ETH zürich

ETH Library

CeNet

AppleTalk-kompatible Kommunikations-Software für CERES

Report

Author(s):
Müller, Markus

Publication date:
1989

Permanent link:
https://doi.org/10.3929/ethz-a-000524686

Rights / license:
In Copyright - Non-Commercial Use Permitted

Originally published in:
ETH, Eidgenössische Technische Hochschule Zürich, Departement Informatik 114

This page was generated automatically upon download from the ETH Zurich Research Collection.
For more information, please consult the Terms of use.

PC. 89. 114

ETH

Eidgenössische
Technische Hochschule
Zürich

Departement Informatik
Fachgruppe
Kommunikationssysteme

Markus Müller

CeNet: AppleTalk - kompatible Kommunikations-Software für CERES

Oktober 1989

Eidg. Techn. Hochschule Zürich
Informatikbibliothek
ETH-Zentrum
CH-8092 Zürich

114

90.01.10
6520

Adresse des Autors:

Institut für Technische Informatik und Kommunikationsnetze
ETH-Zentrum
CH-8092 Zürich, Switzerland

© 1989 Departement Informatik, ETH Zürich

CeNet:

AppleTalk-kompatible Kommunikations-Software

für CERES

Markus Müller

Zusammenfassung

Der vorliegende Bericht beschreibt die Implementierung der AppleTalk-Protokolle auf Ceres unter Medos. Der erste Teil dieses Berichts erklärt die Struktur dieser Software und behandelt die verwendeten Techniken, um diese Software einerseits effizient, anderseits aber fehlerfrei, übersichtlich und wartungsfreundlich zu gestalten. Im zweiten Teil folgt die detaillierte Beschreibung der einzelnen Komponenten von CeNet; dieser Teil dient, zusammen mit dem Anhang, gleichzeitig als Benutzerhandbuch.

Abstract

This report describes the implementation of the AppleTalk protocols on Ceres under Medos. The first part of the report shows the structure of the software and discusses methods used to get efficient, error free, simple and easily maintainable software. The second part contains the detailed descriptions of the various components of CeNet; together with the Appendix, the second part also serves as a user manual for CeNet.

Inhaltsverzeichnis

Zusammenfassung 3

Abstract 3

A. Architektur und Implementationstechnik von CeNet 7

1. Einführung in CeNet 7

2. Ueberblick über die Architektur von CeNet 8
- 2.1. Einleitung 8
- 2.2. Tiefere Schichten und Netzwerkschicht 8
- 2.3. Zugang zu Diensten via Namen 9
- 2.4. Transportschicht 9
- 2.5. Höhere Schichten 10
- 2.6. Die Toolbox 10

3. Implementationskonventionen 10
- 3.1. Einleitung 10
- 3.2. Namensgebung 11
- 3.3. Modul-Dokumentation 11
- 3.4. Effizienz 12
  - 3.4.1. Down- und Upcalls 12
  - 3.4.2. Minimieren des Kopierens innerhalb des Speichers 12
  - 3.4.3. Analysieren und Erzeugen von AppleTalk-Paketen 12
  - 3.4.4. Effizienz der Multi-Prozess-Umgebung 13
- 3.5. Locking-Strategie 14
- 3.6. Terminierung von Programme. 14
- 3.7. Fehlerbehandlung 15
- 3.8. Benutzerschnittstelle der tieferen Schichten 16
- 3.9. Benutzerschnittstelle der höheren Schichten 16
- 3.10. Konventionen für Assembler-Programme 16

4. Realisierte Protokolle 16

5. Realisierte Anwendungen 17
- 5.1. LaserWriter Channel Module für laraprint 17
- 5.2. Software, die auf CeresNet basiert 17

6. Verfügbarkeit der Software 17

7. Rückblick 18

8. Verdankungen 18

9. Literatur 19

B. Benutzerhandbuch **20**

1. CNTools: Die Toolbox **20**
- 1.1. Funktionen zur Handhabung von Datenpaketen 20
- 1.2. Weitere Funktionen der Toolbox 21

2. CNScc: Der Zugang zur Hardware **22**
- 2.1. Zeitkritische Operationen 22
- 2.2. Grundlagen von ALAP 22
  - 2.2.1. Die Knotennummer (Node Number) 22
  - 2.2.2. Allgemeines Format eines ALAP-Paketes 23
  - 2.2.3. ENQ/ACK: Suchen einer freien Knotennummer. 24
  - 2.2.5. Flusskontrolle im RTS/CTS-Protokoll 26
- 2.3. Aufgabenteilung zwischen CNScc und CNAlap 27
- 2.4. Verwaltung der Empfangs-Puffer 28
- 2.5. Modul-Beschreibung 30

3. CNAlap: Der Zugang zum Netzwerk auf tiefster Stufe **31**
- 3.1. Funktion von CNAlap 31
- 3.2. Modul-Beschreibung 31
- 3.3. Initialisierung CNAlap 33

4. CNDdp: Kommunikation in einem Netzwerkverbund **33**
- 4.1. Funktion 33
- 4.2. Modul-Beschreibung 34

5. CNNbp: Symbolische Namen für Netzwerk-Adressen **36**
- 5.1. Funktion 36
- 5.2. Modul-Beschreibung: 36

6. CNMpp: Interne Hilfs- und Verwaltungsfunktionen **38**
- 6.1. Uebersicht 38
- 6.2. RTMP-Stub 38
- 6.3. Echo-Protokoll 39
- 6.4. Modul-Beschreibung 39

7. CNAtp: Ein einfaches Transaktions-Protokoll **40**
- 7.1. Funktion 40
- 7.2. Modul-Beschreibung: Master-Funktionen 41
- 7.3: Modul-Beschreibung: Server-Funktionen 43

8. CNCeresNet: Gesicherte Verbindungen **44**
- 8.1. Funktion 44

8.2. Modul-Beschreibung 44
8.3. Der interne Modul CNPackets 44

9. CNPapMaster: Zugang zu AppleTalk-basierten Printern 45
9.1. Funktion 45
9.2. Modul-Beschreibung 45
9.3. Ansprechen des LaserWriter 46

10. CNZipInfo: Aufteilung des Netzwerkverbundes in Namenszonen 46
10.1. Funktion 46
10.2. Modul-Beschreibung 47

11. Nicht implementierte Protokolle 47

12. Testprogramme 48
12.1. Modul-Testprogramme 48
12.2. Peek und Poke auf Macintosh 48
12.3. CNMonitor, CNMonPrint, CNSpy 48

Appendix

A. Paketformate 49
A.1. Grundstruktur eines Datenpaketes 49
A.2. Datagram Delivery Protocol (DDP) 50
A.3. Name Binding protocol (NBP) 51
A.4. AppleTalk Transaction Protocol (ATP) 52
A.5. Routing Table Maintenance Protocol (RTMP), Workstations 53
A.6. Zone Information Protocol (ZIP), Workstation part 54
A.7. Echo Protocol 55
A.8. Printer Access Protocol (PAP) 56

B. Modulstruktur 57

C. Geschwindigkeitsmessungen 58

A. Architektur und Implementationstechnik von CeNet

1. Einführung in CeNet

Am Institut für Informatik wurde im Zeitraum 1984 bis 1986 ein neuer Arbeitsplatzrechner, die *Ceres* [Eber87], entwickelt, und das Betriebssystem der *Lilith*, *Medos*, darauf portiert. Für die Datenkommunikation verfügt die Ceres-Hardware über zwei RS422-Schnittstellen, die intern vom Baustein Zilog 8530 SCC bedient werden. Die Hardware der Ceres erfüllt damit die Voraussetzungen für den Betrieb eines Netzwerkes gemäss AppleTalk-Spezifikation; das immer grösser werdende Angebot von AppleTalk-kompatiblen Peripheriegeräten (z.B. Apple *LaserWriter*) und Gateways (z.B. Kinetics *KFPS* für Ethernet) hat uns daher bewogen, diesen de-fakto Standard auch auf der Ceres zu implementieren.

AppleTalk ist speziell für ein low-cost LAN konzipiert, mit der ursprünglichen Aufgabe, teure Peripheriegeräte (Laserdrucker, File-Archiv etc.) auf einfache Art mehreren Macintosh-Benutzern zugänglich zu machen. Die Topologie von AppleTalk ist ein Bus; als physikalisches Uebertragungsmedium dient eine Zweidrahtleitung, an die einzelne Geräte über einen Adapter (Transformator) angeschlossen werden.

Die durch AppleTalk definierten Protokolle sind nicht international standardisiert, gewinnen aber zunehmend an Bedeutung durch die grosse Verbreitung von Macintosh-Produkten einerseits und durch die Zusammenarbeit von Apple Inc. mit anderen führenden Rechnerherstellern anderseits.

An dieser Stelle im Detail auf AppleTalk und dessen Protokolle einzugehen würde den Rahmen dieser Beschreibung der Ceres-spezifischen Implementation sprengen. Die genauen Spezifikationen sind in [Apda86] ersichtlich; sehr detaillierte Informationen finden sich aber auch in [Appl85]. Die hier folgende Beschreibung beschränkt sich auf die Implementation von AppleTalk auf Ceres und die dem Benutzer zur Verfügung stehenden Schnittstellen.

CeNet ist ähnlich aufgebaut wie die Implementation von AppleTalk auf dem Macintosh, weist aber einige wichtige Differenzen auf:

- CeNet ist auf *Modula-2* einerseits und auf das Multi-Process-System *Medos* anderseits ausgerichtet. Den verschiedenen Komponenten entsprechen (im allgemeinen) jeweils ein Modul. Programmaufrufe sind je nach Modul entweder nur synchron oder nur asynchron möglich. Die wegen ihrer Universalität sehr komplexen Datenstrukturen der Macintosh-Lösung wurden nicht übernommen; an deren Stelle treten funktionsspezifische, einfache und möglichst anwenderfreundliche Strukturen.

- Die Benutzung von Assembler für zeitkritische Operationen, die sich in der Implementation auf dem Macintosh nicht nur auf die tiefsten

Schichten beschränkt, wurde auf ein absolutes Minimum reduziert. Dies, zusammen mit dem Verzicht auf Aufrufe von Modula-Prozeduren aus Hardware-Interruptroutinen heraus, hat weitgehende Auswirkungen vor allen auf die Implementation der untersten Schichten von CeNet, isoliert aber anderseits die Hardware besser vom Benutzer.

2. Ueberblick über die Architektur von CeNet

2.1. Einleitung

Dieses Kapitel gibt einen Ueberblick über die Architektur von *CeNet* als ganzes, sowie die Zugangsmöglichkeiten zu den verschiedenen Funktionen.

CeNet ist eine in Schichten strukturierte Kommunikations-Software. Sie gliedert sich grob in:

a) Untere Schichten: Netzwerk-Protokolle (entspricht ISO/OSI Schicht 1 bis 3)

b) Mittlere Schichten: Transport-Protokolle (entspricht ISO/OSI Schicht 4)

c) Obere Schichten: anwendungsorientierte Protokolle (entspricht ISO/OSI Schicht 6 und 7)

d) Hilfs- und Verwaltungsroutinen

2.2. Tiefere Schichten und Netzwerkschicht

Die unteren Schichten sind die systemnahen Teile der Kommunikations-Software. Ihre Benutzerschnittstelle zeichnet sich aus durch:

- direkte Zugriffe auf die System-Puffer mittels Primitiven
- Empfang von Paketen durch Aktivierung einer Empfangs-Prozedur

Aufgeteilt ist diese Schichtengruppe in folgende Schichten:

- *CNScc*: Zugang zur Hardware. In Assembler geschrieben da extrem zeitkritisch
- *CNAlap*: ALAP Link Access Protocol. Layer 1 und 2 des ISO/OSI-Referenzmodells. Behandelt einzelnes Netzwerk
- *CNDdp*: DDP Datagram Protocol. Layer 3 gemäss ISO/OSI. Behandelt Netzverbund
- *CNNbp*: Name Binding Protocol. Zugang zu Diensten via Namen und Typ
- *CNMpp*: Diverse, für den Benutzer nicht sichtbare Dienste wie Erkennen der eigenen Netznummer und das Echo-Protokoll

2.3. Zugang zu Diensten via Namen

Auf Stufe Netzwerkschicht werden Dienste über 32-bit-Adressen angesprochen. Diese Adressen werden beim Aufstart eines Dienstes dynamisch zugewiesen, können sich also bei jedem Neustart des Dienstes ändern und sind daher nicht als feste Adressen für einen Dienst geeignet.

Aus diesem Grund wird festen Diensten ein (textueller) Name zugewiesen; zum Suchen eines Dienstes im Netz steht das *Name Binding Protocol* (*NBP*) zur Verfügung. Sehr grosse Netzwerke können in mehrere mit Namen ansprechbare Teilbereiche, den Zonen, unterteilt werden; mit dem *Zone Information Protocol* kann u.a. auch eine Liste der vorhandenen Zonen sowie der eigene Zonennamen abgefragt werden. Innerhalb CeNet werden diese Protokolle durch

- *CNNbp*: Zugang zu Diensten via textuellen Namen
- *CNZipInfo*: Abfrage der Liste vorhandener Zonen und Name der eigenen Zone

angeboten. Die Benutzerschnittstellen zeichnen sich dabei aus durch:

- Senden und Empfang aus / in vom Benutzer bereitgestellten (meist statischen) Puffern
- Synchrone Ausführung: ein Prozess wartet die Rückantworten ab

2.4. Transportschicht

Für die Transportschicht (entsprechend ISO/OSI Layer 4) stehen mehrere Protokolle zur Verfügung. Hauptaufgabe dieser Schicht ist das Anbieten einer sicheren Datenübermittlung, d.h. eine Uebermittlung, die entweder die Daten sicher zum Partner übermittelt, oder, wenn dies wegen Ausfalls des Partners oder des Uebertragungsmediums nicht mehr möglich ist, den Fehler zurückmeldet. Die Transportschicht stützt sich seinerseits auf die Dienste von CNDdp für die Daten-Uebermittlung. Ihre Benutzerschnittstelle zeichnet sich aus durch:

- Senden und Empfang aus / in vom Benutzer bereitgestellten (meist statischen) Puffern
- synchroner Empfang von Datagrammen: ein Prozess wartet ankommende Daten ab.
- Verwendung von 32-Bit-Adressen, dessen genaue Bedeutung der Benutzer (im allgemeinen) nicht kennen muss, da ihm für die Uebersetzung von Namen in Adressen das Name Binding Protocol (NBP) zur Verfügung steht.

Bisher wurden zwei Vertreter dieser Gruppe implementiert: *CNAtp* und *CNCeresNet*. Das Modul *CNAtp* implementiert das *ATP*-Protokoll, ein Transaktionsprotokoll nach dem Master-Server-Prinzip. Die im Protokoll festgelegten Längenbeschränkungen sowie das Fehlen einer Link-

Ueberwachung, die einen allfälligen Ausfall des Partners erkennen würde, begrenzen leider dessen Einsatzfähigkeit. Ueberwachte Links und gesicherte Uebermittlung von Datengrammen werden durch den Modul *CNCeresNet* angeboten. CNCeresNet basiert auf der an der ETH entwickelten *CeresNet*-Software [Pesch87] und verfügt über eine dazu kompatible Benutzerschnittstelle. Netzseitig sind aber CNCeresNet und CeresNet nicht miteinander kompatibel. CNCeresNet ist selber kein im Rahmen von AppleTalk spezifiziertes Protokoll und setzt seinerseits direkt auf CNAlap (nicht aufCNDdp ; somit ist ein Betrieb in einem Netzwerkverbund nicht möglich) auf.

Zwei weitere verbindungsorientiertes Transportprotokolle, das *AppleTalk Session Protocol* (*ASP*) und *AppleTalk Data Stream Protocol* (*ADSP*), sind mittlerweise ebenfalls durch die AppleTalk-Spezifikationen festgelegt. Auf *Ceres* ist aber keines dieses Protokolle implementiert.

