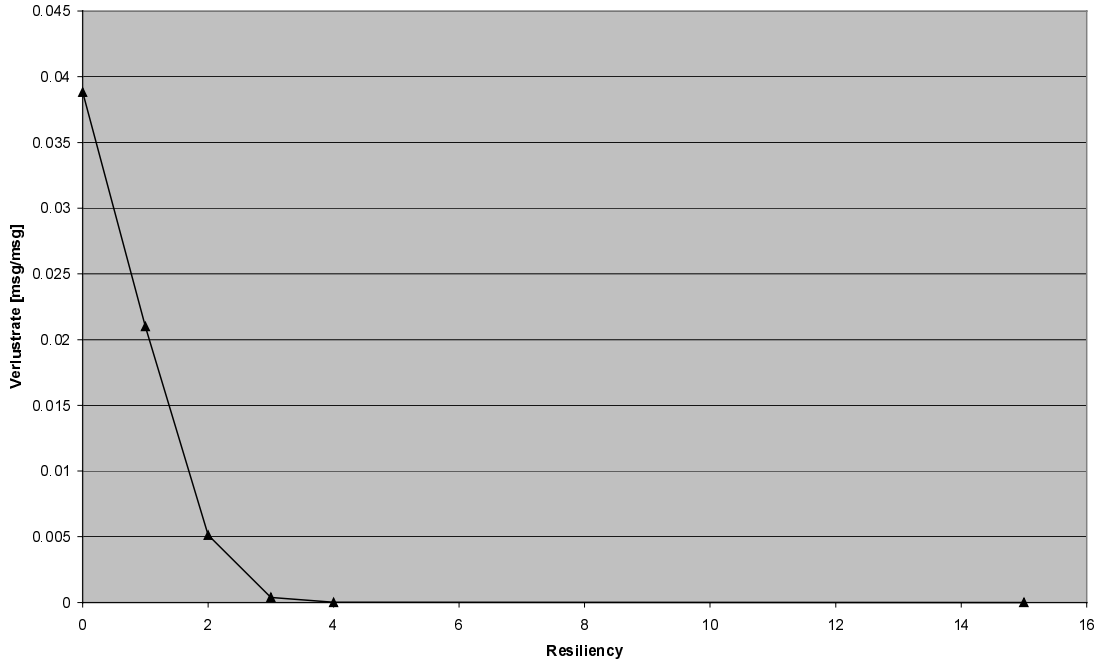


Abbildung 6.3 Die Verlustraten, gemessen in verlorenen Nachrichten pro gebroadcasteten Nachrichten, in Abhängigkeit von der Resiliency bei Anwendung der Fehlerinduktion.

Alle bisher erwähnten Messungen wurden innerhalb eines Büros, also bei relativ günstigen Bedingungen durchgeführt. Um das Protokoll auch unter ungünstigeren Bedingungen zu Testen, wurde ein Software-Fehlerinduktionsmechanismus verwendet. Mit diesem Mechanismus kann die Verbindung zwischen zwei Stationen unterbrochen werden, indem beide Stationen in bestimmten Zeitintervallen alle Nachrichten der jeweils anderen Station unmittelbar über der Socketschnittstelle verwerfen. Jede Station bestimmt die Zeitintervalle, in denen sie die Kommunikation mit einer anderen Station unterbricht, zufällig mit einer spezifizierbaren Wahrscheinlichkeit. Bei den folgenden Messungen betrug diese Wahrscheinlichkeit 2%. Jede Station entschied also für jede andere Station mit einer Wahrscheinlichkeit von 2%, die Verbindung mit dieser Station im nächsten Zeitintervall zu unterbrechen. Damit betrug die Wahrscheinlichkeit, daß die Verbindung zwischen zwei Station in einem bestimmten Zeitintervall unterbrochen war, $2\% + 2\% - 2\% \times 2\% = 3{,}96\%$.

Die gewonnenen Ergebnisse bestätigen die, welche ohne Fehlerinduktion gemessen wurden: Durch eine Steigerung der Resiliency kann die Anzahl der verlorenen Nachrichten stark reduziert werden. Dabei sind schon bei einer Resiliency, die gegen den Omission Degree relativ klein ist, sehr gute Ergebnisse erreichbar. Da der Omission Degree eine kritische Größe ist und auch die seltenen Ausreißer abdecken muß, kann er nicht dieser realistischeren Schätzung angepaßt werden. Durch die Resiliency kann aber der Benutzer mit der erwarteten Anzahl von Nachrichtenverlusten arbeiten, solange Nachrichtenverluste, die durch gelegentliche Ausreißer auftreten könnten, toleriert werden können.

Während der oben erläuterten Messungen mit und ohne Fehlerinduktion traten insgesamt ca. 19800 Nachrichtenverluste auf. Keiner dieser Nachrichtenverluste war asymmetrisch, d. h. alle aufgetretenen Nachrichtenverluste wurden von den Stationen konsistent wahrgenommen. Dies zeigt, daß das Protokoll tatsächlich auch beim Auftreten von Nachrichtenverlusten die Einigung unten den Stationen sicherstellt.

Außer über die Zuverlässigkeit des Protokolls sollten die Messungen aber auch erste Aufschlüsse über den erreichbaren Durchsatz und die tatsächlichen Nachrichtenverzögerungen geben.

