**DIPLOMARBEIT**

# Effiziente Software-Verteilung in Schulinfrastrukturen für computerunterstützte Ausbildung auf Basis eines Reliable Multicast Protocols

zur Erlangung des akademischen Grades
Diplomingenieur
(Dipl.-Ing.)

ausgeführt am

Institut für Rechnergestützte Automation,
Forschungsgruppe Industrial Software,
der Technischen Universität Wien

unter Anleitung von

Univ.-Prof. Dipl.-Ing. Dr. Thomas Grechenig

durch

Christian Schöbel

Felix-Dahn-Straße 41/5
A-1190 Wien

Wien, im März 2007
Version 2007-03-13-27

## Vorwort

„Wenn alles schläft und einer spricht, so nennt man dieses Unterricht". Diese nicht gerade optimistische Betrachtung der Methode des Frontalunterrichts hat doch einen gewissen Wert, wenn man sich rein auf den funktionalen Kontext beschränkt. Letztlich stammt der Frontalunterricht ja aus einer Zeit, in welcher das Verbreiten von Datenwissen keine Unterstützung durch moderne Medien kannte. Es war nuneinmal das einfachste, Informationen in jener Form weiterzugeben, daß einer, nämlich der Lehrer oder die Lehrerin, diese Information verkündete und die SchülerInnen alles in ihren Heften niederschrieben. Heutzutage ist die Fotokopie billig geworden, weshalb man sich im Falle der Weitergabe von Datenwissen dieser Methode bedienen kann. Dadurch haben sich letztlich auch Ziel und Absicht des Unterrichts geändert, weil reines Faktenwissen leicht verfügbar geworden ist, wie zum Beispiel durch Enzyklopädien im Internet. Neue Herausforderungen erwachsen im Unterricht. Deshalb kann heutzutage mehr Zeit für das Verständnis von Zusammenhängen und für den Erwerben von Fähigkeiten aufgebracht werden.

In der Computerwelt gibt es sozusagen nur die Wissensvermittlung, aber keinen Erwerb von Fähigkeiten. Computer können gegenseitig Daten austauschen; aber es wird nicht möglich sein, daß ein Computer einem anderen zum Beispiel ein Protokoll beibringt.

Wegen dieser Tatsache erschien mir die Idee des Frontalunterrichts, die durch das anfangs genannte Zitat ja negativ ausgelegt wird, für die Computerwelt jedoch relativ brauchbar. Einer hat die Daten, das ist die Lehrerin oder der Lehrer. Die Information wird an die SchülerInnen weitergegeben und soll doch möglichst detailgetreu erfaßt werden. Nun kennen LehrerInnen ihre SchülerInnen. Man weiß, daß die Margareta ein fleißiges Mädchen ist und man sie deshalb relativ selten fragen muß, ob sie auch alles mitschreiben konnte. Isidor aber ist ein „Traumentlein", schaut gerne durch die Luft und kann deshalb den Ausführungen der LehrerIn nicht immer folgen. Deswegen wird Isidor öfter gefragt, ob er auch alles mitschreiben konnte.

So werden die SchülerInnen durch die Lehrerin oder den Lehrer befragt, auf welchem Stand sie sich gerade befinden. Dementsprechend wird der Lehrer oder die Lehrerin das

wiederholen, was noch nicht alle mitschreiben konnten. SchülerInnen, die das wiederholte bereits vorher mitgeschrieben haben, müssen sich ruhig verhalten; andernfalls stört der Lärm, den sie produzieren.

Genau dieses Bild habe ich auf die Erstellung meines RMARS Protokolls übertragen. Der Server ist Lehrer, die Clients, welche eine fehlerfreie Kopie der Daten erhalten sollen, sind die SchülerInnen.

Um diese Grundidee sinnvoll realisieren zu können, habe ich vor der Entwicklung von RMARS die Möglichkeiten und Probleme von Gruppenkommunikation im Computerbereich erforscht. Viele Überlegungen und auswärtige Informationsquellen sollten die Forschung dieses speziellen Gebiets bereichern. Ich hoffe, eine übersichtliche und detailreiche Betrachtung in dieser Arbeit gegeben zu haben.


## Danksagungen

Der Punkt der Danksagungen ist eine kritische Sache. Wie viele Menschen sind es allein, die mich bei Idee, Erstellung und Fertigstellung dieser Arbeit unterstützt haben! Um niemanden Unrecht zu tun, möchte ich all jene Menschen – wie es heißt – ,,in pectore" bewahren und jenen durch diese Zeilen hier meinen Dank ausdrücken. Eine namentliche Nennung möchte ich auf jene beschränken, die direkt mit dieser Arbeit in Berührung gekommen sind.

So möchte ich mit Herrn ao. Univ.-Prof. Dipl.-Ing. Dr. techn. Thomas Grechenig beginnen, dem ich – neben vielen anderen Dingen – vor allem für die Tatsache danken möchte, daß er mir diese Arbeit ermöglicht hat; Herrn Proj. Ass. Dipl.-Ing. Dr. techn. Philipp Tomsich, der mir mit seinem fachlichen Rat auch während langer Durststrecken zur Seite stand und mich durch Gespräch und Diskussion unterstützt hat; den Scientific Coordinatrices Mag. Jennifer Hetzl und Mag. Dr. Brigitte Brem möchte ich meinen Dank für wissenschaftliche Unterstützung und organisatorische Details aussprechen, und meinem lieben Kollegen am Rg19 in der Krottenbachstraße, Mag. Martin Dobrowolny, der mir diese Arbeit in Erstkorrektur gelesen hat.

Doch ein Dank darf nicht vergessen werden; dieser gilt dem Leser dieser Arbeit. Wenn Sie, hochverehrter Leser, mir eine Nachricht zukommen lassen möchten, so können sie dies unter der E-Mail-Adresse

`scoe@brg19.at`

tun. Ich freue mich über jegliches Interesse an der Materie und über die eine oder andere Rückmeldung.

Christian Schöbel

Wien, im August 2006

## Eidesstattliche Erklärung

Ich erkläre an Eides statt, daß ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfaßt, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 13. März 2007 _____

## Zusammenfassung

Informations- und Kommunikationstechnologien werden in der Ausbildung zunehmend häufiger als Lehrmittel eingesetzt. Waren es bis Mitte der 80er Jahre noch hauptsächlich Fernseher und Videorecorder, die technologische Innovationen in der Ausbildung repräsentierten, begann spätestens mit der Einführung des verpflichtenden Informatikunterrichts der Aufbau von Informations- und Kommunikationsinfrastrukturen an österreichischen Schulen. Anfänglich meist nebenberuflich von Lehrpersonen betreut, wurde der zunehmende Technologieeinsatz zur administrativen Herausforderung und verlangte nach effizienten Methoden zur Installation und Administration dieser Infrastrukturen. In Hinblick auf die zu investierende Zeit und der damit anfallenden Kosten für wiederholt durchzuführende Installationen stellt neben verschiedenen organisatorischen Lösungsansätzen der Einsatz von Mechanismen zur automatisierten Softwareverteilung einen vielversprechenden technischen Ansatz zur Effizienzsteigerung dar. Eine notwendige Basistechnologie für eine effiziente Softwareverteilung ist ein leistungsfähiger Mechanismus zur ressourcenschonenden Verteilung von Datenpaketen in Netzwerken. Multicast- bzw. Gruppenruf-Protokolle stellen hierfür geeignete Basistechnologien dar, allerdings mit der Einschränkung der fehlenden Zuverlässigkeit der Datenübertragung. Anhand eines um Merkmale zur zuverlässigen Datenübertragung ergänzten Protokolls (Reliable Multicast with Acknowledgments Requested by the Server - RMARS) wird im Rahmen der vorliegenden Arbeit die Eignung dieser Technologie zur effizienteren Gestaltung der Softwareverteilung in Informations- und Kommunikationsinfrastrukturen von Ausbildungsstätten gezeigt, wobei ein unter dem Open-Source Betriebssystem Linux implementierter Prototyp des Protokolls entwickelt wurde.

## Abstract

The use of information and communication technologies as teaching aids for educational purposes has increased enormously during the last decades. Since applied computer science has become compulsory in the school system of Austria, there were new requirements for performing maintenance and repairing computer systems used in teaching environments. The most important task is to provide an efficient and reliable solution for software installation as software installation is most frequently needed due to the fact that operating systems cannot thoroughly prevent damage to local disk data. As networks usually exist in modern computer infrastructures, the process of software installation can be automated for a whole classroom by using broadcast or multicast data transfer. In this thesis a reliable multicast protocol is introduced, designed for the special case of software distribution in educational computer networks, and evaluated in the face of usability for this special purpose.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# 1 Einleitung

Als Bundesminister Dr. Helmut Zilk im Schuljahr 1985/86 den Informatikunterricht als Pflichtfach für die 5. Klassen der allgemeinbildenden höheren Schulen im Lehrplan festsetzte, wurden plötzlich neue Anforderungen an die Art des Unterrichts und an die entsprechend notwendige Ausrüstung gestellt. Im Hinblick auf die Gestaltung des Unterrichts entstand unterschiedlicher Bedarf für die Betriebsmittel, welcher nun von den Schulen selbst erhoben und umgesetzt werden mußte. Finanzielle Förderungen wurden immer häufiger, ebenso die Möglichkeit der Schulen, an sogenannten Halbprojekten teilzunehmen, welche wesentlich die Richtung für die Gestaltung der Hardwareausstattung vorgaben. In vielen Schulen wurde nur eine Lehrerin oder nur ein Lehrer mit der Ausrüstung und Planung der Computerlandschaft betraut, weshalb sich jene oftmals alleingelassen oder überfordert fühlen.

Im Schulbetrieb bestehen andere Anforderungen an Computersysteme als in einem Firmenbetrieb, da letztlich Kenntnisse und Fähigkeiten möglichst effizient und angenehm vermittelt werden sollen. Die LehrerIn soll die Möglichkeit haben, den SchülerInnen gewisse Arbeitsschritte und Vorgänge am Computer zu demonstrieren. Auf der anderen Seite sollen SchülerInnen die von der LehrerIn gestellten Aufgaben erfüllen und sonst möglichst nichts anderes tun, damit sie nicht die Aufmerksamkeit verlieren oder gar den Unterricht stören. Für die meisten Aufgaben im Unterricht muß zudem jeder SchülerIn die gleiche Arbeitsumgebung zur Verfügung gestellt werden, um Komplikationen bei der Vorführung durch die LehrerIn zu vermeiden.

Weiters ergeben sich rechtliche Anforderungen an den Betrieb von Informationstechnologie in der Schule, gerade, wenn Internetanbindung zur Verfügung steht, was heutzutage in keiner Schule mehr fehlen darf.

Demnach soll sich diese Arbeit in Hinsicht der Betrachtung der Probleme im Schulbetrieb in drei Teile gliedern. Allem voran werden die Unterschiede beim Einsatz von Informationstechnologie in der Schule im Vergleich zu den üblichen Anwendungen vorgestellt, vor allem das Problem der Infrastruktur und der sich daraus ergebenden Notwendigkeit, eine Lösung für das Problem der Software-Verteilung bereitzustellen. Eine vielversprechende Methode ist der sogenannte Multicast, weil dadurch Netzwerkressourcen effizient genutzt werden können. Im zweiten großen Abschnitt befaßt sich die Arbeit mit der bereits vorhandenen Theorie und Praxis über Multicast und weshalb diese für den hier behandelten Spezialfall nicht verwendet werden. Der dritte Teil behandelt schließlich die Entwicklung eines neuen Lösungsansatzes für die Verteilung von Software in Schulinfrastrukturen sowie dessen Betrieb und Bewertung.

# 2 Einsatz von Informationstechnologie in der Schule

Wie in Kapitel 1 aufgezeigt wurde, ergeben sich für den Einsatz von Informationstechnologie in der Schule komplett neue Anforderungen, wenn diese mit den ,,üblichen" Anforderungen von Firmen oder technischen Betrieben verglichen werden.

So möge hier zuerst das Standardszenario eines Angestellten einer Firma dargestellt werden. Er hat seinen eigenen Computer, mit welchem er die Aufgaben seines Berufes bewerkstelligen soll. Es ist ihm ein Anliegen, daß sein Computer fehlerfrei funktioniert, damit er seine Arbeit gemäß Forderung des Arbeitgebers erfüllen kann. Im Störungsfall seines Rechners muß dieser gewartet werden, was entweder durch einen externen Dienstleistungsbetrieb oder durch die firmeneigene EDV-Abteilung erledigt wird. Dies produziert jedenfalls Kosten für den Arbeitgeber, zudem entsteht ein weiterer wirtschaftlicher Schaden für die Firma, weil der Mitarbeiter während der Zeit der Wartung seine Aufgaben nicht erledigen kann. Deswegen ist der Mitarbeiter darauf bedacht, den Computer nur für den vorgesehenen Zweck einzusetzen; er wird nicht beginnen, absichtlich irgendwelche Systemdateien zu löschen, um dadurch den Betrieb einzuschränken. Bei einer gewissen Häufigkeit solcher Ereignisse würde die Firma jenem Mitarbeiter wohl die Kündigung aushändigen.

In einem Schulbetrieb herrschen andere ,,Gesetze". Einige SchülerInnen, vor allem während der Pubertät, wollen nicht in die Schule gehen, möchten nicht lernen, weil ihnen das im Moment zu anstrengend scheint und sie vielleicht sowieso ganz andere Dinge im Kopf haben, welche eben mit dem Erwachsenwerden erstmals hervorbrechen. Und um sich mehr Zeit für diese Gedankengänge zu schaffen, greifen jene SchülerInnen gern

einmal zu Methoden, um den Unterricht zu behindern. Abhängig von der Bereitschaft zu harmlosen ,,kriminellen" Handlungen der SchülerInnen sowie deren Gepflogenheiten werden vielleicht Stinkbomben in Klassenräumen gezündet, um den Unterricht kurzzeitig zu unterbrechen, oder ähnliche Aktionen durchgeführt. Bei einer so komplexen Materie wie einem Computersystem sollten sich Unmengen an Möglichkeiten ergeben, dieses zum Erliegen zu bringen. Das Löschen von Systemdateien gehört vielleicht schon fix zum Tagesablauf eines arbeitsunwilligen Schülers.

Weiters müssen im Lehrbetrieb auf allen Arbeitsstationen gleiche Bedingungen für alle SchülerInnen angeboten werden, damit der Lehrer oder die Lehrerin mittels Beamerprojektion den SchülerInnen gewisse Abläufe vorführen kann.

Außerdem benötigt der Schulbetrieb meist spezielle Lernsoftware, weil der Lehrbetrieb nicht durch die Funktionalität gewöhnlicher Office-Anwendungen abgedeckt werden kann.

Zusammengefaßt kann man also die Anforderungen an den Betrieb von Informationstechnologie in der Schule und damit der Aufgaben des EDV-Betreuers durch folgende Stichworte darstellen:

1. Gleiche Arbeitsbedingungen für alle Computer schaffen; betrifft vor allem die Softwareinstallation

2. Benutzer für den Umgang mit dem Computersystem der Schule ausbilden

3. Mit der Zerstörungswut von Benutzern rechnen und dieses Problem bewältigen können

4. Unterrichtsbedingte Spezialanwendungen ermöglichen und Spezialsoftware installieren

5. Rechtliche Anforderungen beachten, vor allem was das Anbieten von Internetdiensten betrifft

Diese Anforderungen haben sich aber im Lauf der Zeit, in welcher Informationstechnologie in der Schule eingesetzt wird, erst entwickelt. So möge zunächst eine typische Entwicklung einer solchen Infrastruktur dargestellt werden.

## 2.1 Genese einer typischen Schulinfrastruktur in der Praxis

Zum Zwecke der Darstellung der Genese einer typischen Schulinfastruktur werden hier die persönliche Erlebnisse des Autors auf diesem Gebiet geschildert, da jenem aufgrund seiner jahrelangen Kenntnis des Schulbetriebs einer AHS, sowohl als Schüler, als auch als Lehrer, eine den Realanforderungen entsprechende Betrachtung möglich ist. Es handelt sich dabei um das Bundesrealgymnasium Wien XIX in der Krottenbachstraße.

Im Jahr 1995 wurde der lehrplangemäße Unterricht in der 5. Klasse in einem kleinen Informatikraum mit sechs Arbeitsstationen durchgeführt. Damals war ein Novell Netzwerk mit Diskless Workstations mit DOS im Einsatz, welches durch den Kustos entsprechend gewartet wurde und für den Unterrichtszweck bestens geeignet war. Der Hauptgrund für etwaige Ausfälle war lediglich das zugrundeliegende Ethernet, bei welchem durch Entfernen eines Abschlußwiderstandes der Totalausfall bewirkt werden konnte. Und so ein Abschlußwiderstand war schnell einmal verschwunden. Dieses Netzwerk wurde im Zuge der Funktionssanierung der Schule im Jahr 1998 ersetzt. Die neue Computerlandschaft bestand aus einem Windows NT Server und 14 Windows NT 4.0 Workstations. Nun war auch ein neuer Kustos tätig, der in mühseliger Arbeit auf allen 14 Workstations die von den LehrerInnen angeforderte Software separat installieren mußte. Der damalige Kustos erzählte, daß er 3 Stunden Arbeit pro Workstation benötigt. Abgesehen davon war die Tätigkeit der Installation nicht gerade fordernd oder erfreulich, weil die einzelnen Installationsvorgänge viel Zeit benötigten, aber dazwischen immer wieder Benutzeraktionen verlangt wurden.

Für die Betriebsart des Windows NT Netzwerks wurde die für so einen Anwendungsfall vorgesehene Methode der Domainanmeldung gewählt. Am Server wurden Benutzeraccounts für alle SchülerInnen angelegt, welche sich dann an beliebigen Workstations anmelden konnten. Der Server war für SchülerInnen physikalisch unzugänglich, weil die-

ser in einem eigenen Raum aufgestellt wurde. Dieser Raum wurde im Zuge der Funktionssanierung der Schule eigens dafür geplant. Durch die Domainanmeldung hätten SchülerInnen eigentlich nur in ihrem eigenen Profil Änderungen vornehmen können sollen. Anwendungssoftware zu installieren oder gar zu entfernen war SchülerInnen durch entsprechende Vergabe von Netzwerkberechtigungen nicht ermöglicht.

Dennoch ergaben sich im Lauf der Zeit Probleme. Es schien, als hätten sich die Windows NT Workstations sogar im normalen Betrieb nach einigen Wochen selbst hingerichtet. Der häufigste Fehler war, daß SchülerInnen sich nicht mehr an der Domain anmelden konnten, weil die Workstation aus irgendeinem Grund die Domain nicht mehr gefunden hat. Aber auch andere Dinge wie plötzlich auftretende Fehlfunktionen bei Programmen, die sonst anstandslos funktionierten, oder auch der Totalabsturz von Windows beim Systemstart, gehörten nach einiger Zeit zur Tagesordnung.

Kurzum, jede Windows NT Workstation hatte sozusagen ein gewisses ,,Ablaufdatum", nach welchem diese neu installiert werden mußte. Da ja wie bereits erwähnt die Tätigkeit der Neuinstallation einer Workstation nicht gerade eine beliebte Aufgabe war, weil diese sehr zeitaufwändig war, fielen regelmäßig zwei oder drei Workstations längerfristig aus. Bei 14 Workstations und Informatikgruppen zu 14 SchülerInnen wurde das zu einem organisatorischen Problem.

Es schien, daß die Windows NT Workstations damit ein Problem hatten, daß sich an einem Computer mehrere Benutzer angemeldet haben; nicht gleichzeitig, aber hintereinander. Für jeden Benutzer wird bei der Anmeldung auch lokal das Profil gespeichert. Obwohl die lokalen Festplatten nicht überfüllt waren, begann eine Workstation mit Fehlfunktionen, wenn cirka 40 verschiedene Profile auf ihr abgelegt waren. In einer Firma kommt es kaum zu so einer Situation, weil meist jeder Mitarbeiter seinen eigenen Arbeitsplatz hat und sich nur dieser selbst bei seinem Arbeitsplatz anmeldet. Da aber in der Schule in jeder Stunde eine andere Gruppe im EDV Saal ist, zudem auch keine fixe Sitzordnung vorhanden ist, passiert ein Wechsel der Arbeitsstationen sehr häufig.

Es sind demnach einige Probleme im Schulbereich vorhanden, welche in der ,,gewöhnlichen Arbeitswelt" nicht bekannt sind. Anhand dieser Schwierigkeiten lassen sich die

Anforderungen an eine typische Schulinfrastruktur relativ einfach extrahieren, wie dies im nachfolgenden Kapitel geschehen soll.

## 2.2 Anforderungen an eine typische Schulinfrastruktur

Als wichtigste Anforderungen erscheinen die Gleichheit der Arbeitsplätze und die schnelle Verfügbarkeit eines Computers nach dem Einschalten.

Da bei der bisherigen Lösung mit Windows NT Workstation sowohl der Startvorgang über 2 Minuten dauerte, als auch die Gleichheit der Installationen nicht gewährleistet werden konnte, weil ja jede Workstation extra installiert wurde, wurde für die Installation von Windows 95 entschieden.

Windows 95 war hinsichtlich der Arbeitsgeschwindigkeit optimal für die vorhandene Hardware geeignet. Außerdem war es relativ leicht, eine Windows 95 Installation auf alle anderen Computer im Netzwerk zu klonen. Mit ein paar Modifikationen in der Windows Registry funktionierte das Festplattenabbild eines sauber installierten Computers auf allen anderen Workstations des Saales fehlerlos.

Für die Übertragung des Festplattenimages wurde auf der Serverseite ein NFS Server eingerichtet, welcher das Image anbot. Die Clients wurden mit einem Boot-ROM bestückt, wodurch man beim Start des Computers die Imagewiederherstellung auslösen konnte.

Da NFS keine Möglichkeit zur gleichzeitigen Übertragung auf mehrere Zielrechner kennt, mußte die Imagewiederherstellung für jeden Computer einzeln geschehen.

Die Gesamtwiederherstellung des Saals benötigte dadurch cirka 2 Stunden und 30 Minuten. So konnte aus organisatorischen Gründen maximal einmal pro Woche diese Prozedur durchgeführt werden. Dennoch ergab sich eine gewaltige Verbesserung der Situation, weil der Vorgang der Wiederherstellung nur eingeleitet, aber keine weitere Benutzeraktion getätigt werden mußte.

Nun galt es noch, die Gesamtwiederherstellung möglichst oft durchzuführen, da ja schon nach einem Tag die Installation der Computer zerstört sein könnte.

Eine organisatorische Lösung für dieses Problem wäre, jeden Abend einen EDV Lehrer einzuteilen, der vor dem Verlassen der Schule die Gesamtwiederherstellung einleitet.

Eine weitere Lösungsmöglichkeit wäre die verstärkte Kontrolle der Tätigkeit der SchülerInnen durch die LehrerIn während des Informatikunterrichts, sodaß wenigstens die mutwillige Zerstörung der Softwareinstallation vermieden werden kann. In Anbetracht der schwierigen Überschaubarkeit der Arbeitsplätze ist dies jedoch nicht leicht zu realisieren.

Wie einfach oder schwierig es ist, ein Festplattenabbild von einer ,,sauberen" Softwareinstallation erstellen zu können, ist abhängig von der verwendeten Software, im speziellen des verwendeten Betriebssystems. Unabhängig davon ist aber der Vorgang der Gesamtwiederherstellung eines Saales, das heißt die Verteilung eines Festplattenabbildes auf die einzelnen Computer, immer das gleiche Problem.

Im Zuge dieser Arbeit wurde für das Problem, die Gesamtwiederherstellung möglichst oft durchführen zu können, eine technische Lösung gewählt, die in den folgenden Kapiteln beschrieben wird.

Es soll die Image Wiederherstellung des gesamten Saales in einem Vorgang realisiert werden, daher ist das Ziel, eine Übertragung des Festplattenimages vom Server für alle Workstations gleichzeitig zugänglich zu machen. Der Ansatz hierfür besteht in der Verwendung von Multicast, einer Art von Gruppenrufprotokoll. Da es sich hier um eine grundsätzlich andere Technologie handelt, muß zunächst geklärt werden, welches Szenario für den Fall der Softwarewiederherstellung im Schulbereich zutrifft, welche Anforderungen und Rahmenbedingungen sich daraus ergeben und wie diese erfüllt werden können. Zur Durchführung dieser Analyse ist es zuerst notwendig, Begrifflichkeiten und Definitionen zu geben, welche für diese Technologie relevant sind.