2.5. Höhere Schichten

Die höheren Schichten sind anwendungsorientiert (Schicht 6 bis 7 gemäss ISO/OSI). Momentan als einziger implementierter Vertreter dieser Schicht sei *CNPap* (Printer Access Protocol) genannt, der Zugang schafft zu der grossen Klasse von AppleTalk-Printern, insbesondere zum *LaserWriter*.

Innerhalb AppleTalk wurde für Anwendungen, die *External Files* verwenden, ein weiteres Protokoll definiert: das *AppleTalk File Protocol* (*AFP*), dessen Inhalt neben Zugriff auf externe Files auch File Locking, Directory-Operationen in flachen oder hierarchischen Directories, Handhabung von Ikonen, Benutzer-Authentifizierung und weiteres enthält. Das sehr mächtige Protokoll ist unter anderem auch in der Lage, stark unterschiedliche File-Systeme miteinander zu verbinden (gegenwärtig sind External Files für Macintosh, IBM PC, VAX/VMS und VAX/UNIX und SUN verfügbar); seine grosse Komplexität spricht jedoch gegen einen Einsatz auf Ceres.

2.6. Die Toolbox

Die Toolbox *CNTools* stellt zahlreiche Werkzeuge zur Verfügung, die in allen Komponenten von CeNet einsetzbar sind. Dies umfasst:

- Konversion von 16- oder 32-bit-Grössen auf Netzwerk-Standard (high byte first)
- Primitiven für Verwaltung von Datenpuffern
- Random Number Generator

3. Implementationskonventionen

3.1. Einleitung

Komplexe Probleme können auf viele Arten gelöst werden. Un einerseits diese Vielfalt etwas einzudämmen, die Programme übersichtlich zu gestalten und von Anfang an zu möglichst fehlerfreien Programmen zu gelangen, wurde ein Satz von Regeln festgelegt. Bei Aenderungen oder

Erweiterungen der CeNet-Software sollte man sich nach Möglichkeit daran halten.

3.2. Namensgebung

Um zusammengehörende Module leichter zu erkennen, beginnen ihre Name alle mit "CN", gefolgt von einer Kurzbezeichnung des Protokolls, das sie implementieren, eventuell noch gefolgt von einer Angabe, welche Untermenge von Funktionalitäten innerhalb eines Protokolls unterstützt werden. Aus Lesbarkeitsgründen werden dabei bei der Kurzbezeichnung des Protokolls nur der erste Buchstaben gross geschrieben. Beispiel: *CNDdp* für *DDP*, *CNPapMaster* für *PAP* Master Implementation. Einzige Ausnahme: *CNCeresNet* wird, zwecks Kompatibilität, unter dem Namen *CeresNet* entwickelt.

3.3. Modul-Dokumentation

Alle Definitions-Module sind in Anlehnung an [Se87] ausgiebig dokumentiert. Die Dokumentationssprache für die Programme ist Englisch. Jedes Definitions-Modul umfasst mindestens die folgende Dokumentation:

- Standardisierter Modulkopf
- Aenderungsgeschichte
- Zweck des Moduls
- Wichtige Informationen, die das Modul als Ganzes betreffen

Die Import-Liste jedes Moduls macht neben den importierten Elementen auch dessen Typen ersichtlich. Innerhalb des Definition Modules werden die Deklarationen nach Konstanten, Typen, Variablen und Prozeduren sortiert aufgeführt.

Mindestens pro Prozedur werden folgende Angaben gemacht:

- Zweck / Funktion der Prozedur
- Parameterbeschreibung (inkl. Art des Parameters)
- Vorbedingungen der Prozedur, mit Angabe der Auswirkungen einer nicht erfüllten Vorbedingungen
- Nachbedingungen der Prozedur, soweit nicht im Zweck der Prozedur genügend präzis umschrieben
- Ev. weitere wichtige Hinweise zur Benutzung

Falls sinnvoll, werden auch Typen und Variablen und deren Gebrauch nach diesem Schema beschrieben; im allgemeinen reicht hier aber ein kurzer Kommentar direkt neben der Deklaration.

3.4. Effizienz

3.4.1. Down- und Upcalls

Kommunikations-SW muss möglichst effizient laufen. Deshalb wurden die Schnittstellen zwischen den verschiedenen Schichten der Software mit Unterprogramm-Aufrufen (im Gegensatz zu Message Passing) realisiert. Hier entsteht das Problem, dass Modula-2 nur Aufrufe von hierarchisch höheren nach tiefer liegenden Modulen erlaubt, während in der Kommunikations-SW sowohl höhere Schichten niedrigere aktivieren (Daten senden) als auch umgekehrt (Daten empfangen). Die einzige Möglichkeit, Aufrufe in beide Richtungen zu implementieren, geht via "Up-Call": Die höhere Schicht übergibt durch Aufruf einer Installationsprozedur die eigenen Prozeduren als Prozedur-Parameter an die tiefere Schicht, wo diese in Prozedur-Variablen abgelegt werden. Durch Aufruf via diese Prozedur-Variablen kann die tiefere Schicht nun Prozeduren der höheren Schicht aufrufen.

3.4.2. Minimieren des Kopierens innerhalb des Speichers

Kopieren innerhalb des Speichers ist zwar schnell (ca. 500 kByte / sec für Ceres-I mit 32032-Prozessor), bewirkt aber im Vergleich zur Uebertragungsleistung der Netzwerk-Hardware (ca. 40 kByte / sec) einen messbaren Leistungsverlust (ca. 8 % für einfaches Kopieren). Aus diesem Grund sollten die Daten vom Puffer des Benutzerprogrammes bis zur Uebermittlung auf das Netzwerk möglichst wenig umkopiert werden. Als obere Grenze wurde maximal eine Kopie festgelegt.

3.4.3. Analysieren und Erzeugen von AppleTalk-Paketen

Jedes AppleTalk-Paket besteht aus einer Serie von Headers, gefolgt vom eigentlichen Datenblock. Jedem Header (oder Gruppe von Headers) entspricht eine bestimmte Schicht der Kommunikations-Software; der vordeste Header entspricht dabei der tiefsten (hardware-nahesten) Schicht. Dieser vordeste Header hat eine fixe Länge und Semantik; diejenige der weiteren Header lässt sich aus dem Inhalt des vorhergehenden Header ableiten.

Verwendet wird daher folgende Realisierung:

1. Verwendung verketteter linearer Puffer auf unterster Stufe.
   Hauptvorteil: Einfachheit und Effizienz an der zeitkritischsten Stelle, der Ansteuerung der Netz-Hardware

   Andere Möglichkeiten wären zirkuläre Puffer mit und ohne Wraparound am oberen Ende des Pufferbereichs. Diese Strukturen sind bedeutend sparsamer im Speicherverbrauch, bedingen aber komplexere und damit (zu) langsame Software.

2. Parsing ankommender Pakete: Ein empfangenes Paket wird von links nach rechts aufgelöst, indem man den jeweils nächsten Header in eine (statische) Datenstruktur kopiert und dort auswertet. Aufgrund dieser Auswertung wird dann die nächst höhere SW-Schicht aktiviert,

die analog fortfährt. Für dieses Zerlegen des Paketes steht im Modul CNTools die Prozedur *Get* zur Verfügung.

Alternativ dazu könnte der Puffer auch durch Ueberlagern ganzer Datenstrukturen direkt über den Pufferbereich analysiert oder aufgebaut werden. Damit wären aber alle Tests auf genügend vorhandene Daten in die einzelnen Programme verstreut, während in unserer Lösung die meisten Tests in der Prozedur *Get* zentral erfolgen.

3. Ein zu sendendes Paket wird von rechts nach links in einem statischen Puffer aufbereitet. Die oberste beteiligte SW-Schicht stellt dabei diesen Puffer bereit und kopiert dann die Benutzerprogramm-Daten an das obere Ende des Puffers. Weitere Headers werden den bereits eingelagerten Daten vorangestellt. Für diesen Aufbau eines Paketes steht ebenfalls im Modul CNTools die Routine *Put* zur Verfügung.

   Durch die Verwendung der Routinen Get und Put von CNTools werden die Daten beim Aufbau und Analysieren des Puffers jeweils einmal innerhalb des Speichers umkopiert. Somit ist unsere Forderung nach minimalem Umkopieren der Datem innerhalb des Speichers erfüllt, sofern nicht noch innerhalb der einzelnen Module grössere Datenbereiche umkopiert werden.

4. Die Modula-2 Typen CARDINAL und LONGINT werden Ceres-intern als Lo-Byte-First dargestellt; bei Uebertragung auf dem Netz müssen sie in die Darstellung Hi-Byte-First überführt werden, sobald das Feld in einem nicht Ceres-spezifischen Protokoll wie z.B. CeresNet enthalten ist. Dazu stehen die Routinen *PrepInt* und *PrepLong* aus CNTools zur Verfügung, welche die Darstellung direkt innerhalb der entsprechenden Variablen umwandelt.

3.4.4. Effizienz der Multi-Prozess-Umgebung

Als einfachste Primitive zur Synchronisation mehrerer Prozesse bietet Medos die Prozedur *Pause* an; diese gibt die CPU zugunsten anderer Prozesse ab, die bereit zum Rechnen sind. Durch wiederholten Aufruf von Pause kann ein Prozess, der auf ein Ereignis warten will, solange auf die CPU verzichten, bis das Ereignis eintritt.

Diese Lösung hat den Vorteil, dass beliebig komplexe Bedingungen für das Warten möglich sind, und dass beliebige Prozesse, also auch Interrupt-Prozesse, auf diese Art einem wartenden Prozess ein Ereignis melden können. Der grosse Nachteil: der wartende Prozess bleibt rechenbereit und erhält entsprechend oft die CPU zugewiesen; diese zahlreichen Prozessumschaltungen verbrauchen erheblich Zeit. Deshalb soll Pause ausschliesslich dort benutzt werden, wo keine Alternative besteht (Interrupt-Routine, die einen Medos-Prozess starten will) oder wo eine Wartesituation nur selten eintritt. In allen übrigen Fällen sollen die Medos-Primitiven *Priority Module*, *Signal*, *Send*, *Wait* und *TimedWait* verwendet werden.

Alternativ könnte anstelle von Pause auch *Delay(1)* aufgerufen werden; der so pausierende Prozess wird erst beim nächsten Tick des Real Time Clocks

wieder aktiviert. Dieser Clock tickt aber nur alle 20 ms (gleich 50 Ticks pro Sekunde), und das ist für gewüschte Datenraten von 100 und mehr Paketen pro Sekunde zu wenig.

3.5. Locking-Strategie

Die CeNet-Software lebt in einer Multiprozess-Umgebung und besteht selber aus mehreren internen Prozessen; deshalb ist gegenseitiger Ausschluss von Prozessen beim Zugriff auf gemeinsame Daten ebenso unerlässlich wie aktive Synchronisation zwischen Prozessen.

Medos unterstützt für gegenseitigen Ausschluss die Prozess-Priorität, die mittels Priority Modules beeinflusst werden kann. An dieser Stelle sei nur auf die für uns wichtigen Eigenschaften hingewiesen: Alle Prozesse mit Priorität ungleich Null laufen mit "non-preemptive scheduling"; werden also vom Scheduler nicht selbstständig zugunsten eines anderen Prozesses unterbrochen. Somit ist gegenseitiger Ausschluss beim Zugriff auf gemeinsame Daten garantiert. Interrupts kommen bei niederen Prozess-Prioritäten (1 bis 7) weiterhin durch. Aktive Synchronisation zwischen Prozessen erfolgt mit den Prozeduren Pause, Send, Wait, TimedWait und Delay.

Sollen Interrupts ebenfalls am Durchkommen gehindert werden, muss die Priorität auf den höchstmöglichen Wert (15) angehoben werden. Dem SCC-Interrupt ist formell die Prioritäts-Ebene 14 zugeordnet; da dieser Interrupt aber extrem zeitkritisch ist, wird er von Medos nie blockiert. Gegenseitiger Ausschluss gegenüber dem SCC-Prozess wird daher anders bewerkstelligt: die Variable *recMode* innerhalb des Moduls CNScc zeigt dem Interrupt-Prozess an, ob auf gemeinsame Daten zugegriffen werden darf. Findet der Interrupt-Prozess diese Variable mit einem anderen Wert als *recAll* vor, wird die Interrupt-Behandlung vorzeitig beendet, ohne dass auf gemeinsame Daten zugegriffen wird.

3.6. Terminierung von Programme.

Medos ist sowohl ein Mehr-Prozess-System als auch ein Mehr-Programm-System: jedes Programm hat die Möglichkeit, ein weiteres Programm (u.a. auch einen Command Interpreter) zu starten, und ein so gestartetes Programm darf wiederum eigene parallele Prozesse erzeugen. Von mehreren Programmen gleichzeitig benutzte Module werden, analog zu parallelen Prozessen, gemeinsam benutzt.

Aus Platzgründen kann an dieser Stelle nicht auf alle Besonderheiten dieser Mehr-Programm-Umgebung eingegangen werden; für und wichtig sind vor allem die anfallenden "Aufräume-Arbeiten", wenn ein parallel laufendes Programm terminiert, ohne sich vorher von CeNet abzumelden:

- Alle zum terminierenden Programm gehörenden Prozesse werden sofort aus der Prozesstabelle von Medos entfernt. Sobald CeNet von einem solchen Prozess eine Antwort erwartet, besteht die Gefahr eines Deadlocks.

- Der Programm-Code des terminierenden Programms wird aus dem Speicher entfernt. Somit werden die Inhalte von Prozedurvariablen,

die Prozeduren des terminierenden Programms referenzieren, ungültig und dürfen nicht mehr länger verwendet werden.

- Der globale Variablenbereich und aller dynamisch alloziierte Speicherbereich des terminierenden Programms wird (im allgemeinen) freigegeben. Betroffene Pointers dürfen somit nicht mehr weiter benutzt werden.

- Vom terminierenden Programm belegte Resourcen innerhalb CeNet wie Tabellen-Einträge müssen wieder freigegeben werden.

Medos bietet für diese "Aufräum-Arbeiten" Terminations-Prozeduren an, die ausgeführt werden, sobald ein parallel ablaufendes Programm terminiert. Angemeldet wird eine solche Terminations-Prozedur durch Aufruf von *TermProcedure* innderhalb des Moduls Processes; sobald ein Programm terminiert, wird die angemeldete Prozedur aktiviert. Programme selber werden unter Medos über deren Programm-Ebene identifiziert; die Programm-Ebene des zum laufenden Prozess gehörenden Programms kann mit der Prozedur *CurrentLevel* aus Processes abgefragt werden.

Um die im Modul Processes vorhandenen (zu kleinen) Tabellen nicht unnötig zu belasten, wird die Prozedur *TermProcedure* mit identischer Funktionalität nochmals in CNTools angeboten.

3.7. Fehlerbehandlung

Fehler werden in vier Kategorien eingeteilt und entsprechend behandelt:

- Fehler durch falsche Benutzung (z.B. Arbeiten mit einem nicht eröffneten Socket)

- Schwere Fehler, die nicht durch den Benutzer verschuldet sind, aber eine Fortsetzung des Programmes sinnlos machen (z.B. CNDdp kann intern den Receiver-Prozess nicht starten)

- Fehler durch nicht vorhersehbare äussere Einflüsse, die jedoch eine sinnvolle Fortsetzung des Programms erlauben (z.B. Partner gibt keine Antwort mehr), oder bewusst erzeugte Fehler (z.B. lokales Abbrechen einer Transaktion)

- Fehler in einem empfangenen Paket, die auf Hardware- oder Softwareprobleme in der absendenden Station schliessen lässt

In den ersten beiden Fällen wird eine Fehlermeldung (via Modul Terminal) ausgegeben und daraufhin das Programm mit HALT abgebrochen. Im dritten Fall wird ein entsprechender Fehlerstatus zurückgegeben. Im letzten Fall wird lediglich eine Fehlermeldung auf dem Schirm ausgegeben.

Durch diese Unterscheidung hat jede Prozedur innerhalb von CeNet einen genau definierten (kleinen) Satz von möglichen Fehlercodes, und der Benutzer ist von der Behandlung von Fehlern, auf die er keinen Einfluss hat, entlastet.