Der Durchsatz des Protokolls wurde zunächst unter physisch günstigen Bedingungen für Nachrichten von 1000 Byte Nutzlast gemessen. In Abhängigkeit von der Resiliency ergaben sich die in Tabelle 6.4 dargestellten Werte.

Reciliency	Durchsatz
0	57,00
1	57,71
2	57,87
3	57,97
15	57,77

Tabelle 6.4 Durchsatz, gemessen in ausgelieferten Broadcast-Nachrichten pro Sekunde, in Abhängigkeit von der Resiliency (Nachrichtengröße 1000 Byte).

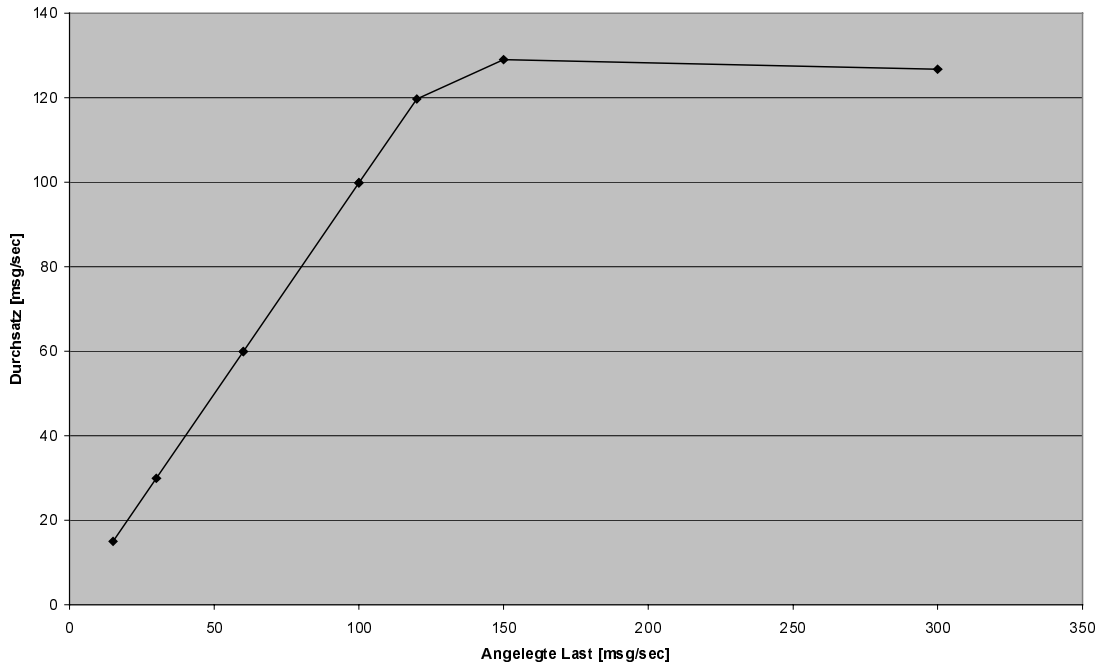


Abbildung 6.4 Durchsatz von Nachrichten mit einer Nutzlast von 100 Byte

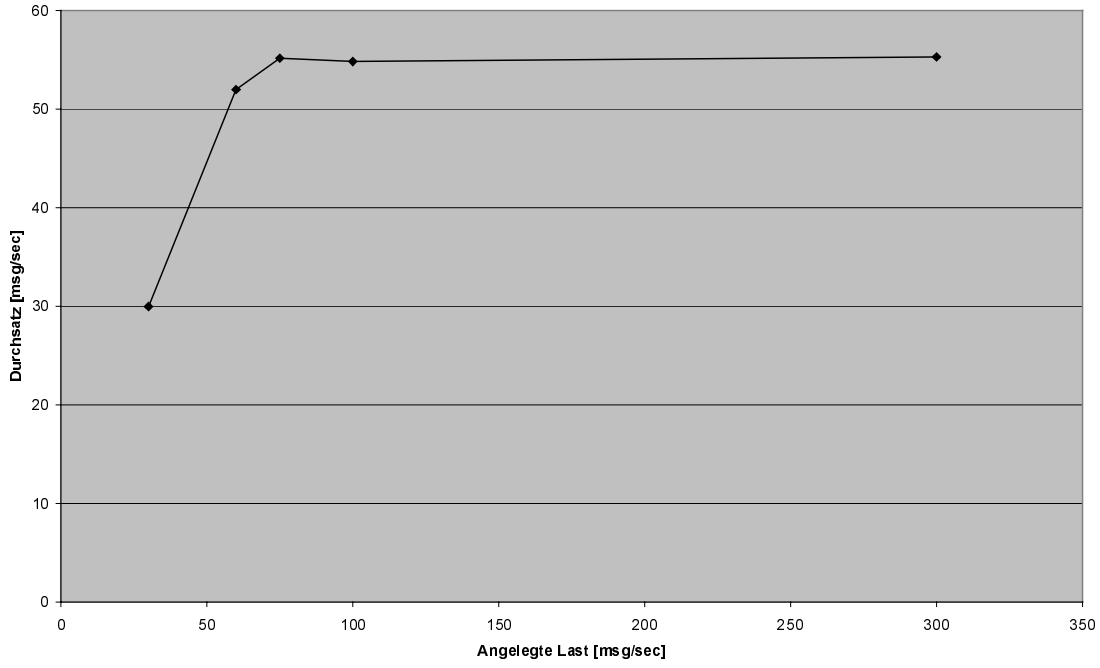


Abbildung 6.5 Durchsatz von Nachrichten mit einer Nutzlast von 1000 Byte

Es zeigt sich, daß die Wahl einer höheren Zuverlässigkeit den tatsächlichen Durchsatz gar nicht oder nur gering beeinflußt. Dies liegt daran, daß die Resiliency der Nachrich-