# 3 Technische Grundlagen und Definitionen

Um die nachfolgenden Ausführungen über eine technische Lösung für das Problem auf ein brauchbares Fundament zu stützen, folgen nun einige notwendige Begriffserklärungen und Definitionen, die im weiteren Verlauf der Arbeit benötigt werden.

## 3.1 Software-Verteilung

Der wesentliche Vorgang, welcher bei der Gesamtwiederherstellung durchgeführt wird, kann als Software-Verteilung bezeichnet werden, da bei diesem Vorgang ja auf allen Computern in einem Unterrichtsraum eine Softwareinstallation durchgeführt werden soll.

Wo mehrere Computer mittels Netzwerk verbunden sind, kann so eine Software-Installation effizient gestaltet werden. Die meisten Netzwerkformen bieten einen Gruppenrufdienst an, welcher die Möglichkeit enthält, ein Datenpaket an mehrere Empfänger gleichzeitig zu adressieren. Ein wesentlicher Begriff für diese Funktionalität heißt Multicast.

Multicast bezeichnet demnach jene Betriebsart der Übertragung in paketvermittelnden Netzwerken, bei der ein ausgesandtes Paket gleichzeitig an mehrere Empfänger übertragen wird. Das Paket wird dupliziert, ist dadurch mehreren Empfänger verfügbar und auch für diese bestimmt.

Ein großer Vorteil von Multicast ist, daß durch dessen Verwendung die Bandbreite des Kommunikationskanals besser genutzt werden kann. Erfordert der Anwendungsfall nämlich die Übertragung derselben Daten an mehrere Teilnehmer, so müßte in direkter Übermittlung, in sogenanntem Unicast, die Übertragung sooft stattfinden, wie

es Empfänger gibt. Dadurch ist der Kommunikationskanal in Abhängigkeit von der Empfängerzahl mehrfach belastet.

Durch diese Tatsache läßt sich auch schon das größte Problem von Multicast erkennen. Durch steigende Teilnehmerzahlen ist Zuverlässigkeit der Übertragung auch schwieriger zu gewährleisten. Gerade dann, wenn der Anwendungsfall Zuverlässigkeit bedingt, muß ein Protokoll extra für diese Eigenschaft Sorge tragen. Je nach Anwendungsfall wird entweder Zuverlässigkeit benötigt oder auch nicht.

Als Beispiel für ein paar Anwendungsfälle, welche die Zuverlässigkeit der Übertragung benötigen, diene folgende Aufzählung: die Übertragung von Daten in Börsenhandelssystemen, das Übermitteln von Videodaten, deren Protokoll keine Datenfehler toleriert, oder auch das Kopieren eines neuen Softwarepaketes an Mirror-Server, die im Internet verteilt sind.

Bei all diesen Beispielen ist die Mehrfachübertragung der Daten unnötige Verschwendung von Bandbreite, bedingt durch die ineffiziente Verwendung des Kommunikationskanals, da jede Übertragung zwar die gleichen Daten beinhaltet, jedoch immer nur ein Empfänger davon profitiert.

Demnach wird jener Anwendungsfall, in welchem die zuverlässige Übertragung an mehrere Empfänger gefordert wird, als Reliable Multicast bezeichnet. Reliable Multicast ist ein gut erforschtes Gebiet und wird durch [16] überblicksmäßig sehr gut erfaßt. Aktuelle Entwicklungen lassen Reliable Multicast eine Renaissance erleben, wie zum Beispiel in der Tätigkeit der IETF Forschungsgruppe RMT im Aufgabenbereich des IPTV, in Mobile Networks [29], sowie bei Videoübertragungen in Wireless Networks [15].

Das Ziel dieser Arbeit ist die Entwicklung eines Protokolls namens Reliable Multicast with Acknowledgments Requested by the Server (RMARS) zum Einsatz in einer typischen Schulinfrastruktur zum Zwecke der Softwareverteilung. Als Vorbereitung auf diese Entwicklung wird das Thema Multicast und Reliable Multicast von vielen Seiten her beleuchtet und betrachtet. Der Zweck von RMARS ist die Entwicklung eines Transport-Protokolls, welches auf Ethernet bzw. IP aufsetzt und auch bei möglichst großen Teilnehmerzahlen zuverlässig und effizient die Übertragung von Daten meistert.

Im Gegensatz zum Reliable Multicast Protokoll, welches in [10] dargestellt wird, soll RMARS aber Daten übertragen, welche keine Echtzeitendaten sind. Deshalb muß keine Garantie über den Zeitpunkt des Eintreffens der Daten gewährleistet sein, wie es bei [10] der Fall ist. Ein weiterer Unterschied ergibt sich aufgrund der Tatsache, daß beim Anwendungsfall von RMARS die Übertragung für einen Client sowieso gescheitert ist, wenn dieser einem schwerwiegendem Fehler unterliegt. Die Übertragung der anderen kann problemlos vervollständigt werden, und die Übertragung des fehlerbehafteten Clients ist aufgrund ihrer Unvollständigkeit ohnedies nutzlos.

## 3.2 Nodes

Teilnehmer an einem Netzwerk, welchen durch ein beliebiges Adressierungsschema eindeutige Namen zugeordnet werden und welche durch diese Namen unterschieden werden können, werden als *Nodes* bezeichnet.

## 3.3 Protokoll

Ein Protokoll im Hinblick auf Kommunikation im Netzwerk bestimmt das Format, den Inhalt und die Bedeutung von Nachrichten, welche von den einzelnen Teilnehmern eines Netzwerks verschickt und empfangen werden [28].

Ein Protokoll ist also eine Kommunikationsvorschrift, welcher sich einzelne Nodes, die miteinander kommunizieren möchten, bedienen können. Bei der oben genannten Definition gemäß [28] entfällt aber die Erwähnung der Strategie, welche das Protokoll zum Verschicken oder Empfangen von Daten anwendet. Diese Strategie ist ein wesentlicher Faktor bei der Leistung und Verwendbarkeit eines Protokolls. In dieser Arbeit soll aber unter dem Begriff Protokoll auch immer die Strategie desselben verstanden werden.

## 3.4 Strategie

Eine Strategie zur Übermittlung von Datenpaketen wäre das sogenannte PAR-Protokoll [13]. Das Prinzip dieses PAR-Protokolls sei durch folgendes Szenario beschrieben. Es existiere ein Client und ein Server, sowie ein gemeinsames Kommunikationsmedium. Der Client möchte vom Server bestimmte Daten anfordern. Dazu übermittelt der Client eine Anfrage an den Server, welche die vom Server gewünschten Daten spezifiziert. Im Standardfall erhält der Server diese Anfrage, bearbeitet sie und generiert eine entsprechende Antwort, die er mittels Kommunikationsschnittstelle dem Client übermittelt. Der Client erhält die Antwort auf seine Anfrage und kann mit den empfangenen Daten weiterarbeiten. (siehe Abbildung 3.1)

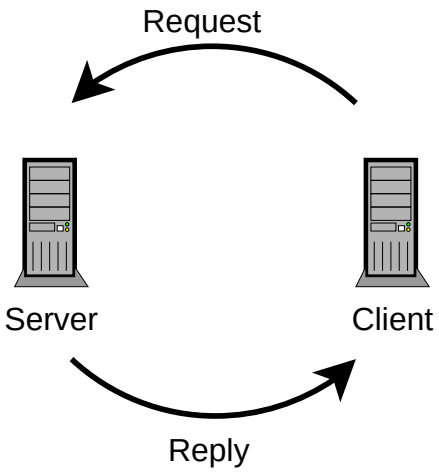


Abbildung 3.1: Server und Client im PAR Protokoll

Da es jedoch bei jeder Übertragung von Daten zu Fehlern kommen kann [27], ist nicht garantiert, daß alle verschickten Datensätze tatsächlich auch beim jeweiligen Empfänger ankommen. Die Strategie des PAR-Protokolls löst dieses Problem, indem der Client beim Versenden seiner Anfrage einen Timer startet. Sobald eine gewisse Zeit abgelaufen ist, innerhalb welcher keine Antwort durch den Client empfangen wurde (TIMEOUT), sendet der Client seine Anfrage erneut (RETRANSMISSION). Der Client wiederholt diesen

Vorgang solange (RETRY), bis er die Antwort vom Server innerhalb der geforderten Zeit erhalten hat. Eine Erweiterung des PAR-Protokolls dieser Art wäre, daß der Client zusätzlich die Anzahl der Wiederholungen des Verschickens der Anfrage überwacht (RETRANSMISSION_COUNTER), und nach einer gewissen Anzahl an Versuchen aufgibt, mit dem Server zu kommunizieren.

Dieses relativ einfache Schema birgt jedoch zumindest ein offensichtliches Problem. Wie groß soll die Zeitspanne sein, die der Client wartet, bevor er die Anfrage erneut sendet? Jedenfalls gilt folgende Bedingung:

$$d_{to} > 2 \cdot d_{tsc} + d_{BCET}$$

mit $d_{to}$ ist das Timeout, $d_{tsc}$ die Übertragungszeit einer Nachricht von Server zu Client und $d_{BCET}$ ist die minimale Zeit, die der Server zur Bearbeitung der Anfrage benötigt.

Bei einem Kommunikationskanal wie zum Beispiel Ethernet ist jedoch die Zeit, die ein Datenpaket benötigt, um vom Client zum Server zu gelangen, äußerst variabel. [27] Zusätzlich ist die Zeit, die der Server zur Verarbeitung der Anfrage benötigt, schwer zu begrenzen, sofern es sich bei dem Server nicht um ein Echtzeitsystem mit harten Deadlines handelt [13].

## 3.5 Multicast

Beim Multicast werden eine definierte Gruppe von einzelnen Nodes, die ja aufgrund eines gewissen Adressierungsschemas eindeutig unterscheidbar sind und einzeln angesprochen werden können, mit einer Gruppenkennung zugleich adressiert. Es kann also eine beliebige Node ein Datenpaket an diese Gruppenkennung adressieren, womit jedes Mitglied dieser Gruppe das Paket empfangen kann. [21]

Eine offensichtliche Voraussetzung ist die Gruppierung der einzelnen Teilnehmer zu einer Gruppe. Im IP Multicast zum Beispiel gibt es dafür das IGMP, das Internet Group Management Protocol [7].

Sobald die Gruppe gebildet ist, existiert für sie eine eindeutige Gruppenkennung. Eine Node, die Datenpakete an diese Gruppenkennung sendet, adressiert die Pakete also

eigentlich an alle Teilnehmer einer Gruppe. Multicast garantiert aber nicht, daß ein an die Gruppe versendetes Paket auch wirklich von allen Teilnehmern empfangen oder gar verarbeitet wird. Der Empfang allein bedeutet ja auch noch nicht, daß der entsprechende Client das Paket verarbeiten konnte. Der Client könnte ja mit anderen Aufgaben beschäftigt gewesen sein, sodaß das Paket zwar empfangen, aber letztlich ignoriert wurde. Es muß also vom tatsächlichen Empfängerprozeß bestätigt werden, daß das Paket auch verarbeitet werden konnte, sonst kann keine Zuverlässigkeit garantiert werden.

Vom sicherheitstechnischen Standpunkt her müßte aber eigentlich auch für einen neuen Bewerber, der Mitglied dieser Gruppe werden möchte, festgestellt werden, ob dieser dazu überhaupt berechtigt ist. Denn ginge es zum Beispiel um vertrauliche Daten, so könnte ansonsten ein unbefugter Dritter diese Daten mitschneiden, in dem er sich einfach für die Gruppe anmeldet. Da der Server nicht weiß, an wen aller er die Pakete schickt, da ihm ja nur die Gruppenadresse bekannt ist, ist das Verschicken vertraulicher Daten gefährdet.

## 3.6 Lokale Netze

Der Zweck eines Netzes ist, daß alle Nodes Daten untereinander austauschen können. Netze werden auch gemäß ihrer Größe klassifiziert. So kommt es zu Begriffen wie Local Area Network (LAN, auch Lokales Netz), Metropolitan Area Networks (MAN) und Wide Area Network (WAN). [17] Tatsächlich ist aber eine Einteilung der Größe nach nicht leicht realisierbar, weshalb die Unterscheidung dieser Netzbezeichnungen eher aufgrund von Übertragungsgeschwindigkeit und maximal möglicher räumlicher Ausdehnung getroffen wird.

Heutzutage findet man meist eine Gruppe von Computern in einem lokalen Netz mittels Ethernet verbunden. Ein Switch simuliert sozusagen den gemeinsamen Bus, den man vom Ur-Ethernet kennt. Der Switch verzögert zwar Pakete, auf der anderen Seite puffert er diese auch, wodurch es bei einem sogenannten *full-duplexed switched ethernet* zu keinen klassischen Ethernet Kollisionen mehr kommen kann. [27] Dieses Problem kann nur noch auf jene Art auftreten, daß ein Puffer für einen gewissen Port des Switches voll

ist. Dann kann der Switch zum Beispiel mittels Back-Pressure den Sender dazu bringen, seine Übertragungsrate zu reduzieren.

## 3.7 Flußkontrolle

Die Steuerung der Ablaufgeschwindigkeit eines Protokolls in Abhängigkeit vom Empfänger wird als *Flußkontrolle* (flow control) bezeichnet. [23]

## 3.8 Überlastkontrolle

Ein anderer Begriff, der oft mit dem Begriff Flußkontrolle verwechselt wird, ist jener der *Überlastkontrolle* (congestion control). Der Begriff Überlastkontrolle bezeichnet die Erkennung und das Setzen von Maßnahmen, um das Netzwerk nicht zu überlasten. [23] Im Gegensatz dazu geht es bei der Flußkontrolle darum, daß der Empfänger nicht überlastet wird. Netzwerküberlastung kann in paketvermittelnden Netzen auftreten, weil Netzwerkgeräte wie Netzwerkkarten, Switches und Router Pakete zwischenspeichern und bei Überfüllung von internen Puffern auch Pakete wegwerfen [20]. Wenn es sich bei den fallengelassenen Paketen zum Beispiel um Kontrollpakete handelt und ein Mechanismus zur Transportsicherung (siehe Abschnitt 3.12.1) aber genau diese Pakete immer erneut sendet, so kann das Protokoll nicht ordnungsgemäß ablaufen. Das Netzwerk ist praktisch mit sinnlosen Daten überfüllt. Ein Protokoll muß also auch Überlegungen dahingehend anstellen.

## 3.9 Topologie

Die Topologie (griech. topos: Ort; griech. logos: Gedanke, Lehre; [9]) beschreibt die räumliche Anordnung eines Netzwerks, sowie die Art und Weise der Verbindungen der Nodes untereinander. Entsprechend kann man folgende Topologien bei Lokalen Netzen unterscheiden. [11] [17]

1. Stern

2. Bus
3. offener und geschlossener Ring
4. Baum
5. Maschen

Abhängig von der verwendeten Topologie ist auch die Realisierung von Multicast unterschiedlich komplex. Zum Beispiel wären Ringstrukturen besser geeignet als Busstrukturen, um Reliable Multicast zu realisieren. Ein Ansatz dazu findet sich in 5.3.1. Typischer Vertreter für eine Busstruktur ist das Ethernet, als Beispiel für eine Ringstruktur kann der IBM Token Ring dienen. [17]

## 3.10 Server und Client

Die begriffliche Unterscheidung von Server und Client wird in dieser Arbeit im Bereich der Nutzdaten getroffen. Nodes, welche Datenquellen besitzen und Daten verschicken, werden als *Server* bezeichnet. Jene Nodes, welche Nutzdaten von Servern empfangen, werden *Clients* genannt.

## 3.11 Kommunikationsszenarien

Obwohl die Kommunikationsszenarien unabhängig davon betrachtet werden können, ob eine zuverlässige Datenübertragung realisiert werden soll oder nicht, möge der folgende Abschnitt über die verschiedenen Anwendungsfälle trotzdem in Hinblick auf einen reliable transport, wie er in 3.12.1 beschrieben wird, ausfallen.

Ein wesentlicher Punkt bei der Behandlung von Kommunikation im Allgemeinen ist die Frage nach dem Kommunikationsmedium und nach der Struktur der Kommunikationskanäle. Es gibt ja Netzstrukturen, sogenannte Maschenstrukturen [17], die es jeder einzelnen Node ermöglichen, mit jeder beliebigen anderen Node direkt zu kommunizieren. Solche Strukturen sind aber sehr teuer und aufwendig, sowohl im Aufbau, als auch

im Betrieb. Daher wird eine solche Form eher selten angetroffen, außer bei speziellen Anwendungen.

Folgende Kommunikationsszenarien können aufgrund der Konfiguration der Teilnehmer unterschieden werden.

### 3.11.1 Punkt-zu-Punkt Kommunikation

Punkt-zu-Punkt Kommunikation kann auch als *peer-to-peer unicast* bezeichnet werden, weil es in diesem Anwendungsfall genau zwei Teilnehmer am Kommunikationsprozeß gibt, wobei beide jeweils Daten senden und empfangen. Sie sind also gleichberechtigt, womit die Bezeichnung *peer-to-peer* gerechtfertigt ist. Da eine Nachricht jedoch nur an einen gewissen Teilnehmer gerichtet ist, nämlich immer an den anderen, ist der Begriff *unicast* angebracht.

Wie beim Szenario zur Beschreibung des PAR-Protokolls zu bemerken war, gibt es bereits bei reinen Punkt-zu-Punkt Verbindungen, bei welchen also nur zwei Nodes einfach miteinander kommunizieren, einige Schwierigkeiten. Ein wesentliches Problem für die Wahl des Timeout für die Retransmission ist aber zusätzlich vorhanden, wenn als zugrundeliegendes Protokoll Ethernet verwendet wird. Für den Client ist es zusätzlich unmöglich, die momentane Last des Servers abzuschätzen, welche sich ja wesentlich auf die Verarbeitungszeit der Anfrage auswirkt. Kurzum kann ein wichtiger Parameter des PAR-Protokolls, der sich mit Ablauf der physischen Zeit stark ändern kann, nur berechnet werden, wenn die Momentanbelastung des Servers und die momentane Übertragungsdauer über das Kommunikationsmedium bekannt ist. Beides steht dem Client aufgrund seiner Position nicht zur Verfügung. Daher wird für das Timeout ein Wert geschätzt.

Wird der Wert für das Timeout zu hoch geschätzt, so wird die Übertragungsgeschwindigkeit im Fall des Datenverlustes von Anfrage oder Antwort wesentlich reduziert. Es gilt also, ein möglichst geringes Timeout zu wählen.

Wird der Wert für das Timeout jedoch zu gering geschätzt, so werden Anfragen öfter wiederholt, obwohl der Server zum Beispiel noch bei der Verarbeitung genau einer sol-

chen Anfrage ist. Abhängig von der jeweiligen Implementierung gibt es hierbei unterschiedliches Verhalten. Wenn die Anfragen, die der Client wiederholt, absolut gleichen Inhalts und gleichen Formats sind, so wird der Client nicht merken, auf welchen Versuch der Server jetzt eigentlich geantwortet hat. Es kann hierbei also vorkommen, daß der Client eine Anfrage abschickt, der Server empfängt diese Anfrage, verarbeitet sie und verschickt die Antwort. Bevor jedoch die Antwort beim Client ankommt, läuft das Timeout des Client zur Wiederholung der Anfrage ab, weil das Timeout zu kurz gewählt wurde. Der Client verschickt seine Anfrage also nochmals, empfängt jedoch kurze Zeit später schon die Antwort des Servers auf die erste Anfrage. Der Server erhält trotzdem die zweite Anfrage, weil diese ja bereits abgeschickt wurde, und startet nochmals die Verarbeitung, deren Ergebnis aber nicht benötigt wird.

Eine mögliche Maßnahme zur Vermeidung solcher unnötiger Verarbeitungen wäre, daß der Server die Anzahl der Anfragen eines Clients, die er pro Zeiteinheit verarbeitet, beschränkt. Dies bereitet jedoch Probleme, wenn die Antwort des Server unterwegs verlorengeht. In diesem Fall nämlich müßte der Client ja gemäß PAR-Protokoll die Anfrage erneut wegschicken, weil er die Antwort ja nicht empfangen hat. Das Timeout des Clients muß aber jetzt auch noch die Rate berücksichtigen, mit welcher der Server Anfragen des Clients beantwortet. Dadurch wird das Timeout wiederum größer.

Es gibt demnach viele Fälle, welche den Durchsatz des Protokolls, also die Datenmenge pro Zeiteinheit, negativ beeinflussen. Je komplexer das Protokoll und je mehr verschiedene Zustände das Protokoll auf Seiten von Client wie von Server einnehmen kann, desto komplizierter wird die Analyse.

### 3.11.2 Kommunikation unter Gruppen

Erweitert man den Begriff einer Punkt-zu-Punkt Verbindung, wie sie im vorhergehenden Abschnitt beschrieben wurde, dahingehend, daß als Teilnehmer auch Gruppen von Nodes zugelassen werden, ergeben sich mehrere mögliche Kombinationen. Diese sollen im folgenden aufgelistet werden.

### 1:n Kommunikation

Es kann eine einzelne Node mit einer Gruppe von Nodes kommunizieren, dabei hat diese eine Node Daten, welche an alle Nodes der Gruppe fehlerfrei übertragen werden sollen. Dies entspräche sozusagen dem Szenario, in welchem ein Server und mehrere Clients existieren, die alle dieselben Daten empfangen sollen. Als Anwendung für solch ein Modell käme jegliche Art von Datendistribution in Frage. Diejenige Node, welche die Datenquelle besitzt, ist der Server. Alle anderen Nodes, die an der Kommunikation teilnehmen, werden als Clients klassifiziert.

### n:1 Kommunikation

Im Gegensatz dazu wäre das Kommunikationsszenario bei Datenkollektion wie folgend. Hierbei gibt es n Nodes, welche Daten zur Verfügung stellen. Alle diese Daten müssen an eine zentrale Sammelstelle übermittelt werden. Bei solch einer Situation gibt es demnach n Server, die einen Client bedienen. Hierbei müssen sich die Server untereinander arrangieren.

### m:n Kommunikation

Eine weitere Möglichkeit wäre, daß mehrere Server ihre Daten an mehrere Clients schicken müssen. Hierbei müssen sich ebenfalls die Server untereinander arrangieren, es muß aber zusätzlich sichergestellt werden, daß auch alle teilnehmenden Clients die Daten aller Server verarbeiten konnten. Die Bezeichnung *peer-to-peer multicast* wäre hier angebracht. Weil es in diesem Anwendungsfall keine Gruppe von ausgezeichneten Nodes gibt, kann keine sinnvolle Unterscheidung von Servern oder Clients vorgenommen werden. Alle Teilnehmer sollen von allen Teilnehmern empfangen und an alle Teilnehmer senden, deshalb der Begriff *peer-to-peer*. Weiters wird beim Versenden einer Nachricht an die Gruppe gesendet, weshalb der Begriff *multicast* zulässig wäre.

## 3.12 Eigenschaften von Protokollen

Nachfolgend werden die Eigenschaften aufgelistet, nach denen einzelne Protokolle klassifiziert werden können. Dabei muß es sich nicht unbedingt um Multicast Protokolle handeln. Die Eigenschaften hier sind aber speziell auf das Thema dieser Arbeit ausgelegt, weshalb sich diese Auflistung nicht auf Vollständigkeit beruft. Es sei an diesem Punkt nochmals darauf hingewiesen, daß der Begriff Protokoll in dieser Arbeit immer Protokoll und Strategie bedeutet (siehe 3.3).

### 3.12.1 Reliable vs. Unreliable

Die Frage, ob ein Protokoll sich als reliable (zuverlässig) oder unreliable (unzuverlässig) erweist, kann auf der Ebene des Transport Layer des *ISO OSI Reference Models* [1] angesiedelt werden, weshalb man auch von *reliable transport* spricht. Der Begriff Zuverlässigkeit möge durch drei Eigenschaften beschrieben werden:

1. duplicate-free
2. ordered
3. lossless

Damit ein Protokoll als reliable klassifiziert werden kann, muß ein sogenannter Mechanismus zur Transportsicherung sicherstellen, daß die drei oben genannten Eigenschaften vorhanden sind. Dieser Mechanismus wird von der *Transportinstanz* [27] ausgeführt.

