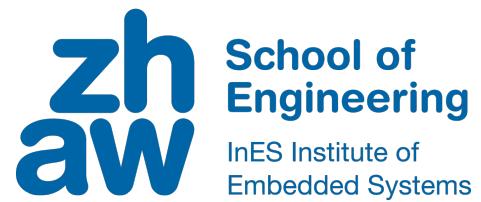


Zurich University
of Applied Sciences



Kommunikationstechnik

Prof. Thomas Müller¹ Prof. Hans Weibel²

Institute of Embedded Systems³

22. Februar 2018

¹thomas.mueller@zhaw.ch

²hans.weibel@zhaw.ch

³www.ines.zhaw.ch

Inhaltsverzeichnis Kapitel 1

1 Das OSI-Referenzmodell	1-1
1.1 Geschichte	1-1
1.2 Protokolle	1-1
1.3 Schichten, Protokolle und Dienste	1-2
1.4 Datenübertragung im Schichtenmodell	1-3
1.5 Die sieben Schichten des OSI-Modells	1-4
1.5.1 Überblick	1-4
1.5.2 Physical Layer (Bitübertragungsschicht)	1-4
1.5.3 Data Link Layer (Sicherungsschicht)	1-5
1.5.4 Network Layer (Vermittlungsschicht)	1-6
1.5.5 Transport Layer (Transportschicht)	1-9
1.5.6 Session Layer (Kommunikationsschicht)	1-10
1.5.7 Presentation Layer (Darstellungsschicht)	1-10
1.5.8 Application Layer (Verarbeitungsschicht)	1-10
1.6 Kritikpunkte zum OSI-Modell	1-11

Inhaltsverzeichnis Kapitel 2

2 Übertragungsmedien	2-1
2.1 Ausbreitungsgeschwindigkeit	2-2
2.2 Signaldämpfung und Dämpfungsbelag	2-2
2.3 Koaxialkabel	2-4
2.4 Paarsymmetrische Kabel (Twisted Pair)	2-5
2.4.1 Funktionsprinzip der paarsymmetrischen Kabel	2-6
2.4.2 Kabelkategorien	2-9
2.4.3 Stecker	2-10
2.5 Lichtwellenleiter	2-11
2.5.1 Grundprinzip der Glasfaser	2-11
2.5.2 Einteilung der Lichtwellenleiter	2-12
2.5.3 Multimode Glasfaser	2-14
2.5.4 Monomode Glasfaser	2-15
2.6 Strukturierte Gebäudeverkabelung	2-16

Inhaltsverzeichnis Kapitel 3

3 Physical Layer (Bitübertragungsschicht)	3-1
3.1 Kopplung der Kommunikationspartner	3-2
3.2 Übertragungsverfahren	3-2
3.2.1 Serielle asynchrone Übertragung	3-3
3.2.2 serielle synchrone Übertragung	3-4
3.3 Leitungscodes	3-5
3.3.1 Gleichspannungsfreiheit	3-6
3.3.2 Taktrückgewinnung	3-7
3.3.3 Nutzung der Bandbreite	3-8

Inhaltsverzeichnis Kapitel 4

4 Data Link Layer (Sicherungsschicht)	4-1
4.1 Aufteilen der Layer-3-Daten in Frames	4-2
4.2 Framing bei asynchroner Übertragung	4-3
4.3 Framing bei synchroner Übertragung	4-3
4.4 Wahl der Frame-Länge	4-5
4.4.1 Einfluss der Frame-Länge auf Jitter	4-5
4.4.2 Einfluss der Frame-Länge auf den Durchsatz	4-6
4.5 Fehlererkennung und Fehlerkorrektur	4-7
4.6 Fehlererkennung	4-7
4.6.1 Hamming-Distanz als Grundlage der Fehlererkennung	4-8
4.7 Fehlererkennung mit Parity	4-10
4.8 Prüfsumme	4-11
4.9 Cyclic Redundancy Check	4-11
4.10 Fehlerkorrektur	4-14
4.10.1 Backward Error Correction	4-14
4.10.2 Forward Error Correction	4-15
4.11 Flusssteuerung	4-16
4.12 Medium Zugriff	4-18
4.12.1 Deterministische Verfahren	4-18
4.12.2 Undeterministische Verfahren	4-20
4.13 Wichtige Layer-2-Protokolle	4-21

Inhaltsverzeichnis Kapitel 5

5 Lokale Netzwerke (Local Area Network)	5-1
5.1 Topologie	5-1
5.1.1 Bustopologie	5-1
5.1.2 Linientopologie	5-1
5.1.3 Ringtopologien	5-2
5.1.4 Stern topologie	5-3
5.1.5 Baumtopologie	5-4
5.2 Übertragungsarten	5-4
5.3 Normung für LAN und MAN	5-5
5.4 Shared Ethernet	5-8
5.4.1 Entstehungsgeschichte	5-8
5.4.2 Physical Layer 10BASE5 und 10BASE2	5-9
5.4.3 Leitungscodierung	5-10
5.4.4 Datenübertragung und Zugriffskontrolle	5-11
5.5 Aufbau des Ethernet-Frames	5-14
5.6 LAN-Erweiterungen mit Repeater	5-16
5.6.1 Konfigurationsregeln für Repeater	5-17
5.7 Übungen	5-21

Inhaltsverzeichnis Kapitel 6

6 Switched LAN und Ethernet-Technologien	6-1
6.1 Bridges	6-1
6.1.1 Funktionsweise	6-2
6.1.2 Vollduplex Betrieb	6-5
6.1.3 Remote- und Multiport-Bridges	6-5
6.1.4 Virtuelle LAN	6-8
6.1.5 Redundanz Protokolle	6-9
6.1.6 Weitere Leistungsmerkmale von Switches	6-11
6.2 Moderne Ethernet-Systeme	6-13
6.2.1 Twisted-Pair-Ethernet 10BASE-T	6-13
6.2.2 100 MBit/s-Ethernet-Systeme (Fast-Ethernet)	6-16
6.2.3 1000 MBit/s-Ethernet-Systeme (Gigabit Ethernet)	6-23
6.2.4 10 GBit/s-Ethernet-Systeme (Gigabit Ethernet)	6-26
6.3 Übungen	6-27
6.3.1 802.3-Normierung	6-27
6.3.2 LAN-Erweiterungen	6-27