ten nur dann von Bedeutung ist, wenn auf dem Medium tatsächlich Nachrichtenverluste auftreten. Deshalb verursacht das Tolerieren von Nachrichtenverlusten einen im Vergleich zur Gesamtheit aller versendeten Nachrichten geringen Aufwand und dies i. A. nur in Situationen, in denen es ohne Neuübertragungen zu Nachrichtenverlusten gekommen wäre. Der geringe Einfuß einer höheren Zuverlässigkeit auf den erzielten Durchsatz liegt also im Einsatz dynamischer Redundanz begründet.

In Abhängigkeit von der angelegten Last (Aufrufe von *send(*idu)* pro Sekunde) steigt der Durchsatz des Protokolls linear bis zu einem Maximalwert an; danach bleibt er für steigende Lasten konstant (s. Abb. 6.4 und Abb. 6.5). Es kommt nicht zum Abfallen des Durchsatzes bei steigender Last, wie dies u. U. bei kollisionsauflösenden Medienzugriffsverfahren der Fall sein kann.

Reciliency	Durchsatz
0	100,14
1	98,89
2	96,98
3	97,88
4	97,82
15	100,09

Tabelle 6.5 Durchsatz des Protokolls mit Fehlerinduktion, gemessen in ausgelieferten Broadcast-Nachrichten pro Sekunde, in Abhängigkeit von der Resiliency (Nachrichtengröße 100 Byte).

Auch die Messungen des Durchsatzes wurden unter Verwendung der Fehlerinduktion durchgeführt (s. Tab. 6.5). Da durch den Fehlerinduktionsmechanismus weiterer Aufwand in Form größerer Nachrichtenheader entsteht, ist hier vor allem die qualitative Aussage von Bedeutung. Auch wenn Fehlerinduktion zum Einsatz kommt, beeinflußt die Resiliency den Durchsatz gar nicht oder nur sehr wenig. Diese Ergebnisse bestätigen also ebenfalls den Vorteil der dynamischen Redundanz.

Die Nachrichtenverzögerungen des Gruppenkommunikationsprotokolls wurden für die selben Nachrichtengrößen bestimmt wie die Verzögerungen der UDP-Broadcasts. Für jede Nachrichtenverzögerung wurden die Verzögerungen von 1200 Nachrichten gemessen. Die Verzögerungen wurden unter der Annahme bestimmt, daß die Anwendung die Nachrichten immer zum optimalen Zeitpunkt an das Protokoll übergibt, d. h. unmittelbar vor dem Empfang der Polling-Nachricht. Die Kenngrößen der ermittelten Meßreihen sind in Abbildung 6.6 und Tabelle 6.6 dargestellt.

Die mittleren Verzögerungen wurden sowohl mit geringer Last – jede Station übertrug nur eine Nachricht pro Sekunde – als auch mit hoher Last – die Stationen übertrugen Nachrichten so schnell wie möglich – durchgeführt. Bei geringer Last sind die gemessenen Verzögerungen kürzer, da sowohl der AP als auch die Stationen häufig Nachrichten ohne Nutzlast versenden, weshalb die Länge einer Runde kleiner ist als bei großer Last.

Die gemessenen mittleren Verzögerungen liegen bereits in einem für viele Anwendungen akzeptablen Rahmen. Weiterhin ist davon auszugehen, daß durch eine Portierung des Protokolls auf ein Echtzeit-Betriebssystems und eine Implementierung auf einem niedrigeren Systemlevel noch kürzere Verzögerungen zu erreichen sind. Auch die zur Zeit noch auftretenden großen Maximalwerte (z. B. 311930 µs bei 1000 Byte Nutzlast) können auf diese Weise wohl vermieden werden (vgl. Paragraph 6.2.1).