Der Datenstrom wird in Pakete aufgeteilt. Diese Pakete können über das Netzwerk verschickt werden.

Die Eigenschaft *duplicate-free* ist insofern wichtig, da es bei paketvermittelnden Netzen, wie sie in dieser Arbeit betrachtet werden, passieren kann, daß Pakete aufgrund der verwendeten Protokolle mehrfach verschickt werden. Als Beispiel sei genannt, daß das PAR-Protokoll ein Paket erneut verschickt, wenn es zu keiner Bestätigung kam. Es ist also auf der einen Seite eine Maßnahme der Transportsicherung, Pakete zu duplizieren,

auf der anderen Seite muß die Transportsicherung aber auch garantieren, daß sie diese Duplikate erkennt und nur einmal verarbeitet.

Wenn ein Datenstrom in Pakete zerlegt wird, so ist auf der Abstraktionsebene des Datenstroms von dieser Zerteilung keine Kenntnis zu erwarten. Das Aufteilen in Pakete ist Aufgabe der Transportsicherung, daher muß die Transportsicherung auch darauf achten, die einzelnen Pakete wieder in genau derselben Reihenfolge zusammenzusetzen, wie sie im zu versendenden Datenstrom vorkommen. Diese Tatsache wird durch die Eigenschaft *ordered* ausgedrückt.

Da nun eine Aufteilung des Datenstroms in Pakete unvermeidbar ist, muß die Transportsicherung auch mit dem Problem des Paketverlusts fertig werden. Die meisten *Data Link Layer* Protokolle können oder wollen keine Garantie für eine zuverlässige Übertragung von Paketen geben, deshalb fällt diese Aufgabe höhergestellten Schichten zu. Maßnahmen der Transportsicherung, welche Paketverlust kompensieren können, bewirken die Eigenschaft *lossless*.

Je nach Anwendung reicht entweder ein unreliable Protokoll, wie zum Beispiel bei gewissen Video- oder Audiostreaming Protokollen, die mit Datenverlust umgehen können, weil für diese Protokolle verspätete Daten schlimmer sind als beschädigte [27], oder es wird ein reliable Protokoll benötigt, wenn aufgrund der Anwendung keine Datenfehler zulässig sind.

### 3.12.2 Server-controlled vs. Client-controlled

Ein Protokoll hat für die Wahl der Kontrollinstanz grob gesehen zwei verschiedene Ansätze zur Verfügung. Zum einen gibt es die Möglichkeit, daß ein Server die Steuerung des gesamten Protokollmechanismus übernimmt (*server-controlled*), zum anderen kann die Steuerung in irgendeiner Weise den Clients übertragen werden. Es sind auch Mischformen dieser beiden Ansätze möglich.

### Server-controlled

Bei dem *server-controlled* Ansatz fällt einem Server die zentrale Stellung im Ablauf des Protokolls zu. Welcher Teilnehmer der Kommunikation zu so einem Server wird, ist meist abhängig von zusätzlichen strategischen Komponenten. Im Fall der 1:n Kommunikation zum Beispiel fällt die Bestimmung des Servers nicht schwer, da es sich situationsbedingt um jenen Teilnehmer handeln muß, welchem einzig und allein die zu verteilenden Daten zur Verfügung stehen.

Die zentrale Stellung des Servers bewirkt, daß dieser auch die zentrale Intelligenz des Protokolls und den *Herzschlag* des Protokollablaufs übernehmen muß. Im Fall des Reliable Multicast sollte der Server deswegen auch genauestens über den Status aller Clients Bescheid wissen. Entsprechende Mechanismen zur Abfrage des Status der Clients von Seiten des Servers müssen vom Protokoll zur Verfügung gestellt werden.

Die Intelligenz des Servers muß folgende Punkte berücksichtigen: Einerseits muß überprüft werden, ob alle Clients noch am Protokolllauf teilnehmen. Aufgrund eines Fehlers könnten Clients ja funktionsuntüchtig werden. Dies muß vom Server erkannt werden, und der Server muß entsprechend der Vereinbarung im Protokoll agieren.

Das, was hier als Herzschlag des Protokolls bezeichnet wird, meint letztlich die Steuerung der Ablaufgeschwindigkeit der Datenübertragung in Abhängigkeit von den Empfängern, also die Flußkontrolle.

Andererseits muß der Server demnach auch feststellen, ob die Clients bei der aktuellen Übertragungsgeschwindigkeit mithalten können. Gelingt dies manchen Clients über einen längeren Zeitraum nicht, so sollte der Server die Datenrate drosseln. Wenn einzelne Clients Datenfragmente nicht oder nur fehlerhaft empfangen konnten, muß der Server dies ebenfalls erkennen und entsprechende Maßnahmen setzen, in dem zum Beispiel gewisse Datenpakete erneut gesendet werden.

Dem Server fallen also aufgrund seiner zentralen Stellung Flußkontrolle und Überlastkontrolle zu. Er kann diese Aufgabe aber nur dann sinnvoll bewältigen, wenn ihm auch ausreichend Informationen über den Status der Clients verfügbar sind.

#### Client-controlled

In diesem Ansatz gibt es rein theoretisch betrachtet keine ausgezeichnete Zentralstelle des Protokolls, wie eine solche beim server-controlled Ansatz zu finden wäre. Die Clients übernehmen die Steuerung des Protokollablaufs, weshalb zum Beispiel ein Server in einer 1:n Kommunikation hauptsächlich durch die Aktionen der Clients gesteuert wird. Dementsprechend wird hier der Begriff der *distributed control* eingeführt, der ausdrücken soll, daß keine eindeutige, singuläre Kontroll- oder Steuerinstanz in diesem Ansatz festzustellen ist. Unabhängig vom zugrundeliegenden Kommunikationsszenario sollen hier die Server lediglich Daten anbieten, mit welcher Geschwindigkeit oder zu welchem Zeitpunkt diese gesendet werden obliegt jedoch der verteilten Kontrolle der Clients.

### 3.12.3 ACK vs. NACK

Hinsichtlich der Bestätigung von Datenpaketen gibt es zwei grundverschiedene Klassen von Ansätzen.

#### ACK

Ein Ansatz mit ACK bedeutet, daß Bestätigungen für erfolgreich empfangene Datenpakete oder Datenportionen verschickt werden. Kurzum wird für nicht empfangene Daten auch nichts bestätigt. Nicht empfangene oder fehlerhaft empfangene Daten bewirken keine Auslösung einer Kontrollnachricht.

Beim TCP [19] zum Beispiel wird ja letztlich eine andauernde, gegenseitige Bestätigung empfangener Datenpakete durchgeführt. Es wäre vergleichbar mit zwei Personen, die sich gegenseitig Briefe schreiben. In diesem Vergleich schickt Person A einen Brief aus. Person B erhält diesen Brief und schickt an Person A einen Brief, der bestätigt, daß Person B den Brief von Person A erhalten hat. Doch weiß Person B jetzt nicht, ob diese Bestätigung auch tatsächlich zu Person A gelangt ist. Person A muß deswegen wiederum eine Bestätigung für die Bestätigung schicken. Analog besteht das Problem aber wiederum für die Bestätigung der Bestätigung. (siehe Abbildung 3.2)

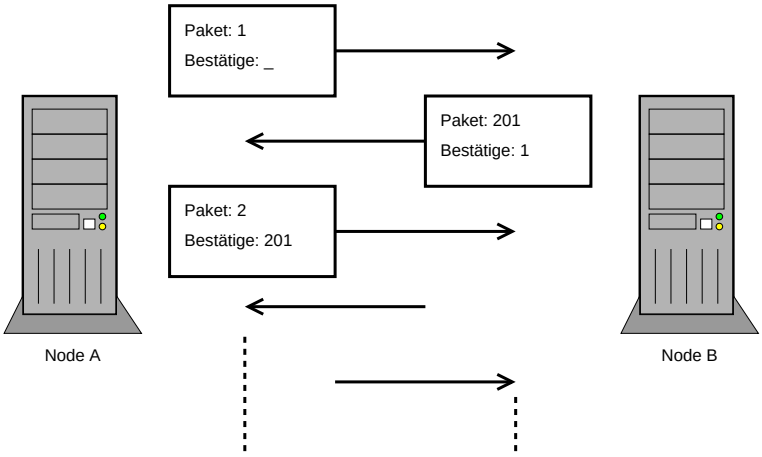


Abbildung 3.2: Gegenseitige Bestätigung

Wenn in diesem andauernden Hin und Her von Bestätigungen zusätzlich Nutzdaten übertragen werden, so macht diese scheinbar unsinnige Vorgangsweise plötzlich Sinn. Um wieder den Zusammenhang zu TCP herzustellen, müßten die Personen zusätzlich noch die Anzahl der Buchstaben, die sie vom anderen erhalten haben, in die Bestätigung hineinschreiben. Dies entspräche dem TCP Mechanismus der Sequence Number und Acknowledgement Number.

Der ACK Ansatz kann sowohl für server-controlled, als auch für distributed control eingesetzt werden.

Wird beim Multicast ein ACK Ansatz verwendet, so kann es zu sogenannten ACK implosions kommen (siehe Abschnitt 5.3.2).

### NACK

Bei einem Ansatz mit NACK, was soviel wie Not Acknowledgment bedeutet, wird jedes fehlerhafte oder nicht empfangene Paket eine Kontrollnachricht auslösen. Der Client generiert eine Kontrollnachricht an den Server, wenn er das Fehlen eines Pakets oder Datenfehler in einem Paket festgestellt hat. In jenem Fall, daß die Übertragung fehlerfrei läuft, wird keinerlei Kontrollnachricht von Seiten des Clients generiert.

Um dieses Prinzip realisieren zu können, werden Datenpakete meist mit Sequenznummern versehen. Diese Sequenznummern werden vom Server vergeben und entsprechen im einfachsten Fall einer Folge, welche als Folgeglieder die Menge der natürlichen Zahlen, aufsteigend angeordnet, besitzt. Der Client erkennt fehlende Pakete nun daran, daß bei den Sequenznummern der empfangenen Datenpakete einzelne Zahlen fehlen. Der Client generiert eine NACK Kontrollnachricht, in welcher er auch gleich die fehlende Sequenznummer einträgt, und schickt diese an den Server. [28]

Das wesentliche Problem bei diesem Ansatz ist die Tatsache, daß der Server keine Bestätigung für den Erfolg der Übertragung hat. Das Prinzip setzt zusätzlich aber schon voraus, daß der Client auch merkt, wenn ein Fehler auftritt, und daß der Client dies auch melden kann. Ein Totalausfall des Clients wird vom Server nicht erkannt und kann in gewissen Anwendungsfällen den Gesamterfolg der Übertragung zunichte machen.

Der NACK Ansatz erfordert in der Gruppenkommunikation zusätzlich meist die Verwendung von distributed control, weil ja nur einer, der ein Client ist, feststellen kann, ob er ein Paket erhalten hat oder nicht.

Wird beim Multicast ein NACK Ansatz verwendet, so kann es zu sogenannten NACK implosions kommen, die ein Analogon zu den ACK implosions darstellen.

## 3.13 Bewertungskriterien für Protokolle

Die für diese Arbeit als relevant betrachteten Kriterien zur Bewertung von Protokollen werden im folgenden ausgeführt.

### 3.13.1 Latenzzeit

Die Latenzzeit gibt an, wie lange ein Datenpaket verzögert wird, wenn es vom Sender zum Empfänger gelangt. Die Latenzzeit ist also jene Zeitspanne, die sich vom Zeitpunkt des Abschicken des Datenpakets bis zum Zeitpunkt dessen Ankunft beim Empfänger erstreckt. Eine positive Bewertung eines Protokolls hinsichtlich dessen Latenzzeit wäre gegeben, wenn das Protokoll eine maximale Latenzzeit garantieren kann. Dies hätte

großen Nutzen bei interaktiver Verwendung oder anderen zeitkritischen Anwendungen. Die Latenzzeit wird in Sekunden gemessen, bei entsprechenden Anwendungen auch in Größenordnungen von Millisekunden oder Mikrosekunden.

### 3.13.2 Durchsatz

Der Durchsatz bezeichnet die durchschnittliche Datenrate, das heißt, wie viele Daten pro Zeiteinheit durchschnittlich übertragen werden. Die Latenzzeit ist für dieses Kriterium irrelevant, es zählt nur die Leistung der Gesamtübertragung. Der Durchsatz wird in Byte pro Sekunden angegeben, entsprechend der Anwendung auch in KiloByte pro Sekunde oder MegaByte pro Sekunde.

### 3.13.3 Scalability

Mit dem Begriff Scalability wird eine Eigenschaft von Protokollen erfaßt, die speziell auf dem Gebiet des Multicast von großer Bedeutung ist. Zur Erklärung dieses Merkmals wird die Definition der Leistung gemäß Abschnitt 3.13.5 verwendet. Ein Protokoll ist scalable (dt. skalierbar), wenn die Leistung des Protokolls sich auch bei hohen Teilnehmerzahlen in der selben Größenordnung befindet wie bei geringen Anzahlen von Clients. Bei den meisten Lösungsansätzen für Protokolle wirken sich hohe Teilnehmerzahlen negativ auf die Kriterien Latenzzeit und Durchsatz aus. Besonders schwerwiegend ist es aber, wenn es auch Auswirkungen auf das Kriterium Reliability gibt, das im nächsten Abschnitt beschrieben wird. In diesem Fall wäre also Zuverlässigkeit für eine geringe Anzahl von Teilnehmern gegeben, bei wachsender Teilnehmerzahl würde jedoch diese höchst nützliche Eigenschaft schwinden.

### 3.13.4 Reliable vs. Unreliable

Stellt das Protokoll den Anspruch auf Reliability (Zuverlässigkeit), d.h. erfüllt es die Punkte duplicate-free, ordered und lossless gemäß 3.12.1, so wird sich diese Eigenschaft auch auf Kosten der Leistung des Protokolls auswirken. Letztlich werden dadurch die oben genannten Kriterien Latenz, Durchsatz und Scalability negativ belastet, weil die

Transportsicherung zur Zuverlässigkeit eben eines gewissen Aufwandes bedarf. Ein Protokoll hingegen, welches auf Zuverlässigkeit verzichtet, kann zum Beispiel leichter eine maximale Latenz garantieren. Wenn ein Paket ankommt, ist es innerhalb der festgelegten Zeit da. Danach kommt es eben überhaupt nicht mehr. Oder ein Protokoll verzichtet auf reliable transport, dafür garantiert es, daß ein minimaler Durchsatz immer gewährleistet ist, wozu auch immer jemand diese Eigenschaft benötigen würde. Hin und wieder gehen Pakete verloren, aber das ist ja erlaubt, weil es sich um ein Protokoll handelt, das unreliable ist.

Deswegen wird für Protokolle auch Reliability als Bewertungskriterium eingeführt, weil diese Eigenschaft auf der einen Seite viele Nachteile in Hinsicht auf andere Kriterien bringt, auf der anderen Seite aber einen wesentlichen Beitrag zum Nutzen und zur Verwendbarkeit eines Protokolls leistet.

### 3.13.5 Leistung

Als Leistung könnte man die Gesamtbewertung eines Protokolls bezeichnen. Dabei fließen die bisher genannten Bewertungskriterien wie Latenzzeit, Durchsatz, Scalability und Reliability ein. Es kann also die Leistung eines Protokolls sehr gut sein, wenn es auf der einen Seite reliable transport zur Verfügung stellt, auf der anderen Seite aber bei Kriterien wie zum Beispiel der Latenzzeit schlechter abschneidet.

## 3.14 Ausblick

In dieser Arbeit soll das 1:n Kommunikationsszenario, wie es in 3.11.2 beschrieben wurde, eingehend behandelt werden. Die anderen vorgestellten Szenarios wurden nur der Vollständigkeit halber erwähnt und werden durch diese Arbeit nicht abgedeckt. In gleicher Weise soll nur der Bereich des reliable transport, wie in 3.12.1 beschrieben, bearbeitet werden, da nur dieser, im Gegensatz zum *unreliable transport*, eine Herausforderung darstellt. Die zugrundeliegende Topologie sei Ethernet, da diese die heutzutage am meisten anzutreffende Struktur bei lokalen Netzen ist. Kurz — aber nicht unbedingt korrekt

definiert — gesagt, wird die fehlerfreie Übertragung von Daten an eine Gruppe von Nodes im Ethernet untersucht.

Da also nur das 1:n Kommunikationsszenario behandelt wird, ist im folgenden mit dem Begriff Server immer jene Node bezeichnet, welche die Datenquelle besitzt. Jene Nodes, welche diese Daten fehlerfrei erhalten sollen, werden als Clients klassifiziert.

# 4 Überblick über potentiell geeignete Multicast-Protokolle

Wie aus den Ausführungen des vorherigen Kapitels hervorgeht soll für die Realisierung der Aufgabe der Software-Verteilung in Schulinfrastrukturen ein Reliable Multicast Protokoll eingesetzt werden. Da es sich bei Multicast um ein durchwegs gut erforschtes Gebiet handelt, kann auf eine breite Masse von bereits vorhandenen Protokollen zurückgegriffen werden, die im nachfolgenden Teil dargestellt und in Hinblick auf die Verwendbarkeit für Software-Verteilung betrachtet werden.

## 4.1 XTP [26]

XTP ist die Abkürzung für Xpress Transport Protocol und wurde ursprünglich für Hochgeschwindigkeitsnetzwerke entwickelt. Entsprechend sollte das gesamte Protokoll auch dahingehend gestaltet werden, daß es leicht in Hardware implementiert werden kann.

Eigentlich handelt es sich bei der ursprünglichen Version von XTP um ein Protokoll, das sehr flexibel in Hinsicht der Erweiterbarkeit, der Funktionalität, sowie der individuellen Anforderungen ist. Dementsprechend kann es sehr leicht adaptiert werden, um in den verschiedensten Anwendungsfällen brauchbar eingesetzt zu werden. Benötigt eine Anwendung garantierte Latenzzeit, so ist dies genauso möglich wie zum Beispiel die Realisierung des Reliable Multicast, nur eben nicht gleichzeitig.

In der ursprünglichen Fassung sollte XTP die Funktionalität von Network Layer und Transport Layer verbinden. Das war auch der Grund, weshalb sich XTP nicht als Transport Protokoll durchgesetzt hat, weil das Internet, damals wie heute, wesentlich auf IP

basiert und deswegen weitreichende strukturelle Änderungen notwendig geworden wären, um XTP zu realisieren. Die letzte Fassung von XTP wurde daher lediglich als Transport Protokoll ausgeführt, welches IP als Grundlage verwendet.

Ein wesentliches Merkmal von XTP ist die Trennung von Kontroll- und Nutzdaten.

Besonders interessant für diese Arbeit ist natürlich die Fähigkeit, Reliable Multicast über XTP zu realisieren. Der Reliable Multicast mit XTP bedient sich einer server-controlled Strategie, für die Bestätigung eines NACK Ansatzes. Es wird darauf hingewiesen, daß es bei der Art des NACK Ansatzes, welche XTP verwendet, zu NACK implosions (3.12.3) kommen kann

XTP verwendet ein *sliding window* Protokoll [3], um Flußkontrolle zu realisieren. Dabei gibt der Empfänger an, welche Menge von Daten er vom Server erwartet beziehungsweise verarbeiten kann. Im XTP Reliable Multicast wird von allen Empfängern so eine Vorgabe verschickt, der Server prüft alle diese und verwendet den kleinsten Wert. Damit ist gewährleistet, daß der Server sich am langsamsten Client orientiert.

Der Bestätigungsmechanismus bei XTP läuft wie folgt ab. Es handelt sich um einen server-controlled NACK Mechanismus. Das bedeutet, daß Clients nur dann Kontrollnachrichten aussenden, wenn sie vom Server dazu aufgefordert werden. Diese Aufforderung kann durch ein Feld, welches im Datenpaket vorgesehen ist, getätigt werden. Daraufhin schicken *alle* Clients gegebenenfalls ihre NACKs an den Server. Da dem Server im XTP die einzelnen Teilnehmer der Gruppe nicht unbedingt bekannt sind, kann keine gezielte Aufforderung zum NACK oder auch ACK geschickt werden. Dementsprechend können nur alle Clients gleichzeitig zum ACK oder NACK aufgefordert werden, was zur NACK implosion führen kann. Die Entwickler des Protokolls empfehlen daher eine Erweiterung, die als *slotting and dumping* bezeichnet wird. Clients sollen auf Bestätigungsanforderung nicht sofort antworten, sondern eine gewisse, zufällige Zeitdauer abwarten. Innerhalb dieser Zeitdauer soll der Client horchen, ob nicht andere Clients schon NACKs schicken, die auch seine Bedürfnisse befriedigen. In diesem Fall soll der Client auf das Aussenden seines NACKs verzichten.

Als Zusammenfassung kann man über den Mechanismus des XTP sagen, daß es eine modulare und erweiterbare Architektur ist, die aber aufgrund einiger Entscheidungen in der Gestaltung hohe Komplexität mit sich bringt. Man beachte nur den Lösungsansatz zur Umgehung der NACK implosion, welche auf Seiten des Clients höhere Komplexität und vor allem Unberechenbarkeit bewirkt. Aufgrund dieser hohen Komplexität auf der Client Seite ist der Einsatz dieses Protokolls für die Anwendung der Software-Verteilung nicht geeignet.

## 4.2 MTP [2]

Bei MTP wird die Menge aller Teilnehmer an der Kommunikation als *Web* bezeichnet. Die Teilnehmer selbst werden in drei Klassen aufgeteilt, nämlich in *Master*, *Producer* und *Consumer*. In einem Web kann es nur einen Master geben. Ein Producer ist quasi ein Server, also ein Teilnehmer, der Daten verschickt. Consumer sind hingegen alle Teilnehmer, die Daten empfangen sollen.

Der erste Teilnehmer wird zum Master. Ein Mechanismus soll verhindern, daß es mehr als einen Master gibt, jedoch kann es bei Ausführung desselben im Extremfall auch dazu kommen, daß gar kein Master mehr existiert.

Durch den Master werden auch die Kommunikationsparameter in einem Web festgelegt. Folgende Parameter können spezifiziert werden:

1. reliable oder unreliable
2. 1:n oder n:m Kommunikation
3. Minimaler Durchsatz
4. Maximale Größe eines Nutzdatenpakets

MTP arbeitet mit einem NACK Ansatz und *retention times*. Das bedeutet, daß ein Producer versendete Daten noch eine Zeit lang in einem Pufferspeicher aufbewahrt. Nach Ablauf der retention time werden die Daten aus dem Speicher entfernt. Wenn Clients es

also nicht bewerkstelligen konnten, ihre NACKs vor Ablaufen der retention time an den Server zu übermitteln, so wird das Protokoll für jene Clients unreliable. Obwohl durch die Anfangsparameter gewünscht, kann Reliability nicht in jedem Fall durch MTP garantiert werden, weshalb der Einsatz von MTP für die Aufgabe der Software-Verteilung, die ja Zuverlässigkeit als wesentliches Kriterium fordert, nicht sinnvoll ist.

## 4.3 SRM [8]

SRM steht für Scalable Reliable Multicast und behauptet durch diesen Namen relativ viel. SRM verwendet einen NACK Ansatz, der in der Ausführung aber ein wenig modifiziert ist. Clients sollen horchen, ob nicht bereits NACKs ausgesendet wurden, die auch ihre Bedürfnisse befriedigen, also ähnlich wie bei XTP. Die Zeitdauer, die ein Client wartet, bevor er NACKs verschickt, wird abhängig von dessen Distanz zum Server gemacht. Dies soll Vorteile bringen, weil dadurch ein nähergelegener Client das NACK früher ausschickt und der weiter entfernte Client dies dann nicht mehr tun muß. Zusätzlich senden bei SRM alle Teilnehmer regelmäßig Statusberichte. Bei großen Teilnehmerzahlen belastet dies aber das Netzwerk. Ein wesentlicher Unterschied von SRM zu anderen Protokollen ist, daß alle Teilnehmer die Aufgabe der Retransmission für andere übernehmen können. Es muß also nicht der Server die Daten erneut aussenden, sondern es kann auch ein Client in der Nähe des zurückgebliebenen Teilnehmers tun.