3.8. Benutzerschnittstelle der tieferen Schichten

Die Benutzerschnittstellen der tieferen Schichten (*CNAlap, CNDdp*) zeichnen sich aus durch:

- Arbeit direkt auf den Datenpaketen
- Senden von Daten nicht blockierend: nach dem Abschicken der Daten treten keine längerdauernden Wartezustände ein
- Empfang von Daten über Handler-Prozeduren, die bei Eintreffen eines Datenpaketes aktiviert werden

3.9. Benutzerschnittstelle der höheren Schichten

Die Benutzerschnittstelle der höheren Schnittstellen (*CNNbp, CNAtp, CNPap*) zeichnen sich aus durch:

- Senden von Daten durch Aufruf von Prozeduren. Im allgemeinen wird dabei eine Empfangsbestätigung (Acknowledge) der Gegenstation abgewartet; trifft diese nicht innerhalb einer gewissen Zeit ein, wird das Datenpaket nochmals übermittelt. Nach einer bestimmten Anzahl erfolglosen Versuchen wird ein Uebermittlungsfehler zurückgemeldet. Die Prozeduren zum Senden von Daten sind daher im allgemeinen blockierend; es existiert aber ein Zeitlimit, nachder der Wartezustand verlassen wird.
- Empfang von Daten ebenfalls durch Aufruf von Prozeduren. Es wird gewartet bis entweder Daten anliegen oder bis die zugehörige Abbruch-Prozedur (durch einen parallelen Prozess) aufgerufen wird.
- Uebergabe von Daten in statischen Puffern, die als Parameter übergeben werden
- Möglichkeit, Prozeduraufrufe, die Daten empfangen wollen und somit beliebig lange Wartezustände erzeugen können, durch den Aufruf einer Abbruch-Prozedur vorzeitig zu beenden. Die Benutzung solcher Abbruch-Prozeduren setzt dabei die Benutzung paralleler Prozesse voraus.

3.10. Konventionen für Assembler-Programme

Um auch in Assemblerprogrammen die Uebersichtlichkeit zu wahren, wurden einige Konventionen betreffend globaler Belegung von Registern mit häufig gebrauchten Werten, Aufruf von lokalen Unterprogrammen, Uebergabe von Parametern und und andere spezielle Verwendung von Registern der 32032-CPU aufgestellt. Die exakten Regeln sind jeweils als Kommentar im Kopf der Assembler-Programme aufgeführt.

4. Realisierte Protokolle

Die CeNet-Software realisiert in der gegenwärtigen Form folgende Protokolle oder Untermengen von Protokollen:

- Medium Access / AppleTalk Link Access Protocol (ALAP)
- Datagram Delivery Protocol (DDP)
- Name Binding Protocol (NBP)
- Routing Table Maintenance Protocol (RTMP; nur Stub)
- Echo Protocol (nur Serverseite)
- Zone Information Protocol (ZIP; nur ATP-Style Masterseite)
- AppleTalk Transaction Protocol (ATP)
- Printer Access Protocol (PAP; nur Masterseite)
- CeresNet-Emulation

5. Realisierte Anwendungen

5.1. LaserWriter Channel Module für *laraprint*

Das Modul *CNPapMaster* realisiert den Zugang zu Druckern via AppleTalk auf der Basis des Printer Access Protocols. Für den AppleTalk LaserWriter wurde daher auf Ceres ein Printer Channel Module, *LIB.PrCh.Atk.OBN*, entwickelt, das das Ansteueren dieses Druckers aus *laraPrint* gestattet.

5.2. Software, die auf CeresNet basiert

An Institut für Computersysteme wurde für die Ceres umfangreiche Kommunikations-Software (Laserdruck, Mailbox, Clearinghouse) entwickelt, die alle auf dem Modul *CeresNet* [Pesch87] basieren. Das Modul *CNCeresNet* basiert auf dem identischen Definitions-Modul wie CeresNet und ist funktionell mit CeresNet so weit identisch, dass nahezu alle unter CeresNet laufenden Anwendungen auch unter CNCeresNet betrieben werden können.

6. Verfügbarkeit der Software

Primär ist CeNet eine Programmbibliothek für AppleTalk-Protokolle auf Ceres, die den Benutzer sowohl bei der Entwicklung von verteilten Applikationen als auch für die Implementation von neu festgelegten AppleTalk-Protokollen auf Ceres-1 unter dem Betriebssystem Medos unterstützt. Die Unterstützung anderer Ceres-Betriebssysteme wie Oberon oder Vamos, oder anderer Hardware wie Ceres-2, ist jedoch nicht vorgesehen.

Da (mit Ausnahme von CNScc) alle Module in Modula-2 geschrieben sind, können diese auf beliebige andere Systeme portiert werden, für die ein entsprechender Compiler verfügbar ist und die eine Mehrprozessumgebung in irgend einer Form anbieten. Der Quelltext der CeNet-Module kann beim

Autor sowohl in gedruckter als auch in maschinenlesbarer Form bezogen werden.

An dieser Stelle muss noch angefügt werden, dass die gegenwärtige Implementation sich stark an die unter Medos verfügbare Form von parallelen Prozessen anlehnt und sich bei einer Portierung auf einen Rechner mit ereignisorientiertem Betriebssystem grössere Anpassarbeiten ergeben.

7. Rückblick

Die CeNet-Software wurde von einer Person im beschriebenem Umfang im Verlauf der letzten 2 Jahre in Teilzeitarbeit entwickelt , implementiert und zur Produktionsreife gebracht. Das Ziel, einerseits effiziente, anderseits aber robuste und wartungsfreundliche Software zu realisieren, wurde erreicht.

Die wohl wichtigste Anwendung der CeNet-Software ist der laraprint Printer Driver für die Ansteuerung eines Apple LaserWriters, der sich auf zahlreichen Systemen innerhalb und ausserhalb des Instituts für Informatik gut bewährt hat. Daneben können auch zahlreiche andere Kommunikationsprobleme mit dieser Software gelöst werden. Mit der Einführung von CNCeresNet ist es nun auch möglich, andere am Ifl entwickelte Applikationen wie Ceres-Mail auf gemischten Ceres/Macintosh-Netzwerken einzusetzten.

Der modulare Aufbau der Software erlaubt auch eine einfache Portierung auf andere Systeme, die parallele Prozesse unterstützen. Gegenwäritg läuft ein Projekt zur Portierung von CeNet auf die H-Box, einem am RZ der ETH entwickelten Rechner für den Zugang zum LocalNet-Netzwerk der Firma Sytek.

Im Verlaufe der Zeit zeigte sich aber auch, dass die prozess-orientierte Implementation der höheren Protokolle mit einigen Nachteilen behaftet ist. Zum einen bedeuten komplexe Protokolle meist eine hohe Zahl paralleler Prozesse, die eine nicht unerhebliche Menge Speicher für ihre Stacks verbrauchen und dessen Verwaltung nicht trivial ist. Zum andern arbeiten viele Implementationen von AppleTalk auf anderen Hosts (VAX/Unix, VAX/VMS, Mac, IBM PC, Ceres/Oberon) in einer ereignisorientierten Umgebung, was Portierung von Programmen von und nach Ceres/Medos erschwert. Aud diesem Grund sollten bei einer Ueberarbeitung der gesamten Software auch die höheren Schichten der Implementation auf Ceres, ähnlich der Macintosh-Implementation, ereignisorientiert gestaltet werden.

8. Verdankungen

An dieser Stelle möchte ich mich bei Herrn Prof. Dr. B. Plattner für die Betreuung und Unterstützung dieser Arbeit bedanken. Mein Dank gilt aber auch den Mit-Assistenten C. Pfister für die geleistete Vorarbeit und M. Wille und F. Peschel für die technische Unterstützung betreffend Ceres, SCC-Baustein und Assembler-Programmierung sowie für den Zugang zu den Quelltexten der CeresNet-Software. Mein weiterer Dank gilt aber auch jenen

hier nicht namentlich aufgeführten Personen, die mir den Zugang zu normalerweise nicht verfügbarer Fachliteratur ermöglichten.

9. Literatur

[Apda86] Inside AppleTalk
APDA, Renton 1986

[Apda87] AppleTalk Filing Protocol (AFP) Engineering Notes
APDA, Renton 1987

[Appl85] Inside Macintosh, Band 2, Kapitel "AppleTalk"
Addison-Wesley, Reading 1985

[Grie88] M. Grieder:
Elektronische Post im Netz der Ceres Arbeitsplatzrechner
Gelber Bericht Nr. 94
Institut für Informatik, Zürich 1988

[Scc82] AmZ8030 / AmZ8530 Serial Communications Controller
Technical Manual
Advanced Micro Devices Inc., Sunnyvale 1982

[Scc85] Z8530 SCC Serial Communications Controller
Preliminary Product Specification
Zilog Inc., Campbell 1985

[Eber87] J. Eberle:
Development and Analysis of a Workstation Computer
Diss. ETH, Zürich 1987

[Pesch87] F. Peschel:
CeresNet: ein low-cost Netzwerk für Ceres
Internes Memo Institut für Informatik, Zürich 1987

[Se87] Gruppe "Software Engineering" der ETH Zürich
Richtlinien zur Entwicklung von Software
Internes Arbeitspapier
Zürich 1987

[Wagn86] B. Wagner:
Cipon: A Model for Distributed Systems
Diss. ETH, Zürich 1986

B. Benutzerhandbuch

1. CNTools: Die Toolbox

1.1. Funktionen zur Handhabung von Datenpaketen

Auf der untersten Stufe (CNScc) werden Pakete als Block gesendet und empfangen; empfangene Pakete müssen also in ihre Teile zerlegt und abgehende Pakete aus einzelnen Teilen zusammengesetzt werden. Wir haben uns dabei festgelegt, einerseits diese Daten höchstens einmal umzukopieren, und anderseits Werkzeuge für das Umkopieren anzubieten, die u.a. eine vereinfachte Fehlerbehandlung gestatten.

Die Datenpuffer werden durch geeignete Deskriptoren dargestellt, die alle Angaben für eine effiziente Handhabung des Puffers beinhalten. *RxDesc* beschreibt Puffer für empfangene Daten; *dataPtr* gibt dabei die Adresse des nächsten noch unverarbeiteten Bytes, *remain* die Anzahl im Puffer verbleibenden Bytes an. Werden n Bytes aus dem Puffer geholt, wird *dataPtr* entsprechend erhöht und *remain* entsprechend reduziert.

```
CONST

   maxPacketSize = 603;


TYPE

   AtBuffer = ARRAY [0..maxPacketSize-1] OF BYTE;

   RxDesc = RECORD
                dataPtr : ADDRESS;
                remain  : INTEGER
            END;

   TxDesc = RECORD
                dataPtr : ADDRESS;
                dataLen : INTEGER;
                remain  : INTEGER
            END;


   PROCEDURE PrepInt(VAR i : INTEGER);

   PROCEDURE PrepLong(VAR i : LONGINT);

   PROCEDURE Get(VAR rxDesc : RxDesc;
                 VAR dest   : ARRAY OF BYTE;
                     len    : INTEGER);

   PROCEDURE RPut(VAR txDesc : TxDesc;
                  VAR src    : ARRAY OF BYTE;
                      len    : INTEGER);

   PROCEDURE RxDescInit(VAR rxDesc : RxDesc;
                        VAR buf    : ARRAY OF BYTE;
                            len    : INTEGER);
```

```
PROCEDURE TxDescInit(VAR txDesc  : TxDesc;
                     VAR buf     : ARRAY OF BYTE);
```

Für das Umkopieren von Daten in einen privaten Puffer steht die Prozedur *Get* zur Verfügung. Diese hat die spezielle Eigenschaft, bei vorzeitig leerem Puffer nicht auf die Daten hinter dem Puffer (oder gar jenseits der Speichergrenze) zuzugreifen, sondern Null-Bytes an die Ziel-Adresse zu kopieren. Gleichzeitig wird das Feld *remain* negativ, um den Fehler anzuzeigen. Damit vereinfacht sich für die meisten Programme die Fehlerbehandlung beim Lesen eines einzelnen Puffersegmentes.

Beim Aufbau eines Sendepuffers muss zusätzlich über die bereits in den Puffer geschriebene Daten Buch geführt werden; das Feld *dataLen* im *TxDesc* übernimmt diese Aufgabe. Analog zur Prozedur *Get* steht zum Umkopieren aus einem privaten Puffer die Prozedur *RPut* zur Verfügung, die ebenfalls über einen eingebauten Schutz gegen Pufferüberlauf verfügt. Zu beachten ist aber, dass *RPut* den Puffer *rückwärts* aufbaut, neu zugefügte Daten also *vor* den bisherigen Pufferinhalt setzt. Der Grund dafür ist einfach: im Datenpaket befinden sich die für die niedrigste Schicht bestimmten Daten zuvorderst, werden aber beim Abschicken des Paketes zuletzt eingefügt.

Für die Initialisierung von Puffer-Deskriptoren stehen die Prozeduren *RxDescInit* und *TxDescInit* zur Verfügung.

Bei Uebertragung von Integer-Werten wird gemäss AppleTalk-Spezifikationen auf dem Netz prinzipiell das most significant byte zuerst übertragen; die Speicherdarstellung von Integers auf Ceres ist jedoch genau umgekehrt. Deshalb müssen Integer-Werte vor dem Abschicken und nach dem Empfang zuerst in die entsprechenden Darstellung überführt werden; für diesen Zweck stehen die Prozeduren *PrepInt* und *PrepLong* zur Verfügung.

1.2. Weitere Funktionen der Toolbox

```
PROCEDURE Random(max: INTEGER): INTEGER;

PROCEDURE TermProcedure(term: PROC; VAR done: BOOLEAN);
```

An verschiedenen Stellen innerhalb der CeNet-Sofware werden Zufallszahlen gebraucht. Innerhalb CNAlap wird mittels Zufallszahlen sowohl eine freie Knotennummer gesucht als auch die Verzögerungszeit vor dem Abschicken eines Paketes zur Vermeidung von Kollisionen variiert.. Zu diesem Zweck ist ein der Random Number Generator *Random* innerhalb CNTools untergebracht. Dieser Random Generator ist sehr einfach gestaltet und wäre für statistische Anwendungen kaum geeignet, reicht aber für unsere Bedürfnisse aus. Ein Schutz gegen simultanen Benutzung durch mehrerer Prozesse besteht keiner, da ein solcher Konflikts wenig Auswirkungen auf die Qualität der erzeugten Zufallszahlen hat. Die Funktion *Random*(max) gibt als Resultat eine Zufallszahl im Bereich 0 bis max-1 zurück.

Die im System-Modul Programs angebotene Prozedur *TermProcedure* wird in CNTools ebenfalls angeboten. Funktionell ist sie mit der Version im Modul Programs identisch, benutzt aber eigene Tabellen und erlaubt einen höhere Zahl simultaner Registrierungen (momentan bis 20), ohne die Tabellen innerhalb des Moduls Programs zu belasten.

2. CNScc: Der Zugang zur Hardware

2.1. Zeitkritische Operationen

CNScc ist für die Ansteuerung der Hardware aus dem Programm verantwortlich; dies umfasst das Initialisieren des SCC-Bausteines und das Senden und das Empfangen von Steuer- und Datenpaketen. Dieses Modul ist vollständig in Assembler geschrieben, da alle Prozeduren extrem zeitkritisch sind.

Die Datenrate auf dem Netz beträgt 230 KBits pro Sekunde; das sind umgerechnet 28.75 Bytes pro Millisekunde oder jeweils 1 Byte alle 34.8 Mikrosekunden. Diese Datenrate ist zu hoch, um pro Byte einen Interrupt verarbeiten zu können. Gesendet wird daher mit Polling (busy waiting); der Empfang von Daten wird durch einen Interrupt beim Empfang des ersten Zeichens angestossen, die folgenden Bytes werden danach aber ebenfalls mittels Polling eingelesen.