Inhaltsverzeichnis Kapitel 7

7 Internet Protokolle des Network Layers	7-1
7.1 Network Layer	7-1
7.1.1 Adressierungsschema	7-2
7.1.2 Router	7-3
7.1.3 Brouter und Layer 3 Switches	7-6
7.2 Die Geschichte der TCP/IP-Protokollfamilie	7-6
7.3 IP Adressierung	7-8
7.3.1 Darstellung und Aufbau der Adresse	7-9
7.3.2 Subnetze und Subnetzmasken	7-10
7.3.3 Netz- und Broadcast-Adresse	7-11
7.3.4 Classful-Routing	7-12
7.3.5 Verwaltung der Adressen	7-14
7.4 Routing	7-15
7.4.1 Funktion und Routing-Tabelle	7-15
7.4.2 Route-Befehl zur Anzeige und Manipulationen der Routing-Tabelle	7-16
7.4.3 Flaches und hierarchisches Routing	7-17
7.5 IP Protokoll	7-19
7.5.1 IP-Protokoll-Header	7-19
7.5.2 Fragmentation and Reassembly	7-21
7.6 Kapselung und Umsetzung der IP-Adresse in die Hardware-Netzadresse . .	7-26
7.6.1 Kapselung eines IP-Datagramms	7-26
7.6.2 Adressauflösung	7-27
7.6.3 ARP	7-29
7.7 RARP – Reverse Address Resolution Protocol	7-34
7.8 Internet Control Message Protocol (ICMP)	7-35
7.8.1 Meldungs-Format	7-35
7.8.2 Destination Unreachable Message (Type 3)	7-37
7.8.3 Time Exceeded Message (Type 11)	7-39
7.8.4 Echo or Echo Reply Message (Type 8 / Type 0)	7-42
7.9 Übungen	7-44
7.9.1 IP-Protokoll	7-44
7.9.2 Routing	7-46
7.9.3 ICMP	7-48

Inhaltsverzeichnis Kapitel 8

8 Transport Layer	8-1
8.1 Einleitung	8-1
8.1.1 Kapselung	8-2
8.1.2 Multiplexen und Demultiplexen	8-2
8.2 User Datagram Protocol (UDP)	8-3
8.2.1 UDP-Header	8-3
8.2.2 Ports	8-4
8.3 Transmission Control Protocol (TCP)	8-5
8.3.1 Einleitung	8-5
8.3.2 TCP-Dienste für Anwendungen	8-7
8.3.3 Ende-zu-Ende-Dienst	8-8
8.3.4 Zuverlässigkeit	8-9
8.3.5 Fluss-Steuerung	8-13
8.3.6 Zuverlässiger Verbindungs-Aufbau und -Abbau	8-16
8.3.7 Überlastüberwachung	8-21
8.3.8 Das Format des TCP-Headers	8-23
8.4 Übungen	8-25

Inhaltsverzeichnis Kapitel 9

9 Netzwerk-Applikationen und Protokolle	9-1
9.1 Das Domain Name System	9-1
9.1.1 Überblick über das Domain Name System	9-2
9.1.2 DNS-Abfragen auswerten	9-5
9.1.3 Die Organisation des Domain Name Space im Internet	9-6
9.1.4 Adressen Namen zuordnen	9-9
9.2 Bootstrap Protocol (BootP)	9-10
9.2.1 Funktionsprinzip des BootP	9-10
9.2.2 BootP-Paketformat und Bedeutung der Felder	9-13
9.2.3 Dynamic Host Configuration Protocol (DHCP)	9-15
9.3 Trivial File Transfer Protocol (TFTP)	9-16
9.3.1 TFTP-Funktionen	9-16
9.3.2 TFTP-Funktion im Überblick	9-17
9.4 Simple Mail Transfer Protocol (SMTP)	9-18
9.4.1 SMTP 822-Format	9-19
9.4.2 SMTP-Kommandos	9-21
9.4.3 SMTP-Bestätigungen	9-22
9.4.4 Anwendungsbeispiel	9-23
9.4.5 Der MIME-Standard	9-24
9.4.6 Content-Transfer-Encoding	9-26
9.5 Hypertext Transfer Protocol (HTTP)	9-28
9.5.1 Funktionsweise	9-28
9.5.2 HTTP-Request	9-30
9.5.3 HTTP-Reply	9-30

Inhaltsverzeichnis Kapitel 10

10 Die Schnittstelle zum Transport-Layer	10-1
10.1 Einleitung	10-1
10.1.1 Das Socket-Application Program Interface	10-2
10.1.2 Socket und UNIX-Ein-/Ausgaben	10-3
10.1.3 Socket-Ein-/Ausgaben und Descriptoren	10-3
10.1.4 Socket-API und Parameter	10-4
10.2 Funktionen zur Integer-Konversion	10-5
10.3 Prozeduren zur Implementierung des Socket-API	10-6
10.3.1 Die Prozedur <code>socket</code>	10-6
10.3.2 Die Prozedur <code>close</code>	10-6
10.3.3 Die Prozedur <code>bind</code>	10-7
10.3.4 Die Prozedur <code>listen</code>	10-8
10.3.5 Die Prozedur <code>accept</code>	10-9
10.3.6 Die Prozedur <code>connect</code>	10-10
10.3.7 Die Prozedur <code>send</code>	10-10
10.3.8 Die Prozedur <code>recv</code>	10-10
10.4 Socket-Datenaustausch mit <code>read</code> und <code>write</code>	10-12
10.5 Weitere Socket-Prozeduren	10-12
10.6 Client-/Server-Beispiel	10-13
10.6.1 Verbindungsorientierte Kommunikation	10-13
10.6.2 Beispieldienst	10-13
10.6.3 Befehlszeilenargumente für die Programmbeispiele	10-14
10.6.4 Ablauf der Socket-Prozeduraufälle	10-14
10.6.5 Stream-Dienst und mehrfache <code>recv</code> -Aufrufe	10-15
10.6.6 Code für den Beispiel-Server	10-16
10.6.7 Code für den Beispiel-Client	10-19
10.6.8 Socket-Prozeduren und Blockierung	10-22
10.6.9 Codeumfang und Fehlerkontrolle	10-22
10.6.10 Test der Kommunikation	10-23
10.7 Sockets und Prozesse	10-24
10.7.1 Unix-Beispiel für einen parallelbetriebenen Server	10-24