Beim vorgestellten NACK Ansatz kann es aber durchaus zu NACK implosions kommen, weil das Mithören von NACKs nur in glücklichen Fällen eine Reduktion der Anzahl an NACKs bringt. Außerdem wird komplizierte Zeitsynchronisation unter den Teilnehmern notwendig, um das SRM NACK Prinzip zu realisieren.

Die dadurch entstehende, höhere Komplexität und das Vorhandensein von NACK Implosions lassen das SRM Protokoll für den Einsatz zur Software-Verteilung als nicht sinnvoll erscheinen.

## 4.4 PGM [24]

PGM steht für Pragmatic General Multicast und hat folgende Eigenschaften. Im PGM sind die Teilnehmer einer Gruppe nicht bekannt, deshalb kann Reliability nicht in allen Fällen gewährleistet werden. Das Protokoll verwendet einen client-controlled Ansatz mit einem speziellen NACK Prinzip.

Wenn man die Anordnung der Teilnehmer an der Gruppenkommunikation im Netzwerk beschreiben möchte, so kann dies am besten durch eine Baumstruktur geschehen. Dabei steht der Sender an der Wurzel, die Clients sind die Blätter des Baums. Im PGM werden Pakete zur Pfadanalyse vom Sender zu den Clients geschickt, sogenannte Source Path Messages (SPM). Router und andere Zwischenstellen in der Paketvermittlung verarbeiten diese Nachrichten, um Auskunft über die Struktur des Netzes zu erhalten. Dabei kann jedes Gerät oder jeder Teilnehmer feststellen, wer sein Vorgänger im Pfad ist. Dadurch wird es möglich, daß NACKs an Verzweigungspunkten des Baumes verschmolzen werden, sodaß beim Sender im besten Fall nur ein NACK ankommt.

Es gibt im PGM für den Client zwei Möglichkeiten, um NACKs zu verschicken. Erstens kann er sein NACK per Unicast an seinen Vorgänger im Pfad schicken. Eine andere Möglichkeit wäre, das NACK per Multicast an alle Teilnehmer im lokalen Subnet zu schicken, das heißt, mit einer TTL von 1.

NACKs werden von Clients so lange wiederholt, bis diese bestätigt werden. Wenn NACKs an die Multicastadresse geschickt werden, so müssen Clients zuvor eine zufällige Zeit abwarten und horchen, ob nicht NACKs mit denselben Anforderungen verschickt wurden. In diesem Fall wird der Client auf die Aussendung seines NACKs verzichten.

Durch die Pfadanalyse wird es dem Sender letztlich auch möglich, Retransmissionen nur für jene Teile des Netzwerks durchzuführen, welche diese brauchen.

Dadurch, daß Router die Verschmelzung mehrerer NACKs vornehmen, ist das Problem der NACK implosions großteils verhindert. Ähnlich wie bei der Analyse von SRM ist aber das Abwarten von NACKs anderer Clients ein Ansatz, bei dem Erfolg oder Mißlingen letztlich unvorhersehbar ist. Dementsprechend ist auch dieses Protokoll für den Einsatz zur Software-Verteilung nicht geeignet.

## 4.5 MFTP [22]

Beim MFTP, dem Multicast File Transfer Protocol, ist ein wesentliches Designziel, Dateien an mehrere Clients zu übertragen. Dabei werden zwei grundlegende Betriebsarten unterschieden, nämlich ob die Kommunikationsteilnehmer bekannt oder unbekannt sind. Wenn die Teilnehmer bekannt sind, wird ein Subprotokoll des MFTP angewandt, das sogenannte Multicast Control Protocol (MCP).

Im ersten Durchlauf des Protokolls wird die Datei vollständig übertragen, danach wird der Sendevorgang solange wiederholt, bis alle Clients eine fehlerfrei Kopie bekommen haben.

Der Sender schickt hin und wieder ein allgemeines NACK Request aus, auf welches all jene Clients antworten sollen, welche Fehler bei der Übertragung bemerkt haben. Durch diesen Mechanismus kann es zu NACK implosions kommen, weshalb das Protokoll nicht scalable ist.

## 4.6 Eignung bestehender Protokolle für eine Schulinfrastruktur

Für die Software-Verteilung in Schulinfrastrukturen ist das oberste Ziel die Zuverlässigkeit der Übertragung, danach folgt als zweitwichtigste Anforderung die Geschwindigkeit der Übertragung, weil der Vorgang der Gesamtwiederherstellung ja möglichst oft durchgeführt werden können soll.

Alle vorgestellten Protokoll weisen bei dieser Betrachtungsweise aber Schwierigkeiten auf. Den meisten Reliable Multicast Protokollen ist gemein, daß sie zu komplex sind und deswegen von Seiten des Clients hohe Anforderungen stellen, die wiederum auf den Protokollablauf wirken und die Übertragung verlangsamen. In extremen Fällen können Protokolle nicht einmal mehr die Eigenschaft der Reliability gewähren, wie zum Beispiel beim vorgestellten MTP. Überhaupt nicht einsetzbar sind Protokolle, die nichteinmal über die teilnehmenden Clients Bescheid wissen. In solchen Fällen ist die Forderung von Reliability unmöglich umzusetzen.

Da es sich bei Anwendung der Software-Verteilung in Schulinfrastrukturen scheinbar um einen derartigen Spezialfall handelt, daß bestehende Protokoll nicht sinnvoll einsetzbar sind, schlägt diese Arbeit die Entwicklung eines eigenen, speziell für diesen Anwendungsfall gestalteten Protokolls ein.

# 5 Software-Verteilung in Schulinfrastrukturen mittels Reliable Multicast

Das universelle Reliable Multicast Protokoll wird es nicht geben, weil es aufgrund der sich ergebenden Komplexität nicht möglich ist, alle Anforderungen gleichermaßen zu erfüllen. Betrachtet man Begriffe wie Latenzzeit, Durchsatz und Scalability, wie sie im Abschnitt 3.13 beschrieben wurden, so wird bei näherer Betrachtung und Überlegung die Unvereinbarkeit aller dieser Kriterien offensichtlich. Es hängt demnach stark vom geplanten Einsatzgebiet ab, in welcher Richtung sich die optimale Lösung für das gestellte Problem befindet. Eine optimale Lösung kann sehr leicht zu finden sein, wenn nur das Problem ausreichend beschränkt und klar definiert ist, was ja für den Einsatz von Software-Verteilung in Schulinfrastrukturen leicht möglich ist. Im folgenden sollen einige Anwendungsfälle, sowie bereits existierende Lösungsansätze vorgestellt und beschrieben werden.

## 5.1 Anwendungsfälle

Die Anwendungsfälle sind immer auf dem Gebiet der Datendistribution, weil ja aufgrund des Ausblicks der Arbeit in 3.14 nur das 1:n Kommunikationsszenario behandelt werden soll.

Der Unterschied bei den nachfolgend behandelten Anwendungsfällen ist also nur, ob es sich um Daten mit Echtzeitcharakter handelt oder nicht. Datendistribution ist es auf jedenfall.

Wird eine obere Schranke für die Latenzzeit und gleichzeitig eine zuverlässige Übertragung gewünscht, so sind dies zwei Eigenschaften, die sich nicht leicht vereinbaren lassen. Dadurch, daß die Übertragungsgarantie nicht nur für eine Node, sondern für die ganze Gruppe gewährleistet sein muß, kann es wegen Maßnahmen der Transportsicherung öfter zu Verzögerungen kommen, wodurch das Garantieren einer maximalen Latenzzeit erschwert wird. Es handelt sich demnach um einen Trade-Off (Gratwanderung) zwischen garantiert maximaler Latenz (siehe 3.13.1) und Reliable Transport (siehe 3.13.4).

Gibt es keine Echtzeitanforderungen für die zu übertragenden Daten, so wird das Problem des Reliable Multicast auf einen sehr einfachen Spezialfall reduziert. Das in dieser Arbeit entwickelte und vorgestellte Protokoll ist für diesen Spezialfall konzipiert und ausgeführt. Näheres dazu findet sich dann in Abschnitt 6.

### 5.1.1 Echtzeitanwendungen

Sind die zu transportierenden Daten Echtzeitdaten gemäß [13], so ist vor allem eine vom Protokoll garantierte Latenzzeit notwendig, um prüfen zu können, ob das Protokoll für den jeweiligen Einsatz geeignet ist. Zusätzlich muß für die Feststellung der Verwendbarkeit das Kriterium des Durchsatzes berücksichtigt werden.

Latenzzeit allein wäre wichtig, wenn es sich um eine interaktive Anwendung handelt, wie zum Beispiel SSH [30]. Da für SSH aber keine sinnvolle Anwendung in einem Multicastbetrieb erdenkbar ist, scheidet diese Variante aus. Ein anderes Beispiel wäre ein Warnsystem. Dabei tritt ein Ereignis ein, welches dem Server gemeldet wird. Der Server hat die Aufgabe, alle Clients sofort über das Auftreten dieses Ereignisses zu informieren. Hierbei wäre eine garantiert maximale Latenzzeit brauchbar.

Wenn es jedoch zum Beispiel um die Übertragung eines Videostreams an eine Gruppe geht, wäre die Garantie sowohl einer oberen Schranke für die Latenzzeit, als auch einer unteren Schranke für den Durchsatz notwendig, um die Anwendung durchführen zu

können. Die Übertragung eines MPEG2 Video und Audio Streams in höchster Qualität des DVD Standards kann durch eine Datenrate von 10 Mbps realisiert werden. Um das Video ohne Unterbrechungen wiedergeben zu können, sollte am Client eine Pufferung der Daten vorgenommen werden. Wenn jedoch keine maximale Latenzzeit garantiert wird, so kann es passieren, daß der clientseitige Puffer irgendwanneinmal leer wird und die Wiedergabe zum Stocken kommt. [27]

### 5.1.2 Datendistribution

Im vorhergehenden Punkt wurde bereits darauf verwiesen, daß es sich streng genommen eigentlich immer um Datendistribution handelt, weil der Ausblick dieser Arbeit auf diesen Bereich spezialisiert ist. Was sich also hier hinter dem Begriff Datendistribution verbirgt, ist die Tatsache, daß es sich bei den zu übertragenden Daten um Nicht-Echtzeitdaten handelt. Hier ist die Gültigkeit und Brauchbarkeit der Daten unabhängig von deren Ankunftszeitpunkt. Es werden also keine sonderlichen Forderungen an den Garant maximaler Latenzzeiten gestellt. Trotzdem wird gefordert, daß die Datendistribution möglichst schnell abläuft. Deswegen wird in diesem Fall das Hauptaugenmerk auf den Durchsatz gelegt; es gilt, diesen zu maximieren.

## 5.2 Fehlerhafte Clients

Bevor auf die eigentlichen Lösungsansätze, die es für Reliable Multicast gibt, eingegangen werden kann, muß eine wichtige Frage behandelt werden. Wie soll das Protokoll mit Clients verfahren, die schwerwiegenden Fehlern unterliegen?

Bei Punkt-zu-Punkt Kommunikation gemäß Abschnitt 3.11.1 ist das Auftreten eines schwerwiegenden Fehlers bei einem der beiden Kommunikationspartnern mit klaren Reaktionen verbunden. Die Kommunikation wird definitiv sinnlos, sobald bei einem der beiden Teilnehmer schwerwiegende Fehler auftreten, sodaß eine Wiederherstellung des Protokollablaufs nicht mehr möglich ist.

Im Fall der Gruppenkommunikation ist es aber nicht möglich, eine dermaßen klare Vorgangsweise zu definieren. Vielmehr ist die sinnvollste Behandlung des Totalausfalls eines Clients abhängig vom Anwendungsfall.

Wenn es sich zum Beispiel um die Übertragung eines Festplattenimages von einem Server zu n Clients handelt, so macht es durchaus Sinn, bei Totalausfall eines Clients den Rest der Gruppe weiterzubetreuen. Da der ausgefallene Client ja offensichtlich einen entsprechend schwerwiegenden Fehler hat, sodaß jener nichteinmal das Festplattenimage vollständig empfangen kann, würde dieser ja auch nicht als Arbeitsstation dienen können.

Handelt es sich hingegen um eine verteilte Anwendung, die unbedingt alle Teilnehmer des Multicast benötigt, so kann der gesamte Protokollauf abgebrochen werden, da nach Ausfall eines Clients die Aufgabe sowieso nicht mehr erfüllt werden kann.

Es hängt also stark vom Anwendungsfall ab, welche Möglichkeiten das Protokoll bei schwerwiegenden Fehlern von Clients ausschöpfen kann.

## 5.3 Lösungsansätze und Methoden

Im folgenden Teil sollen einige Lösungsansätze und Methoden, die bei der Realisierung von Reliable Multicast Protokollen angewendet werden können, aufgeführt und dargestellt werden.

### 5.3.1 Reliable Multicast in einer Ring-Topologie

Liegt bei der verwendeten Topologie eine Ringstruktur vor und ist diese entsprechend adaptionsfähig, so kann Reliable Multicast nach folgendem Ansatz relativ leicht verwirklicht werden.

Gegeben sei eine geschlossene Ringstruktur. Es werden nur jene Nodes betrachtet, die am Reliable Multicast teilnehmen. Der Server generiert ein Datenpaket, welches aus einer Organisationsstruktur und den Nutzdaten besteht. In der Organisationsstruktur findet sich eine Sequenznummer und ein sogenannter Empfangsbitvektor. In diesem Empfangsbitvektor ist für jeden Client ein Bit reserviert, in welchem dieser den erfolg-

reichen Empfang eines Pakets eintragen kann. Trägt das Bit den Wert 1, so konnte der Client das Paket empfangen; ist der Wert des Bits hingegen 0, so ist der Empfang mißlungen.

Der Server erzeugt einen Empfangsbitvektor, der so groß ist, daß für jeden Client ein Bit vorhanden ist. Wenn der Server ein Datenpaket ausschickt, so werden alle Bits des Empfangsbitvektors mit 0 initialisiert, da ja noch kein Client dieses Paket empfangen haben kann.

Das Paket wird über den Ring auf die Reise geschickt. Es gelangt zum ersten Client. Ist dieser Client beschäftigt, so wird das Paket unverändert weitergereicht. Dementsprechend hat sich auch nichts am Empfangsbitvektor geändert, weshalb für diesen Client das Bit auf 0 bleibt. Hat der Client jedoch Zeit, das Paket zu verarbeiten, so soll dieser versuchen, das Paket in einen lokalen Puffer zu kopieren. Dieser lokale Puffer wird vom Transport Layer zur Verfügung gestellt und wird von der Anwendung, die den Reliable Multicast verwendet, ausgelesen. Gelingt es dem Client, das Paket im lokalen Transportpuffer abzulegen, so soll beim Empfangsbitvektor des ausgehenden Datenpakets für das entsprechende Empfangsbit 1 eingetragen werden. Konnte das Paket im lokalen Transportpuffer nicht abgelegt werden, so bleibt das Bit des entsprechenden Clients auf 0.

Damit dieser Mechanismus funktioniert, muß der Ring folgende Bedingungen erfüllen:

1. Die Pakete müssen gemäß *Store-and-Forward* [17] behandelt werden
2. Das Umschreiben der Pakete durch das Reliable Multicast Protokoll muß möglich sein

Daraus ergeben sich aber Probleme. Ein *Store-and-Forward*-Betrieb erhöht die Latenzzeit, da ja jedes Paket zuersteinmal zwischengespeichert wird, danach folgt die Verarbeitung und am Schluß wird das Paket erst weitergeschickt. Beim Original IBM Token Ring ist dies gar nicht möglich, da jede Node nur einen 1 Bit Speicher hat, um die momentanen Daten des Ringes zu halten. Es wird ein Bit empfangen, dieses wird im lokalen Speicher der Netzwerkkarte abgelegt, auf diesem Speicher folgt die Verarbeitung, und sobald das nächste Bit empfangen wird, muß das Bit aus dem Speicher bereits wieder

auf die Reise geschickt worden sein. In diesem Fall würde die Station also den Empfangsbitvektor einfach unverändert weiterschicken, weil sie ja die Daten noch gar nicht empfangen hat und demnach nicht entscheiden kann, ob diese im lokalen Transportpuffer abgelegt werden konnten. Folgende Erweiterung würde hier Abhilfe schaffen. Wenn vor dem Empfangsbitvektor noch ein Datenfeld vorhanden ist, in welchem die Größe der Nutzdaten abgelegt wurde, so kann der Client im Voraus entscheiden, ob er genug Platz im lokalen Transportpuffer für dieses Paket hat. Er könnte also im Voraus den erfolgreichen Empfang eintragen, obwohl die Daten noch gar nicht empfangen wurden. Falls während des Kopierens der Nutzdaten in den Transportpuffer ein Fehler passiert, so hätte der Client aber schon den erfolgreichen Empfang bestätigt. Mögliche Lösungen dieser Probleme wären, den Empfangsbitvektor zum Beispiel hinter den Nutzdaten zu platzieren, oder nach den Nutzdaten ein Prüfsummenfeld einzurichten, welches der Client im Extremfall noch zerstören kann, um das Paket ungültig zu machen und so dem Server zu signalisieren, daß die Übertragung dieses Pakets nicht fehlerlos stattgefunden hat.

Ein weiteres Problem ergibt sich aus der zweiten geforderten Bedingung. Wenn jede Node das Paket am Ring beliebig umschreiben kann, so könnte der Empfangsbitvektor auch mutwillig oder aufgrund eines Fehlers zerstört werden, und so könnten falsche Aussagen über den Erfolg der Übertragung entstehen.

Dieses Verfahren hat ein Scalability-Problem in Hinblick auf den Speicherbedarf. Da für jeden Client ein Bit im Datenpaket vorgesehen werden muß, ist die maximale Anzahl von Clients durch die maximale Paketgröße beschränkt.

### 5.3.2 Reliable Multicast mit ACK Strategie

Hier wird das Prinzip des PAR-Protokolls als Vorbild genommen. Die Rollen von Server und Client sind jedoch anders belegt als bei der Vorstellung des PAR-Protokolls. Der Server sendet ein Datenpaket an alle Clients. Danach erwartet der Server von allen Clients eine positive Bestätigung über den Empfang dieses Datenpakets. Eine solche Lösung erweist sich jedoch als problematisch in Hinsicht auf Scalability, da bei einer

Anzahl von n Clients für ein Datenpaket des Servers auch n Bestätigungen befürchtet werden. Bei hohen Werten für n kommt es zu einem Effekt, der in der Literatur als *ACK implosion* [28] bezeichnet wird. Weil der Server für das Aussenden eines Datenpakets auch n Bestätigungen verarbeiten muß, ist die Anzahl n der teilnehmenden Clients durch die Kapazitäten von Kommunikationsmedium und Server stark begrenzt.

### 5.3.3 Reliable Multicast mit NACK Strategie [28]

Reliable Multicast Protokolle mit NACK Strategie haben meist eine bessere Leistung und sind skalierbarer als Reliable Multicast Protokolle mit ACK. Es gibt jedoch einige Probleme, die bei einer NACK Strategie auftreten. Zunächst weiß der Server nicht, wann das NACK eines Clients kommt. So könnte es passieren, daß ein Client für eine längere Zeit ausfällt und somit auch keine NACKs an den Server schickt. Sobald jener Client aber wieder am Multicast teilnimmt, bemerkt er, daß er schon einige Sequenzen versäumt hat. Er schickt ein NACK an den Server, der nun einen Teil der Übertragung wiederholen muß. Zusätzlich zu einer starken Verzögerung des gesamten Übertragungsvorganges muß der Server nun auch noch alle Datenpakete in einem Puffer gespeichert haben, um die Übertragung wieder von dort beginnen zu können, wo jener Client aufgehört hat, den Paketstrom mitzuschreiben.

In der Praxis entfernt der Server nach einer gewissen Zeit, welche geschätzt wird, die ältesten Pakete aus dem Puffer, wie es zum Beispiel bei MTP passiert. Solche Schätzungen können aber, ähnlich wie beim Timeout des PAR-Protokolls, danebengehen. Wenn das NACK eines Clients erst nach Ablauf dieser Lebenszeit für Daten eintrifft, so muß die Übertragung für diesen Client abgebrochen werden, weil sie unbrauchbar geworden ist.

### 5.3.4 Server-controlled Reliable Multicast

Der Vorteil eines server-controlled Ansatzes ist die leichtere Überschaubarkeit des Protokollablaufs und der einzelnen Protokollphasen. Die a-priori Analyse der möglichen Fälle

wird wesentlich einfacher als es zum Beispiel bei einem client-controlled Ansatz wäre, wie er im nächsten Abschnitt beschrieben wird.

Wesentlicher Nachteil dieses Ansatzes ist jedoch die Tatsache, daß der Server eben aufgrund seiner zentralen Aufgaben auch zum *single point of failure* [17] wird. Die negative Auswirkung dieses Nachteils reduziert sich aber wesentlich, wenn der Server aufgrund des Anwendungsfalles sowieso den single point of failure darstellt, wie es zum Beispiel beim 1:n Kommunkationsszenario ist, wie es in Abschnitt 3.11.2 dargestellt wird. In diesem Fall stehen ja nur dem Server die zu verteilenden Daten zur Verfügung, weshalb ein schwerwiegender Fehler auf Seiten des Servers immer zu einem totalen Versagen des Ablaufs führt.

### 5.3.5 Client-controlled Reliable Multicast

Der große Vorteil dieses Ansatzes ist, daß höhere Ausfallssicherheit besteht, weil es a-priori keinen single point of failure gibt. Da ja alle Clients an der Steuerung und Kontrolle beteiligt sind, ist der Ausfall eines Clients für den Rest der Gruppe nicht sonderlich problematisch.

Der große Nachteil dieses Ansatzes ist jedoch, daß ein solches Protokoll schwer zu entwickeln und zu optimieren ist. In der Planungsphase einer distributed control müssen alle möglichen Zustände, welche die gesamte Gruppe einnehmen kann, berücksichtigt werden. Dabei nimmt aber im Sinne des Kriteriums Scalability, wie es in Abschnitt 3.13.3 beschrieben wird, die Anzahl der Clients wesentlichen Einfluß auf die Komplexität der Planung. Es wird demnach auch schwer möglich sein, für eine große Anzahl von Clients gleiche Garantien wie für einen Minderlastfall zu gewährleisten. Das Verhalten der Clients muß bei so einem Ansatz auch stark reglementiert werden, um die Aktionen der Clients koordinieren zu können. Durch diese starke Einschränkung kommt es wiederum meist zu Leistungseinbußen.

Bei einer großen Anzahl von Clients wird der Übertragungsprozeß schließlich unberechenbar, gerade, wenn zusätzliche Faktoren wie die Verwendung von Ethernet eine Rolle spielen.

# 6 Prototyp eines adaptierten Reliable Multicast Protokolls

Im folgenden Textteil werden die unterschiedlichen Merkmale jenes Protokolls beschrieben, welches im Zuge dieser Arbeit entwickelt wurde. Die Merkmale ergeben sich aufgrund der Anforderungen. Diese Merkmale grenzen aber auch die Fähigkeiten des Protokolls ab, teilweise aufgrund des beabsichtigten Einsatzgebietes, aber auch, um die Komplexität zu reduzieren. Das Protokoll ist für den Einsatz der Übertragung eines Festplattenimages für gleichartige Clients gedacht.

## 6.1 Einleitung zu RMARS

Die Abkürzung RMARS steht für Reliable Multicast with Acknowledgments Requested by the Server, ein Protokoll, das im Zuge dieser Arbeit entwickelt und in diesem Kapitel definiert wird.

## 6.2 Eigenschaften von RMARS

RMARS ist ein server-controlled (3.12.2) Protokoll, das für ein 1:n Kommunikationsszenario (3.11.2) konzipiert ist; es handelt sich um den Anwendungsfall der Datendistribution für Nicht-Echtzeitdaten (5.1.2). Es wird ein reliable transport (3.12.1) zur Verfügung gestellt, zur Bestätigung von empfangenen Daten wird ein modifizierter ACK Ansatz (3.12.3) verwendet. Um ACK implosions (5.3.2) zu vermeiden, kontrolliert der Server,

ob und wann bestimmte Clients ein ACK verschicken sollen. Fehlerhafte Clients werden durch den Server erkannt und von der Kommunikation ausgeschlossen.