Umgerechnet auf die Prozessor-Geschwindigkeit stehen für den Empfang oder das Senden eines einzelnen Bytes 10 - 20 Maschinen-Instruktionen zur Verfügung. Weitere zeitliche Randbedingungen des SCC-Bausteins wie die Notwendigkeit zweier Zugriffe auf den Baustein für die Abfrage eines Statusregisters oder das Einhalten der minimalen Erholzeit zwischen zwei Zugriffen reduzieren diese Zahl weiter. Ein weiteres kritischer Punkt war die benötigte Zeit für die Aktivierung der Interrupt-Prozedur, wo es nur mit einem Trick gelang, den SCC vor Datenüberlauf zu bewahren. Diese Probleme erfordern eine extrem hardwarenahe Programmierung, die nur in Assembler möglich ist.

2.2. Grundlagen von ALAP

2.2.1. Die Knotennummer (Node Number)

ALAP befasst sich mit der Uebertragung von Datenpaketen zwischen Stationen, die am gemeinsamen Uebertragungsmedium angeschlossen sind. Jeder Station ist eine 8-Bit-Nummer, die Knotennummer, zugewiesen; beim Einschalten einer Station wird eine freie Nummer selbstständig gesucht. Bei diesem Suchprozess wird zwischen Master, d.h. reguläre Benutzerstation, und Server, d.h. reservierte Station, die irgendwelche Dienste anbietet, unterschieden. Bei Server ist die Suche nach einer freien Knotennummer und dessen Zusicherung wesentlich strikter und zeitraubender als bei Master.

Zwei Werte der Knotennummern haben eine spezielle Bedeutung: Nummer 0 ist die ungültige Nummer und darf nie von einer Station belegt werden; der Versuch, ein Paket an Knotennummer 0 zu schicken, wird von der ALAP-

Software als Benutzerfehler gewertet. Der Wert 255 hat als *Broadcast-Adresse* ebenfalls eine spezielle Bedeutung: an diese Adresse geschickte Pakete werden von allen am Netz angeschlossenen Stationen empfangen.

Zusammengefasst ergibt sich:

0	ungültige Knotennummer
1 - 127	Master
128 - 254	Server
255	Broadcast-Adresse

2.2.2. Allgemeines Format eines ALAP-Paketes

Die Einheit der Datenübertragung auf Ebene *ALAP* ist das *Paket* (*Frame*); es kann eine Länge von 3 bis 603 Bytes umfassen. Die Grundstruktur eines Paketes ist immer die gleiche, wie in Fig. 2.1 dargestellt:

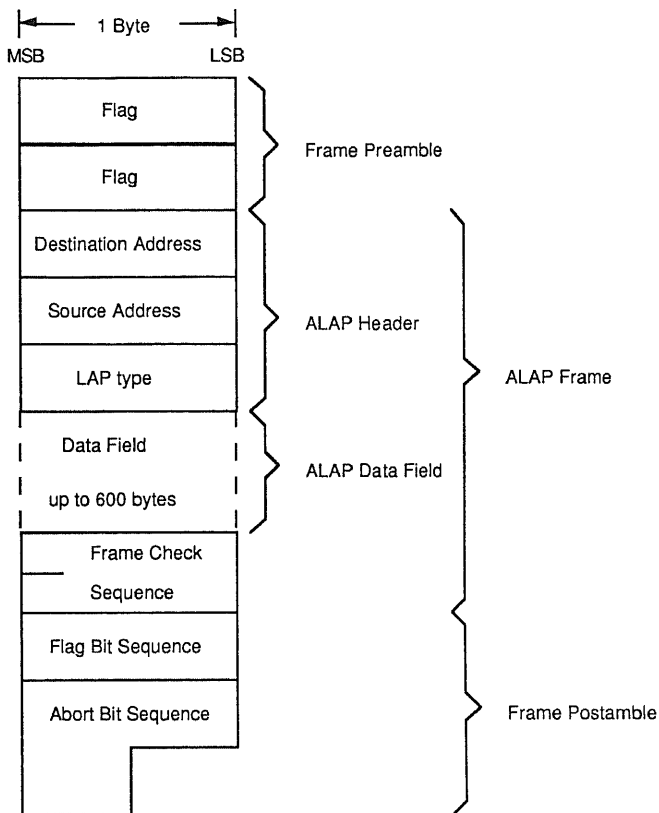


Fig 2.1. Grundstruktur eines ALAP-Datenpaketes

Die Preamble und Postamble dienen der Steuerung und der Synchronisation zwischen dem sendenden und empfangenden SCC-Baustein; die Frame Check Sequence dient der Erkennung von Uebertragungsfehler. An dieser Stelle soll lediglich auf den Header und den Datenteil von ALAP-Paketen näher eingeganen werden; für weitere Details sei auf [Apda86] verwiesen.

Es wird zwischen zwei Pakettypen unterschieden: Steuerpakete und Datenpakete. *Datenpakete* enthalten die zu übermittelnde Daten, während *Steuerpakete* den korrekten Protokoll-Ablauf auf Ebene ALAP steuern. Die Unterscheidung erfolgt über das Pakettyp-Feld; in CeNet sind folgende Werte definiert:

Pakettyp	Bedeutung
Pakettyp 1 - 127	Datenpakete
1 - 63	AppleTalk Standard-Protokolle
1	DDP short header
2	DDP long header
64 - 127	Lokale Protokolle
126	Lokale Tests
127	reserviert für CeresNet
Pakettyp 128 - 255	Steuerpakete
129 ($81)	RTS
130 ($82)	CTS
132 ($84)	ENQ
133 ($85)	ACK

Die Steuerpakete werden in den folgenden Abschnitten erklärt.

2.2.3. ENQ/ACK: Suchen einer freien Knotennummer.

Eine Station, die eine freie Knotennummer sucht, wählt sich irgend eine zulässige Nummer aus und sendet dann eine Reihe von *ENQ-Paketen* mit Absender = Empfänger = gewählte Knotennummer. Sollte die Knotennummer durch eine andere Station schon belegt sein, empfängt diese andere Station diese Pakete (da Empfänger = eigene Knotennummer auch für diese Station)und antwortet mit *ACK-Paketen*, die ebenfalls Absender = Empfänger = Knotennummer enthalten. Diese ACK-Pakete werden wiederum von der suchenden Station empfangen und dort als "Nummer besetzt" interpretiert. Ist dagegen die Knotennummer frei, antwortet keine Station mit ACK, und die rufende Station kommt beim Warten auf Antworten auf Timeout.

Dieser Test ist nicht absolut zuverlässig: sowohl ENQ- als auch ACK-Pakete können verloren gehen und so eine besetzte Netznummer als frei vortäuschen. Ausserdem existieren Server, die sich gelegentlich für einige Sekunden vom Netz trennen müssen, un in zeitkritischen Phasen nicht unterbrochen zu werden; genannt sei hier der LaserWriter, bei dem während der 6 Sekunden dauernden Belichtungsphase die CPU vollumfänglich für den Transfer der Daten in die Belichtungseinheit gebraucht wird. Aus diesem Grund muss der ENQ/ACK-Test viele Male und über einen gewissen Zeitraum verteilt ausgeführt werden; empfohlen sind 50 Tests für Master und

500 Test für Server; für letztere soll der Test über 8 Sekunden verteilt werden.

2.2.4. RTS/CTS: Datenübertragung mittels 3-Way-Handshake

Wie jedes CSMA-Zugriffsprotokoll sinkt auch der Durchsatz von AppleTalk bei grosser Last stark ab, bedingt durch das zunehmende Auftreten von Kollisionen. Da AppleTalk im Gegensatz zu anderen Technologien wie Ethernet keine Kollisionserkennung hat, wird ein Paket stets ganz gesendet; nach einer Kollision zwischen langen Datenpaketen bleibt somit das Netz bis zum Paketende belegt. Durch die Verwendung zusätzlicher Steuerpakete und einem speziellen *3-Way-Handshake* wird nun versucht, Kollisionen auf Steuerpakete zu begrenzen und so möglichst wenig Zeit zu verlieren; gleichzeitig wird die sendende Station über die aufgetretene Kollision informiert und kann den Sendeversuch wiederholen.

Bevor eine Station Daten auf das Netzwerk schickt, muss sie sicherstellen, dass nicht gerade eine andere Station das Netz belegt. Dazu horcht sie eine gewisse Minimalzeit, der *IDG* (*Inter-Dialog Gap*; 400 ns), auf Aktivität auf dem Netzwerk. Werden in diesem Zeitraum keine Daten erkannt, gilt das Netz als frei. Durch Variieren dieser Horchzeit um eine zufällige Grösse wird verhindert, dass nach einer Kollision die beteiligten Stationen nicht erneut gleichzeitig das Netzwerk als frei erkennen und zu senden beginnen.

Eine Datenübertragung besteht selber aus mehreren Paketen wie unten beschrieben. Damit nicht eine andere sendebereite Station das Netz als frei erkannt, darf der Abstand der einzelnen Pakete einer Datenübertragung nicht grösser als die *IFG* (*Inter-Frame gap*; 200 ns) sein.

Nun zum genauen Protokoll einer Datenübertragung von Station A nach B: Station A schickt zunächst ein *RTS-Steuerpaket* an Station B. Diese beantwortet dieses Paket umgehend mit einem *CTS-Steuerpaket*, das von der Station A empfangen wird. A sendet daraufhin das Datenpaket (Fig. 2.2, Teil A).

Kollisionen können nur noch auftreten, wenn zwei Stationen exakt gleichzeitig ihre RTS-Pakete abschicken; die beiden RTS-Pakete vermischen sich zu einer unlesbaren Bitfolge und werden von den entsprechenden Zielstationen nicht mehr erkannt (Fig. 2.2, Teil C). Somit antwortet keine der Zielstationen mit einem CTS-Steuerpaket; für die Absender bedeutet das Ausbleiben von CTS eine Kollision und damit die Notwendigkeit, den Datenübertragungsversuch zu wiederholen.

Broadcast-Pakete müssen gesondert behandelt werden, da sonst alle Stationen gleichzeitig mit einem CTS-Steuerpaket antworten würden. Deshalb wird bei Broadcasts generell auf das CTS-Steuerpaket verzichtet (Fig. 2.2, Teil B) mit dem Risiko, dass eine Kollision zwischen Broadcast-Paketen unerkannt bleibt.

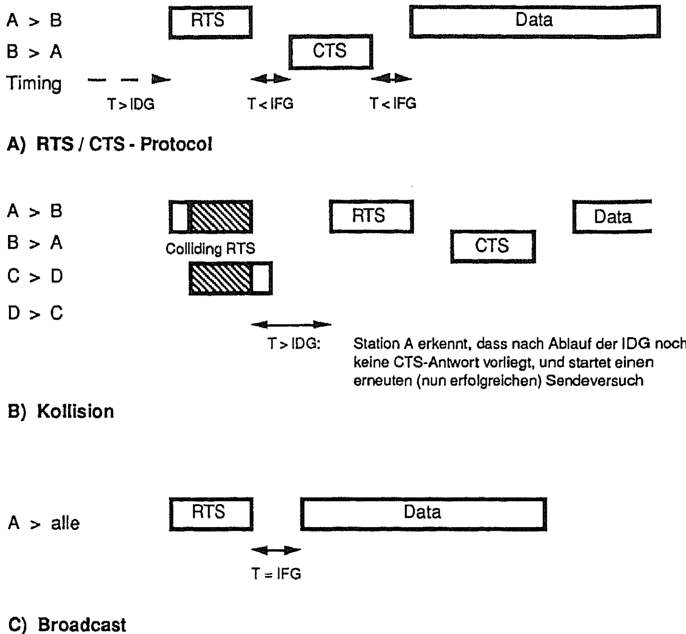


Fig. 2.2. RTS / CTS - Protokoll

Für die genaue Definition dieser Protokolle siehe [Apda86], Kapitel ALAP.

2.2.5. Flusskontrolle im RTS/CTS-Protokoll

Wie bei allen Netzwerken, die nach dem *CSMA*-Zugriffsverfahren arbeiten, sinkt auch die Leistungsfähigkeit eines AppleTalk-Netzwerkes bei hoher Netzlast rapide ab. Durch die Verwendung des RTS/CTS-Protokolls wird zwar das Verhalten eines AppleTalk-Netzwerkes unter mittlerer bis hoher Last stark verbessert; zusätzlich wird bei sehr hoher Netzlast noch eine *Flusskontrolle* aktiv, die die Senderate der einzelnen Stationen reduziert und damit den Datenverkehr auf dem Netzwerk selber auf ein erträgliches Mass limitiert.

Diese Flusskontrolle basiert auf dem Mechanismus, der das Netz als frei oder belegt erkennt, und auf der Erkennung von Kollisionen in Form von ausbleibenden CTS-Paketen. Sobald eine Station eine Häufung von Kollisionen feststellt, verlängert sie die Zeit, während der sie auf dem Netzwerk horcht, um festzustellen, ob es frei ist. Anderseits reduziert sie diese Zeit wieder, wenn das Netz nur noch selten besetzt vorgefunden wird.

Auf Ceres wurde diese Flusskontrolle aus folgenden Gründen jedoch nicht implementiert:

- Beim Austesten des Moduls CNScc wurde festgestellt, dass bei ansteigender Netzlast die Zahl der Kollisionen auch dann sehr klein blieb, wenn bereits über 98 % aller Sendeversuche wegen belegtem Netzwerk scheiterten. Ein derart hoch belastetes Netzwerk, auf dem pro Datenpaket im Schnitt 50 Sendeversuche notwendig werden, ist in der Praxis nicht mehr tragbar.

- In der Praxis sind Kollisionen meist nicht lastmässig bedingt, sondern resultieren aus Situationen, wo der Partner aus irgend einem Grund nicht empfangsbereit ist und daher das empfangene RTS nicht mit CTS quittiert. Diese Situation wird von der sendended Station als Kollision erkannt, bewirkt eine entsprechende Verlängerung der Horchzeit und damit eine unnötige Reduktion des Datenflusses.

- Gefahr des "Verhungerns": Hat eine einzelne Station auf einem mittel belasteten Netzwerk mehrmals Kollisionen erkannt (z.B. weil die Gegenstation nicht antwortete), erhöht sie die Horchzeit stark und kann damit im Kampf um Netzwerkzeit gegen die übrigen Stationen, die weiterhin mit kurzen Horchzeiten arbeiten, nicht mehr konkurrieren.

Anstelle dieser Flusskontrolle wurde auf Ceres nur die auch in den AppleTalk-Empfehlungen vorgesehene Verlängerung der Horchzeit nach dem ersten misslungenen Sendeversuch implementiert.

2.3. Aufgabenteilung zwischen CNScc und CNAlap

Unsere Implementation realisiert im wesentlichen den Vorschlag gemäss [Apda86], Kapitel ALAP; um die Grösse des Assemblerprogramms möglichst klein zu halten, sind die Aufgaben von ALAP in die beiden Module CNScc (Assembler) und CNAlap (MODULA) aufgeteilt.

Folgende Funktionalitäten finden sich in CNScc:

- Initialisieren des *SCC-Bausteins* für Netzbetrieb, inkl. Installieren der Interrupt-Prozedur mittels Processes.CreateDriver

- Sendeversuch für ein einzelnes Paket, mit 3-Way-Handshake. Misslingt der Sendeversuch, z.B. weil das Netzwerk belegt vorgefunden wurde, wird zum Hauptprogramm zurückgekehrt, sobald das Netzwerk wieder frei ist.

- Absenden eines einzelnen ENQ-Paketes und Horchen auf ein zurückkommendes ACK-Paket. Wiederum wird nur ein Sendeversuch unternommen.

- Interruptroutine zum Empfang von Paketen, mit 3-Way-Handshake. Empfangene ENQ-Pakete werden protokollgerecht mit ACK beantwortet.