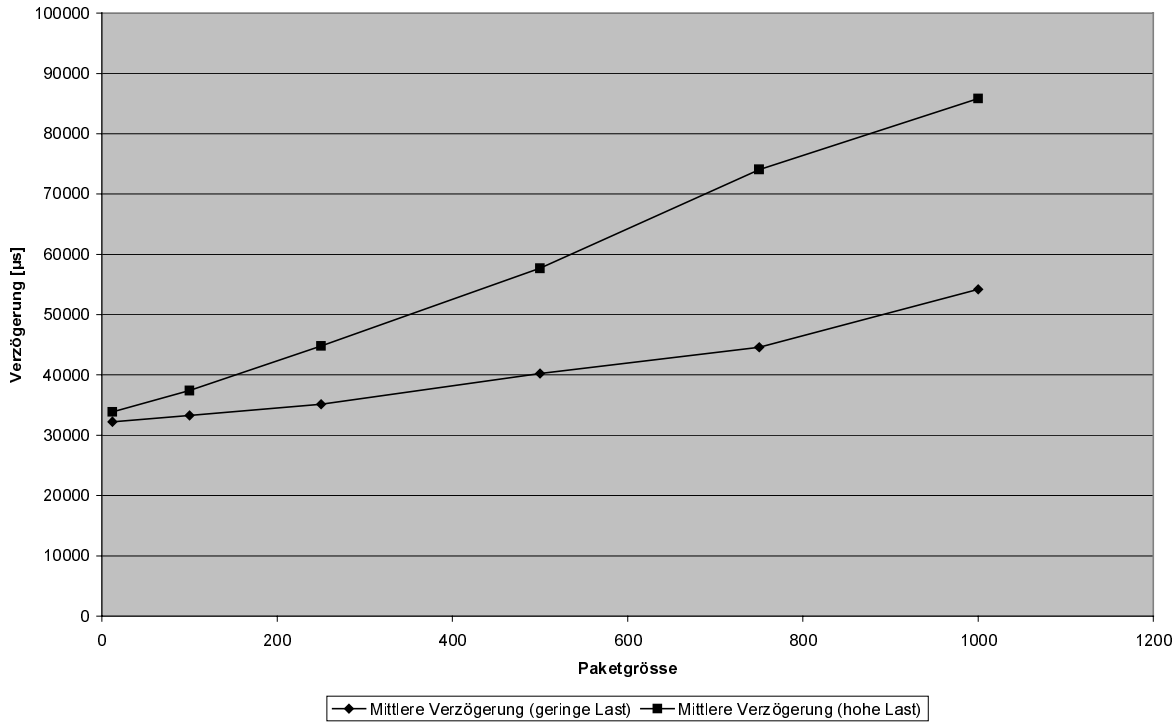


Abbildung 6.6 Nachrichtenverzögerung des Gruppenkommunikationsprotokolls

Paketgröße	12	100	250	500	750	1000
Mittel (geringe Last)	32249	33281	35134	40249	44576	54182
Mittel (hohe Last)	33875	37396	44806	57684	74072	85815

Tabelle 6.6 Nachrichtenverzögerungen des Gruppenkoummunikationsprotokolls in µs

7 Fazit

Im Rahmen der Diplomarbeit wurde ein zuverlässiges Echtzeit-Gruppenkommunikationsprotokoll für Gruppen von autonomen mobilen Systemen entwickelt, die über ein lokales Funknetz verbunden sind. Dieses Protokoll stellt den Anwendungen Dienste zur Verfügung, deren Sematik die Entwicklung von Kooperationskonzepten auf höheren Ebenen erheblich vereinfacht. Dabei stellt das Protokoll sicher, daß alle Dienste in ihrem zeitlichen Verhalten vorhersagbar sind, so daß sie für die Entwicklung von verteilten Echtzeitanwendungen verwendet werden können.

Konkret realisiert das Protokoll rechtzeitige, atomare Broadcasts und einen rechtzeitigen Teilnehmerservice. Das heißt, das Protokoll stellt die folgenden Eigenschaften sicher:

- Für jede Nachricht m und jede Station s gilt: s liefert m höchstens einmal aus und nur dann, wenn m gebroadcastet wurde.

- Wenn eine korrekte Station eine Nachricht m mit Resiliency OD broadcastet, dann liefert jede korrekte Station m aus.

- Wenn eine Station eine Nachricht m ausliefert, dann liefert jede korrekte Station m aus.

- Wenn zwei Stationen s_1 und s_2 die Nachrichten m_1 und m_2 ausliefern, dann liefert Station s_1 Nachricht m_1 vor Nachricht m_2 aus, genau dann wenn Station s_2 die Nachricht m_1 vor Nachricht m_2 ausliefert.

- Wenn eine Station eine Nachricht m mit Resiliency res zum Zeitpunkt t broadcastet, dann liefert keine Station m nach dem Zeitpunkt $t + 2 \times (res + 1) \times \Delta_{Round} + (OD + 1) \times \Delta_{Slot}$ aus. Dabei ist $\Delta_{Slot} = 3 \times t_m$, $\Delta_{Round} = \Delta_{Slot} \times n$ und n die maximale Gruppengröße.

- Wenn eine Station zum Zeitpunkt t ausfällt oder die Gruppe verläßt, dann liefert jede korrekte Station spätestens zum Zeitpunkt $t+ (OD + 1) \times \Delta_{Round} + (OD + 1) \times \Delta_{Slot}$. eine Änderungsnachricht aus, die den Ausfall von s anzeigt.

- Wenn eine Station s zum Zeitpunkt t eine Änderungsnachricht ausliefert, die den Ausfall von Station s' anzeigt, dann ist s' vor dem Zeitpunkt t ausgefallen.

- Wenn eine Station s eine Änderungsnachricht ausliefert, die den Ausfall von Station s' anzeigt, dann liefert jede korrekte Station diese Nachricht ebenfalls aus.

- Wenn die Stationen s_1 und s_2 Änderungsnachrichten ausliefern, die den Ausfall der Stationen s_3 und s_4 anzeigen, dann liefern beide Stationen die Änderungsnachrichten in der gleichen Reihenfolge aus.

- Wenn die Stationen s_1 und s_2 Änderungsnachrichten ausliefern, die den Ausfall der Stationen s_3 und s_4 anzeigen, dann liefern beide Stationen zwischen diesen Änderungsnachrichten die gleichen Broadcast-Nachrichten aus.