Das Ziel von Optimierungen dieses Protokolls ist das Erreichen eines hohen Datendurchsatzes, erwünscht wäre auch die Garantie einer maximalen Latenzzeit.

Es wurde für einen server-controlled Ansatz entschieden, weil der client-controlled Ansatz komplexe Analysen erfordert und letztlich nicht als scalable erscheint.

Das Prinzip des server-requested ACK verdient besondere Aufmerksamkeit, weil es ACK implosions verhindern soll. Server-requested ACK bedeutet, daß Clients nicht von sich aus entscheiden, wann sie ACK Kontrollpakete schicken. Ein bestimmter Client verschickt ACK Kontrollpakete *nur* dann, wenn der Server es von genau diesem verlangt. Der Client wird also gezielt angesprochen.

## 6.3 Voraussetzungen für RMARS

Das Protokoll benötigt einen Unicast Dienst, um Kontrollpakete von Client zu Server und umgekehrt zu schicken. Der Server braucht zusätzlich einen Broadcast- oder Multicastdienst [28], um Datenpakete an die Gruppe zu übertragen.

Serverseitig wird ein Puffer benötigt, der als Ringpuffer (circular buffer [25]) betrieben wird und Pakete für die Retransmission speichert. Die Clients benötigen keinen Puffer.

## 6.4 Definitionen zu RMARS

Das RMARS Protokoll hat bestimmte Eigenschaften, um Reliable Multicast zu realisieren, die nun festgelegt werden sollen.

### 6.4.1 Strategie

Die Strategie, die ja Teil des Protokolls ist, wurde im vorhergehenden Abschnitt bereits angedeutet.

Clients verhalten sich prinzipiell ruhig. Sie warten auf Nutzdatenpakete vom Server und verarbeiten diese. Im speziellen prüfen Clients die Sequenznummer des ankommen-

den Pakets. Wird die Sequenznummer, die im Paket eingetragen ist, vom Client als nächste erwartet, so werden die Nutzdaten dieses Pakets ausgegeben. Der Client berechnet nun die Sequenznummer, die er als nächstes erwartet. Erhält der Client ein Nutzdatenpaket mit einer Sequenznummer, die ungleich der von ihm erwarteten ist, so ignoriert der Client dieses Paket.

Der Server fordert ACKs gezielt jeweils von einem bestimmten Client an. Er ist damit Herzschlag und zentrale Intelligenz und muß sich daher auch um Aufgaben wie Flußkontrolle und Überlastkontrolle kümmern. Dies ist dem Server aber auch vernünftig möglich, weil ihm aufgrund seiner zentralen Position die für Entscheidungen von Flußkontrolle und Überlastkontrolle notwendigen Daten zur Verfügung stehen. [27] Zum Beispiel kann der Server die Umlaufzeiten von Paketen zu den einzelnen Clients messen und individuell darauf reagieren.

Der Server arbeitet nach folgendem Prinzip. Er legt einen Ringpuffer an. Dieser Ringpuffer wird von drei unterschiedlichen Instanzen bearbeitet: Datenleser, Datensender und Datenlöscher. Der Datenleser liest von der Datenquelle und legt die gelesenen Daten im Ringpuffer ab, wenn dafür Platz ist. Der Datensender verschickt Daten, die im Ringpuffer liegen und nicht als löschbar markiert sind. Der Datenlöscher markiert Daten im Ringpuffer als löschbar, wenn diese bereits von allen Clients bestätigt wurden.

Das Zusammenwirken dieser drei Instanzen kann am besten durch den Vergleich mit drei Läufern dargestellt werden, die im Kreis laufen. Der Kreis ist der Serverringpuffer. Die Läufer sind die drei Instanzen Datenleser, Datensender und Datenlöscher. Der Datenleser läuft als erster los, Datensender als zweiter und Datenlöscher als dritter.

Der Leser wird den Löscher nicht überholen, da ja sonst Daten überschrieben würden, die noch nicht von allen Clients bestätigt und somit noch nicht als löschbar markiert wurden.

Der Datensender läuft vorerst unabhängig von den beiden anderen.

Unterschiedliche Ereignisse lassen sich benennen, deren Auftreten damit zusammenfällt, wenn Läufer einander überholen oder auf die Fersen rücken.

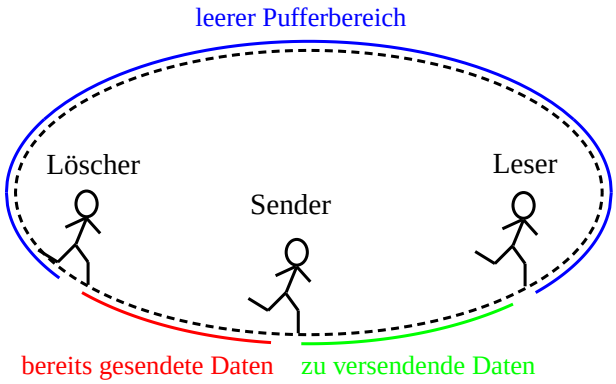


Abbildung 6.1: Die drei Läufer im Normalbetrieb

Für die Analysen sei die Reihenfolge der Läufer so, wie diese gestartet sind, also Leser der erste, danach Sender und am Schluß der Löscher, wie in Abbildung 6.1 ersichtlich.

Wenn der Sender den Leser überholt, so beginnt ein sogenannter server buffer loop (siehe 6.4.2). Vor dem Leser befinden sich nämlich ausschließlich Daten, die bereits ausgesendet wurden und welche nun erneut gesendet werden. Es kann auch passieren, daß Pufferspeicher zwischen Leser und Löscher frei ist. Da der Sender aber leeren Pufferspeicher im Schnelldurchlauf überstreicht, überholt er sofort auch den Löscher. Deshalb wird als Ereignis, welches einen server buffer loop auslöst, jene Situation genannt, in welcher der Sender den Leser überholt. (Abbildung 6.2)

Es kann passieren, daß der Löscher den Sender überholt, zum Beispiel wäre dies der Fall, wenn der Sender gerade einen server buffer loop ausführt, die ACKs der Clients aber verzögert ankommen und daher der Löscher alle bereits bestätigten Daten entfernt. Der Löscher bleibt beim ersten Paket stehen, das noch nicht von allen Clients bestätigt wurde, weil es zum Beispiel noch gar nicht versendet oder von einigen Clients noch nicht empfangen wurde. Implizit wartet der Löscher so lange, bis der Sender ihn wieder überholt hat. Da der Sender aber nur Daten aussendet, die nicht als löschbar markiert

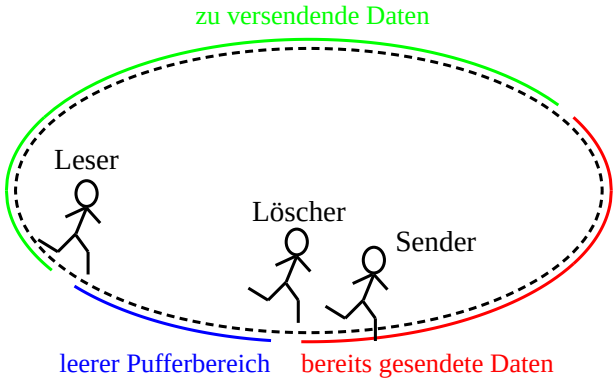


Abbildung 6.2: Server buffer loop: Sender überholt Leser

sind, läuft er sehr schnell bis zum Löscher, um diesen bei seinem Standplatz wieder zu überholen. (Abbildung 6.3)

Wenn aber der Datenlöscher unmittelbar hinter dem Datenleser ist und der Datenleser stillsteht, so darf auch der Sender auf dem selben Platz wie der Leser stehenbleiben. Das wäre nämlich der Fall, wenn schon alle Daten fehlerfrei an alle Clients geschickt wurden und die Quelle im Moment keine weiteren Daten anbietet. (Abbildung 6.4)

### 6.4.2 Unproduktive Pufferdurchläufe

Als unproduktiven Pufferdurchlauf bezeichnet man bei RMARS einen Lauf des Senders durch den Ringpuffer, in welchem keine neuen Pakete zum Versand eingetragen werden. Es werden in solchen Runden also ausschließlich Pakete verschickt, die schon mindestens einmal an die Gruppe versandt wurden. Der Puffer muß ganz durchlaufen werden, weil Clients ja Pakete mit Sequenznummern, die ungleich der von ihnen erwarteten sind, nicht zwischenspeichern. Daher muß ein Client alle Pakete vom ersten fehlenden bis zum aktuellen erneut zugesendet bekommen. Unproduktive Pufferdurchläufe werden auch als *server buffer loops* bezeichnet.

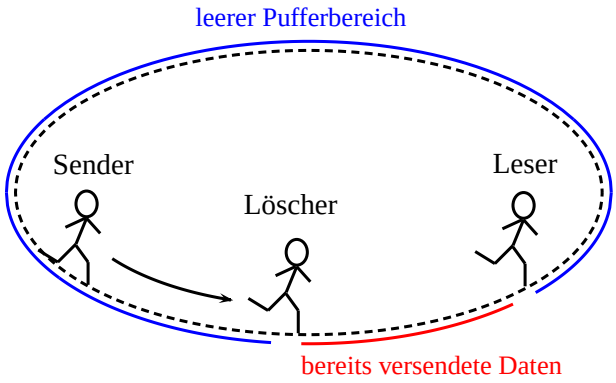


Abbildung 6.3: Verspätete ACKs: Löscher hat Sender überholt

### 6.4.3 ACK Selection Algorithmus

Die Datensenderinstanz, welche Pakete aus dem Ringpuffer an die Gruppe verschickt, bestimmt vor dem Verschicken eines Datenpakets, welcher Client zum ACK aufgefordert wird. Die Entscheidung, welcher Client das ist, wird vom *ACK Selection Algorithmus* getroffen.

### 6.4.4 Runden

Die Größe des Serverringpuffers wird in sogenannten Runden gemessen. Eine Puffergröße von einer Runde entspricht soviel Speicher, daß für jeden Client ein Paket im Puffer abgelegt werden kann. Die Anzahl der Pakete in einer Runde ist also gleich der Anzahl der Clients, die an der Kommunikation teilnehmen.

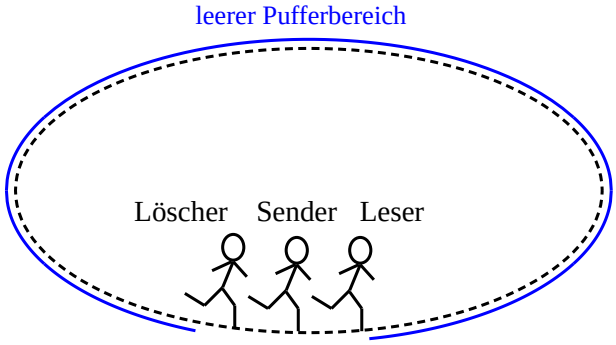


Abbildung 6.4: Alles erfolgreich verschickt, Datenquelle blockt: Sender wartet gemeinsam mit dem Leser

## 6.5 Protokollablauf

### 6.5.1 Kurzbeschreibung

Der Protokollablauf geschieht in drei verschiedenen Phasen.

1. Phase I: Grouping
2. Phase II: Transmit Data
3. Phase III: Final

Der Protokollablauf beginnt mit Phase I, Grouping. Dabei registrieren sich die einzelnen Clients beim Server für eine Session. Nach Abschluß der Registrierung wechselt das Protokoll zu Phase II, Transmit Data.

Die Clients sind sehr einfach aufgebaut. Sie empfangen Datenpakete vom Server und antworten auf Bestätigungsanforderungen, die der Server mitteilt. Sobald die Anzahl der Nutzbytes im Datenpaket Null ist, wechseln Clients zu Phase III, Final.

Der Server ist das Herzstück in diesem Protokoll, er hat zentrale Steuerungs- und Organisationsaufgaben. Er bestimmt die Geschwindigkeit, mit welcher Datenpakete ausgesandt und wann von welchem Client Bestätigungen angefordert werden.

Die nachfolgenden Abbildungen stellen die Zustandsdiagramme von Server und Client dar.

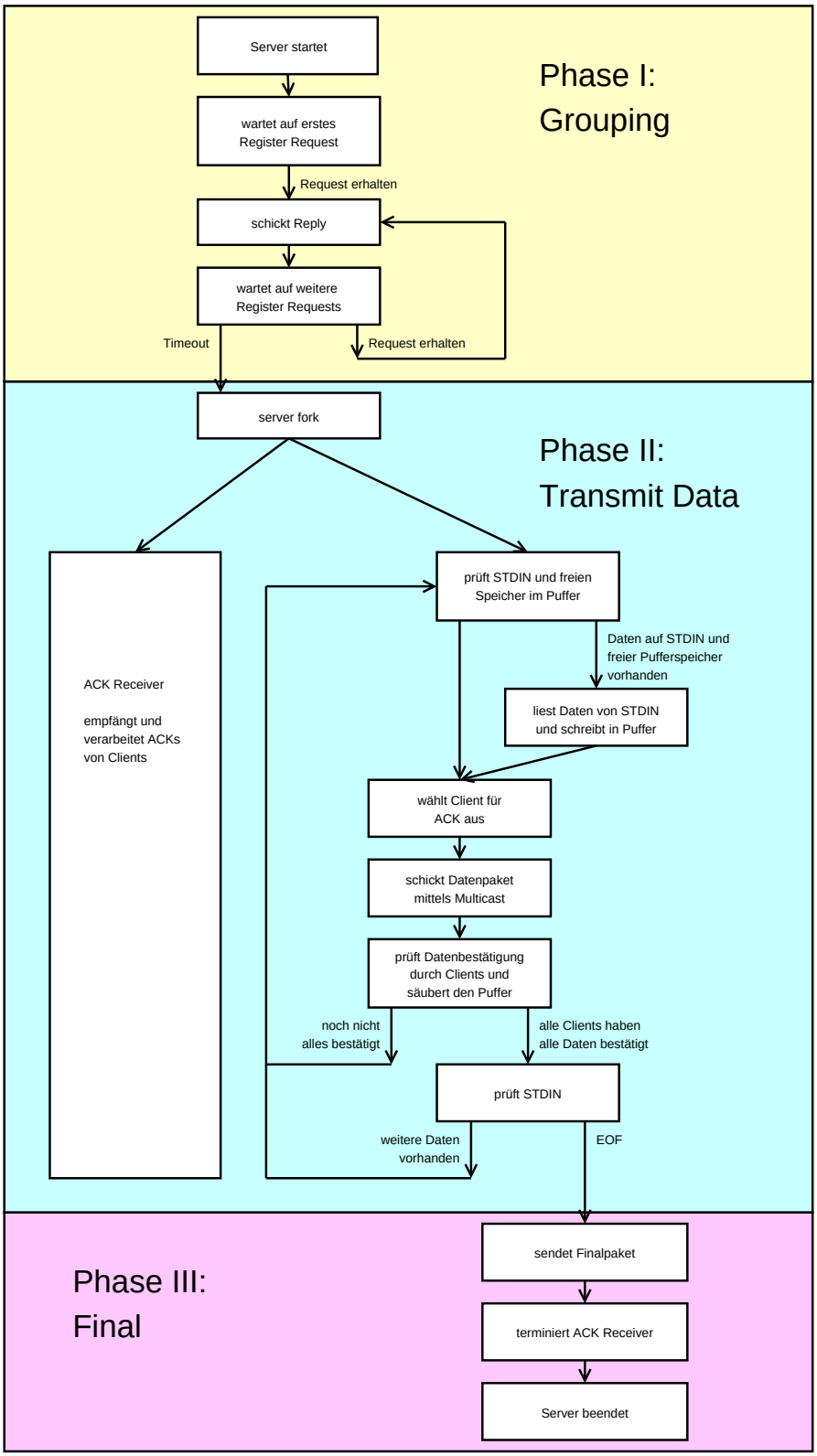


Abbildung 6.5: Zustandsdiagramm des Servers

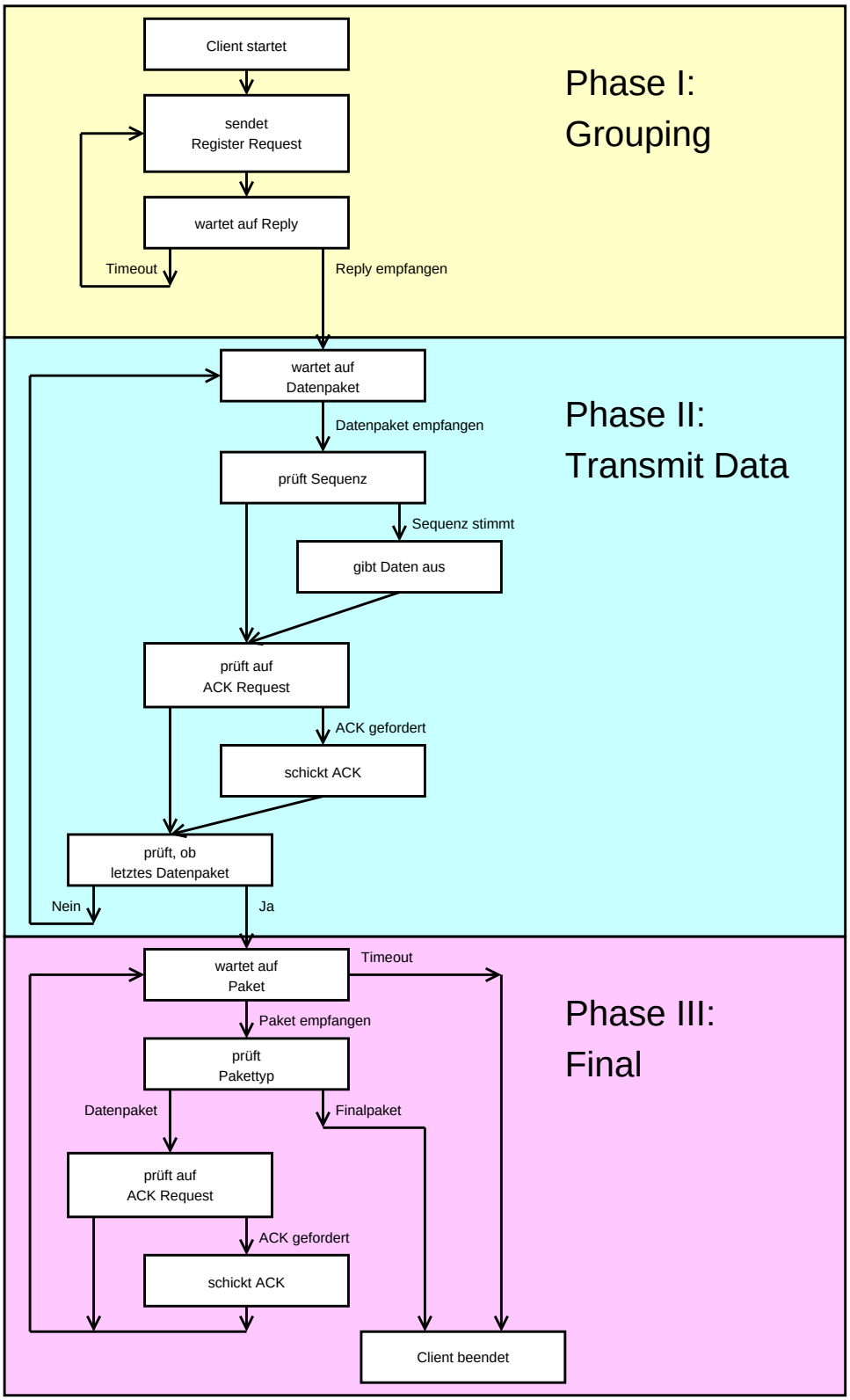


Abbildung 6.6: Zustandsdiagramm des Clients

### 6.5.2 Ausführliche Beschreibung

#### Phase I: Grouping

Die Initialsituation hat folgende Gestalt. Dem Server steht der Datenstrom zur Verfügung. Es wird aber noch nicht von diesem Datenstrom gelesen. Der Server befindet sich in Phase I, Grouping. Er wartet auf die erste Anfrage eines Clients, welcher den Datenstrom bekommen möchte. Bei Empfang einer solchen Anfrage beginnt eine Session. Der Server erteilt dem Client eine *Client Number* und identifiziert ihn dadurch eindeutig für diese Session. Der Server wartet nun eine Zeit, ein Timeout, ab. Wird innerhalb dieses Timeouts eine Anfrage eines weiteren Clients vom Server empfangen, so wird auch dieser Client registriert und der Ablauf des Timeout beginnt von neuem. Wirkt das Timeout, weil sich kein weiterer Client registrieren wollte, so wechselt der Server zu Phase II, Transmit Data.

#### Phase II: Transmit Data

In dieser Phase ist der Server sozusagen der Herzschlag des gesamten Mechanismus. Der Datenstrom wird in Datenpakete aufgeteilt. Diese Pakete erhalten für jene Session eindeutige Sequenznummern. Die Pakete werden mittels Broadcast oder Multicast an alle Clients verschickt. Der Server bestimmt die Übertragungsgeschwindigkeit des Datenstroms. Er muß jedoch auch darauf achten, daß die einzelnen Clients alle Daten mitbekommen, und er muß entsprechend Daten erneut senden, welche noch nicht bei allen Clients angekommen sind. Der Server kann in den Datenpaketen, welche er an alle Clients schickt, von einzelnen Clients Bestätigungen anfordern. Pro Datenpaket kann der Server aber nur von einem Client eine Bestätigung anfordern. Dazu wird in so einem Paket das ACK REQUESTED Flag gesetzt und für die Client Number die Identifikation des entsprechenden Clients in jener Session eingetragen. Der angesprochene Client übermittelt nun ein ACK Paket an den Server, in welchem der Client die Sequenznummer des letzten Datenpakets eines durchgängigen Stromes einträgt, den er empfangen hat. Der Sinn dieses sehr einfachen Mechanismus ist, die Komplexität des Clients möglichst gering

zu halten. Denn die Komplexität des Clients wirkt sich n-fach aus, wenn n Clients am Protokoll teilnehmen. Wenn Clients viele verschiedene Programmzustände einnehmen können, so gibt es bei n Clients umsomehr Kombinationsmöglichkeiten der Zustände einzelner Clients.

Der Server kann aber aufgrund seiner zentralen Stellung auch eine Anpassung der Übertragungsgeschwindigkeit durchführen, zum Beispiel aufgrund der Häufigkeit fehlender Pakete bei den Clients. So wäre zum Beispiel die Geschwindigkeit zu verringern, wenn häufig Pakete fehlen, oder aber auch probeweise zu erhöhen, wenn über längere Zeit keine Pakete verlorengegangen sind, um die schnellstmögliche Übertragung zu gewährleisten.

Die Sequenznummern der Datenpakete sind eine Folge, welche die Menge der natürlichen Zahlen, aufsteigend angeordnet, als Folgeglieder besitzt. Das erste Folgeglied ist 1.

In den Datenpaketen wird auch gespeichert, wie viele Bytes der Nutzdaten des Pakets gültig sind. Liefert der Datenstrom EOF oder wird das Lesen vom Datenstrom abgebrochen, so generiert der Server ein Datenpaket, welches die nächste Sequenznummer hat, aber in der Anzahl der Nutzdaten Null eingetragen hat. Es wird also von den Clients gefordert, daß auch dieses Datenpaket, wie alle anderen auch, von allen Clients empfangen und letztlich bestätigt wird. Der Server wechselt danach in Phase II-III, in welcher schließlich die Beendigung des Protokolls versucht wird. Phase II-III ist eine Übergangsphase zwischen Phase II Transmit Data und Phase III Final, da ja zum Zeitpunkt des Abbruchs oder des Auftretens von EOF im Datenstrom noch nicht garantiert werden kann, daß alle Clients schon all jene Datenpakete erhalten haben, die tatsächlich noch Nutzdaten des Datenstroms beinhalten.

Erhält ein Client ein Datenpaket mit einer Anzahl von Null Bytes an Nutzdaten, so wechselt er intern zu Phase III. Der Server befindet sich zu diesem Zeitpunkt in der Übergangsphase II-III. Clients, welche sich in Phase III befinden, bestätigen Datenpakete des Servers, wenn es von ihnen gefordert wird. Sie tun dies solange, bis sie vom Server ein FIN_ACK Paket empfangen, oder wenn sie innerhalb eines Timeouts überhaupt kein

Datenpaket mehr vom Server empfangen haben. Danach gilt die Session für den Client als beendet.