Folgende Funktionalitäten finden sich in *CNAlap*:

- Ablaufsteuerung zur Belegen einer Knotennummer

- Wiederholte Sendeversuche für Pakete, die nicht im ersten Versuch erfolgreich abgeschickt wurden; Fehlerbehandlung, wenn auch nach etlichen Versuchen kein Erfolg
- Festlegen der *Defer-Zeit*, d.h. die Zeitdauer, während der das Netzwerk unbelegt sein muss, bevor ein Paket abgeschickt wird.
- *Loopback*-Interface: Pakete, die an die eigene Station adressiert sind, werden ebenfalls empfangen; die Station "hört sich selbst".

2.4. Verwaltung der Empfangs-Puffer

Empfangene Daten können mit oder ohne Zwischenpufferung weiterverarbeitet werden. Wird auf Zwischenpufferung verzichtet, wie dies beim Macintosh der Fall ist, muss jedes empfangene Paket sofort verarbeitet werden, bevor ein neues angenommen werden kann; damit besteht die Gefahr, dass in der Zwischenzeit Pakete verloren gehen können oder, als Folge von wiederholten Sendeversuchen, eine überhohe Netzlast resultiert. Wir entschieden uns daher für Zwischenpufferung mehrerer Pakete und wählten aus der Vielfalt möglicher Pufferstrukturen (lineare und zirkuläre Puffer mit und ohne separaten Indexpuffer, mit und ohne Wraparound am Pufferende) die verkettete Liste von linearen Puffern. Diese Methode ist zwar sehr speicherintensiv (für jedes zu empfangende Paket wird ein Puffer der maximalen Länge bereitgestellt, obwohl durchschnittliche Pakete sehr viel kleiner sind), erlaubt jedoch eine einfache und schnelle Implementation.

Ausschnitt aus CNScc.DEF:

```
TYPE

   RxPtr = POINTER TO RxBuf;     (* input packets *)
   RxBuf = RECORD
              next : RxPtr;
              len  : INTEGER;
              data : ARRAY [0..sccRxBufSize] OF BYTE;
   END;

VAR

   rxNext : RxPtr;        (* ptr to next rx buffer to fill *)
   rxRead : RxPtr;        (* ptr to next unread rx buffer *)
   rxFree : RxPtr;        (* ptr to last freed rx buffer *)
```

RxBuf und *RxPtr* definieren die Struktur der Einzelpuffer. Die Puffer werden zu einer linearen Liste verknüpft, dessen Ende mit einem NIL-Pointer markiert wird.

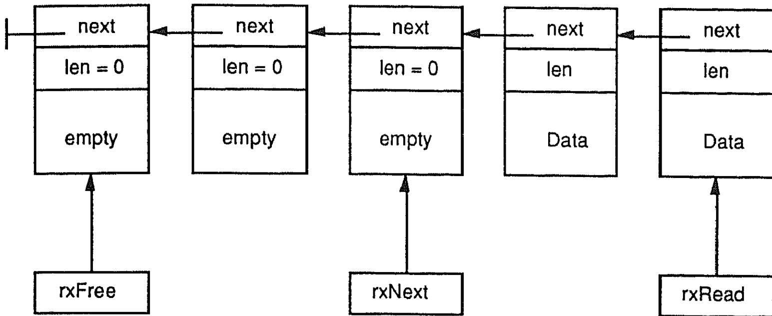


Fig. 2.3. Verknüpfungen der Empfangspuffer

Drei Pointer erlauben den Zugand zu diese Liste. *RxRead* zeigt auf das erste Element am Anfang, ist also die Verankerung der gesamten Liste. Durch Weiterbewegen von *RxRead* zum nächsten Element in der Liste wird das erste Element aus der Liste entfernt. *RxFree* zeigt auf das Element am Ende der Liste und erlaubt, auf einfache Weise neue Elemente am Ende der Liste anzufügen. *RxNext* ist ein Pointer, der auf irgend ein Element in der Liste zeigt.

Die Bedeutung dieser Pointer: *RxNext* zeigt jeweils auf den Puffer, der beim nächsten Datenempfang gefüllt wird. Wird ein Datenpaket erfolgreich empfangen und ist noch ein weiterer Datenpuffer hinter *RxNext^* vorhanden, so wird *RxNext* um ein Element weiter bewegt. Alle Puffer von *RxRead^* bis und ohne *RxNext^* sind somit mit empfangenen Daten gefüllte Puffer, die durch Weiterbewegen von *RxRead* aus der Liste entfernt und damit konsumiert werden. Die Puffer von *RxNext^* bis und mit *RxFree^* sind die noch leeren Empfangspuffer; neue leere Puffer können am Ende der Liste mittels *RxFree* angesetzt werden.

Diese Technik ergibt eine strikte Trennung der Zuständigkeiten zwischen dem SCC-Interruptprozess, der Daten in die Puffer füllt, und einem Konsumentenprozess, der die gefüllten Puffer aus der Liste entfernt, die empfangenen Daten verarbeitet und die nun leeren Puffer wieder an der Liste ansetzt. RxNext wird nur durch den SCC-Interruptprozess verändert, während der Konsumentenprozess RxFirst, RxFree und die Verkettungen innerhalb der Liste nachführt. Auf diese Art kann mit einem Minimum von gegenseitigem Ausschluss dauernde Konsistenz garantiert werden. In der Praxis reicht dies allerdings nicht aus, da bei speziellen Operationen wie das Senden von Datenpaketen an die eigene Station auch andere Prozesse den Pointer RxNext verändern. Gelöst wird das Problem mit einem Flag (recMode, s. unten), der dem SCC-Interruptprozess mitteilt, dass momentan nicht auf die Datenstruktur der Pufferverwaltung zugegriffen werden darf. Der SCC-Interruptprozess selber ist nicht unterbrechbar; temporär inkonsistente Zustände in diesem Prozess bleiben nach aussen unsichtbar.

2.5. Modul-Beschreibung

Das Modul CNScc wird im Normalfall auschliesslich von CNAlap angesprochen; Benutzerprogramme sollten dieses Modul nie ansprechen.

```
CONST

   sccRxBufSize    = maxPacketSize + 2;
   sccTxBufSize    = maxPacketSize;

   scOk            = 0;     (* no error *)
   scTechProblem   = 1;     (* technical SCC problem *)
   scCableBusy     = 2;     (* had to defer *)
   scNoAnswer      = 3;     (* no answer / collisions *)

   recNone         = '0';   (* receive nothing *)
   recEnq          = '1';   (* receive / act on ENQ only *)
   recAll          = '3';   (* receive everything *)


TYPE

   RxPtr = POINTER TO RxBuf;     (* input packets *)
   RxBuf = RECORD
              next : RxPtr;
              len  : INTEGER;
              data : ARRAY [0..sccRxBufSize-1] OF BYTE;
           END;


VAR

   (* --- buffer management --- *)

   rxNext  : RxPtr;
   rxRead  : RxPtr;
   rxFree  : RxPtr;

   (* --- receive mode / buffer access lock flag --- *)

   recMode : CHAR;

   (* Possible values:
         - recNone : No data reception; ENQ are not replied
         - recEnq  : Do react to ENQ, but do not receive data
         - recAll  : Normal mode
   *)

   PROCEDURE InitDriver(VAR ok: BOOLEAN);

   PROCEDURE SetAdr(myAdr: INTEGER);

   PROCEDURE StopDriver();

   PROCEDURE TransENQ(): INTEGER;

   PROCEDURE Trans(backOff : INTEGER;
                   len     : INTEGER;
                   data    : ADDRESS): INTEGER;
```

Das Definition Module enthält eine sehr umfangreiche und präzise Dokumentation, die der Uebersicht wegen hier nicht wiedergegeben wurde.

3. CNAlap: Der Zugang zum Netzwerk auf tiefster Stufe

3.1. Funktion von CNAlap

Während das Modul *CNScc* lediglich Bindeglied zur Hardware war, wird innerhalb *CNAlap* das eigentliche ALAP-Protokoll erzeugt. Dies umfasst:

- Finden einer freien Node Number während der Initialisierung
- Behandlung des ALAP Headers
- Fehlerbehandlung
- Flusssteuerung bei hohem Verkehrsaufkommen (momentan stark vereinfacht implementiert)
- Bereitstellen von Pufferspeicher für den Paket-Empfang
- Verwaltung der Puffer für empfangene Pakete
- Aufruf einer Handler-Prozedur für die Weiterreichung empfangener Pakete an die nächsthöhere Schicht

Innerhalb CNAlap werden die Felder *Node Number* (8 bit) und *LAP-Type* (8 bit) als Paketadresse verwendet. *Node Number* identifiziert eine Station (in unserem Falle eine Ceres) auf dem gleichen Netzwerk; der Wert 255 ist dabei für Broadcast reserviert, während 0 ein nicht zulässiger Wert ist. *LAP-Type* gibt an, nach welchem Protokoll das Paket verarbeitet werden soll. *CNDdp* (s. weiter unten) belegt die Werte 1 und 2.

CNAlap initialisiert ebenfalls das Modul *CNScc*. Bei Programm-Ende hat jedoch *CNAlap* keine Möglichkeit, *CNScc* zu terminieren, da eine Termination Procedure die Terminierung der eigenen Programm-Ebene nicht erkennt. Somit wird der SCC-Baustein nicht zurückgesetzt und erzeugt weiterhin Interrupts; Medos selber jedoch desaktiviert bei Ende eines Programmes dessen Gerätetreiber und sperrt Interrupts vom zugehörigen Gerät.

3.2. Modul-Beschreibung

```
CONST

   (* error codes *)

   cnOk          = 0;     (* no error *)
   cnBufTooSml   = 1;     (* buffer too small *)
   cnDataTooLg   = 2;     (* data part too long *)
   cnTableFull   = 3;     (* registration tables are full *)
   cnNotFound    = 4;     (* key / object was not found *)
   cnDoubleKey   = 5;     (* key / object already present *)
   cnCableErr    = 10;    (* any sort of problem on the cable *)
```

```
TYPE

   ProtoHandler = PROCEDURE(     INTEGER,       (* from node *)
                                 INTEGER,       (* ALAP protocol type *)
                                 BOOLEAN,       (* was broadcast *)
                             VAR RxDesc);       (* data buffer desc *)


   SpyProc = PROCEDURE(ADDRESS,          (* data adr *)
                       INTEGER,          (* length *)
                       BOOLEAN);         (* outgoing *)

VAR

   alapNode : INTEGER;    (* local ALAP node number; READ-ONLY *)


   PROCEDURE ALAPOpenProto(     proto         : INTEGER;
                                protoListener : ProtoHandler;
                            VAR status        : INTEGER);

   PROCEDURE ALAPCloseProto(proto: INTEGER);

   PROCEDURE ALAPWrite(     toNode : INTEGER;
                            type   : INTEGER;
                            txDesc : TxDesc;
                        VAR status : INTEGER);

   PROCEDURE InstallSpy(spy : SpyProc);

   PROCEDURE RemoveSpy;
```

ProtoHandler definiert den Prozedurkopf für eine Handler-Prozedur, die alle nicht durch DDP belegten Protokolle behandelt.

OpenAlapProto installiert eine Handler-Prozedur für einen bestimmten ALAP-Protokoll-Typ. Sobald ein Paket dieses Typs eintrifft, wird die als Parameter übergebene Prozedur aufgerufen.

Mit *CloseAlapProto* wird eine derartige Handler-Prozedur wieder entfernt.

Weitere Hinweise:

- Der Prozess, der den Handler aufruft, ist ein normaler MEDOS-Prozess (ohne Priorität); es gelten keinerlei Einschränkungen betreffend Betriebssystem-Aufrufen. Der diesem Prozess zugewiesene Stackbereich ist aber auf umgefähr 1000 Bytes beschränkt.

- Es wird nur ein einziger Handler aufs Mal aufgerufen. Kommt dieser in einen Wartezustand, wird die Verarbeitung nachfolgender Pakete blockiert.

- Terminiert ein Programm, ohne seinen Protocol Handler mit CloseAlapProto zu entfernen, werden diese selbstständig entfernt.

ALAPWrite erlaubt das Verschicken von Paketen auf Ebene ALAP.

Folgende Elemente dienen dem Debugging oder dem Erzeugen künstlicher Netzstörungen:

SpyProc legt den Prozedurkopf der "Spion-Prozedur" fest. Die Prozedur erhält jedesmal die Kontrolle, wenn ein Paket abgeschickt oder empfangen wird. Sie kann das Paket (mit ALAP-Header) analysieren, verändern oder auch ein Paket "wegzuwerfen", indem das dritte Byte auf einen im Netzwerk nicht definierten ALAP-Protokolltyp gesetzt wird.

InstallSpy installiert eine "Spion-Prozedur". Nur ein "Spion" kann aufs Mal gleichzeitig installiert werden; weitere Aufrufe von InstallSpy entfernen den jeweils vorher installierten "Spion".

UninstallSpy entfernt die installierte "Spion-Prozedur".

3.3. Initialisierung CNAlap; File "DK.Param.CNNodeNum"

Um in einem Netzwerk stabile Verhältnisse zu haben und die Anzahl Konflikte bei der Wahl von Knotennummern zu minimieren, legt jede Station ihre Netzwerk-Nummer im File *DK.Param.CNNodeNum* ab. Zwecks Editierbarkeit wurde das File als 3 byte langes Text-File mit folgendem Format ausgelegt:

1. und 2. Byte: Knotennummer, in hex (Grossbuchstaben benutzen!). Nummern grösser als hex 80 sind für permanente Server reserviert.

3. Byte: Optionsmaske, in hex:
   - 1: Lange ENQ-Initialisierung für Server
   - 2: Neue Knotennummer nicht zurückschreiben
   - 4: Abbrechen wenn Knotennummer besetzt

Wird beim Start dieses File nicht gefunden, wählt *CNAlap* selbstständig irgend eine freie Knotennummer unter hex 80 und legt ein neues Parameterfile an. Die Optionsmaske im neu angelegten File wird dabei auf Null (keine Optionen) gesetzt; für permanente Servers oder zu Testzwecken könne nachträglich mit dem normalen Texteditor sowohl Knotennummer als auch Optionsmaske editiert werden.

4. CNDdp: Kommunikation in einem Netzwerkverbund

4.1. Funktion

CNDdp erweitert einerseits die Möglichkeiten von CNAlap auf Netzwerkverbunde, die aus mehreren AppleTalk Netzwerken bestehen, die über Gateways untereinander verbunden sind, und führt zusätzlich *Netznummer* (16 bit), *Socket* (8 bit) und *Protocol* (8 bit) ein. Die *Netznummer* bezeichnet ein bestimmtes Netz innerhalb des Verbundes; der Wert 0 bedeutet dabei das eigene Netz. Ein *Socket* ist eine Identifikation eines Kommunikationskanals auf der lokalen Maschine und erlaubt, innerhalb einer Maschine mehrere logische Kanäle zu unterhalten. Einige Sockets wie derjenige für Name Binding Protocol (NBP) sind *statisch* definiert; vom

Benutzer angebotene Dienste erhalten *dynamisch* zugewiesene Socket Numbers. Insgesamt sind 127 verschiedene Sockets dynamisch zuweisbar, welche in unserer Implementation alle gleichzeitig eröffnet sein können.

Netzwerk-Adressen bezeichnen nun einen bestimmten Dienst innerhalb einer Station auf irgend einem Netzwerk im Verbund, und bestehen aus den Komponenten *Network Number*, *Node Number* und *Socket Number*. Diese Kombination, die 32 bits = 4 bytes belegt, wird als *Internet-Adresse* (nicht zu verwechseln mit *DARPA-Internet-Adressen*, die ebenfalls 32 bit belegen) bezeichnet.

Protocol identifiziert das verwendete Protokoll oberhalb DDP (d.h. der mittleren Schicht) und soll verhindern, dass zwei Anwendungen mit verschiedenen Protokollen versuchen, miteinander Kontakt aufzunehmen.

Empfang und Senden von Paketen erfolgt nach der gleichen Logik wie in CNAlap. Ein Socket wird wiederum durch Bereitstellen einer Empfangsprozedur (Socket listener, Socket handler) eröffnet; sobald auf dem Socket Daten empfangen werden, werden diese durch Aktivierung der Empfangsprozedur an das betreffende Programm übergeben.