Als Grundlage für die Entwicklung des Protokolls dient ein lokales Funknetzwerk nach dem IEEE 802.11 Standard. Dieses hat zum einen den Vorteil, daß es den Einsatz von Standard-Hardwarekomponenten für die Realisierung des lokalen Funknetzes erlaubt. Zum anderen stellt der Standard mit den beiden Zugriffsverfahren DCF und PCF sowohl ein verteiltes, kollisionsauflösendes Zugriffsverfahren als auch ein zentralisiertes, kollisionsfreies Zugriffsverfahren zur Verfügung. Da in der PCF der Zugriff aus das Medium zeitlich vorhersagbar erfolgen kann, ist diese als Grundlage für ein Echtzeitprotokoll gut geeignet.

Das besondere Problem, das in der vorliegenden Diplomarbeit gelöst wurde, bestand darin, sowohl die Zuverlässigkeit als auch die zeitliche Vorhersagbarkeit der Übertragung von Broadcast-Nachrichten zu gewährleisten. Dabei waren auch die relativ geringe Bandbreite des Funknetzes und die spezielle Kommunikationsstruktur des IEEE 802.11 Standards zu beachten.

Unter der Annahme, daß die Anzahl aufeinanderfolgender Nachrichtenverluste stets kleiner oder gleich dem Omission Degree *OD* ist, wurde das Problem durch den Einsatz dynamischer Redundanz gelöst. Der große Vorteil dynamischer Redundanz gegenüber statischer besteht darin, daß der verursachte Aufwand abhängig von der tatsächlichen Anzahl von Fehlern ist und nicht von der angenommenen maximalen Anzahl von Fehlern. Dieser Vorteil konnte im Rahmen der Messungen verdeutlicht werden, bei denen sich zeigte, daß der tatsächliche Durchsatz durch die Resiliency nur geringfügig beeinflußt wird. Ein hoher tatsächlicher Durchsatz ermöglicht es dem AP, die CFP vor der eingeplanten Maximaldauer abzuschließen, so daß mehr Zeit für das Versenden von Nicht-Echtzeitnachrichten in der CP zur Verfügung steht.

Bei der Entwicklung des Protokolls wurde besonderer Wert darauf gelegt, den Bestätigungsmechanismus, der zum Erkennen von Nachrichtenverlusten benötigt wird, in effizienter Weise zu realisieren. Dabei bestand das Ziel vor allem darin, zusätzliche Nachrichten, die nur dem Zwecke der Bestätigung dienen, nach Möglichkeit zu vermeiden, da jede weitere Nachricht unabhängig von ihrer Größe einen bestimmten Mindestaufwand verursacht (s. Kapitel 6 Abb. 6.1) und da jede weitere Protokollnachricht, die nicht vom AP gesendet wird, zu einer weiteren Polling-Nachricht des AP führt. Durch die Strukturierung des Protokolls in Runden, kann jede Station die Request-Nachrichten, die sie an den AP sendet, benutzen, um alle Broadcast-Nachrichten der Vorrunde positiv oder negativ zu bestätigen. Dazu mußten die Request-Nachrichten lediglich um ein Bitfeld erweitert werden, dessen Länge der maximalen Gruppengröße entspricht. Zusätzliche Nachrichten müssen nur dann versendet werden, wenn eine Station keine Benutzernachricht an den AP übertragen möchte und daher die Request-Nachricht nur wegen der Bestätigungen an den AP sendet.

Auf die beschriebene Weise kann das Protokoll die Rechtzeitigkeit und Zuverlässigkeit der Nachrichtenübertragung sicherstellen. Solange die Zuverlässigkeit der Nachrichtenübertragung gewährleistet wird, ist allerdings die garantierbare maximale Verzögerung vom Omission Degree *OD* abhängig und damit sehr groß (z. B. für OD=15, t_m=2,8 ms und n=4: 1209,6 ms). Daher kann es sein, daß diese Zeitschranke für bestimmte Anwendungen zu lang ist. Das Protokoll schafft die Möglichkeit, daß diese Anwendungen die maximalen Verzögerungen verkürzen und damit Zeitfehler vermeiden, indem sie die Zuverlässigkeit der Nachrichtenübertragung einschränken. Hierzu wurde das Konzept der Resiliency von Nachrichten entwickelt, das es dem Benutzer ermöglicht, eine vom Omission Degree abweichende maximale Anzahl von Neuübertragungen zu spezifizieren und damit kürzere Zeitschranken zu erreichen (z. B. für OD=15, t_m=2,8 ms, n=4 und res=3: 403,2ms). Wenn die Resiliency einer Nachricht kleiner als *OD* gewählt wird, kann Zuverlässigkeit der Nachrichtenübertragung nicht mehr gewährleistet werden. Dennoch ist für viele Anwendungen diese eingeschränkte Funktionalität noch hinreichend, vor allem, da das Protokoll auch beim Auftreten von Nachrichtenverlusten die Einigung unter den Stationen sicherstellt, d. h. dafür sorgt, daß Nachrichtenausfälle konsistent wahrgenommen werden.