Dieser Mechanismus beugt folgendem Fall vor: Angenommen, der Server befindet sich in Phase II-III, hat also schon alle Datenpakete mit Nutzdaten mindestens einmal ausgesandt. Nun gäbe es zwei Gruppen von Clients. Jene, welche sich schon in Phase III befinden, weil sie schon alle Pakete in der entsprechenden Reihenfolge empfangen haben, und jene, welche sich noch in Phase II befinden, weil ihnen Pakete verlorengegangen sind. Clients in Phase III haben ja das Timeout, das zurückgesetzt wird, wenn sie Datenpakete vom Server erhalten. In dieser Phase III müssen Clients ja nur mehr Bestätigungen ausschicken, wenn es von ihnen gefordert wird, aber nicht mehr Daten mitschreiben oder ausgeben. Der Server überträgt also gemäß Phase II-III noch Datenpakete an jene Clients, welche noch nicht alle Pakete erhalten haben. Durch diese Übertragung werden jene Clients, welche bereits in Phase III sind, sozusagen hingehalten.

#### Phase III: Final

Der Server wechselt zu Phase III, wenn alle Clients das Paket mit der letzten Sequenznummer bestätigt haben. Jenes Paket hat als Anzahl der Nutzdaten Null Bytes. So kann der Server feststellen, ob jeder Client jener Session alle Datenpakete erhalten hat. In Phase III Final schickt der Server an alle Clients mittels Broadcast oder Multicast ein FIN_ACK Paket, welches dem Client das Ende der Session bekanntgibt. Dieses Paket bleibt unbestätigt, sodaß der Server nach Verschicken des FIN_ACK Pakets die Session beendet. Clients sind nicht unbedingt abhängig vom erfolgreichen Empfang des FIN_ACK Pakets, weil die Clients ja mittels Timeout auch für sich selbst die Session beenden können.

## 6.6 Bewertung der vorgestellten Variante

Die durch die vorhergehenden Kapitel definierte Protokollidee soll nun von unterschiedlichen Gesichtspunkten betrachtet werden, um eine Bewertung durchführen zu können.

### 6.6.1 Fähigkeiten

Der Server kennt aufgrund der Gruppenbildungsphase alle Teilnehmer an der Kommunikation. Er kann daher feststellen, ob die Übertragung für alle Clients funktioniert hat. RMARS ist nur für lokale Netze gedacht, die sich nur über ein physikalisches Segment erstrecken. Um RMARS in größeren Netzen sinnvoll einsetzen zu können, wird eine Erweiterung vorgestellt, die in Abschnitt 6.7.1 behandelt wird.

Jene Clients, die sich beim Server erfolgreich registrieren konnten, werden auch versorgt.

### 6.6.2 Betrachtungen und Probleme

#### Speicherbedarf am Server

Das größte Problem bei RMARS ist ein Problem der Scalability in Hinsicht des Speicherplatzes: der Server braucht mehr Speicherplatz, um mehr Clients bedienen zu können. Der Speicherplatzbedarf pro Client ist aber sehr gering, weshalb auch große Anzahlen von Clients leicht betreut werden könnten. In der Prototypimplementierung bedarf jeder weitere Client zusätzlichen Speichers von 2 Kilobyte. Der Speicheraufwand ist also linear. Die Größe des serverseitigen Ringpuffers ist bei kleinen Anzahlen von Clients verhältnismäßig groß, weil ja die Verzögerung der ACKs kompensiert werden muß. Daher wird der Puffer, abhängig von der Verzögerungszeit der ACKs, meist auf mehrere Runden ausgelegt, im Prototyp (siehe Abschnitt 6.8) für vier. Bei großen Teilnehmerzahlen muß der Puffer jedoch nur Platz für eine Runde haben, da ja die Runde länger geworden ist und daher die ACKs auch schon eingetroffen sind, bis der Puffer wieder von vorne bearbeitet wird.

#### ACK Mechanismus

Durchschnittlich geht sich die Belastung des Servers durch clientseitige ACKs aus. Wenn aber die ACKs einzelner Clients verzögert ankommen, könnte es auch zu Ansammlungen von ACKs kommen, die aber nicht als ACK implosions bezeichnet werden sollten, da es

sich um einen wesentlich kleineren Rahmen und eigentlich um einen grundverschiedenen Sachverhalt handelt. Da einzelne Clients ja nicht von sich aus ACKs senden, sondern nur, wenn der Server es von ihnen gezielt anfordert, kommt es bei Überlastung des Servers zu keinen weiteren Kontrollnachrichten von Seiten der Clients. Es kann also nicht passieren, daß eine eskalierende Situation eintritt, d.h. der Server und auch das Netzwerk können sich von Lastspitzen erholen und die Übertragung kann ohne großen Schaden fortgesetzt werden. Das Protokoll realisiert also implizit auch Überlastkontrolle.

#### Finalisierung von Clients

Gemäß der Vereinbarung im Protokoll werden Clients in Phase III solange hingehalten, bis alle Clients der Session das Protokoll durchgemacht haben. Alle Clients einer Session schließen im Standardfall gleichzeitig ab, nämlich wenn der Server das FIN_ACK Paket verschickt. Lediglich Clients, die dieses Finalpaket nicht erhalten, werden durch ihr lokales Timeout finalisiert. Clients, welche die Übertragung als erste fertig haben, werden demnach unnötig verzögert.

## 6.7 Erweiterungsmöglichkeiten

Folgende Ideen stellen Erweiterungen des RMARS Protokolls dar, welche in Implementierungen aber bisher noch keine Verwendung gefunden haben.

### 6.7.1 Verwendung von RMARS über lokale Netze hinaus

Eine starke Einschränkung in der Spezifikation von RMARS ist die Tatsache, daß das Protokoll nur für den Einsatz in lokalen Netzen oder kleinen Netzen gedacht ist. Letztlich wird wohl der langsamste Client der gesamten Gruppe beeinflußen, ob der Einsatz von RMARS für den Anwendungsfall noch brauchbar ist. Wie langsam ein Client ist, hängt einerseits von dessen Verarbeitungsgeschwindigkeit ab, auf der anderen Seite aber auch von der Zeit, die ein Datenpaket von Server zu Client benötigt.

Als Abhilfe für diesen Mangel wird ein Prinzip vorgeschlagen, das hier als *subtreeing* bezeichnet wird. Die Idee für dieses Prinzip ist [6] entnommen.

Dabei wird ein Router gleichzeitig zu einem RMARS Client und zu einem RMARS Server. Der RMARS Client auf dem Router empfängt die Daten und übergibt diese gleich an seinen lokalen RMARS Server, welcher nun das vom Router versorgte Subnetz betreut.

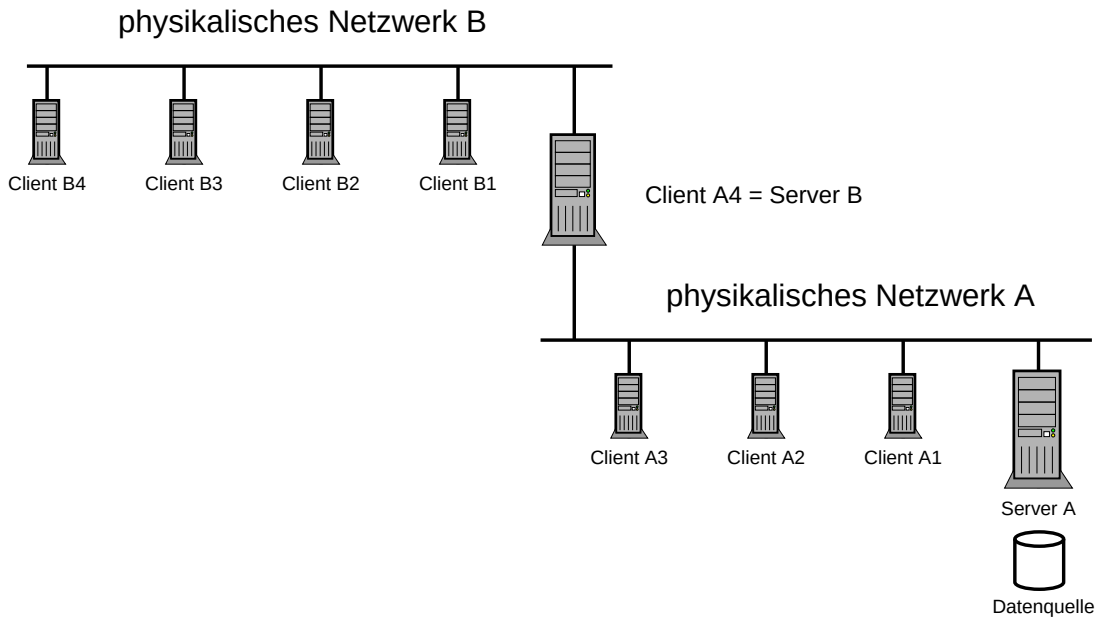


Abbildung 6.7: Subtreeing mit RMARS

Die Implementierung von RMARS, wie sie in Abschnitt 6.8 beschrieben wird, ist direkt für diese Maßnahme einsetzbar, weil der Server von STDIN liest und der Client auf STDOUT schreibt. Es kann also der Datenstrom, der vom Client generiert wird, sofort dem Server für das betreute Subnetz zugeführt werden.

Betreut ein Router mehrere Subnetze, so muß der Datenstrom am Router vervielfacht werden. Für jedes vom Router betreute Subnetz kann so ein RMARS Server gestartet werden, der das jeweilige Subnetz versorgt.

### 6.7.2 Flußkontrolle

Für die Flußkontrolle im RMARS wird folgendes Prinzip vorgeschlagen. Der Server kann für jeden Client messen, wie lange es dauert, bis auf einen ACK Request geantwortet wird. Dadurch hat der Server für jeden Client ein ungefähres Maß, wie schnell der Client an der Kommunikation teilnehmen kann. In diesem Maß sind viele Parameter bereits inkludiert, nämlich die Geschwindigkeit des Kommunikationskanals zwischen Server und diesem speziellen Client, wie auch die Verarbeitungsgeschwindigkeit des Clients selbst.

Der Server muß die Übertragungsgeschwindigkeit dem langsamsten Mitglied der Gruppe anpassen. Dies ist gemäß der Vereinbarung in der Strategie des Protokolls situationsbedingt. Registriert der Server nun häufig Paketverluste bei einem Client, so muß Überlastung dieses Clients angenommen und daher die Übertragungsgeschwindigkeit gedrosselt werden. Läuft jedoch die Kommunikation ohne Paketverluste ab, so muß der Server testweise die Geschwindigkeit erhöhen, um den optimalen Durchsatz für diese Gruppe zu erreichen. Außerdem muß diese Regelungen permanent, aber mit einer gewissen Verzögerung, durchgeführt werden, da sich ja die Zustände einzelner Clients ändern können, weil zum Beispiel am Anfang der Übertragung bei den Clients noch in einen Cache [18] geschrieben wird, bei größeren Datenmengen dieser Cache aber voll wird und daher nur noch mit der geringeren Transferrate der Festplatte gearbeitet werden kann. Auf solche Änderungen im Verhalten der Übertragungsgeschwindigkeiten muß das Protokoll reagieren können.

### 6.7.3 Adaptive Steuerung der Retransmission

Der entwickelte Prototyp verwendet einen sehr primitiven ACK Selection Algorithmus, bei dem alle Clients der Reihe nach durchgegangen werden. (siehe Abbildung 6.8)

Aus dem Vergleich mit den drei Läufern ist ersichtlich, daß ein server buffer loop passiert, sobald der Sender den Datenleser überholt. Die Retransmission wird also im Prototyp erst gestartet, wenn der Ringpuffer bis zum Ende durchgegangen wurde. Der dadurch entstehende Zeitverlust ist damit abhängig von der Puffergröße.

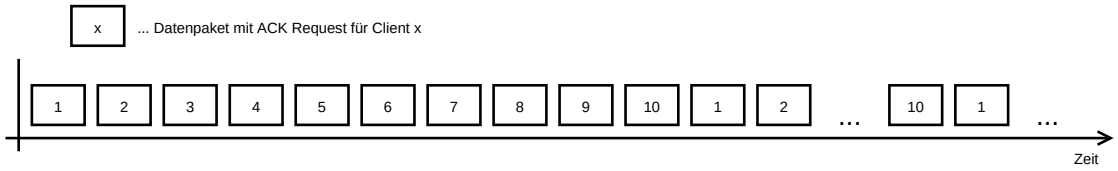


Abbildung 6.8: Primitiver ACK Selection Algorithmus

Bei der hier vorgestellten Erweiterung geht es um eine Modifikation des ACK Selection Algorithmus und um eine gezielte Form der Retransmission von Datenpaketen aus dem Serverpuffer. Das Ziel ist, daß der Server den Verlust eines Pakets bei einem der Clients möglichst rasch feststellen kann und sofort die Retransmission einleitet, die auch gleich bei dem verlorengegangen Paket beginnt, ohne daß der ganze Puffer einmal durchlaufen werden muß.

Auch könnte die Auswahl eines Clients durch den ACK Selection Algorithmus davon abhängig gemacht werden, wie lange dieser schon nicht mehr Bestätigungen ausgesendet hat, um einen fehlerhaften Client zum Beispiel schneller aus der Gruppe entfernen zu können.

Es gilt also, zwei Fälle zu behandeln: den Totalausfall eines Clients und den Paketverlust.

Jedenfalls muß berücksichtigt werden, daß die Antwort eines Clients mit einer gewissen Verzögerung beim Server ankommt. Diese Verzögerungen aller Clients können vom Server gemessen und als ungefähre Richtwerte für die Größenordnungen angesehen werden. Der Server führt also eine Liste, in welcher für jeden Client ein Schätzwert für die Verzögerung abgespeichert wird. Der Server errechnet für jeden Client ein Erwartungsfenster, in dessen Rahmen er die Antwort auf das ACK Request erhofft.

### Totalausfall eines Clients

Für den Totalausfall eines Clients möge diese Erweiterungsmöglichkeit durch ein Beispiel näher erläutert werden. An einer Session nehmen 10 Clients teil, der Serverringpuffer hat eine Größe von drei Runden, daher kann der Puffer 30 Datenpakete zwischenspeichern.

Ein primitiver ACK Selection Algorithmus, wie dieser auch im Prototypen verwendet wird, fragt alle Clients linear ab. Nach fünf vom Server ausgesendeten ACK Requests, die vom Client nichteinmal beantwortet wurden, wird ein Client vom Server aus der Session entfernt.

Sei es nun, daß alle Clients bis auf einen korrekt arbeiten. So werden bei linearer Abfrage insgesamt zwei server buffer loops ausgeführt, um den fehlerhaften Client entfernen zu können. Die Gesamtdauer der Gruppenübertragung wird unnötig verzögert.

Da aber in diesem Beispiel im Zuge eines Pufferdurchlaufs jeder Client insgesamt dreimal nach einer Bestätigung gefragt wird, weil der Puffer drei Runden groß ist, geschieht dies auch mit unserem fehlerhaften Teilnehmer. Kommt die Bestätigung beim Server nicht innerhalb des Erwartungsfensters an, so wird nun der intelligente ACK Selection Algorithmus den entsprechenden Client öfter pro Runde abfragen, zum Beispiel zweimal. Ein anderer Client wird in jener Runde dafür ausgelassen.

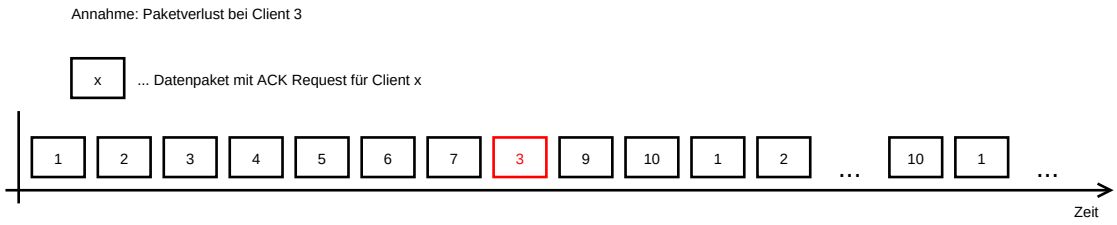


Abbildung 6.9: Intelligenter ACK Selection Algorithmus

So kann also innerhalb eines Pufferdurchlaufs, der drei Runden beträgt, ein Client insgesamt fünfmal abgefragt werden. Bleiben alle diese fünf ACK Requests unbeantwortet, so wird der Server den entsprechenden Client bereits ausschließen und somit einen server buffer loop vermeiden.

### Paketverlust

Paketverlust kann auf zwei Wegen passieren. Entweder kommt das Datenpaket, welches per Broadcast oder Multicast vom Server an die Clients geschickt wird, bei einem Client

niemals an, oder die Bestätigung, die ein Client gemäß einer Aufforderung an den Server zurückschickt, geht verloren.

Für beide Fälle kann analog wie beim Totalausfall des Clients vorgegangen werden. Der Unterschied ist, daß der Server ja feststellen kann, ob eine Bestätigung von dem entsprechenden Client empfangen wurde oder nicht. Wurde auf ein ACK Request hin keine Bestätigung empfangen, so kann es trotzdem noch beide Ursachen haben, also entweder Totalausfall oder einmaligen Paketverlust. Wird nach mehrmaliger Wiederholung des ACK Requests noch immer nicht geantwortet, so muß der Totalausfall angenommen werden, siehe vorhergehenden Abschnitt. Sind jedoch Antworten beim Server eingelangt, die aber keinen Fortschritt vermitteln, weil sie immer die gleiche Sequenznummer bestätigen, so hat es Paketverlust auf dem Weg von Server zu Client gegeben oder der Client konnte manche Pakete nicht verarbeiten. Dieser Fehlerfall kann durch Retransmission korrigiert werden.

Der Paketverlust wird also durch den intelligenten ACK Selection Algorithmus möglichst schnell festgestellt. Zusätzlich wird sofort nach Feststellen des Paketverlusts die Retransmission eingeleitet, mit den Daten aber an jener Stelle beginnend, wo sich der Datenlöscher gerade befindet. Im Vergleich mit den drei Läufern würde das heißen, daß der Sender zum Löscher zurückgeworfen wird und von dort nochmals loslaufen muß.

### 6.7.4 Automatisches Anpassen der Größe des Serverringpuffers

Da der server buffer loop hohe Kosten für den Zeitbedarf der Übertragung produziert, gilt es, diese Kosten zu minimieren. Bei Paketverlust eines Datenpakets, also eines Pakets von Server zu Client, wird ohne adaptive Steuerung der Retransmission (siehe 6.7.3) mindestens ein server buffer loop notwendig.

Auf der anderen Seite darf der Serverpuffer aber auch nicht zu klein sein, da sonst die Bestätigungen der Clients noch gar nicht angekommen sind, wenn der Sender dem Löscher schon wieder auf die Fersen rückt und bei Überholen desselben einen server buffer loop beginnt.

Es gilt also einen Mechanismus zu entwickeln, der den idealen Wert für die Puffergröße, welcher sich zwischen diesen beiden Extremlagen befindet, bestimmt. Da dieser Wert sehr von der Verarbeitungsgeschwindigkeit abhängig ist, wirkt als stark beeinflussende Größe die physikalische Beschaffenheit der Hardware. Dahingehend wirken auf einer hohen Ebene Prozessorleistung, Netzwerkleistung und Festplattenleistung ganz wesentlich auf den Idealwert. Bei näherer Betrachtung wird die Netzwerkleistung ja wiederum durch die Länge des Netzwerksegments beeinflußt.

Das Führen einer Statistik über die individuellen Verzögerungszeiten der Clients als Größenordnung kann auch hier Abhilfe schaffen.

Ein Ansatz wäre die dynamische Ausmessung des Serverringpuffers während der Übertragung.

Zu Beginn wäre hierbei der Serverringpuffer nur eine Runde groß. Der Puffer wird solange vergrößert, bis die Bestätigung des letzten Clients angekommen ist. Für die Vergrößerung des Puffers wird der Einfachheit halber vorgeschlagen, immer eine Runde hinzuzufügen. Auf der anderen Seite muß das Vergrößern auch Grenzen haben. Wenn zum Beispiel der Totalausfall eines Clients eintritt, so darf der Puffer nicht *auf ewig* vergrößert werden, sondern der fehlerhafte Client muß erkannt und entfernt werden.

Wenn der Serverpuffer aber klein ist, sind server buffer loops auch nicht so kostspielig.

Es wird also empfohlen, den Serverpuffer solange zu vergrößern, bis die Anzahl der server buffer loops um Größenordnungen sinkt.

So kann die optimale Größe für den Serverringpuffer für die jeweilige Session bestimmt werden. Bei der Darstellung der Meßergebnisse in Kapitel 7 wirkt die eben beschriebene Vorgangsweise sinnvoll.

### 6.7.5 Permanentes Streaming

Wenn die Datenquelle, die dem Server zur Verfügung steht, immer Daten generiert, also letztlich zum Beispiel keine Datei, sondern ein Datenstrom, wie zum Beispiel ein Live-Video, übertragen wird, so wären weitere Modifikationen erforderlich, um diesen Anwendungsfall vernünftig abzudecken. In diesem Fall würde das Protokoll sozusagen

ewig in Phase II verharren, das bedeutet, eine einmal gebildete Session würde *ewig* andauern.

Da ja nur zu Beginn des Protokollablaufs, nämlich in Phase I, die Gruppenbildung durchgeführt wird, könnten Clients, die später teilnehmen wollen, sich an dieser Session nicht registrieren. Erst bei der nächsten beginnenden Session, die durch ein erneutes Durchlaufen von Phase I gekennzeichnet wäre, ist die Aufnahme neuer Clients möglich. Zusätzlich hat der Server ja die Option, fehlerhafte Clients aus der Gruppe auszuschließen. Wenn so ein ausgeschlossener Client nach Reparatur oder Erholung wieder teilnehmen möchte, gilt für diesen die analoge Situation wie für später teilnehmenwollende Clients.

Für den Anwendungsfall einer *ewigen* Session wird also der Ruf nach dynamischer Gruppenbildung laut. Das Ein- und Aussteigen von Clients muß jederzeit möglich sein, wobei neu einsteigende Clients nicht alle Daten von Beginn der Session an erhalten, sondern lediglich die neu zu übertragenden. Gedacht ist diese Anwendung ja für das Verteilen eines Videostreams, zum Beispiel im MPEG2 Format [27], und in dieser Anwendung hätte es ja keinen Sinn, die alten Videodaten vom Beginn der Session einem neu einsteigenden Client im Nachhinein zu übermitteln. Der Client beginnt bei seinem Einstieg daher mit den aktuell übertragenen Daten.

Für permanentes Streaming ist eine weitere Modifikation notwendig. Die Sequenznummer ist eine ganze Zahl, die in einem Feld des Datenpakets abgelegt wird. Es gibt daher nur eine endliche Anzahl von Sequenznummern. Daher muß ein Mechanismus vorgesehen werden, der das Zurücksetzen von Sequenznummern ermöglicht. Vorgeschlagen wird ein Flag im Datenpaket, welches anzeigt, daß beim nächsten Paket die Sequenznummer wieder mit 1 beginnt. Der Server setzt bei Bedarf dieses Flag, und Clients müssen dieses Flag entsprechend beachten.

Wenn Videostreams übertragen werden sollen, so muß auch berücksichtigt werden, daß das Kriterium der Latenzzeit durch eine brauchbare obere Schranke eingegrenzt sein muß. Der Client soll in diesem Anwendungsfall zwar auch puffern, damit das Video ohne Unterbrechung wiedergegeben werden kann. Um diese clientseitige Pufferung richtig

modellieren zu können, muß aber zunächst eine brauchbare Kombination von garantiertem minimalen Durchsatz und garantierter maximaler Latenzzeit durch das Protokoll gewährleistet werden, damit es nicht zu Unterbrechungen bei der Live-Wiedergabe des Videostreams kommt [27].