4.2. Modul-Beschreibung

```
MODULE CNDdp;

CONST

   (* error codes *)

   cnOk          = 0;     (* no error *)
   cnBufTooSml   = 1;     (* buffer too small *)
   cnDataTooLg   = 2;     (* data part too long *)
   cnTableFull   = 3;     (* registration tables are full *)
   cnNotFound    = 4;     (* key / object was not found *)
   cnDoubleKey   = 5;     (* key / object already present *)
   cnCableErr    = 10;    (* any sort of problem on the cable *)
   cnNoInet      = 20;    (* we are not an inet *)

   (* other DDP constants *)

   ddpMaxSize    = 586;   (* maximum ddp data size *)


TYPE

   InterAddr = RECORD
      net    : INTEGER;      (* net number; 0 means local *)
      node   : INTEGER;      (* node number within net *)
      socket : INTEGER;      (* socket number *)
   END;

   SockHandler  = PROCEDURE(VAR InterAddr,   (* from addr *)
                                INTEGER,     (* DDP protocol number *)
                                BOOLEAN,     (* was broadcast *)
                                INTEGER,     (* local socket *)
                            VAR RxDesc);     (* data buffer desc *)


VAR
```

```
   ddpNode    : INTEGER;  (* local node number *)
   ddpNet     : INTEGER;  (* local net number  *)
   ddpABridge : INTEGER;  (* Any-Bridge number *)


   PROCEDURE DDPOpenPrivSocket(    socket      : INTEGER;
                                   sktListener : SockHandler;
                               VAR status      : INTEGER);

   PROCEDURE DDPOpenSocket(VAR socket      : INTEGER;
                               sktListener : SockHandler;
                           VAR status      : INTEGER);

   PROCEDURE DDPCloseSocket(socket: INTEGER);

   PROCEDURE DDPWrite(VAR dest     : InterAddr;
                          protocol : INTEGER;
                          locSock  : INTEGER;
                          ckSum    : BOOLEAN;
                          txDesc   : TxDesc;
                      VAR status   : INTEGER);

END CNDdp;
```

InterAddr ist die Datenstruktur zur Beschreibung von Internet-Adressen und umfasst Netz-Nummer, Node-Nummer und Socket Nummer.

SockHandler ist der vorgeschriebene Header für Socket-Handler-Prozeduren.

ddpNode ist eine read-only Variable, die die lokale Node Number beinhaltet.

ddpNet ist eine read-only Variable, die die lokale Netz-Nummer beinhaltet. Das Feld wird durch RTMP-Stub (innerhalb CNMpp) gesetzt; andere Module dürfen das Feld nur lesen.

ddpABridge ist eine read-only Variable, die die Node Number irgend einer Bridge beinhaltet. Einzig RTMP-Stub darf hier hineinschreiben. Zusammenhang: ist ddpNet ungleich Null, ist immer auch ddpABridge ungleich Null.

DDPOpenPrivSocket öffnet einen Socket im statischen (privilegierten oder experimentellen) Wertebereich, d.h. trägt unter dem entsprechenden Socket eine Socket-Handler-Prozedur ein.

DDPOpenSocket öffnet einen Socket im dynamischen Bereich; die gewählte Socket-Nummer wird zurückgegeben.

DDPCloseSocket schliesst einen vorgängig geöffneten Socket.

DDPWrite schickt ein Paket aufs Netz.

5. CNNbp: Symbolische Namen für Netzwerk-Adressen

5.1. Funktion

Die Angabe von Internet-Adressen für die Angabe eines Dienstes oder Kommunikations-Partners für dem Benutzer nicht zuzumuten; einerseits weil sie für einen Benutzer schwierig zu merken oder zu interpretieren sind, anderseits weil Node Numbers und Sockets erst beim Einschalten der Maschine bzw. beim Starten des Dienstes festgelegt werden und sich somit ändern können. Deshalb bietet CNNbp die Möglichkeit, Diensten einen Namen zuzuweisen und vor der Benutzung eines Dienstes dessen Namen in die entsprechende Internet-Adresse umzuwandeln. Dazu wird ein verteilter Verzeichnisdienst benutzt: jede Station führt eine Liste jener Namen, die zu Diensten innerhalb der eigenen Station gehören. Sucht nun irgend eine Station einen Dienst nach Namen, sendet sie einen spezielles Broadcast-Paket (das sich, mit Hilfe der Bridges, über das gesamte Internet ausbreiten kann); jede Station, die das Paket empfängt, durchsucht die eigenen Tabellen und antwortet mit der gewünschten Adress-Information, wenn der entsprechende Eintrag gefunden wird.

Um verschiedene Dienstarten voneinander zu trennen, wird neben dem *Namenfeld* (24 char) noch ein *Typfeld* (ebenfalls 24 char) verwendet; einen Apple LaserWriter ist zum Beispiel der Typ "LaserWriter" zugewiesen. Zusätzlich unterstützt CNNbp Wildcards; werden Name und/oder Typ auf "=" gesetzt, bleibt das entsprechende Feld ungeprüft. Es ist also möglich, eine Liste aller mit Namen ansprechbaren Dienste abzufragen.

Wie schon vorher erwähnt, müssen NBP Broadcast sich über das gesamte Internet ausbreiten, was bei sehr grossen Netzen zuviel Datenverkehr mit sich zieht, die Verwaltung von (zu vielen) Namen erschwert und das Netz als homogen über den gesamten Betrieb erscheinen lässt, ohne Strukturen wie verschiedene Departemente, Institute etc. zu wiederspiegeln. Diesen Problemen wird durch Aufteilen den Netzes in verschiedene Zonen begegnet, die wiederum über Namen ansprechbar sind; für uns ist aber momentan nur wichtig, dass die lokale Zone stets als "*" ansprechbar ist und dass das Verteilen von NBP Broadcasts über Netzgrenzen hinweg, mit Berücksichtigung von Zonen, Sache jener *Bridges* ist, die innerhalb des Internets die verschienenen Einzelnetzwerke verbinden.

5.2. Modul-Beschreibung:

```
DEFINITION MODULE CNNbp;

FROM CNDdp IMPORT (* type *) InterAddr;

CONST

   (* error codes *)

   cnOk          = 0;     (* no error *)
   cnTableFull   = 3;     (* registration tables are full *)
   cnNotFound    = 4;     (* key / object was not found *)
   cnDoubleKey   = 5;     (* key / object already present *)
   cnCableError  = 10;    (* network problem *)
```

```
   cnNoInet        = 20;    (* no inet; unable to retrieve zone info *)

   (* other NBP constants *)

   maxNames        = 5;        (* max local names *)
   seekShort       = 1000;     (* short seek time for other servers *)
   seekLong        = 8000;     (* long seek time for laser printers *)
   nameLen         = 32;       (* maximum name length in NBP *)


TYPE

   NameString = ARRAY[0..nameLen] OF CHAR;

   EntityName = RECORD           (* Entity names *)
      name    : NameString;      (* name *)
      type    : NameString;      (* type *)
      zone    : NameString;      (* zone *)
      iaddr   : InterAddr;       (* internet addr *)
      enum    : CARDINAL;        (* enum; internal *)
   END;


VAR

   asterisk    : NameString;     (* String holding '*' *)
   eqSign      : NameString;     (* String holding '=' *)


   PROCEDURE RegisterName(      name     : ARRAY OF CHAR;
                                type     : ARRAY OF CHAR;
                                socket   : INTEGER;
                                seekTime : INTEGER;
                            VAR status   : INTEGER);

   PROCEDURE RemoveName(      name   : ARRAY OF CHAR;
                              type   : ARRAY OF CHAR;
                          VAR status : INTEGER);

   PROCEDURE NameDirectory(      name     : ARRAY OF CHAR;
                                 type     : ARRAY OF CHAR;
                                 zone     : ARRAY OF CHAR;
                                 increm   : BOOLEAN;
                                 seekTime : INTEGER;
                             VAR hits     : INTEGER;
                             VAR entities : ARRAY OF EntityName;
                             VAR status   : INTEGER);

   PROCEDURE LookUpName(      name   : ARRAY OF CHAR;
                              type   : ARRAY OF CHAR;
                              zone   : ARRAY OF CHAR;
                          VAR iaddr  : InterAddr;
                          VAR status : INTEGER);

   PROCEDURE ConfirmName(      name   : ARRAY OF CHAR;
                               type   : ARRAY OF CHAR;
                               zone   : ARRAY OF CHAR;
                               iaddr  : InterAddr;
                           VAR status : INTEGER);


END CNNbp.
```

NameString definiert einen String mit maximal zulässiger Länge.

EntityName definiert die für NameDirectory notwendige Datenstruktur zur Ablieferung der gefundenen Namen.

RegisterName registriert einen Namen auf dem Netzwerk, zusammen mit der lokalen Socket-Nummer.

RemoveName entfernt einen lokal registrierten Namen.

NameDirectory erlaubt Namenssuche mit Wildcard "=" für Namen und Typ. Die Namen werden in einer Array der Form ARRAY [0..max] OF EntityName abgeliefert; die Grösse des Arrays ist dem Benutzer überlassen. Einschränkung: wenn mehr als 9 Stationen Namen zurückmelden, wird innerhalb CNScc/CNAlap der Pufferbereich knapp und das Resultat unsicher. Suchen mit einem expliziten Zonen-Namen statt "*" für lokale Zone setzt eine Internet-Umgebung voraus.

LookUpName sucht einen ganz bestimmten Namen im Netzwerk und gibt dessen Internet-Adresse zurück.

ConfirmName prüft die Zusammengeörigkeit zwischen einem Namen und dessen Internet-Adresse. Sollte benutzt werden, wenn zwischen der Namenssuche und dem Gebrauch der Adresse ein längerer Zeitraum (einige Minuten) verstrich, um nicht einen mittlerweise aufgehobenen Dienst anzusprechen.

6. CNMpp: Interne Hilfs- und Verwaltungsfunktionen

6.1. Uebersicht

CNMpp erledigt Verwaltungsaufgaben, die pro Station anfallen, aber für den Benutzer nicht sichtbar sind. Dazu gehört einerseits das Finden der eigenen Netznummer und irgend eines Gateways zu anderen Netzen, anderseits das Anbieten eines Echo-Dienstes, mit dem Testprogramme die Ansprechbarkeit von Stationen im gesamten Internet überprüfen können.

6.2. RTMP-Stub

Um die Verwaltung eines AppleTalk-Internets möglichst einfach zu gestalten, wird Information über die Netzstruktur nur in den Bridges gespeichert; vom Netzwerk-Betreiber fest einprogrammiert werden dabei nur die Netznummern der unmittelbar angrenzenden Netz-Segmente. Einmal gestartet, beginnen die Bridges in gewissen Zeitabständen (z.B. 10 sec), untereinander Informationen über die Netzstruktur auszutauschen, und innerhalb wenigen Minuten kennt jede Bridge alle existierenden Netzwerk-Segmente mit Netznummer und Adresse des nächsten Bridges auf dem Pfad zu jenem Segment. Diese Information wird als Broadcast-Paket nach einem speziellen Protokoll, dem *Routing Table Maintenance Protocol* (*RTMP*), ausgesandt. Die Details dieses Protokolls sind nur für Bridges wichting; als allgemeine Information für alle Teilnehmer enthält aber jedes Paket die eigene Netznummer sowie die Knotennummer der Absender-

Bridge. Jede Station kann diese Pakete mithören und so sowohl die eigene Netznummer als auch die für Internet-Betrieb notwendige Knotennummer einer Bridge ermitteln. Mit einem speziellen RTMP-Anfragepaket ist es auch möglich, bei einer Bridge diese Information direkt zu erfragen; jede Station startet eine solche Anfrage automatisch bei Eintritt ins Netz.

Pragmatisch gelöst ist das Verhalten von Rtmp-Stub bei Ausfall des / der Bridges auf dem eigenen Netz. Gemäss [Apda85] soll die Netznummer für ungültig erklärt werden, sobald während einer gewissen Zeit (25 sec) keine RTMP-Pakete mehr empfangen wurde. Auf Ceres wurde eine einfachere Lösung gewählt, um nicht mit Timeouts und einem zusätzlichen Clock-Prozess arbeiten zu müssen: Ist ein Gateway zweimal hintereinander nicht ansprechbar, wird die Netznummer bis zum Eintreffen eines weiteren RTMP-Paketes für ungültig erklärt.

6.3. Echo-Protokoll

Im Einzelnetzwerk kann die Verfügbarkeit einer Station durch einfaches Schicken irgend eines Paketes überprüft werden. Ist die Station nicht annahmebereit, wird bereits innerhalb des 3-Way-Handshake die fehlende Antwort des Partners festgestellt, und das aufrufende Programm kriegt eine Fehlermeldung. Anders im Netzverbund: Pakete, die eine Bridge nicht am Ziel abliefern kann, werden einfach weggeworfen; die sendende Station wird nicht informiert. Aus diesem Grunde wurde das Echo-Protokoll definiert: empfängt der Echo-Socket ein spezielles Anfragepaket, so antwortete dieser mit einem entsprechenden Antwortspaket.

Da diese Möglichkeit von wenig Nutzen für Applikationprogramme ist und daher ausschliesslich in Diagnoseprogrammen verwendet wird, wurde die Master-Seite des Echo-Protokolls nicht realisiert; Programme, die diese Möglichkeit nutzen wollen, müssen dazu CNDdp benutzen. Die Server-Seite, die in jedem Rechner vorhanden sein muss, wurde direkt in CNMpp integriert.

6.4. Modul-Beschreibung

Das Modul *CNMpp* exportiert (mit Ausnahme einer Dummy-Konstanten, die den Compiler zufriedenstellt) nichts. Dennoch muss jedes Programm, das *CNDdp* oder *CNNbp* benutzt, dieses Modul mitladen, da die beiden Module auf Zusammenarbeit mit eventuell vorhandenen Bridges angewiesen sind.

Dies bedeutet, dass Programme, die die Module CNDdp und/oder CNNbp direkt benutzen, ohne dabei gleichzeitig höhere Module wie CNAtp oder CNPapMaster zu importieren, das Modul CNMpp mittels

IMPORT CNMpp

explizit importieren müssen!

CNCeresNet ist davon nicht betroffen, da dieses Modul auf CNAlap basiert und daher nicht auf die Dienste von CNMpp angewiesen ist. Module der höheren Softwareschichten wie *CNAtp* oder *CNPapMaster* importieren CNMpp selbstständig.

7. CNAtp: Ein einfaches Transaktions-Protokoll

7.1. Funktion

CNAtp realisiert das *AppleTalk Transaction Protocol* (*ATP*); alle Optionen (Multi Packet Response, Exactly Once, Transaction Cancel) sind implementiert. ATP ist ein einfaches Master-Server-Protokoll, bei dem der Master einen Request an einen Server richtet; der Server behandelt diesen Request und schickt abschliessend eine Response zurück. Bei Nichteintreffen der Server-Antwort innerhalb einer gewissen Zeit infolge eines Fehlers wird die Transaktion wiederholt; nach einer vorgebbaren Anzahl Versuche wird mit Fehler abgebrochen. Optional kann auch eine Transaktion gegen mehrfache Ausführung im Server als Folge verlorener Antwortspakete geschützt werden. Eine eigentliche Link-Ueberwachung, die Zusammenbrüche des Masters oder des Servers zwischen einzelnen Transaktionen erkennt, existiert dagegen nicht.

Ausgelöst wird eine Transaktion mit *SendRequest*; es wird gewartet, bis die Transaktion regulär terminiert oder die Timeout-Zeit abläuft. Nach Timeout wird versucht, die Transaktion zu wiederholen; bleibt dies auch nach der angegebenen maximalen Anzahl ohne Erfolg, wird mit Fehlerstatus *cnAtpTimeout* abgebrochen. Adressiert wird der Partner durch seine Internet-Adresse, die man aus seinem Namen durch vorgängiges Name Binding (Normalfall) oder auf anderen Wegen erhalten hat.