Literaturverzeichnis

- [1] A. S. Tannenbaum: *Computer Networks*, Prentice Hall
- [2] D. Comer: *Computer-Netzwerke und Internets*, Prentice Hall
- [3] Mathias Hain: *TCP/IP Internetprotokolle im professionellen Einsatz*, International Thomson Publishing Company
- [4] W. Richard Stevens: *TCP/IP Illustrated Volume 1*, Addison Wesley
- [5] Ford, Lew, Spanier, Stevenson: *Handbuch Netzwerk-Technologien*, Cisco Press

1

Das OSI-Referenzmodell

1.1 Geschichte

Die International Standards Organisation (ISO) setzte 1977 eine Arbeitsgruppe ein, um ein Referenzmodell zur Beschreibung der Interprozesskommunikation zwischen räumlich entfernten Kommunikationspartnern zu entwickeln. Dieses Modell, das Open Systems Interconnection (OSI) Modell, wurde 1983 verabschiedet (ISO 7498 bzw. ITU-T X.200). Das ISO-Referenzmodell selbst definiert keine Netzarchitektur sondern beschreibt nur die Rahmenbedingungen für die Normung von Kommunikationsprotokollen. Das Modell erleichtert das Verstehen der komplexen Vorgänge bei der Datenübertragung zwischen zwei Kommunikationspartnern, indem die einzelnen Funktionen einer solchen Übertragung bestimmten Schichten zugeordnet werden.

1.2 Protokolle

Im zwischenmenschlichen Bereich kennt man Protokolle beispielsweise in der Diplomatie. Es sind Vorschriften, die unter anderem bei Staatsbesuchen den genauen Ablauf bis hin zu Kleiderordnungen, Anreden, Sitzordnung etc. beschreiben. Mit den Protokollvorschriften soll eine Umgebung geschaffen werden, damit die Teilnehmer ungeachtet ihrer Herkunft und kulturellen Hintergründe störungsfrei miteinander kommunizieren können. Unnötige Missverständnisse oder gar Streitigkeiten sollen damit vermieden werden.

Analog dazu ist in der Technik ein Kommunikationsprotokoll eine Vereinbarung, die festlegt, wie eine Datenübertragung zwischen zwei oder mehreren Kommunikationspartnern abläuft. Dabei können die Kommunikationspartner Bruchteile von Millimetern (Funktionsblöcke auf einem Chip) oder Lichtjahr (Raumsonde) entfernt sein. Die Protokolle selber können in Hardware oder Software implementiert sein; oft besteht eine gewisse Wahlfreiheit mit entsprechenden Vor- und Nachteilen.

1.3 Schichten, Protokolle und Dienste

Modularisierung ist ein gängiges Mittel zur Bewältigung der Komplexität von technischen Systemen. Ein System wird beschrieben und verständlich gemacht, indem es auf in sich geschlossene überblickbare Module aufgeteilt wird, die auf eine durchschaubare Art zusammenwirken. Wenn es um Kommunikationssysteme geht, so ist die Aufteilung in Module zu einem gewissen Grad naheliegend, weil es immer ähnliche Probleme sind, die es zu lösen gilt. Als geeignetes Mittel haben sich übereinanderliegende Protokollsichten (Layer) herausgestellt, die eine Dienstleistungs-hierarchie bilden. Eine Schicht hat die Aufgabe, der darüberliegenden Schicht bestimmte Dienste anzubieten. Im Beispiel von Abbildung 1.1 wird ein Modell mit drei Schichten betrachtet. Die