Wird nämlich nur ein minimaler Datendurchsatz gewährleistet, aber keine maximale Latenzzeit, so kann der clientseitige Puffer irgendwann auch leer werden. Man nehme zum Beispiel an, der Videostream benötigt 1 MByte pro Sekunde an Durchsatz. Wenn der clientseitige Puffer nun eine Größe von 5 MByte hat, so kann dieser Puffer eine Zeit von maximal fünf Sekunden überbrücken, in welcher vom Protokoll keine Daten geliefert werden. Wenn nun eine minimale Latenzzeit von vier Sekunden durch das Protokoll garantiert wird, so bedeutet dies, daß vom Server neue gelesene Daten sicher innerhalb von vier Sekunden beim Client angekommen sind. Die Pufferung stützt demnach eine Verzögerung von maximal vier Sekunden, weil ja erst nach fünf Sekunden der Empfang neuer Daten zwingend erforderlich wäre. Für einen gewissen Paketverlust kann diese Pufferkonfiguration also eine Stützung des Datenstromes garantieren. Wenn diese vier Sekunden garantierte Latenzzeit ausgenützt werden müssen, der clientseitige Puffer danach nicht wieder vollständig gefüllt werden kann und gleich darauf erneut Paketverlust auftritt und die minimale Latenzzeit von vier Sekunden wieder vollständig ausgenützt werden müßte, so reicht diese Pufferkonfiguration nicht mehr aus, um eine unterbrechungsfreie Wiedergabe des Videos zu gewährleisten. Letztlich kann keine Pufferung verhindern, daß nicht irgendwanneinmal möglicherweise doch der Datenstrom unterbrochen wird [25].

## 6.8 Eine Implementierung von RMARS

Es wurde im Laufe der theoretischen Überlegungen auch eine Implementierung derselben als Prototyp realisiert. Dieser Prototyp des RMARS Protokolls sollte als Grundlage dieser Arbeit und für die weitere Erforschung einzelner Designentscheidungen dienen.

### 6.8.1 Eigenschaften des Prototypen

Der Prototyp hat folgende Eigenschaften:

1. Als Programmiersprache wird C unter Linux verwendet.
2. Grundlage für die Übertragung ist UDP Broadcast.
3. Nutzdatenpakete transportieren maximal 512 Byte an Nutzdaten.
4. Für die Sequenznummer wird ein unsigned 32 Bit Integer verwendet.
5. Der Server liest die zu versendenden Daten von STDIN.
6. Der Client schreibt die empfangenen Daten auf STDOUT.
7. Der Server liest mittels fread (2) und einer Blockgröße von 1.
8. Der Client schreibt mittels fwrite (2) und einer Blockgröße von 1.
9. Der Server spaltet sich in zwei Prozesse: DATASender und ACKReceiver.
10. Der Serverringpuffer hat eine konstante Größe von vier Runden bei 14 Clients.
11. Der Server implementiert keine Flußkontrolle.

Viele dieser Eigenschaften ergaben sich aufgrund der Einfachheit. Bei genauer Rekapitulation erscheinen mancherlei Mechanismen, welche sich negativ auf die Leistung des Protokolls auswirken:

1. Es gibt viele *context switches* [3] auf Seiten des Clients, weil auf STDOUT geschrieben wird.
2. Es gibt viele context switches auf Seiten des Servers, weil erstens von STDIN gelesen wird, und zweitens der Server sich in die zwei Prozesse DATASender und ACKReceiver aufteilt.

3. Die im Prototyp für fread und fwrite verwendete Blockgröße von 1 wirkt sich äußerst negativ auf die Leistung, insbesondere was den Durchsatz des Protokolls betrifft, aus.

4. Die Gesamtmenge an Daten, die in einer Session transportiert werden kann, ist $(2^{32} - 2) \cdot 512$ Byte, also cirka 2 TB, weil ein Datenpaket maximal 512 Byte Nutzdaten transportieren kann und nur $2^{32} - 2$ Sequenznummern für Nutzdatenpakete zur Verfügung stehen, weil die Sequenznummer Null nicht existiert und das Abschlußpaket mit einer Nutzdatenanzahl von Null Byte auch eine Sequenznummer benötigt.

### 6.8.2 Implementierung des Prototypen im Detail

Es wird eine für Server und Client gemeinsame Definition für Datenpakete (DataPacket) und Organisationspakete (DispatchPacket), sowie für die Pakettypen (Packet Types = PT) verwendet.

#### Phase I: Grouping

Das Grouping in Phase I läuft nach einem PAR-Protokoll [13] ab. Der Server horcht auf Anforderungen von Clients auf dem Organisationskanal (SERVER_DISP_PORT). Clients schicken ihre Anforderung (PT_CLIENT_REGISTER) an den Server, der Server gibt dem Client ein eindeutige Nummer (ClientNumber) innerhalb der Session und schickt eine Antwort auf die Registrierungsanforderung (PT_SERVER_REGISTER_ACK), welche die zugeteilte ClientNumber beinhaltet.

Server und Clients verwenden jeweils blockendes recv (2) und sendto (2). Es wird setitimer (2) verwendet, um Timeouts für ein blockendes recv zu realisieren.

Sobald der Client das Bestätigungspaket vom Server erhalten hat, wechselt er zu Phase II.

Nach Ablauf des Timeouts in der Grouping Phase (WAIT_START_TRANSMISSION) wechselt auch der Server zu Phase II.

### Phase II: Transmit Data

**Von Seiten des Clients** Clients warten in Phase II auf Datenpakete vom Server (PT_SERVER_DATA). Sobald ein Paket empfangen wurde, wird geprüft, ob es sich um ein PT_SERVER_DATA Paket handelt. Wenn ja, folgt eine Verarbeitung.

Der Client muß immer feststellen, ob der Server eine Bestätigung von ihm anfordert. Dazu werden die ClientNumber und das ACK REQUESTED Flag geprüft. Wird der Client durch dieses Paket zum ACK aufgefordert, so muß er nach Verarbeitung der Nutzdaten entsprechend reagieren.

Daher prüft der Client als nächstes die Sequenznummer des gerade empfangenen Pakets. Stimmt diese nicht mit der von ihm als nächste erwarteten überein, so wird das Paket gemäß Protokoll ignoriert. Andernfalls folgt eine weiterführende Verarbeitung: im Paket sind die Nutzdaten (Data) und die Anzahl der gültigen Bytes im Nutzdatenfeld (DataSize) eingetragen. Der Client schreibt DataSize viele Byte vom Nutzdatenfeld (Data) mittels fwrite (2) auf STDOUT. Sobald in DataSize der Wert Null eingetragen ist, wechselt der Client zu Phase III.

**Von Seiten des Servers** Der Server ist gemäß den Vorstellungen des Protokolls das Herzstück des gesamten Ablaufes, es fallen ihm zentrale Steuerungs- und Organisationsaufgaben zu.

Der Server legt einen Ringpuffer (buffer) an. Der Server speichert ein Paket solange in dem Puffer, bis alle Clients dieses Paket bestätigt haben, denn dann ist dieses nicht mehr notwendig. Solange ein Paket aber noch nicht von allen bestätigt wurde, müssen die Daten noch zur Verfügung stehen, da der Server jene erneut senden muß.

Der Server teilt sich mittels fork (2) in einen ACKReceiver und einen DATASender. Die beiden Prozessen kommunizieren über ein *Shared Memory* [25], es wird in der Implementierung SDA (Shared Data Area) genannt.

Der ACKReceiver hat als einzige Aufgabe, wie schon sein Name sagt, die ACKs der Clients zu empfangen und die Sequenznummer, welche von einem Client bestätigt wird, im gemeinsamen Datenbereich SDA einzutragen. Da es sich hierbei nur um *einen* Pro-

zessorbefehle handelt, nämlich einen MOVE-Befehl, der einen Integerwert in das Shared Memory schreibt, wurde hier auf die Verwendung von Synchronisationsmitteln wie Semaphores verzichtet. Es kann unter keinen Umständen zu Inkonsistenzen kommen, weil der Scheduler nicht während eines Prozessorbefehls unterbrechen kann [25].

Der DATASender ist nun derjenige, der den Hauptteil des Protokolls steuert. Es ist eine Schleife (Hauptschleife), die mit dem Auftreten von EOF auf STDIN beendet. Sobald nämlich EOF auftritt, muß der Server in eine andere Phase wechseln, nämlich in Phase II-III. Diese Phase gibt es nur für den Server, nicht für die Clients.

Der Ringpuffer in Phase II wird wie folgt verwendet. Der Ringpuffer speichert vorgefertigt Datenpakete, wie sie direkt auf das Netzwerk geschickt werden. Zu jedem Datenpaket wird die Gültigkeit vermerkt. Wenn in einem Datenpaket des Ringpuffers bei der Sequenznummer Null eingetragen ist, so darf es beschrieben werden. Anfangs sind also die Sequenznummern aller Datenpakete im Ringpuffer Null.

Es werden folgende Variablen zur Puffer- und Übertragungssteuerung verwendet:

1. Sequence
2. NextBufferToBeSent
3. NextACKClient
4. NextSEQToBeACKed

Die Variable sequence wird direkt auf den Ringpuffer abgebildet, ähnlich wie bei einem direct-mapped Cache [18]. Im Vergleich mit den drei Läufern entspricht die Variable NextBufferToBeSent der Position des Senders, die auf den Ringpuffer abgebildete Variable sequence der Position des Lesers sowie die Variable NextSEQToBeACKed der Position des Löschers.

Die Hauptschleife prüft nun, ob das durch sequence bestimmte, entsprechende Element des Ringpuffers mit Daten beschrieben werden darf. Wenn ja, so wird mittels fread (2) von STDIN gelesen, und diese Daten werden am entsprechenden Teil im Ringpuffer abgelegt. Wenn nicht geschrieben werden darf, passiert nichts; es wird auch nicht von

STDIN gelesen. Die Funktion fread (2) blockt den gesamten Ablauf, wenn auf STDIN keine weiteren Daten zur Verfügung stehen.

Unabhängig davon kümmert sich die Hauptschleife auch darum, die Pakete aus dem Ringpuffer zu übertragen. Dazu wird die Variable NextBufferToBeSent verwendet. Diese wird bei jedem Schleifendurchlauf im Wertebereich des Ringpufferindex erhöht, d.h., bei jedem Durchlauf der Hauptschleife wird auch ein Datenpaket aus dem Puffer verschickt.

Der Durchsatz des Protokolls ist also dann hoch, wenn das Eintragen neuer Datenpakete im Ringpuffer nicht gebremst wird, d.h., wenn der Ringpuffer ausreichend groß ist, daß alle Bestätigungspakete der Clients für ein Datenpaket A angekommen sind, bevor der DATASender Prozeß dieses Datenpaket A überschreiben möchte. Auf der anderen Seite darf der Ringpuffer auch nicht zu groß sein, da ja in dieser Variante von RMARS die Größe des Ringpuffers bestimmt, wie viele Pakete *nochmals* übertragen werden, wenn es zu einem server buffer loop kommt.

Server buffer loops treten auf, wenn die Variable NextBufferToBeSent die Variable NextSEQToBeACKed überholt. In diesem Fall nämlich werden Datenpakete wiederholt, die schon mindestens einmal ausgesendet worden sind.

Unabhängig von alledem gibt es in der Hauptschleife noch den ACK Selection Algorithmus. In der Prototyp-Implementierung ist diese Auswahl sehr primitiv, es wird einfach Client für Client durchgegangen.

Wiederum unabhängig davon wird in der Hauptschleife eine Variable NextSEQToBeACKed geführt, die angibt, welches Datenpaket als nächstes von allen bestätigt sein sollte, kurzum, welches Datenpaket der DATASender gerne als nächstes aus dem Ringpuffer löschen möchte. Eine primitive Funktion prüft, ob schon alle Clients eine Bestätigung für dieses Datenpaket implizit oder explizit durchgeführt haben. Wenn ja, so wird das entsprechende Paket im Ringpuffer als zu überschreibendes markiert.

Wird von STDIN ein EOF gelesen, so wechselt der Server in Phase II-III. Diese ist Phase II sehr ähnlich, bis auf die Tatsache, daß in dieser Phase klarerweise nicht mehr von STDIN gelesen wird, sondern das Abschlußpaket *einmal* in den Ringpuffer eingefügt wird. Dieses Abschlußpaket hat als Anzahl der Nutzdaten (DataSize) den Wert Null und

signalisiert den Clients damit, daß die Übertragung der Daten zuende ist. Da für dieses Abschlußpaket aber eine weitere Sequenznummer vergeben wird, kann von allen Clients auch in gewohnter Weise und ohne eine weitere Verkomplizierung des Mechanismus eine Bestätigung für dieses Abschlußpaket verlangt werden. Erst, wenn alle Clients dieses Abschlußpaket bestätigt haben, wechselt der Server zu Phase III, in welcher der Abschluß des Protokolls versucht wird.

#### Phase III: Final

In Phase III macht der Client nichts anderes mehr, als auf Nutzdatenpakete (PT_SERVER_DATA) zu antworten, welche von ihm Bestätigungen anfordern. Diese Phase wird vom Client erst verlassen, wenn entweder vom Server ein Finalpaket geschickt wird (PT_SERVER_FIN_ACK), oder wenn das lokale Timeout des Clients für den Empfang von Datenpaketen (PHASE_III_CLIENT_TIMEOUT) abläuft. Solange der Client Datenpakete empfängt, wird das Timeout nicht wirken. Sollte jedoch der Client das Finalpaket des Servers nicht empfangen, so wird das Timeout den Client beenden, da der Server mit Verschicken des Finalpakets die Session als beendet erklärt. Ferner muss darauf geachtet werden, daß gilt:

$$PHASE\_III\_CLIENT\_TIMEOUT < WAIT\_START\_TRANSMISSION$$

Sonst könnte folgendes Problem auftreten. Es wird vom Server eine Session (Session 1) abgeschlossen. Client A bekommt das Finalpaket aus Session 1 aber nicht mit. Der Server wird nun erneut gestartet und wartet auf Client Register Anforderungen. Client B fordert nun eine Session an. Client A wartet aber noch immer auf das Finalpaket, das Timeout ist noch nicht abgelaufen. Wenn der Server nun eine neue Datentransmission gemäß Phase II startet, bevor das Timeout bei Client A abgelaufen ist, so würde Client A auch noch den Ablauf dieser neuen Session abwarten müssen.

### 6.8.3 Verwendung des Prototypen

RMARS in der Prototypimplementierung wird wie folgt verwendet. Zunächst muß der Server gestartet werden. Da der Server von STDIN liest, muss ein Datenstrom zur Verfügung gestellt werden. Zum Beispiel wird der Befehl

```
cat dez2005.img | rmars-server
```

die Datei dez2005.img dem RMARS Server auf STDIN zur Verfügung stellen. Im Zuge dieser RMARS Session wird also jene Datei übertragen.

Auf der Clientseite muß ebenso der Datenstrom, der auf STDOUT vom Client produziert wird, umgeleitet werden. Mit dem Befehl

```
rmars-client > /dev/hda
```

wird das Festplattenimage, welches über RMARS empfangen wird, direkt auf die Festplatte geschrieben.

Analog funktioniert das mit einem Live-Videostream, der zum Beispiel von /dev/video0 gelesen werden kann. So wäre auf der Serverseite

```
cat /dev/video0 | rmars-server
```

auszuführen und am Client mittels

```
rmars-client | mplayer --stdin
```

ein Media Player zu starten, der den Datenstrom von STDIN liest.

### 6.8.4 Modifikationen und Erweiterungen

Da es sich bei der vorgestellten Implementierung um einen Prototypen handelt, werden hier noch wichtige Modifikationen und Erweiterungen vorgestellt, die für einen Leistungsbetrieb wesentlich sind.

Folgende Veränderungen werden empfohlen; auch, um einige der Probleme zu beseitigen, die bei den Eigenschaften des Prototypen (siehe 6.8.1) angeführt wurden.

1. UDP Multicast anstelle von UDP Broadcast verwenden
2. Blockgröße für fread und fwrite vergrößern
3. Das Aufspalten des Servers in zwei Prozesse vermeiden
4. Alternativen zum STDIN/STDOUT Mechanismus bieten
5. Flußkontrolle implementieren
6. Adaptive Steuerung der Retransmission implementieren
7. Automatische Anpassung der Größe des Serverringpuffers implementieren

Für die Verwendung von UDP Multicast anstelle von Broadcast erscheinen keine Probleme. Im Gegenteil, das Registrieren mittels IGMP ist den Vorgängen in der Grouping Phase sehr ähnlich.

Die Blockgröße von fread und fwrite zu erhöhen ist auch lediglich eine kleine Veränderung, die aber große Steigerungen des Durchsatzes erwarten läßt. Als Vorschlag soll mittels fread ein größerer Block in der Speicher gelesen werden. Von diesem Block wird von der Datenleserinstanz dann immer nur ein Paket in den Ringpuffer kopiert. Dadurch fällt der Overhead durch I/O, zum Beispiel für eine Festplatte, wesentlich seltener an. Auf dem Client soll genauso zunächst vom Netzwerk in den Speicher geschrieben und erst, wenn der Puffer voll ist, auf STDOUT weitergegeben werden.

Um context switches zu vermeiden, soll auf dem Server auch das Empfangen von ACKs im Hauptprogramm erledigt werden, ohne auf einen eigenen Prozeß zurückgreifen zu müssen.

Auch beim Lesen von STDIN bzw. Schreiben auf STDOUT passieren context switches. Als Alternative dafür bietet sich zum Beispiel der direkte Zugriff auf das Dateisystem, was die context switches bereits ersparen würde. Eine weitere Möglichkeit unter Linux wäre die Verwendung von Memory Mapped I/O.

Für die Realisierung von Flußkontrolle erscheint für die Implementierung unter Linux die Verwendung der Funktionen nanosleep (2) oder setitimer (2) sinnvoll. Der Zahlenwert

für die Pausendauer ist abhängig von der momentanen Leistung des Servers und der Gruppe und muß daher permanent durch eine adaptive Methode bestimmt werden.

### Reine Dateiübertragung

Eine andere Modifikation, die sich nicht auf die Leistung, sondern nur auf die Bedienbarkeit des Servers auswirkt, wäre eine Betriebsart, in welcher der Server bei jeder Session dieselbe Datei überträgt. Dies wäre äußerst wünschenswert, wenn es zum Beispiel um die Übertragung eines Festplattenimages geht. Insofern empfähle sich eine Ausführung des RMARS Servers als *daemon* [3].

# 7 Vergleich und Bewertung von RMARS

Um die Brauchbarkeit von RMARS für die Software-Verteilung in einer Schulinfrastruktur zu bewerten, mußten Tests durchgeführt werden, bei welchen für diese Anwendung wichtige Parameter gemessen wurden. Für diese Messungen wurde eine Umgebung eingerichtet, die im nächsten Abschnitt näher beschrieben wird.

## 7.1 Versuchsaufbau und Methodik

Aufgrund der Anforderungen des Anwendungsfalles ist die entscheidende Meßgröße für die Bewertung eines Protokolls die Gesamtdauer der Übertragung, also jene Zeit, die vom Zeitpunkt des Beginns der Übertragung bis zu jenem Zeitpunkt vergeht, zu welchem alle Clients die fehlerfreie Kopie der Daten erhalten und geschrieben haben.

Folgende Meßgrößen erschienen für die Bewertung von RMARS sinnvoll:

1. Gesamtdauer der Übertragung (grundlegend)
2. Anzahl der server buffer loops
3. Größe des Serverringpuffers

Es sei darauf hingewiesen, daß die Anzahl der server buffer loops und die Größe des Serverringpuffers aber wiederum die Gesamtdauer der Übertragung beeinflußen. Für diese Ereignisse wiederum gilt als Ursache auftretender Paketverlust.

Mit Hilfe der Gesamtdauer und der Gesamtdatenmenge kann der Durchsatz berechnet werden.

In der Testumgebung befinden sich ein Server und 14 Clients, welche alle — bis auf Exemplarstreuung — gleiche Hardware besitzen, die durch ein *100 Mbps full-duplexed switched ethernet* [27] untereinander verbunden sind.

## 7.2 Ergebnisse des Prototypen

Es wurde im Rahmen der Protokollweiterentwicklung der in Abschnitt 6.8 beschriebene Prototyp implementiert. Mit Hilfe dessen wurden einige Messungen durchgeführt. Das Protokoll sollte aufgrund dieser Erkenntnisse weiterentwickelt werden können. Am Protokoll des Prototyps wurde für die Messungen nichts verändert; es wurden lediglich kleine Softwarefehler ausgebessert.

Erstes Hauptaugenmerk bei den Messungen betraf die Auswirkungen der Puffergröße auf die Übertragungsdauer. Wie in oben bereits erwähnten Überlegungen zu finden (siehe 6.7.4), würde sich eine zu gering gewählte Puffergröße negativ auf die Übertragungsleistung auswirken. Die Anzahl der unproduktiven Pufferdurchläufe (siehe 6.4.2) erhöht sich, da der Server mit der Übertragung wieder am Anfang des Puffers angelangt ist, bevor alle Bestätigungen der Clients beim Server eingetroffen und verarbeitet wurden. Eine zu großzügige Wahl für die Pufferlänge hingegen wirkt sich ebenfalls negativ aus, wenn Pakete häufig verlorengehen, weil bei auftretendem Paketverlust auch ein größerer Datenbereich wiederholt wird.

Für die Messung der Übertragungsleistung gelten folgende Bedingungen.

1. Es wurden 14 Clients in einem lokalen Netzwerk verwendet
2. Die Clients waren hardwaremäßig baugleich, das heißt, leistungsmäßig sehr ähnlich, abgesehen von Exemplarstreuung.
3. Die Puffergröße wurde durch den Parameter Runden verändert. Eine Runde beinhaltet Speicher derart, daß für jeden Client ein Paket gespeichert werden kann.
4. Es wurde jeweils die Anzahl der unproduktiven Pufferdurchläufe, sowie die Gesamtdauer der Übertragung gemessen.

Es wurden bisher cirka 30 Sessions durchgeführt, bei jeder dieser Sessions wurden cirka 1.5 Gigabyte an Daten übertragen. An jeder Session nahmen 14 Clients teil. Die Clients sind gleichartiger Hardware und ähnlichen Alters.

Die Übertragung mittels Prototyp-Implementierung des RMARS Protokoll beträgt für die gesamte Gruppe von 14 Clients cirka 8 Minuten. Es sei nochmals darauf hingewiesen, daß bei der Prototyp-Implementierung noch keinerlei Optimierungen zur Leistungssteigerung durchgeführt wurden.