Die Einigung unter den Stationen wird mit Hilfe des synchronen Kanals erreicht, der es dem AP erlaubt, Entscheidungen in Form von Bittupeln an die Stationen zu übertragen. Die starken Eigenschaften dieses Kanals, vor allem die Eigenschaften der Zuverlässigkeit, Rechtzeitigkeit und FIFO-Ordnung, bilden eine gute Grundlage bei der Realisierung der Protokolleigenschaften. Der synchrone Kanal wurde realisiert, indem die Broadcast-Nachrichten um ein Bitfeld der Länge $2 \times (OD+1)$ erweitert wurden, so daß der AP die Entscheidungen via Piggy-Backing übertragen kann. Auf diese Weise werden zusätzliche Protokollnachrichten für den synchronen Kanal nur dann erforderlich, wenn der AP leere Broadcast-Nachrichten versenden muß, d. h. solche, die keine Benutzernachricht enthalten.

Die Messungen zeigen (Kapitel 6, Abb. 6.2 und Abb. 6.3), daß i. A. mit einer Resiliency, die deutlich unter dem Omission Degree liegt, schon keine oder nur sehr wenige Nachrichtenverluste auftreten. Folglich spiegelt eine Resiliency unter dem Omission Degree häufig die tatsächlichen Verhältnisse erheblich realistischer wider als der Omission Degree selbst. Das Protokoll erlaubt es dem Anwender, mit einer solchen, realistischeren Abschätzung zu arbeiten. In diesem Sinne zeigt das Protokoll eine Analogie zum Konzept des Taskpair-Scheduling ([NS97], [Str95]), das es ermöglicht, Aufgaben mit ihren erwarteten Ausführungszeiten einzuplanen, statt Annahmen über den ungünstigsten Fall zugrunde zu legen, der höchst selten oder sogar nie eintreten wird. Auf diese Weise kann es zwar vorkommen, daß die eingeplante Zeit nicht ausreicht, um die volle Funktionalität einer Aufgabe auszuführen, der Scheduler ist aber in der Lage, dieses zu erkennen und eine vom Benutzer definierte Ausnahmebehandlung rechtzeitig durchzuführen. Ebenso werden auch hier Zeitfehler verhindert, indem u. U. auf die volle Funktionalität verzichtet und statt dessen gelegentlich eine Minimalfunktionalität erreicht wird.

Durch die Integration von Teilnehmer- und Broadcastprotokoll und die Wahl des AP als zentralen Koordinator wurde erreicht, daß das Teilnehmerprotokoll neben den Nachrichten des Broadcastprotokolls keine zusätzlichen Nachrichten benötigt. Das Teilnehmerprotokoll nutzt die Request-Nachrichten, um Stationsausfälle rechtzeitig und genau zu erkennen. Weiterhin verwendet es den synchronen Kanal, um die festgestellten Änderungen an die Stationen zuverlässig und rechtzeitig zu übertragen. Auf diese Weise stellt das Teilnehmerprotokoll die total geordnete und virtuell synchrone Auslieferung der Änderungsnachrichten sicher und gewährleistet die Einigung unter den Stationen.

Das entwickelte Protokoll wurde unter Windows NT implementiert und es wurden erste Messungen vorgenommen. Diese zeigen vor allem, daß es möglich ist, mit einer begrenzten Anzahl von Sendeversuchen die Zuverlässigkeit der Übertragung sicherzustellen. Weiterhin zeigt sich, daß durch die Wahl der Resiliency einer Nachricht die Zuverlässigkeit der Nachrichtenübertragung beeinflußt wird, wobei durch eine höhere Resiliency eine höhere Zuverlässigkeit erreicht werden kann. Der erzielte Durchsatz wird durch die Wahl einer höheren Resiliency aber nicht beeinflußt, da beim Einsatz dynamischer Redundanz Neuübertragungen von Nachrichten nur im Falle von Nachrichtenverlusten durchgeführt werden.

In einem veränderten Anwendungskontext, in dem die Verträglichkeit mit dem Standard nicht unbedingt notwendig ist, kann das vorgestellte Protokoll auch zeitgesteuert eingesetzt werden. Im zeitgesteuerten Betrieb senden die Stationen die Request-Nachrichten zu eingeplanten Zeitpunkten auf einer globalen Zeitachse. Die hierfür erforderliche globale Uhr kann mit Hilfe eines Uhrensynchronisationsprotokolls realisiert werden, das sicherstellt, daß es eine bekannte maximale Differenz zwischen den lokalen Uhrzeiten der Stationen gibt. Da beim zeitgesteuerten Einsatz auf das Versenden von Polling-Nachrichten verzichtet werden kann, benötigt das Protokoll weniger Bandbreite und erreicht eine höhere Zuverlässigkeit.

Nachdem das Protokoll entwickelt und in einer ersten Form implementiert ist, besteht ein interessanter nächster Schritt in der Portierung des Protokolls auf die angestrebte Zielplattform: autonome mobile Systeme. Nach der Portierung können Tests des Protokolls und weitere Messungen in der Zielumgebung stattfinden. Vor allem kann hierbei eine Vermessung des Protokolls unter den angestrebten Bedingungen mehrerer, sich schnell bewegender Systeme stattfinden. Des weiteren kann durch eine Evaluierung des Protokolls anhand stochastischer Modelle die Zuverlässigkeit und Leistungsfähigkeit des Protokolls unter unterschiedlichen Annahmen über die Umgebung untersucht werden. Auf diese Weise können z. B. Zusammenhänge zwischen der betrachteten Umgebung, der Resiliency und der Zuverlässigkeit der Nachrichtenübertragung verdeutlicht werden.