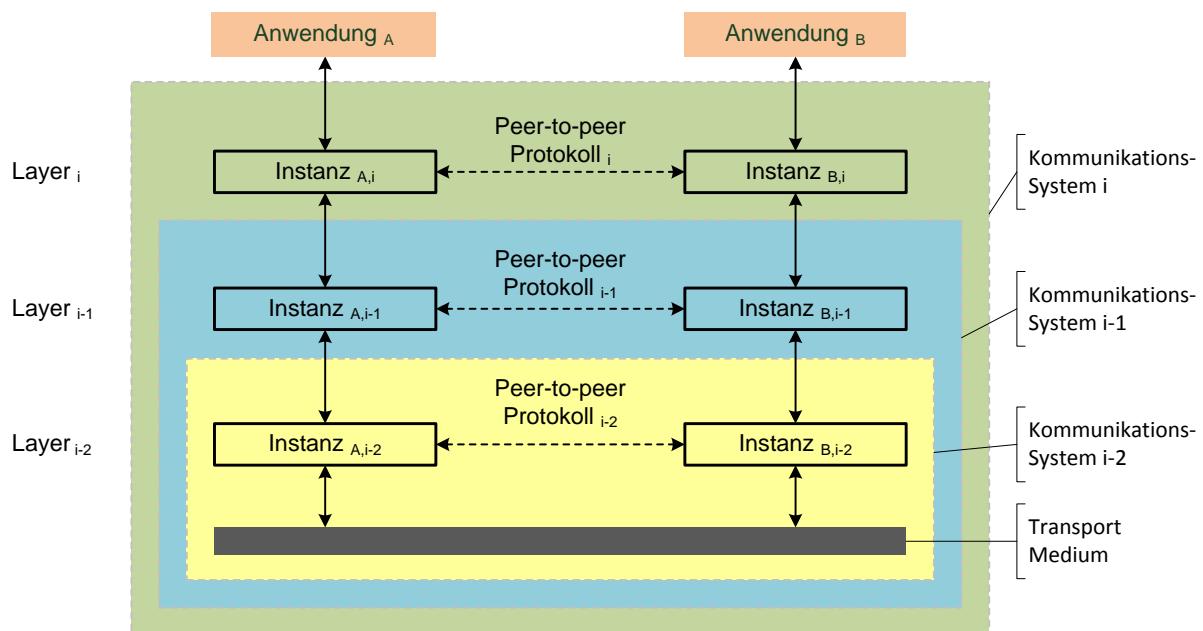


Abbildung 1.1: Modell mit drei Schichten

Schicht **i-1** bietet der höheren Schicht **i** einen Dienst an. Die Schicht **i** nutzt diesen Dienst ohne Wissen über dessen Realisierung und allfällige darunterliegende Schichten. Die Schicht **i** hat den angebotenen Dienst mit Hilfe der tieferen Schichten zu realisieren. Die Vereinbarung über die Dienste einer Schicht wird Schnittstelle oder Interface genannt. Eine bestimmte Schicht **i** ist durch kommunizierende gleichartige Instanzen am Ort A bzw. B realisiert. Die Regeln, wie die Kommunikation zwischen der Instanz **A_i** und **B_i** abläuft, werden als Peer-to-Peer-Protokoll bezeichnet (Peer-to-Peer heisst „unter Gleichgestellten“).

Ein Protokoll legt also das Format und die Bedeutung der von den Partnereinheiten innerhalb einer Schicht ausgetauschten Nachrichten fest. Eine Schicht kann ihr Protokoll beliebig ändern, ohne dass die übergeordnete Schicht davon betroffen ist. Eine Änderung des Dienstes hat jedoch direkte Auswirkungen auf die übergeordnete Schicht.

Die Anwendung kommuniziert mit der Partneranwendung im entfernten System, indem sie die Dienste der höchsten Schicht beansprucht. Diese wiederum benutzt die Dienste der darunterliegenden Schichten. Die unterste Schicht sorgt für die physikalische Übertragung der Daten. Die zusammenarbeitenden Protokolle übereinanderliegender Schichten werden als **Protokoll-Stack** bezeichnet (Stack = engl. Stapel).

1.4 Datenübertragung im Schichtenmodell

Will eine Anwendung Daten an eine Anwendung im entfernten System senden, so übergibt sie einen Datenblock über einen Service Access Point (SAP) der höchsten Schicht. Jede Schicht ergänzt normalerweise die von der übergeordneten Stelle erhaltenen Daten für Steuerzwecke mit zusätzlicher Information, dem sogenannten Header (H). In gewissen Fällen wird zusätzliche Information den Daten angefügt; typischerweise eine Prüfsumme am Ende der Daten um allfällige Übertragungsfehler zu erkennen. Man spricht dann von einem Trailer.

Diese Vorgehensweise ist vergleichbar mit dem Verpacken und Adressieren eines Briefes in einem Couvert. Da dies auf jeder Ebene geschieht, gleichen die übertragenen Daten schlussendlich einer Babuschka (Puppe in Puppe in Puppe...). Auf der Empfängerseite interpretiert und entfernt jede Schicht die für sie bestimmte Information und übergibt die Nutzdaten der höheren Schicht. So gelangen die Daten von der Ursprungs- zur Ziel-Anwendung, ohne dass die Anwendung wissen muss, wie der Transfer realisiert ist.

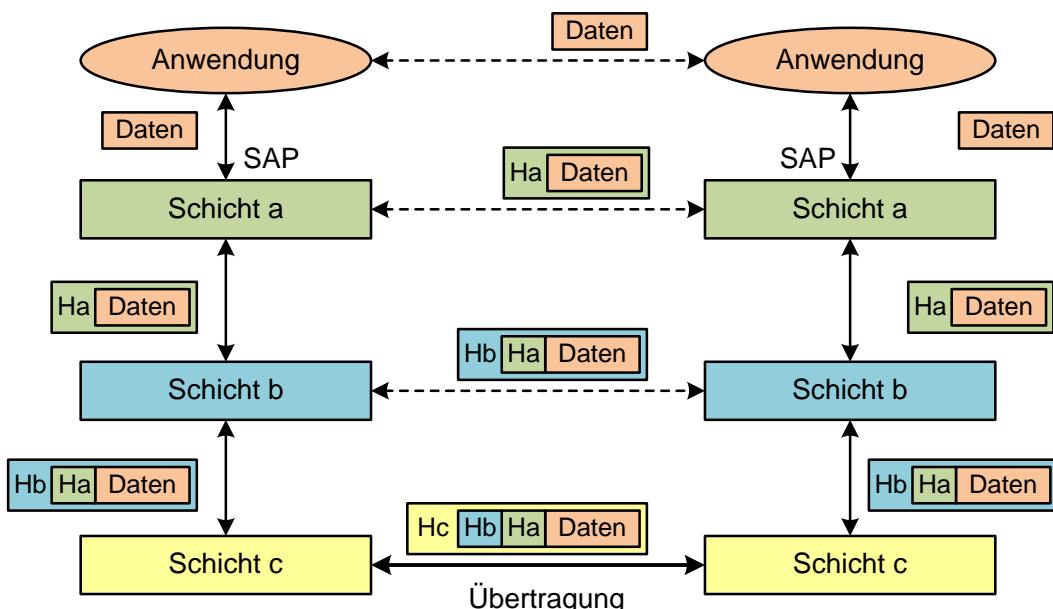


Abbildung 1.2: Horizontale und vertikale Kommunikation zwischen den Protokoll-Instanzen

Die zusätzlichen Informationen, die von den Schichten zugefügt werden, dienen dazu, die protokollspezifischen Funktionen zu steuern. Logisch kommunizieren die Partnereinheiten innerhalb einer Schicht (horizontal) untereinander. Physisch erfolgt die Kommunikation vertikal also zwischen den Schichten und erst auf der untersten Stufe (horizontal) über die Übertragungsstrecken. Die zusätzlichen Informationen (Header / Trailer) der Schichten belasten die Übertragungsstrecken (also die Datenverbindung zwischen zwei Punkten) und verringern damit die von der Applikation effektiv nutzbare Übertragungsrate.