Den nachfolgenden Abbildungen ist die Auswirkung der Größe des Serverringpuffers auf die Übertragungsdauer zu entnehmen. Zur Messung dieser Daten wurde lediglich der Parameter Runden verändert.

| Runden | Loops | Dauer | Runden | Loops | Dauer |
|---|---|---|---|---|---|
| 1 | 430451 | 00:29:20 | 10 | 1366 | 00:09:10 |
| 2 | 105354 | 00:17:36 | 20 | 1254 | 00:09:35 |
| 3 | 32167 | 00:12:40 | 30 | 1147 | 00:09:58 |
| 4 | 1850 | 00:08:58 | 40 | 1029 | 00:10:14 |
| 5 | 1707 | 00:08:56 | 50 | 934 | 00:10:28 |
| 6 | 1625 | 00:08:58 | 60 | 815 | 00:10:34 |
| 7 | 1638 | 00:09:02 | 70 | 828 | 00:10:55 |
| 8 | 1572 | 00:09:04 | 80 | 854 | 00:11:21 |
| 9 | 1430 | 00:09:04 | 90 | 839 | 00:11:38 |

Tabelle 7.1: RMARS mit unterschiedlichen Puffergrößen

Aus diesem Diagramm ist leicht erkennbar, daß die in Kapitel 6.7.4 angestellten Überlegungen betreffend die Größe des Serverringpuffers zumindest durch Messungen belegt werden können. Bei Ansteigen der Größe des Serverringpuffers auf cirka 4 Runden nimmt die Anzahl der server buffer loops um Größenordnungen ab, ebenso reduziert sich der Zeitbedarf für die Übertragung. Bei Puffergrößen von mehr als 5 Runden steigt die Gesamtdauer der Übertragung, wenn auch nur gering, weil bei Auftreten von server buffer loops ja ein größerer Datenbereich wiederholt wird. Die entscheidende Größe von 5 Run-

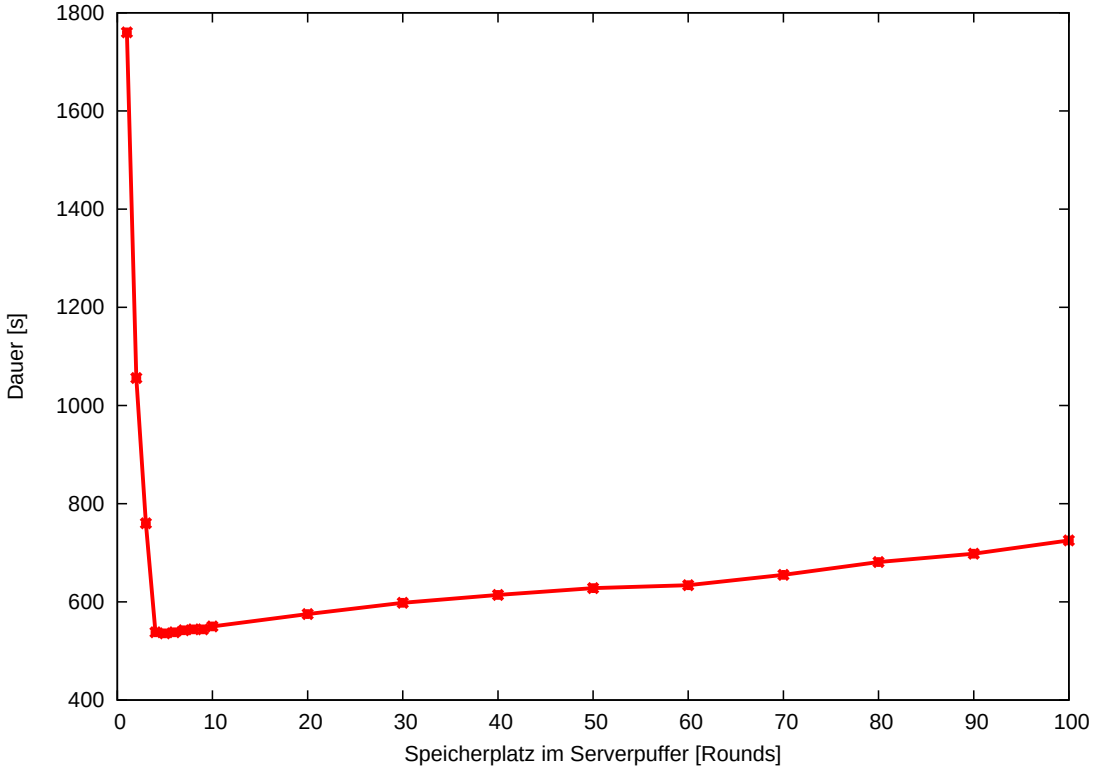


Abbildung 7.1: RMARS mit unterschiedlichen Puffergrößen

den ist aber individuell für diese Testumgebung. Für eine andere Umgebung wird es für den Serverringpuffer auch eine andere optimale Größe geben. Dieser Wert muß durch automatische Anpassung der Größe des Serverringpuffers für die jeweilige Session ermittelt werden.

## 7.3 Vergleich mit ähnlichen Protokollen

### 7.3.1 Vergleich mit singulärer NFS Übertragung (Single NFS)

Mit singulärer NFS Übertragung ist gemeint, daß zur selben Zeit immer nur ein Client das Festplattenimage über NFS [5] vom Server bezieht. Dabei handelt es sich rein um eine Beziehung zwischen einem Server und einem Client, welchen das Netzwerk exklusiv zur Verfügung stehen. Die Clients melden ihren Wunsch, das Festplattenimage herun-

terzuladen, beim Server an. Dieser verwaltet eine Warteschlange und bedient die Clients in jener Reihenfolge, in welcher sich diese angemeldet haben. Da für alle Clients dieselben Bedingungen gelten, gestaltet sich auch der Zeitbedarf der Übertragung bei allen ähnlich. Pro Client mußte in der Testumgebung ein Zeitbedarf von durchschnittlich fünf Minuten gerechnet werden. Da insgesamt 14 Clients wiederhergestellt werden sollten, ergab sich ein Gesamtbedarf von 70 Minuten. Diese Lösung weist offensichtlich ein Problem der Scalability hinsichtlich des Zeitbedarfs auf. Je mehr Clients an der Übertragung teilnehmen, desto größer ist der Gesamtzeitbedarf.

### 7.3.2 Vergleich mit mehrfacher NFS Übertragung (Multi NFS)

Im Gegensatz zur Einzelvariante sollten hier alle Clients gleichzeitig das Image vom Server über NFS beziehen. Um diesen Test durchzuführen, wurden zunächst alle Clients synchronisiert, damit die gemeinsame NFS Übertragung bei allen zur gleichen Zeit beginnt, um die Gesamtdauer der Übertragung leichter bestimmen zu können. Dies sollte auch bewirken, daß der Server Datensätze nur einmal von der Festplatte lesen mußte und diese aber mehrfach über das Netzwerk übertragen konnte. Der erste Lesezugriff eines NFS Serverprozesses mußte also noch die Daten von der Festplatte beziehen, weitere Prozesse konnten aber großteils über den Cache bedient werden. Dadurch wirkte sich hauptsächlich die Leistung des Netzwerks auf die Gesamtdauer der Übertragung aus, und nicht die Parameter des Festplattenzugriffs. Bei dieser synchronisierten Variante betrug der Zeitbedarf für die 14 Clients insgesamt 28 Minuten. Eine Variante ohne Synchronisation erscheint aus vorher genannten Gründen nicht sehr sinnvoll. Den nachfolgend angegebenen Zahlen ist der Zeitbedarf jedes einzelnen Clients zu entnehmen. Dabei fällt auf, daß alle Clients nahezu gleichzeitig fertig geworden sind.

Die durchschnittliche Übertragungsdauer bei dieser Messung beträgt 27 Minuten und 52 Sekunden, die Toleranz beträgt $\pm$ 9 Sekunden. Diese Streuung kann durch das individuelle Verhalten der Clients bei Cachingmaßnahmen erklärt werden.

Auch bei diesem Ansatz gilt, daß es ein Problem der Scalability gibt, aber im Vergleich zum singulären Ansatz in Hinsicht des Netzwerks. Wenn die zehnfache Menge

| Client | Anfangszeit | Endzeit | Dauer |
|---|---|---|---|
| 01 | 17:26:46 | 17:54:30 | 0:27:44 |
| 02 | 17:26:48 | 17:54:34 | 0:27:46 |
| 03 | 17:26:43 | 17:54:26 | 0:27:43 |
| 04 | 17:26:44 | 17:54:28 | 0:27:44 |
| 05 | 17:26:44 | 17:54:38 | 0:27:54 |
| 06 | 17:26:47 | 17:54:32 | 0:27:45 |
| 07 | 17:26:44 | 17:54:44 | 0:28:00 |
| 08 | 17:26:44 | 17:54:43 | 0:27:59 |
| 09 | 17:26:45 | 17:54:33 | 0:27:48 |
| 10 | 17:26:42 | 17:54:37 | 0:27:55 |
| 11 | 17:26:43 | 17:54:39 | 0:27:56 |
| 12 | 17:26:44 | 17:54:44 | 0:28:00 |
| 13 | 17:26:43 | 17:54:33 | 0:27:50 |
| 14 | 17:26:42 | 17:54:43 | 0:28:01 |

Tabelle 7.2: Dauer der Einzelübertragungen

an Clients gleichzeitig den NFS Server verwenden, wäre dieser hoffnungslos überfordert. Da die einzelnen NFS Serverprozesse zusätzlich nicht darauf achten, den Festplattencache zum gemeinsamen Vorteil zu nützen, ist es nur eine Frage der Zeit, bis bei diesem Lösungsansatz der Zeitbedarf explosionsartig ansteigt.

### 7.3.3 Vergleich mit UDPCAST

UDPCAST ist ein OpenSource Projekt, welches von der URL http://udpcast.linux.lu/ bezogen werden kann. Da sich auf dieser Webseite jedoch leider keine genaue Beschreibung des verwendeten Protokolls befindet, wurde diese Lösung nicht in Kapitel 4 aufgeführt. Der Autor des Projekts hat jedoch in einer Mailingliste Auskunft über den

Ablauf des Protokolls gegeben. Dieses Mail ist unter [12] zu finden. Angaben über den genauen Ablauf sucht man dort aber vergebens.

Der Server kann von STDIN lesen, und die Clients können auf STDOUT schreiben. Der Anwender hat die Auswahl zwischen der Verwendung von Ethernet Broadcast oder Ethernet Multicast.

Diese Lösung produzierte das beste Ergebnis im Vergleich zu den anderen Varianten mit einer durchschnittlichen Gesamtdauer der Übertragung von 2 Minuten und 21 Sekunden. Der Serverteil meldete zusätzlich durchschnittlich 16000 Retransmissions.

Letztlich wurden zur Analyse des Protokolls die Beschreibung aus der Mailingliste, der Source-Code selbst und die Manual Pages des Projekts zu Rate gezogen. Unter Zuhilfenahme dieser Quellen ist nachfolgende Beschreibung der Funktionsweise des Protokolls entstanden.

Es wird der Begriff *slice* eingeführt. Der Datenstrom wird in slices aufgeteilt, slices bestehen wiederum aus *blocks*. Nach jedem slice fragt der Server die Clients, ob sie alle Blöcke des slice empfangen haben. Dem Source-Code ist zu entnehmen, daß selektives ACK möglich ist, das heißt, daß einzelne Clients gezielt zum ACK aufgefordert werden können, daß aber auch mit einem einzigen ACK Request des Servers gleichzeitig mehrere Clients zum ACK aufgefordert werden können. Es scheint jedoch, daß der Server nach Übertragung eines slices immer alle Clients zugleich abfragt. Jedenfalls antworten Clients auf server requested ACKs entweder mit einem OK oder einem Retransmit Request.

UDPCAST kann beauftragt werden, zusätzlich Forward Error Correction (FEC) zu verwenden, um eine gewisse Anzahl fehlender Pakete auf der Clientseite kompensieren zu können.

Wie der Server entscheidet, wann welche Clients zum ACK aufgefordert werden, ist aufgrund der fehlenden Spezifikation nur mühsam dem Source-Code zu entnehmen. Da es aber die Möglichkeit gibt, mehrere Clients gleichzeitig aufzufordern, sind prinzipiell auch ACK implosions möglich.

Alles in allem dürfte UDPCAST ein naher Verwandter von RMARS sein. Für den Vergleich sei darauf hingewiesen, daß von RMARS lediglich ein Prototyp existiert, für den Test von UDPCAST jedoch eine Produktionsversion verwendet wurde.

Hier werden die Gemeinsamkeiten der beiden Protokolle aufgezeigt:

1. Es kann mittels STDIN/STDOUT gearbeitet werden.
2. Der Datenstrom wird in atomare Einheiten aufgeteilt, die erst aus dem Serverpuffer entfernt werden, wenn diese von allen Clients bestätigt wurden.
3. Clients werden nach einer gewissen Anzahl von unbestätigten ACK Requests durch den Server von der Gruppe ausgeschlossen.
4. Alle Clients erhalten Pakete von Retransmissionen, auch wenn sie diese nicht benötigen.

Da von UDPCAST zusätzlich keine Spezifikation existiert, müssen Softwarefehler des Programms als Protokollfehler angenommen werden, weil der Source-Code des Projekts ja eigentlich das Protokoll spezifiziert. Deshalb kann eigentlich nichteinmal Reliability für dieses Protokoll behauptet werden, da der Server in irgendeinem pathologischen Fall möglicherweise Pakete aus seinem Puffer entfernt, bevor alle Clients diese fehlerfrei empfangen haben.

Jedenfalls bestehen in folgenden Punkten Unterschiede:

1. RMARS kann mit jedem Datenpaket ein gezieltes ACK anfordern, UDPCAST fordert ACKs nur nach Übertragung eines kompletten Slice an.
2. RMARS verwendet ein simples Puffermanagement, UDPCAST teilt in slices und blocks auf.
3. RMARS ist ein sehr primitives und einfach realisiertes Protokoll, UDPCAST ist aufwendig und kompliziert.

Bis auf die Verwendung von FEC, welche von RMARS nicht vorgesehen wird, kann RMARS die Funktionalität von UDPCAST doch größtenteils emulieren. Da es sich bei

der Version von UDPCAST, die im Rahmen dieser Arbeitet getestet wurde, um eine optimierte und schon intensiv weiterentwickelte Variante handelt, ist der Vergleich mit dem sehr jungen Prototypen des RMARS Protokolls relativ unfair.

### 7.3.4 Übersicht der Varianten

Hier findet sich eine Übersicht der vorgestellten Varianten mit berechnetem Durchsatz.

| Protokoll | Teilnehmerzahl | Datenmenge | Gesamtdauer | Durchsatz |
|---|---|---|---|---|
| Single NFS | 14 | 1.5 GB | 70 min. | 2,9 Mbps |
| Multi NFS | 14 | 1.5 GB | 28 min. | 7,3 Mbps |
| RMARS | 14 | 1.5 GB | 8 min. | 25,6 Mbps |
| UDPCAST | 14 | 1.5 GB | 2,4 min. | 87,1 Mbps |

Tabelle 7.3: Vergleich der Varianten

Es sei darauf hingewiesen, daß der Durchsatz bei Variante Single NFS nur als Bewertung betrachtet werden darf, da ja die einzelnen Übertragungen der Clients mit höherer Geschwindigkeit abgelaufen sind.

# 8 Schlußfolgerungen

Wie in dieser Arbeit auf vielerlei Weise dargestellt handelt es sich bei der Verwaltung von IT-Technologie in einer Schulinfrastruktur um einen Spezialfall.

Aufgrund der Rahmenbedingungen des Einsatzes in der Schule ist das wesentliche Problem, die Softwareinstallation auf den einzelnen Arbeitsplätzen wiederherstellen zu können.

Für die Wiederherstellung der Softwareinstallation ist ausschließlich eine Image-Lösung geeignet. Da in modernen Schulinfrastrukturen Netzwerke standardmäßig vorhanden sind, bietet es sich an, die Image-Wiederherstellung über das Netzwerk zu betreiben und höchstmöglich zu automatisieren.

Da die Image-Wiederherstellung möglichst oft durchgeführt werden können soll, muß die Übertragung über das Netzwerk möglichst effizient gestaltet werden, weshalb Multicast Protokolle für diesen Aufgabenbereich gewählt wurden.

Bestehende Multicast Protokolle eigneten sich aber nicht für den Spezialfall der Software-Wiederherstellung in Schulinfrastrukturen, weil diese entweder in Belangen der Zuverlässigkeit oder des Protokollablaufs nicht zweckentsprechend sind, weshalb im Zuge dieser Arbeit eine Speziallösung mit dem Namen RMARS geschaffen wurde.

RMARS enstand aufgrund der Analyse der unüblichen Anforderungen, welcher die Software-Wiederherstellung in Schulinfrastrukturen bedarf. Natürlich kann auch jedes andere Multicast Protokoll, welches diese Anforderungen erfüllt, für diesen Spezialfall eingesetzt werden.

Der Prototyp von RMARS hat sich für den Einsatz nun schon einige Monate bewährt. Demnach haben die Entwicklung von RMARS und die entsprechende Vorarbeit des Auffindens der Anforderungen ihren Zweck erfüllt.

# Literaturverzeichnis


[1] ISO/IEC 10731:1994: Information technology — Open Systems Interconnection — Basic Reference Model: Conventions for the definition of OSI services, 1994.

[2] S. Armstrong, A. Freier, and K. Marzullo. Multicast Transport Protocol, 1992. RFC 1301.

[3] Johann Blieberger, Johann Klasek, Alexander Redlein, and Gerhard-Helge Schildt. *Informatik*. Springer Verlag, third edition, 1996.

[4] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. A digital fountain approach to reliable distribution of bulk data. In *SIGCOMM*, pages 56–67, 1998.

[5] B. Callaghan, B. Pawlowski, and P. Staubach. NFS Network File System, Version 3, 1995. RFC 1813.

[6] D. Chiu, S. Hurst, M. Kadansky, and J. Wesley. Tram : A tree-based reliable multicast protocol, 1998.

[7] W. Fenner. IGMP Internet Group Management Protocol, Version 2, 1997. RFC 2236.

[8] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang. A reliable multicast framework for light-weight sessions and application level framing. 5(6):784–803, December 1997. IEEE/ACM Transactions on Networking.

[9] Wilhelm Gemoll. *Griechisch-Deutsches Schul- und Handwörterbuch*. G. Freytag Verlag, 1965.

[10] G. Grunsteidl and H. Kopetz. A reliable multicast protocol for distributed real-time systems, 1991.

[11] Franz-Joachim Kauffels. *Personal-Computer und lokale Netzwerke*. Markt und Technik Verlag, 1992.

[12] Alain Knaff. A description of the reliable multicast protocol used in udp-cast. http://udpcast.linux.lu/pipermail/udpcast/2005-December/000436.html, link checked at 2006-05-04 13:30.

[13] Hermann Kopetz. *Real-Time Systems – Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[14] John C. Lin and Sanjoy Paul. RMTP: A reliable multicast transport protocol. In *INFOCOM*, pages 1414–1424, San Francisco, CA, March 1996.

[15] In Wireless Networks. Qos-directed error control of video multicast.

[16] K. Obraczka. Multicast transport mechanisms: A survey and taxonomy. 36(1):94–102, January 1998.

[17] Alfred Olbrich. *Netze, Protokolle, Spezifikationen*. Vieweg Verlag, first edition, 2003.

[18] David A. Patterson and John L. Hennessy. *Computer Organisation and Design*. Morgan Kaufmann Publisher, 1998.

[19] J. Postel. Transmission Control Protocol, 1981. RFC 793.

[20] Walter E. Proebster. *Rechnernetze: Technik, Protokolle, Systeme, Anwendungen*. Oldenbourg Verlag, 1998.

[21] Terry Quinn-Andry and Kitty Haller. *Netzwerk-Design: die komplette Anleitung für den Aufbau professioneller Netzwerke*. Markt und Technik Verlag, 1998.

[22] K. Robertson, K. Miller, M. White, and A. Tweedly. StarBurst Multicast File Transfer Protocol (MFTP) Specification, April 1998. IETF.

[23] Werner Sinz. *Lokale Netze: ein Überblick*. Heise Verlag, 1996.

[24] T. Speakman, N. Bhaskar, R. Edmonstone, D. Farinacci, S. Lin, A. Tweedly, L. Vicisano, and J. Gemmell. PGM Reliable Transport Protocol Specification, 2001. RFC 3208.

[25] William Stallings. *Operating Systems: Internals and Design Principles*. Prentice Hall, third edition, 1998.

[26] W. Timothy Strayer, Bert Dempsey, and Alfred Weaver. *XTP: The Xpress Transfer Protocol*. Addison-Wesley, 1992.

[27] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, fourth edition, 2002.

[28] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems - Principles and Paradigms*. Prentice Hall, 2002.

[29] K. Tang, K. Obraczka, S. Lee, and M. Gerla. Reliable Adaptive Lightweight Multicast Protocol, 2003.

[30] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture, 2006. RFC 4251.

[31] O. Vorndran and F. Zotta. Regionale IT-Planung für Schulen, Materialien zur Entscheiderberatung, Verlag Bertelsmann, Gütersloh 2003.

[32] R. Varughese. Strategic Enterprise Management, An IT Manager's Desk Reference, Thomson Computer Press, 1997.

[33] W. Stangl. Die Einführung des Internets an Schulen am Beispiel Österreichs, in R. Groner und M. Dubi. Das Internet und die Schule, Huber, Bern, 2000.

[34] C. Stoll. Log Out, Warum Computer nichts im Klassenzimmer zu suchen haben und andere High-Tech-Ketzereien, S. Fischer Verlag GmBH, Frankfurt am Main, 2001.

[35] W. Stangl. Hemmende Rahmenbedingungen bei der Einführung des Internets an österreichischen Schulen, in A. Bruck und G. Geser. Schulen auf dem Weg in die Informationsgesellschaft, StudienVerlag, Innsbruck, 2000.

[36] R. Schulmeister. Grundlagen hypermedialer Lernsysteme, Theorie – Didaktik – Design, Addison Wesley Verlag, 1996.

[37] H. Schaumburg. Besseres Lernen durch Computer in der Schule? Nutzungsbeispiele und Einsatzbedingungen, in L. Issing und P. Klimsa. Information und Lernen mit Multimedia und Internet, Lehrbuch für Studium und Praxis, 3. vollständig überarbeitete Auflage, Beltz Verlag, 2002.

[38] Arbeitskreis Netzwerkeinsatz im Unterricht an beruflichen Schulen Hamburg: Initiative für eine Professionalisierung der pädagogischen und technischen Netzwerkbetreuung an Hamburger Schulen, PROFSYS, 1998. http://www.adminforum.de/service/profsys.html

[39] S. Papert. Revolution des Lernen, Kinder, Computer, Schule in einer digitalen Welt, im Original: The Childrens Machine, Basic Books, 1993.

[40] R. Nolan. The Stages Theory: A Framework for IT Adoption and Organizational Learning, Harvard Business School, 1993.

[41] R. Nolan. Managing the Computer Resource: A Stage Hypothesis, Communications of the ACM. Vol. 16 (1973), No. 7, S. 399-405.

[42] M. Nadenau. Wer betreut das Schul-Intranet? Gedanken zu einer didaktisch begründeten Netzwerkadministratoren durch Lehrerinnen und Lehrer, Zeitschrift Computer und Unterricht, Nr. 39, 2000

[43] F. Mayrhofer. Betreuung des Rechnerbetriebs an (bayer.) Schulen, Entwicklung, Situation, Misere, dringende Konsequenzen, April 1998, http://www.weihenstephan.org/ mayrhoff/rebe.htm

[44] T. DeMarco und T. Lister. Wien wartet auf Dich!, Der Faktor Mensch im DV-Management, 2. Auflage 1999, im Original: Peopleware, Productive Projects and Teams Dorset House Publishing Company, Incorporated.

[45] IDC: Understanding the Total Cost and Value of Integrating Technology in Schools, An IDC White Paper Sponsored by Apple Computer, Inc., 1997.

[46] Gartner: TCO Analyst – A white paper on Gartner Group's Next Generation Total Cost of Ownership Methodology, Bericht No. Project 17013571, Gartner Consulting 1997.

[47] Gartner: Total Cost of Ownership: reducing PC/LAN Costs in the Enterprise, Bericht No. R-TCO-104, Gartner Consulting 1996.

[48] A. Bruck und G. Geser. Schulen auf dem Weg in die Informationsgesellschaft, StudienVerlag, Innsbruck, 2000.

[49] R. Mittermeier (Hrsg.). From Computer Literacy to Informations Fundamentals, International Conference on Informatics in Secondary Schools – Evolution Perspectives, ISSEP 2005, Proceedings, ISBN 0302-9743

[50] C. Borchel, L. Humbert, M. Reinertz. Design of an Informatics System to Bridge the Gap Between Using and Understanding in Informatics, in R. Mittermeier (Hrsg.). Informatics in Secondary Schools. Evolution and Perspectives, Klagenfurt, 30th March to 1st April 2005. Wien : Ueberreiter Verlag, pp. 53–63.

[51] T. van Weert (Hrsg.), R. Munro (Hrsg.). Let's teach informatics – empowering pupils, students and teachers in T. van Weert (Hrsg.), R. Munro (Hrsg.). Informatics and the Digital Society – Proceedings of Conference on Social, Ethical and Cognitive Issues of Informatics and ICT, Norwell, Massachusetts : Kluwer Academic

Publishers, April 2003. (Open IFIP-GI-Conference – July 22–26, 2002, University of Dortmund, Germany), pp. 141–147.

[52] A. Schwill (Hrsg.). Grundkonzepte der Informatik und ihre Umsetzung im Informatikunterricht. in A. Schwill. Informatik und Schule – Fachspezifische und fachübergreifende didaktische Konzepte. Springer Verlag Berlin, 1999. (Informatik aktuell), S. 175–189.

[53] Fessl, Gansterer. Konzeption einer Fortbildungsveranstaltung: eLearning by Doing – Moodle im eigenen Unterricht einsetzen.

[54] Gansterer. Moodle in der Lehrerausbildung – Ein Erfahrungsbericht aus der Praxis, Beitrag, 3. internationale Moodle-Konferenz in Österreich, September 2006, FH-Campus Hagenberg.