Quellenverzeichnis

[ADKM92a] Y. Amir, D. Dolev, S. Kramer und D. Malki. Membership Algorithms for Multicast Communication Groups. In *Proceedings of the 6th International Workshop of Distributed Algorithms*., S. 292-312, Haifa, Israel, 1992.

[ADKM92b] Y. Amir, D. Dolev, S. Kramer und D. Malki. Transis. A communication subsystem for high availability. In *22nd International Symposium on Fault-Tolerant Compting*, S. 76-84, Boston, Massachusetts, USA, 1992.

[AMMAC95] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal und P. Ciarfell. The Totem Single-Ring Ordering and Membership Protokoll. *ACM Transactions on Computer Systems* 13(4), S. 311-342, 1995.

[Ark92] R. C. Arkin. Cooperation without Communication: Multi-agent Schema Based Robot Navigation. *Journal of Robotic Systems* 9(3), S. 351-364, 1992.

[AV95] C. Almeida und P. Veríssimo. An Adaptive Real-Time Group Communication Protocol. In *Proceedings of the 1st IEEE Workshop on Factory Communication Systems*, Leysin, Schweitz, Oktober 1995.

[BA94] T. Balch und R. C. Arkin. Communication in Reactive Multiagent Robotic Systems. *Autonomous Robots* 1, S. 27-52, 1994.

[BJ87] K. Birman und T. A. Joseph. Exploiting Virtual Synchrony in Distributed Systems. *ACM Operating Systems Review* 21(5), S. 123-138, 1987.

[BSS91] K. Birman, A. Schiper und P. Stephenson. Lightwight Causal and Atomic Group Multicast. *ACM Transactions on Computer Systems* 9(3), S. 272-314, August 1991.

[CASD85] F. Cristian, H. Aghili, R. Strong und D. Dolev. Atomic Broadcast: From Simple Message Diffusion to Byznatine Agreement. In *Proceedings of the 15th International Symposium on Fault-Tolerant Computing*, S. 200-206, Ann Arbor, Michigan, USA, 1985.

[CFK97] Y. U. Cao, A. S. Fukunaga und A. B. Kahng. Cooperative Mobile Robotics: Antecedents and Directions. *Autonomous Robots* 4, S. 7-27, 1997.

[CM84] J.-M. Chang und N. F. Maxemchuck. Reliable Broadcast Protocols. *ACM Transactions on Computing Systems* 2(3), S. 251-273, August 1984.

[CS95] F. Cristian und F. Schmuck. Agreeing on Processor Group Membership in Timed Asynchronous Distributed Systems. UCSD Technichal Report CSE95-428, 1995.

[Cri91] F. Cristian. Reaching Agreement on Processor-Group Membership in Synchronous Distributed Systems. *Distributed Computing* 4, S. 175-187, 1991.

[Cri96] F. Cristian. Synchronous and Asynchronous Group Communications. *Communications of the ACM* 39(4), S. 88-97, April 1996.

[CT96] T. Deepak C. und S. Toueg. Unreliable Failure Detectors for Reliable Distriubted Systems. *Journal of the ACM*. 43(2), S. 225-267, März 1996.

[DDS87] D. Dolev, C. Dwork und L. Stockmeyer. On the Minimal Synchronism Needed for Distributed Consensus. *Journal of the ACM* 34(1), S. 77-97, Januar 1987.

[DKM92] D. Dolev, S. Kramer und D. Malki. Total Ordering of Messages in Broadcast Domains. Technical Report CS92-9, Hebrew University of Jerusalem, Jerusalem, Israel, November 1992.

[EL90] P. D. Ezhilchelvan und R. de Lemos. A Robust Group Membership Algorithm for Distributed Real-Time Systems. In *Proceedings of the Real-Time Systems Symposium*, S. 173-179, Lake Buena Vista, Florida, USA, 1990.

[FLP85] M. J. Fischer, N. A. Lynch und M. S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM* 32(2), S. 374-382, 1985.

[Gar82] H. Garcia-Molina. Elections in a Distributed Computing System. *IEEE Transactions on Computers* 31(1), Jan. 1982.

[GP96] A. Galleni und D. Powell. Consensus and Membership in Synchronous and Asynchronous Distributed Systems. Raport LAAS (Laboratoire d'Analyse et d'Architecture des Systemes) No. 96104, Contrat Esprit De-Va Project No. 20072, S. 295-339, Dezember 1996.

[HT93] V. Hadzilacos und S. Toueg. Fault-Tolerant Broadcast and Related Problems. In Sape Mullender (Ed.): Distributed Systems, Addison-Wesley, 1993.

[HS95] M. A. Hiltunen und R. D. Schlichting. Unterstanding Memberschip. Technical Report 95-07, Department of Computer Science, University of Arizona, Tuscon, Arizona, USA, 1995.

[IEEE97] IEEE. 802.11 Standard: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. 1997.

[IITM98] Y. Inoue, M. Iizuka, H. Takanashi und M. Morikura. A Reliable Multicast Protocol for Wireless Systems with Representative Acknowledgement Scheme. In *Proceedings of the 5th International Workshop on Mobile Multimedia Communication*, Berlin, Deutschland, 1998.