1.5 Die sieben Schichten des OSI-Modells

1.5.1 Überblick

Das OSI-Referenzmodell (Abbildung 1.3) teilt die Kommunikation in sieben Schichten (1...7) ein und definiert die Funktionen und Dienste jeder Schicht. Man spricht darum auch vom Schichtenmodell. Die Schichten 1 bis 4 sind für die Übertragung der Daten zwischen den Endeinrichtungen verantwortlich, während die Schichten 5 bis 7 bei der Datenübertragung das Zusammenwirken mit dem Anwendungsprogramm und dem Betriebssystem koordinieren. Die Schichten 1 bis 4 werden darum auch als **Transportschichten** oder Transportsystem, die Schichten 4 bis 7 als **Anwendungsschichten** bezeichnet.

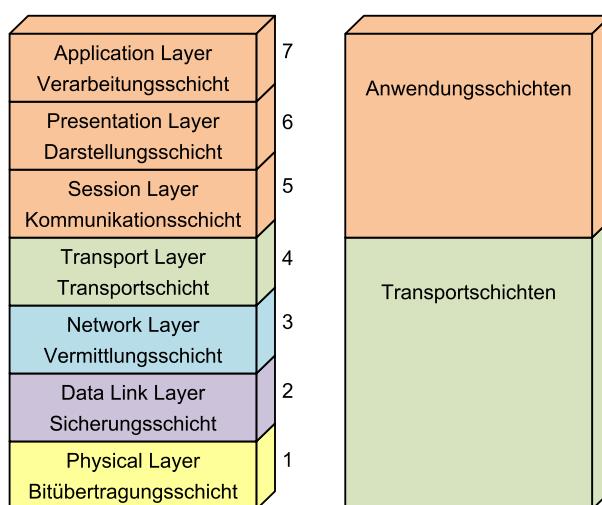


Abbildung 1.3: OSI-Modell mit 7 Schichten

Anmerkung: In diesem Lehrgang wird der Internet-Protokollwelt besonders Rechnung getragen. Der Ursprung der Internet-Protokolle geht auf die Zeit vor der Entwicklung des OSI-Modells zurück. Obwohl Internet- und OSI-Referenzmodell nicht in allen Punkten übereinstimmen, ist die grundsätzliche Aufteilung von Protokollfunktionen auf Schichten in beiden Modellen vergleichbar. Im Internet werden allerdings die Funktionen der 3 höchsten OSI-Layer nicht auseinander gehalten. Sie sind durch einen einzigen Layer abgebildet (Abbildung 1.4).

1.5.2 Physical Layer (Bitübertragungsschicht)

Diese Schicht sorgt für die ungesicherte Übertragung eines Bit-Stromes zwischen zwei Knoten über das physikalische Medium. Es geht also um Fragen der Anpassung von Endeinrichtungen an die Übertragungseinrichtung und das Übertragungsmedium selbst:

- elektrische oder optische Eigenschaften (Pegel, Timing, Frequenzen, usw.)
- Codierung (Return to Zero, non return to Zero, AMI, Manchester, FSK, usw.)
- mechanische Eigenschaften (Stecker, Pinbelegung, usw.)

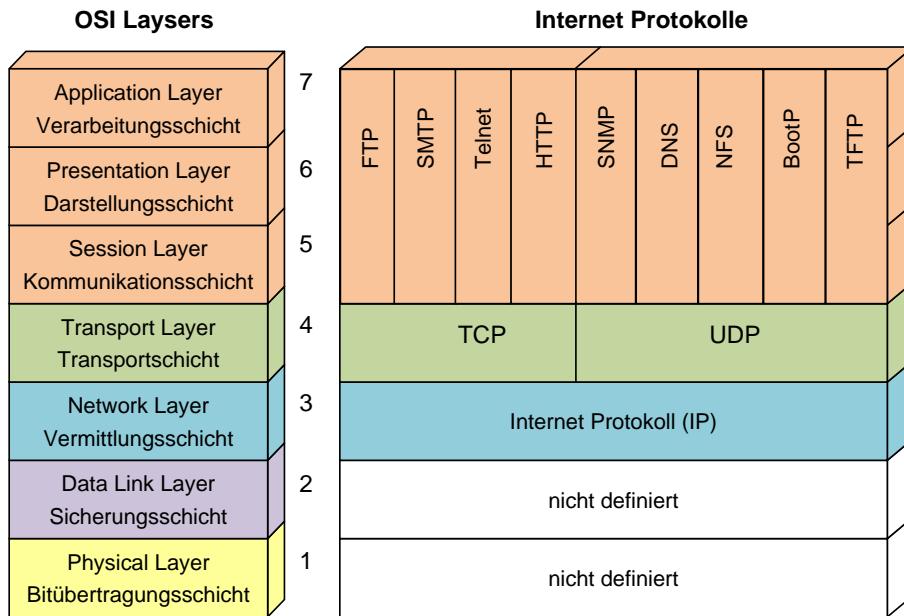


Abbildung 1.4: Internet-Protokollwelt versus OSI-Modell

Das eigentliche Übertragungsmedium (z.B. Kabel) liegt unterhalb des Physical Layer und damit ausserhalb des OSI-Referenzmodells. Die physikalische Schicht ist, wie jede OSI-Schicht, austauschbar, ohne dass die anderen Schichten betroffen sind (im Rahmen der physikalischen Grenzen des Mediums). So kann z.B. eine Kupferleitung, ohne Rückwirkung auf den Data Link Layer, durch eine Glasfaserleitung ersetzt werden, indem der Physical Layer ausgetauscht wird.

1.5.3 Data Link Layer (Sicherungsschicht)

Der Data Link Layer stellt der höheren Schicht eine gesicherte Übertragungsstrecke zwischen zwei direkt miteinander verbundenen Knoten zur Verfügung. Die Anforderungen an den Data Link Layer sind davon abhängig, ob genau zwei oder mehr Knoten durch das Übertragungsmedium miteinander verbunden sind.