Die Server-Seite differiert aber stark von anderen Implementationen (z.B. cipon) oder von der Macintosh-Implementation. Unser Ziel war es, einerseits mit statischen Datenstrukturen arbeiten zu können, anderseits aber mehrere Prozesse zuzulassen, die Transaktionen des gleichen Dienstes parallel bearbeiten. Damit fällt die sonst elegante Lösung mit Parameter-Prozeduren ausser Betracht; programmiert wird eine Transaktion als endloser Loop, der mit *ReceiveRequest* einen Request holt, diesen behandelt und mit *SendResponse* abschickt. Der Loop wird verlassen, sobald man als Fehlerstatus CNAtpCancel kriegt, der anzeigt, dass der Service durch irgend einen Prozess auf der Server-Maschine aufgehoben wurde.

Gelangen Request an einen Server, der zwar den Dienst definiert hat, aber auf dem keine freien Prozesse existieren, die den Request behandeln können, wird gemäss AppleTalk-Definition der Request ignoriert; der Master wird nach Ablauf seiner Zeitüberwachung einen neuen schicken. Damit erübrigt sich ein spezielles Zwischenpuffern von empfangenen Requests mit Duplikat-Filterung; doch die Sache hat eine wesentlich schlimmere Kehrseite: sendet der Master nach Empfang der Response sofort wieder den nächsten Request, bevor der Server wieder Gelegenheit hatte, wieder ReceiveRequest aufzurufen, geht das Paket verloren. In unserer Lösung wird daher der zuletzt empfangene Request zwischengepuffert und dem nachfolgenden ReceiveRequest übergeben.

SendRequest ignoriert intern Fehler, die beim Senden von Paketen entstehen; über den Timeout-Mechanismus wird das Paket erneut gesendet. Läuft der Retry-Count ab, so resultiert der allgemeine Status *cnAtpTimeout*.

ReceiveRequest kennt nur den einen möglichen Fehler "*cnAtpCancelled*", wenn der Service mittlerweise (durch einen anderen Prozess) aufgehoben wurde. Dagegen kann *SendResponse* die gesamte Palette von Fehlercodes auf CNAlap und CNDdp produzieren, die anzeigen, dass die Response nicht ablieferbar war. Bei wiederholbaren Transaktionen können solche Fehler ignoriert werden; bei Benutzung nicht wiederholbarer Transaktionen (exactly once-Option) muss eine der Anwendung entsprechende Behandlung erfolgen. In keinem Fall darf *SendResponse* wiederholt werden, da die Transaktion intern als abgeschlossen betrachtet wird.

Bei Verwendung der exactly once-Option wird der Verlust des vom Master geschickten Abschlusspaketes (*T-Release*) im Gegensatz zu anderen Implementationen nicht als Fehler angezeigt, da die Situation von CNAtp selber korrekt behandelt wird und ein Benutzer an diesem Hinweis kaum interessiert ist.

7.2. Modul-Beschreibung: Master-Funktionen

```
DEFINITION MODULE CNAtp;


FROM SYSTEM      IMPORT (* type *) ADDRESS;
FROM Processes   IMPORT (* type *) Signal;
FROM CNDdp       IMPORT (* type *) InterAddr;


CONST

   (* error codes *)

   cnOk           = 0;     (* no error *)
   cnTableFull    = 3;     (* registration tables are full *)
   cnNotFound     = 4;     (* key / object was not found *)
   cnDoubleKey    = 5;     (* key / object already present *)

   cnAtpTimeout   = 30;    (* no transaction reply within time *)
   cnAtpBufOver   = 31;    (* reply buffers overflowed *)
   cnAtpCancel    = 32;    (* request was cancelled *)
   cnAtpClosed    = 33;    (* server socket was closed *)

   (* other ATP constants *)

   atpMaxSize     = 578;   (* max packet size w/o user bytes *)
   infinRetry     = 0;     (* infinite retrying *)
   nilBufLen      = -1;    (* length marking non-existing buf *)


TYPE


   (* --- stuff mainly for internal use --- *)


   State  = (inactive, active, cancelled, gotsts,
             finished, failed, unsched, replied, rereply);

   (* inactive  : Not in use                                    MS *)
   (* active    : Transaction started, awaiting response        MS *)
   (* cancelled : cancel request was issued for that mcb        MS *)
```

```
 (* gotsts     : last response requested re-request via sts   M  *)
 (* finished   : last missing packet has arrived              M  *)
 (* failed     : timed out                                    M  *)
 (* unsched    : trx waiting to be scheduled                  S  *)
 (* replied    : xo-trx has sent reply; waiting for tRel      S  *)
 (* rereply    : xo-trx has gotten a request to re-reply      S  *)

TCBPtr = POINTER TO TCB;

TCBSysInfo = RECORD                  (* Transaction control block *)
   locSocket  : INTEGER;             (* local socket number *)
   state      : State;               (* transaction state *)
   partner    : InterAddr;           (* partner of transaction *)
   tid        : INTEGER;             (* transaction id *)
   bitmap     : BITSET;              (* bitmap of outstanding pkts *)
   xo         : BOOLEAN;             (* exactly once option *)
   overflow   : BOOLEAN;             (* buffer overflow; master only *)
   wakeUp     : Signal;              (* wake up the attached process *)
   progLev    : INTEGER;             (* allocating program level *)
   next       : TCBPtr;              (* link to next TCB *)
END;


UserBytes = ARRAY [0..3] OF CHAR;        (* User bytes *)

BufferDesc = RECORD
   adr       : ADDRESS;
   maxLen    : INTEGER;
   len       : INTEGER;
   userBytes : UserBytes;
END;

TCB = RECORD                          (* ATP Control Block *)
   sys        : TCBSysInfo;           (* system internal info *)
   reqBuf     : BufferDesc;           (* request buffer descriptor *)
   replBuf    : ARRAY [0..7] OF BufferDesc;      (* buffer desc *)
END;


PROCEDURE ZeroBufDesc(VAR tcb : TCB);


PROCEDURE SendRequest(VAR mcb       : TCB;
                          partner   : InterAddr;
                          xo        : BOOLEAN;
                          timeout   : INTEGER;
                          retries   : INTEGER;
                      VAR status    : INTEGER);


PROCEDURE CancelSendRequest(VAR mcb      : TCB;
                            VAR status   : INTEGER);

END CNAtp.
```

Tcb ist die Definition des Transaktions-Kontrollblock sowohl für Master- und Server-Seite. Der Benutzer muss darin seine verwendeten Sende- und Empfangspuffer eintragen; nicht existente Puffer werden durch den speziellen Wert nilBufLen als Länge markiert. Das Subfeld sys enthält all jene Felder, die rein intern benutzt werden und den Benutzer nichts angehen.

ZeroBufDesc ist eine Hilfsprozedur, die alle Sende- und Empfangspuffer-einträge im TCB auf nicht existente Buffer setzt.

SendRequest schickt einen ATP-Request ab. Das Feld returned sagt, weiviele Pakete in der Antwort zurückgegeben wurden. Adressiert wird der Partner mit Internet-Adresse, die man ggf. vorher in CNNbp auf eine Namen-Anfrage erhalten hat. Spezialfall: Null Empfangspuffer, retries = 1 und timeout = 0 schickt ein Paket, ohne auf Antwort zu warten;

CancelSendRequest bricht einen hängigen ATP-Request ab; der abgebrochene Request erhält einen entsprechenden Fehlerstatus. Wird meist von einem parallel laufenden Prozess aufgerufen, der die Aufgabe hat, Transaktionen aufzuräumen.

7.3: Modul-Beschreibung: Server-Funktionen

```
PROCEDURE OpenRespondingSocket(VAR socket : INTEGER;
                               VAR status : INTEGER);

PROCEDURE CloseRespondingSocket(VAR socket : INTEGER);

PROCEDURE ReceiveRequest(     socket : INTEGER;
                          VAR scb    : TCB;
                          VAR status : INTEGER);

PROCEDURE SendResponse(VAR scb     : TCB;
                           timeout : INTEGER;
                       VAR status  : INTEGER);
```

Tcb definiert auch den Transaktions-Kontrollblock auch für die Server-Seite. Der Benutzer muss darin ebenfalls seine verwendeten Sende- und Empfangspuffer eintragen; nach ReceiveRequest können darin ausserdem die Parameter Lokaler Socket (tcb.sys.locSocket), Absender (tcb.sys.partner), erwartete Antwortspakete (tcb.sys.bitmap) und XO-Modus (tcb.sys.xo) ausgelesen werden.

OpenRespondingSocket öffnet einen Socket für Gebrauch durch ATP Server.

CloseRespondingSocket schliesst einen ATP-Server-Socket.

ReceiveRequest wartet auf einen Request am entsprechenden Socket. Wird auf diesen Socket CloseRespondingSocket aufgerufen, wird ReceiveRequest mit einem entsprechenden Fehlerstatus cnAtpCancel abgebrochen.

SendResponse sendet eine Rückantwort. Spezialfall: Wird SendResponse mit null Rückantwortspaketen geschickt, wird die Transaktion als erledigt aus der Buchhaltung entfernt.

8. CNCeresNet: Gesicherte Verbindungen

8.1. Funktion

CNCeresNet ist die auf AppleTalk portierte Version von *CeresNet* [Pesch87]. Der Modul-Name lautet ebenfalls *CeresNet*; durch Auswechseln des Objekt-Files, wahlweises Einlinken von CNCeresNet.OBN oder LIB.CeresNet.OBN in Programme oder Laden eines speicherresidenten Programms, in das vorgängig CNCeresNet.OBN eingebunden wurde, kann die gewünschte Version aktiviert werden.

CNCeresNet verwendet intern *ALAP*-Protokoll; ein Betrieb über Bridges hinweg ist daher nicht möglich.

8.2. Modul-Beschreibung

Das Definitions-Modul von *CNCeresNet* ist absolut identisch mit demjenigen von *CeresNet*; die Beschreibung von *CeresNet* [Pesch87] behält seine Gültigkeit, mit folgenden Anmerkungen:

- Kompatibel mit *AppleTalk*

- Für *CeNet* wird die Zuleitung zum AppleTalk-Kabel am *B-Port* angeschlossen. Die CeresNet-Software gemäss [Pesch87] arbeitet über den A-Port; es ist also möglich, eine Station an beide Netzwerke anzuschliessen.Die jeweils geladene Software entscheidet, welches Netzwerk tatsächlich angesprochen wird; es ist aber nicht möglich, gleichzeitig beide Netzwerke anzusprechen.

- Die maximale Grösse eines Datenblocks ist auf 595 Bytes (*CeresNet*: 600 Bytes) beschränkt. Bislang existiert aber noch keine Anwendersoftware, die an dieser Limite scheitert.

- Anwendungen, die auf *CeresNet* aufbauen, können so ohne Aenderung auch in der *CeNet*-Umgebung laufen.

8.3. Der interne Modul CNPackets

CeresNet verwendet intern das (in Assembler geschriebene) Modul *Packets*, das den Zugang zur Hardware bewerkstelligt. *CNCeresNet* benutzt intern ALAP-Protokoll, womit das Modul *Packets* nicht mehr benötigt würde. Um aber die Portierarbeit vor allem für neue Versionen von *CNCeresNet* zu reduzieren, wurde das interne Modul unter dem Namen *CNPackets* beibehalten und so die notwendigen Aenderungen innerhalb *CNCeresNet* auf ein absolutes Minimum beschränkt.

9. CNPapMaster: Zugang zu AppleTalk-basierten Printern

9.1. Funktion

CNPapMaster implementiert die Master-Seite des *AppleTalk Printer Access Protocols* (*PAP*), ein spezielles Datenstrom-Protokoll, das über Link-Ueberwachung verfügt und von allen am AppleTalk anschliessbaren Printern, wie LaserWriter, verstanden wird. Obwohl diese Printer meist byteorientiert arbeiten, ist die Schnittstelle datagramm-orientiert, wobei die Grösse der Datengramme im allgemeinen keine logische Bedeutung hat. Der EOF-Flag zeigt das letzte Datengramm mit logisch zusammengehörenden Daten an. Im übrigen ist die Schnittstelle von CNPapMaster weitgehend selbst erklärend.

9.2. Modul-Beschreibung

```
DEFINITION MODULE CNPapMaster;


FROM CNDdp IMPORT (* type *) InterAddr;

CONST

   (* error codes *)

   cnOk            = 0;     (* no error *)
   cnNotFound      = 4;     (* printer not found *)
   cnAtpTimeout    = 30;    (* transaction timeout *)
   cnAtpBufOver    = 31;    (* reply buffers overflowed *)
   cnSessBroken    = 40;    (* PAP session broken or closed *)


   PROCEDURE PAPOpen(     iaddr  : InterAddr;
                      VAR status : INTEGER);

   PROCEDURE PAPClose;

   PROCEDURE PAPAlife(): BOOLEAN;

   PROCEDURE PAPRead(VAR buffer : ARRAY OF CHAR;
                     VAR len    : INTEGER;
                     VAR eof    : BOOLEAN;
                     VAR status : INTEGER);

   PROCEDURE PAPWrite(VAR buffer : ARRAY OF CHAR;
                          len    : INTEGER;
                          eof    : BOOLEAN;
                      VAR status : INTEGER);

   PROCEDURE PAPStatus(     iaddr  : InterAddr;
                        VAR text   : ARRAY OF CHAR;
                        VAR status : INTEGER);

END CNPapMaster.
```

PAPOpen eröffnet eine *PAP*-Session (einen Job) zu einem bestimmten Printer. Die gegenwärtige Version unterstützt nur einen geöffneten Printer aufs Mal.

PAPClose schliesst eine *PAP*-Session regulär ab. Reagiert der Printer nicht auf diesen regulären Abschluss, wird die Verbindung notfallmässig getrennt. Ausstehende *PAPRead* und *PAPWrite* terminieren mit Fehlerstatus. Auf einem LaserWriter werden unvollständig übertragene Seiten nicht ausgedruckt.

Bricht die Verbindung wegen Uebertragungsproblemen zusammen, wird intern aufgeräumt, als ob *PAPClose* aufgerufen worden wäre. Ein Aufruf von *PAPClose* auf eine zusammengebrochene Verbindung wird ignoriert.

Alife zeigt an, ob gegenwärtig eine *PAP*-Session existiert.

PAPRead liest Daten vom Printer.

PAPWrite schickt Daten zum Printer.

PAPStatus frägt den textuellen Status des Printers ab; ob zum Printer gegenwärtig eine *PAP*-Session besteht, ist gleichgültig.

9.3. Ansprechen des LaserWriter

Beim Ansprechen des *LaserWriter* sind etliche Details zu beachten, die aus den verschiedenen Manuals schlecht oder gar nicht hervorgehen:

- Der LaserWriter verwendet *LF* als Zeichen für End-Of-Line. In Eingaben wird auch *CR* als End-Of-Line akzeptiert.

- Während mit *PAPWrite* Daten an den Printer geschickt werden, soll stets parallel dazu aus einem separaten Prozess *PAPRead* ausgeführt werden. Der Printer nimmt keine neuen Daten mehr entgegen, wenn er selber Fehlermeldungen oder andere Daten an den Master zurücksenden will, dieser aber nicht empfangsbereit ist.

- Der LaserWriter schickt jede Fehlermeldung und den zu einem "flush"-Befehl gehörenden Output in einem, ev. mehreren separaten Paketen.

- *EOF* identifiziert das logische Ende eines Jobs und setzt den LaserWriter in den Grundzustand zurück. Der LaserWriter quittiert das EOF des Masters seinerseits mit einem EOF-Datengramm, sobald der *physikalische* Druckvorgang abgeschlossen wurde.

10. CNZipInfo: Aufteilung des Netzwerkverbundes in Namenszonen

10.1. Funktion

Um sehr grosse Netzwerke nach geographischen oder organisatorischen Kriterien weiter unterteilen zu können, wurde beim Name Binding Protocol der Begriff *Zone* eingeführt. Eine einzelne Zone umfasst eine Anzahl

Einzelnetze des Netzwerkes und trägt einen Namen; jedes Einzelnetz gehört zu genau einer Zone.