[JKN96] W. Jia, J. Kaiser und E. Nett. RMP: Fault-Tolerant Group Communication. *IEEE Micro* 16(2), S. 59-67, April 1996.

[KG93] H. Kopetz und G. Grünsteidl. TTP – A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems. In *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing*, Toulouse, France, 1993.

[KN98] J. Kaiser und E. Nett. Echtzeitverhalten in dynamischen, verteilten Systemen. *Informatik Spektrum* 21(6), S. 356-365, Dezember 1998.

[Kop97] H. Kopetz. Real-Time Systems – Design Principals for Embedded Applications, Kluwer Academic Publisher, 1997.

[KSY84] J. F. Kurose, M. Schwartz und Y. Yemini. Multiple-Access Protocols and Time-Constrained Communication. *Computing Surveys* 16(1), S. 43-47, 1984.

[KT91] M. F. Kaashoek und A. S. Tanenbaum. Group Communication in the Amoeba Distributed Operating Systems. In *Proceedings of the 11th International Conference on Distributed Computing Systems*, S. 222-230, Arlington, Texas, 1991.

[KZ97] C. R. Kube und H. Zhang. Task Modelling in Collective Robotics. *Autonomous Robots* 4, S. 53-72, 1997.

[Lam78] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM* 21(7), S. 558-565, 1978.

[Lap92] J. C. Laprie (ed.). Dependability: Basic Concepts and Terminology, Springer, 1992.

[LW92] J. Lehn und H. Wegmann. Einführung in die Statistik, Teubner, Stuttgart, 1992.

[Mar97] P. Martini. Rechnernetze I. Vorlesung an der Universität Bonn. Wintersemester 1997/98.

[MMA90] P. M. Melliar-Smith, L. E. Moser und V. Agrawala. Broadcast Protocols for Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems* 1(1), S. 17-25, Januar 1990.

[MMA93] L. E. Moser, P. M. Melliar-Smith und V. Agrawala. Asynchronous Fault-Tolerant Total Ordering Algorithms. *Siam Journal on Computing* 22(4), S. 727-750, August 1993.

[MMA94] L. E. Moser, P. M. Melliar-Smith und V. Agrawla. Processor Membership in Asynchronous Distributed Systems. *IEEE Transactions on Parallel and Distributed Sytems* 5(5), S. 459-473, 1994.

[MMABL96] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia und C. A. Lingley-Papadopoulos. Totem: A Fault-Tolerant Multicast Group Communication System. *Communications of the ACM* 39(4), S. 54-63, April 1996.

[MN99] M. Mock und E. Nett. Real-Time Communication in Autonomous Robot Systems. In *Proceedings of the 4th International Symposium on Autonomous Decentralized Systems*, Tokyo, Japan, 1999.

[MNS99] M. Mock, E. Nett und S. Schemmer. Efficient Reliable Real-Time Group Communication for Wireless Local Area Networks. In *Proceedings of the 3rd European Dependable Computing Conference* (EDCC-3), Prag, Tschechische Republik, September 1999.

[Net91] E. Nett. Supporting Fault Tolerant Computations in Distributed Systems. Habilitationsschrift, Universität Bonn, 1991.

[NS97] E. Nett und H. Streich. The GMD-Snake – Real-Time Scheduling of a Flexible Robot Application at Run-Time. In *Proceedings of the International Workshop on Parallel Computations and Scheduling*, Ensanada, Mexiko, 1997.

[PBS89] L. L. Peterson, N. C. Buchholz und R. D. Schlichting. Preserving and Using Context Information in Interprocess Communication. *ACM Transactions on Computer Systems* 7(3), S. 217-246, August 1989.

[Pow92] D. Powell. Failure Mode Assumptions and Assumption Coverage. *Proceedings of the 22nd International Symposium on Fault-Tolerant Computing*, Boston, Massachusetts, USA, 1992.

[RV92] L. Rodrigues und P. Veríssimo. xAMP: a Multi-primitiv Group Communication Service. In *Proceedings of the 11th Symposium On Reliable Distributed Systems*, S. 112-121, Houston, Texas, 1992.

[RFV95] L. E. T. Rodrigues, H. Fonseca und P. Veríssimo. Reliable Computing over Mobile Networks. In *Proceedings of the 5th Workshop on Future Trends in Distributed Computer Systems*, Cheju Island, Korea, August 1995.

[Str95] H. Streich. Task Pair-Scheduling: An Approach for Dynamic Real-Time Systems. *International Journal of Mini & Microcomputers* 17(2), S. 77-83, 1995.

[VM90] P. Veríssimo und J. A. Marques. Reliable Broadcast for Fault-Tolerance on Local Computer Networks. In *Proceedings of the 9th Symposium on Reliable Distributed Systems*, Huntsville, Alabama, USA, 1990. (auch als Technical Report AR/24-90 des INESC – Instituto de Engenharia de Sistemas e Computadores, Lissabon, Portugal)

[VRR91] P. Veríssimo, J. Rufino und L. Rodrigues. Enforcing Real-Time behavior on LAN-Based Protocols. In *Proceedings of the 10th IFAC Workshop on Distributed Computer Control Systems*, Semmering, Österreich, 1991