Für Punkt-Punkt-Verbindungen zwischen genau zwei Teilnehmern muss der Data Link Layer folgende Aufgaben erfüllen:

- Realisieren einer sicheren Verbindung zwischen zwei direkt miteinander verbundenen Einrichtungen. Dazu werden Massnahmen zur Fehlererkennung und Fehlerkorrektur durch wiederholte Übertragung notwendig.
- Verpacken der vom Network Layer erhaltenen Datenblöcke in Datenrahmen für die Übertragung und das Auspacken der Datenblöcke aus den empfangenen Datenrahmen.
- Fluss-Steuerung (Flow Control). Darunter versteht man alle Massnahmen, die getroffen werden, damit der Sender Daten nicht schneller sendet, als sie der Empfänger aufnehmen kann.

Bei Konfigurationen mit mehreren durch ein Übertragungsmedium verbundenen Teilnehmern (z.B. LAN, Funknetz) sind zusätzlich folgende Funktionen notwendig:

- Adressierung der Teilnehmer: Jeder Knoten muss über eine innerhalb der Konfiguration eindeutige Adresse angesprochen werden können. Es handelt sich hier um eine Layer 2 Adresse (z.B. MAC-Adresse im Ethernet), die nicht mit der Network Layer Adresse (z.B. IP-Adresse) verwechselt werden darf.
- Steuerung des Zugriffs auf das Medium (Medium Access Control): Da mehrere Teilnehmer ein gemeinsames Übertragungsmedium teilen, muss definiert werden, wie darauf zugegriffen werden darf.

Es ist zu beachten, dass der Transport Layer für netzweite Verbindungen Funktionen definiert, die mit denen des Data Link Layers für direkt miteinander verbundene Knoten vergleichbar sind.

1.5.4 Network Layer (Vermittlungsschicht)

Der Network Layer hat die Aufgabe, den Datenaustausch zwischen Knoten auf eine einheitliche Weise zu ermöglichen, die unabhängig von den darunterliegenden Übertragungsabschnitten ist (Abbildung 1.5). Dazu wird eine netzweite eindeutige Layer 3 Adressierung benötigt, sowie ein Verfahren, das erlaubt, in einem Netz, das mehrere Wege bereitstellt, einen geeigneten Weg zu wählen (Routing).

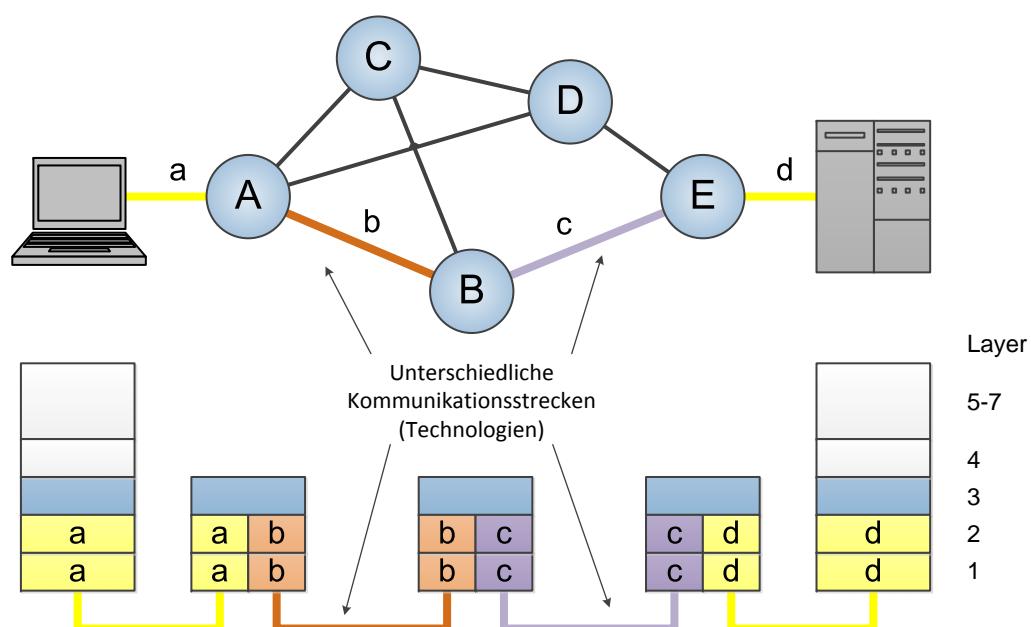


Abbildung 1.5: Einheitliche Kommunikation über unterschiedliche Übertragungsabschnitte

Auf dem Network Layer unterscheidet man zwischen verbindungsorientierten und verbindungslosen Diensten.

a) Verbindungsorientierter Dienst

Ein verbindungsorientierter Dienst überträgt Daten über eine Verbindung (Abbildung 1.6). Die Eigenschaften einer solchen Verbindung sind vergleichbar mit denen eines Schlauches. Das heisst, die zu transportierenden Elemente gehen alle den gleichen Weg und kommen in der Reihenfolge an, in der sie gesendet wurden. Zur Erkennung der Pakete tragen diese im Header eine Verbindungsnummer (in Abbildung 1.6 mit X bezeichnet). X ist keine Adresse, sondern lediglich ein lokaler Identifier, der für den Weiterleitungsentscheid verwendet wird.