Das *Zone Information Protocol* hat die Aufgabe, die Zoneninformation, die anfänglich nur auf einigen wenigen Bridges bekannt ist, über das gesamte Netz auszubreiten. Als weitere Funktionalität wird dem Benutzer die Möglichkeit geboten, von irgend einer Bridge auf seinem lokalen Netzwerk den eigenen Zonennamen oder eine Liste aller bekannten Zonennamen abzufragen. Diese beiden Funktionalitäten werden vom Modul *CNZipInfo* an Benutzerprogramme angeboten.

10.2. Modul-Beschreibung

```
DEFINITION MODULE CNZipInfo;

FROM CNNbp IMPORT (* type *) NameString;

CONST

   (* error codes *)

   cnOk            = 0;    (* no error *)
   cnNoInet        = 20;   (* we are not an inet *)
   cnAtpTimeout    = 30;   (* transaction timeout *)


   PROCEDURE GetZoneList (VAR zoneList : ARRAY OF NameString;
                          VAR count    : INTEGER;
                          VAR status   : INTEGER);

   PROCEDURE GetMyZone (VAR myZone : NameString;
                        VAR status : INTEGER);

END CNZipInfo.
```

GetZoneList ruft eine Liste aller im Netz existierenden Zonen ab. Die momentane Version hat eine in der Praxis kaum wichtige Einschränkung: Werden sehr viele Zonen gefunden (Summe der Länger aller Zonennamen über 580), geht ein Teil der Namen verloren.

getMyZone gibt den Namen der eigenen Zone zurück. Werden im Netzwerk keine Zonen unterstützt, wird als Antwort "*" zurückgegeben.

11. Nicht implementierte Protokolle

Folgende *AppleTalk*-Protokolle mit grösserer Verbreitung sind auf *Ceres* nicht verfügbar:

- PAPServer
- ASP (AppleTalk Session Protokoll)
- AFP (AppleTalk Filing Protocol)
- DARPA-Protokolle (ARP, IP, UDP, TCP)

12. Testprogramme

12.1. Modul-Testprogramme

Zum getrennten Austesten der verschiedenen Software-Komponenten wurden zahlreiche Testprogramme erstellt. Zu jedem einzelnen Modul wurden eines oder mehrere Testprogramme erstellt, um das korrekte Funktionieren des Moduls zu überprüfen.

12.2. Peek und Poke auf Macintosh

Als weiteres Testwerkzeug wurde auch ein Macintosh mit den Programmen *Peek* und *Poke* eingesetzt, mit denen sich einerseits der Datenverkehr auf dem Netzwerk beobachten liess, anderseits vorbereitete Pakete ins Netz eingespiesen werden konnten.

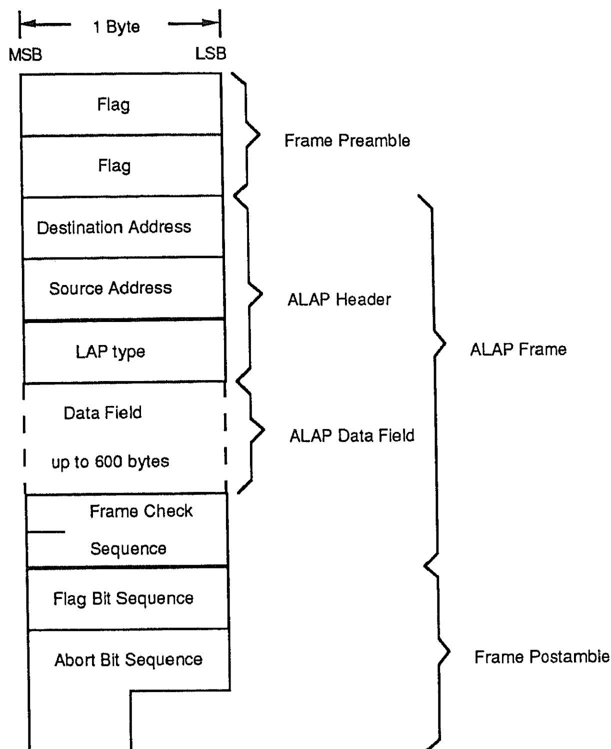
12.3. CNMonitor, CNMonPrint, CNSpy

Weitere Testwerkzeuge wurden auch auf Ceres entwickelt. *CNMonitor* ist ein Programm, das den gesamten Datenverkehr auf dem Netzwerk in ein File mitprotokolliert; das Programm *CNMonPrint* bereitet diese Daten in einer textuellen Form auf.

Da Pakete, die die Station an sich selber schickt, nie auf dem Netz erscheinen, wurde zum Mitprotokollieren solcher Pakete ein weiteres Programm, *CNSpy*, entwickelt, das über den Spy-Mechanismus von *CNAlap* allen Datenverkehr von und zur lokalen Station mitprotokolliert.

Appendix A. Paketformate

A.1. Grundstruktur eines Datenpaketes



A.2. Datagram Delivery Protocol (DDP)

ALAP Basic Format

ALAP Basic Format
Dest. Address
Source Address
LAP Type

DDP Short Format

DDP Short Format
Dest. Address
Source Address
LAP Type = 01
zero
Datagram len
Dest. socket
Source socket
DDP proto type
Datagram data up to 586 bytes

DDP Long Format

DDP Long Format
Dest. Address
Source Address
LAP Type = 02
hop count
Datagram len
DDP Checksum
Destination Network Nr.
Source Network Nr.
Dest. node id
Source node id
Dest. socket
Source socket
DDP proto type
Datagram data up to 586 bytes

A.3. Name Binding protocol (NBP)

Format of Name Touple

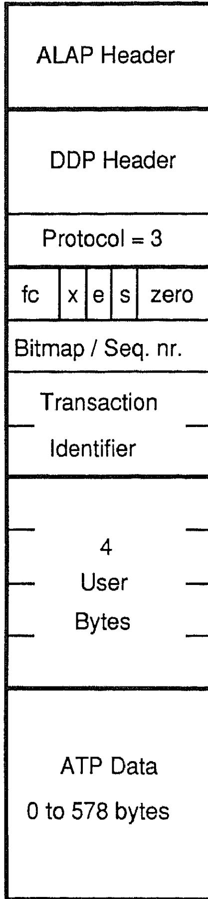
Network Number
Node Number
Socket Number
Enumerator
Name Length
Name
Type Length
Type
Zone Length
Zone

NBP Packet

ALAP Header	
DDP Header	
Protocol = 2	
Control	Touple Count
NBP ID	
NBP Tuple	
* * *	
NBP Touple	

Control:
1 = BrRq
2 = LkUp
3 = LkUp-Reply

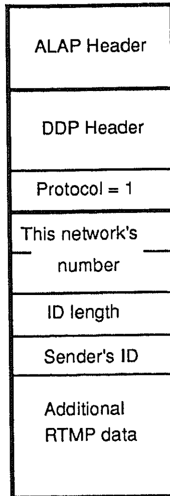
A.4. AppleTalk Transaction Protocol (ATP)



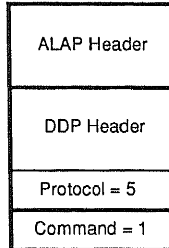
fc function code:
1 = TReq, 2 = TRepl, 3 = TRel
x XO bit
e EOM bit
s STS bit

A.5. Routing Table Maintenance Protocol (RTMP), Workstations

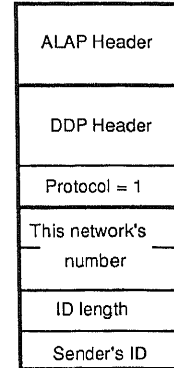
RTMP data packet



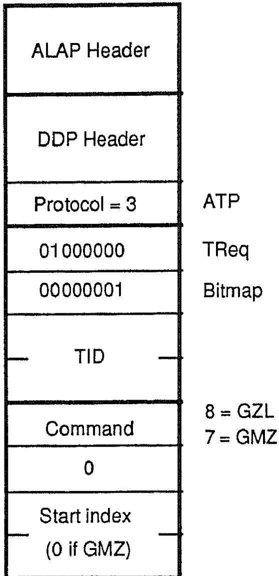
RTMP Request

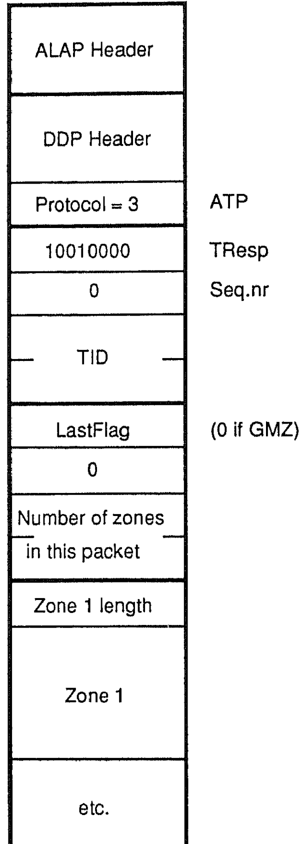


RTMP response

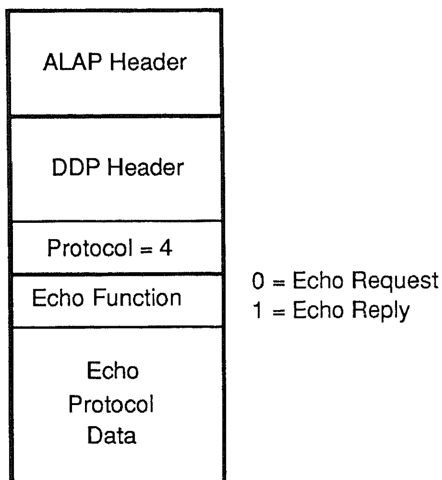


A.6. Zone Information Protocol (ZIP), Workstation part

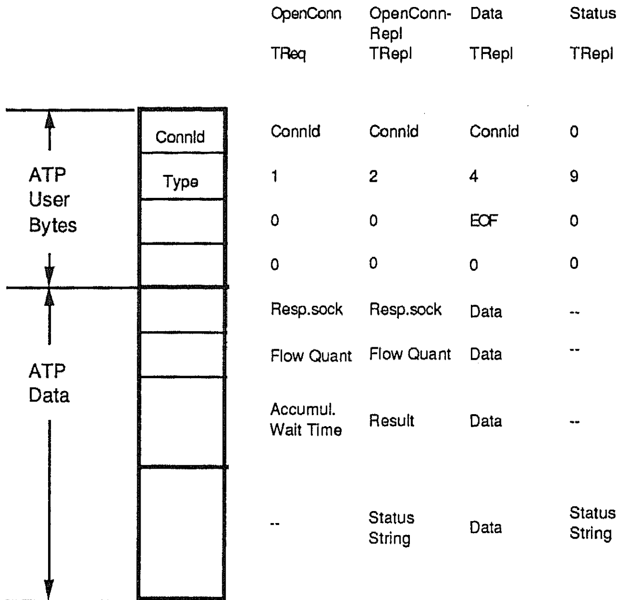




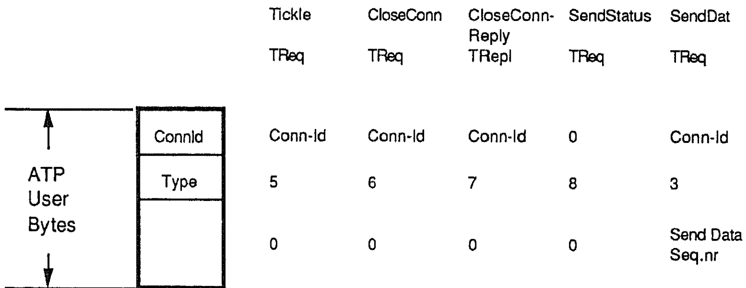
A.7. Echo Protocol



A.8. Printer Access Protocol (PAP)

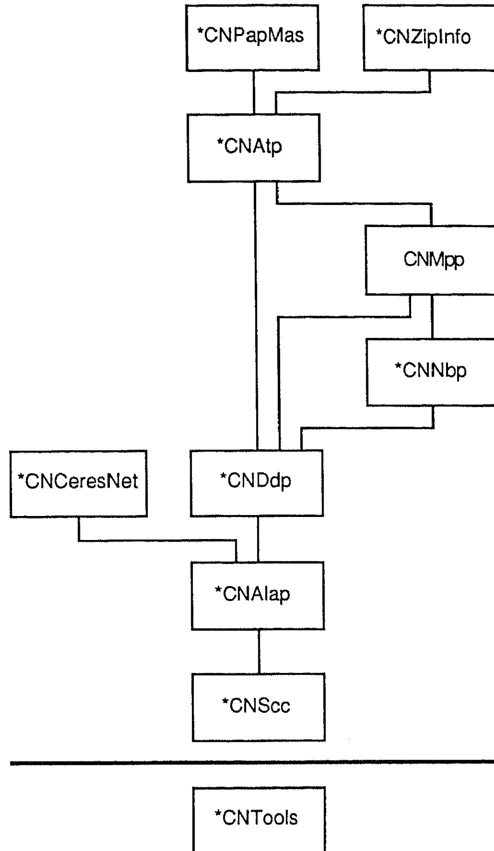


		OpenConn TReq	OpenConn-Repl TRepl	Data TRepl	Status TRepl
ATP User Bytes	ConnId	ConnId	ConnId	ConnId	0
	Type	1	2	4	9
		0	0	EOF	0
		0	0	0	0
ATP Data		Resp.sock	Resp.sock	Data	--
		Flow Quant	Flow Quant	Data	--
		Accumul. Wait Time	Result	Data	--
		--	Status String	Data	Status String



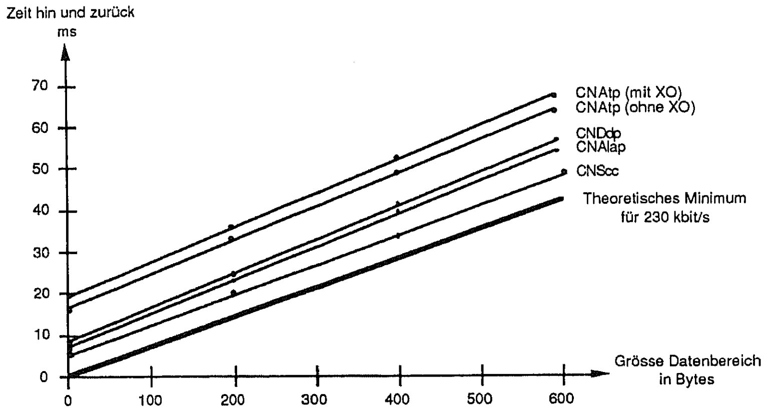
		Tickle TReq	CloseConn TReq	CloseConn-Reply TRepl	SendStatus TReq	SendDat TReq
ATP User Bytes	ConnId	Conn-Id	Conn-Id	Conn-Id	0	Conn-Id
	Type	5	6	7	8	3
		0	0	0	0	Send Data Seq.nr

Appendix B. Modulstruktur



Appendix C. Geschwindigkeitsmessungen

Für die Module CNScc, CNDdp (Alap-Protokoll), CNDdp (Ddp-Protokoll) und CNAtp wurden Geschwindigkeitsmessungen zwischen zwei getrennten Stationen (hier A und B genannt) durchgeführt. Alle Messungen waren identisch organisiert: Station A schickt an Station B ein Datenpaket mit vorgegebener Länge; Station B schickt als Antwort ein Datenpaket identischer Länge zurück. Die folgende Grafik zeigt die dazu benötigte Zeit (Round Trip Time) in Abhängigkeit der Anzahl Datenbytes. Unter ATP (mit oder ohne XO-Option) wird jeweils die Zeit für eine komplette Transaktion angegeben, wobei sowohl das Request- als auch das Response-Paket die entsprechende Menge Benutzerdaten (d.h. User Bytes nicht mitgerechnet) enthalten. Zum Vergleich sind auch die aufgrund der Geschwindigkeit des physikalischen Mediums (230 kbit/s) erreichbaren minimalen Zeiten angegeben.



Eidg. Techn. Hochschule Zürich
Informatikbibliothek
ETH-Zentrum
CH-8092 Zürich