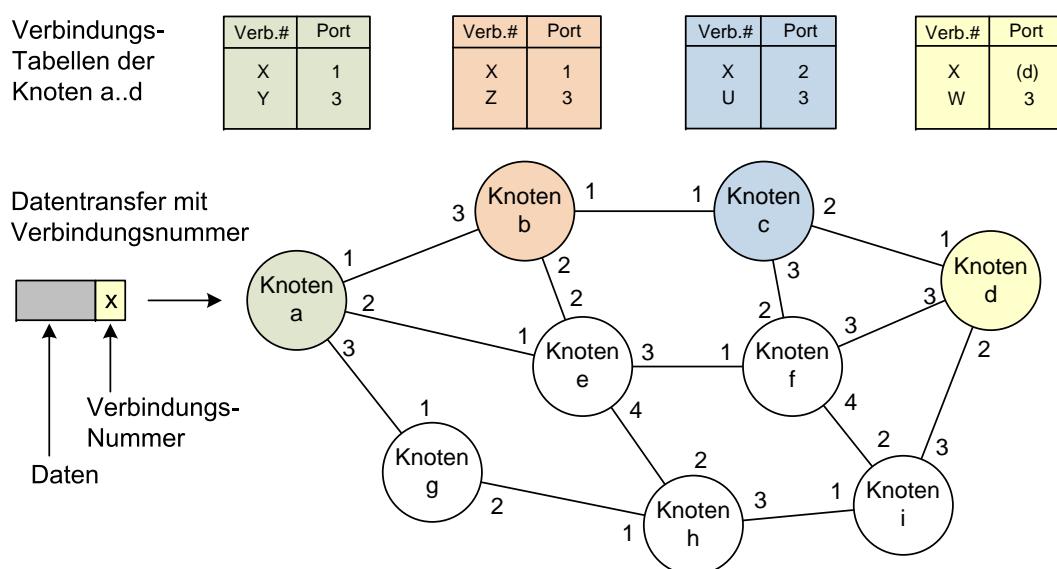


Abbildung 1.6: Verbindungsorientierte Vermittlung

Vor der eigentlichen Übertragung von Nutzdatenpaketen muss also eine Verbindung aufgebaut oder eingerichtet werden. Dabei wird ein Pfad durch das Netzwerk festgelegt und alle Transit-knoten entlang des Pfades werden so eingestellt, dass sie die der Verbindung angehörenden Pakete erkennen und richtig behandeln können. Nach erfolgter Datenübertragung muss die Verbindung wieder abgebaut werden, um die entsprechenden Ressourcen wieder freizugeben.

Auf der Dienstebene unterscheidet man, ob eine fest vorgegebene Verbindung verwendet wird (Permanent Virtual Circuit – PVC) oder eine Verbindung, die bei Bedarf dynamisch aufgebaut und nach der Übertragung wieder abgebaut wird (Switched Virtual Circuits – SVC).

Ein wichtiges verbindungsorientiertes Verfahren ist das Multi Protocol Label Switching (MPLS), das bevorzugt in Provider Backbones verwendet wird. Der Name kommt daher, dass die Verbindungsnummer als Label bezeichnet wird.

b) Verbindungsloser Dienst (Datagram)

Ein verbindungsloser Dienst des Network Layers funktioniert analog zur Briefpost (auch ein verbindungsloser Dienst). Das beschriebene Blatt (Daten) wird in einem Umschlag verpackt, der mit der vollständigen Adresse des Empfängers versehen ist. Der Brief (Datagram) wird unabhängig von anderen Briefen zum Ziel transportiert. Schickt man jeden Tag einen Brief an den gleichen Empfänger, so ist nicht sichergestellt, dass die Briefe beim Empfänger ankommen und dass deren Reihenfolge eingehalten ist.

Bei einem verbindungslosen Dienst trägt jedes zu transportierende Datenpaket (Datagram) die volle Network-Layer-Adresse des Empfängers und wird unabhängig von anderen Datenpaketen vom Sender zum Empfänger übertragen (Abbildung 1.7). Der Network Layer in den einzelnen Knoten entscheidet aufgrund der Zieladresse, auf welchem Weg die Datenpakete weitergeleitet werden. Jeder Knoten muss Informationen über den Netzaufbau haben, um für die Weiterleitung (Forwarding) den optimalen Weg bestimmen zu können. Diese Information wird in Tabellen (Routing-Tabellen) abgelegt, die für jedes Ziel mindestens einen Weg enthalten. Ändert sich die Situation im Netzwerk müssen die verschiedenen Datenpaket nicht den gleichen Weg wie die vorgängigen nehmen. Als Folge davon können nachfolgende Datenpakete vorgängige überholen und in vertauschter Reihenfolge beim Empfänger eintreffen.

Das Internet Protocol (IP) ist ein typisches Beispiel eines verbindungslosen Dienstes über ein paketvermitteltes Netz.

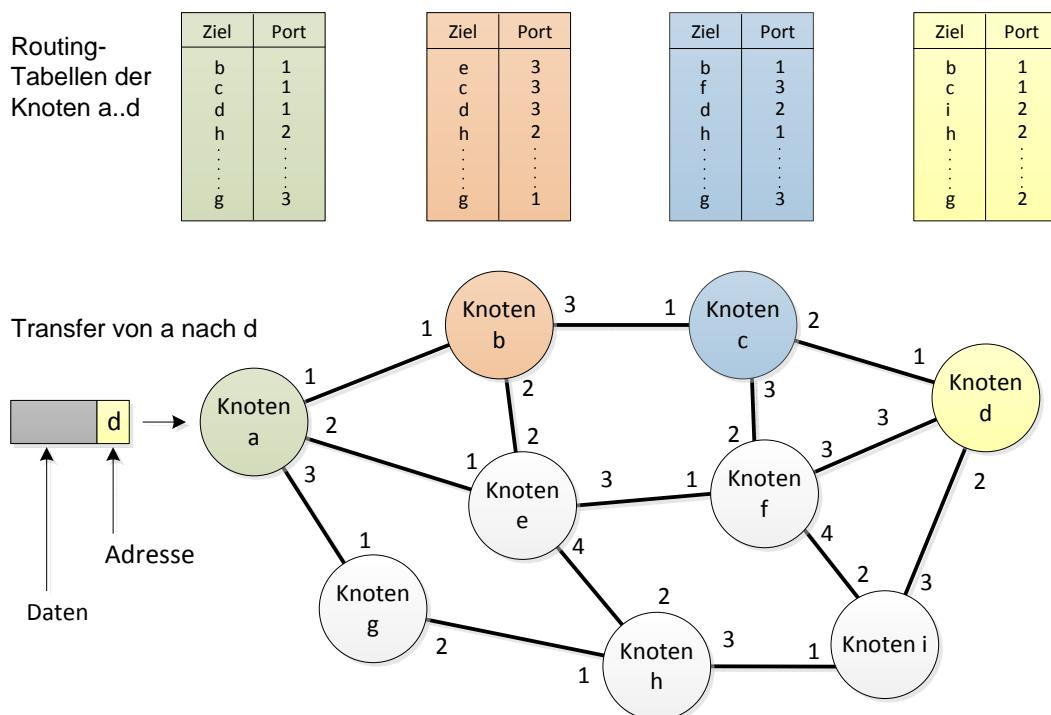


Abbildung 1.7: Network Layer mit verbindungslosem Dienst

c) Gegenüberstellung der Network Layer Dienste

Welches sind die Vor- und Nachteile dieser beiden Kommunikationsarten?

- Während bei **verbindungsloser Paketvermittlung** Datenpakete **spontan** und ohne irgendwelche Vorbereitungen an ein beliebiges Ziel gesendet werden können, muss im verbindungsorientierten Fall zuerst die Verbindung aufgebaut werden. Das verzögert den Ablauf und verursacht Zusatzaufwand.
- Beim Verbindungsauflauf kann durch entsprechende Massnahmen (z.B. Reservation von Bandbreite entlang der Strecke) sichergestellt werden, dass die für die Verbindung geforderte Charakteristik (Durchsatz, Delay, Verlust) eingehalten wird.
- **Verbindungsorientierte Vermittlung** erlaubt die Kontrolle und **gezielte Lenkung** von Verkehrsströmen. Durch geeignete Wahl der Route kann die Last gezielter im Netzwerk verteilt werden.
- Verbindungslose Vermittlung wird (ohne weitere Massnahmen von Außen) bei einem Unterbruch oder Überlastung einer Teilstrecke andere Routen benutzen.
- Bei einem **verbindungsorientierten Dienst** ist **sicher gestellt**, dass die **Reihenfolge** der Daten beibehalten bleibt. Bei einem **verbindungslosen Dienst** muss (falls gefordert) die **Reihenfolge durch den Transport-Layer wiederhergestellt** werden.
- Die verbindungsorientierte Vermittlung bedingt, dass in allen Transitknoten des Pfades für jede aktive Verbindung ein Eintrag vorliegt. Dies bindet Ressourcen, die auch dann wieder freigegeben werden müssen, wenn die Verbindung nicht ordnungsgemäß abgebaut wird. Im Gegenzug für diesen Aufwand ist der Weiterleitungsentscheid aufgrund einer Verbindungsnummer einfacher zu treffen als mit Routing-Tabellen.

1.5.5 Transport Layer (Transportschicht)

Der **Transport Layer** ist die **höchste Schicht des Transportsystems** und hat die Aufgabe, den Prozessen des Session Layers einen **effizienten Datentransport** durch das Netz anzubieten. Das Ziel dabei ist, die vielen Transitknoten einfach zu halten und den Aufwand den beiden Endsystemen der Übertragungsstrecke zu überlassen. Wie Abbildung 1.8 zeigt, ist die Transportschicht darum nur in den Endsystemen, also den Sende- und im Empfangsknoten (u, y) vorhanden. Die Transitknoten (n, x) benötigen nur die untersten drei Schichten.

Da der Anwender oft keinen Einfluss auf die Qualität des Netzes (Layer 1...3) hat, ist es die Aufgabe des Transport Layers, eine End-zu-End-Übertragung mit definierter Qualität zu gewährleisten. Die Funktionen des Layer 4 müssen also auf die Zuverlässigkeit der Layer 1...3 abgestimmt sein. Benötigt die Applikation einen zuverlässigen Dienst und arbeiten die unteren Schichten nicht zuverlässig, so muss der Transport Layer den zuverlässigen Dienst realisieren. Arbeiten die unteren Schichten bereits zuverlässig, so kann der Transport Layer einfacher gestaltet werden.

Wie im Network Layer gibt es auch im **Transport Layer verbindungslose und verbindungsorientierte Dienste**. Ein verbindungsorientiertes Protokoll hat sicherzustellen, dass die Reihenfolge der Datenelemente wiederhergestellt wird und ist im Allgemeinen auch gegen Datenverlust und Datenverfälschung gesichert.

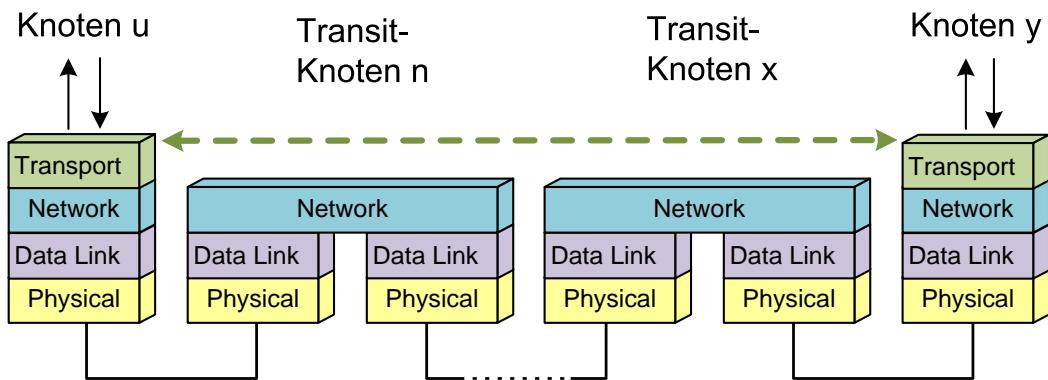


Abbildung 1.8: Transport Layer

Wichtige Layer 4 Protokolle sind:

- User Data Protocol (**UDP**): verbindungsloser, unsicherer Dienst im Internet
- Transmission Control Protocol (**TCP**): verbindungsorientierter, sicherer Dienst im Internet

1.5.6 Session Layer (Kommunikationsschicht)

Im Session Layer wird festgelegt, wie die Kommunikation zwischen den Partnern abläuft. Dazu werden Dienste zum Auf- und Abbau von Sitzungen bereitgestellt, so dass einer oder mehrere Prozesse auf das Transportsystem zugreifen können. Session Layer Verbindungen werden mit Hilfe von Transport Layer Diensten aufgebaut. Bricht die Transportverbindung zusammen, so ist es die Aufgabe des Session Layers, eine neue Verbindung aufzubauen.

1.5.7 Presentation Layer (Darstellungsschicht)

Der Presentation Layer stellt Dienste zur **Darstellung der zu übertragenden Daten zur Verfügung**. Die Information der Anwendungsschicht wird in eine neutrale Form des Kommunikationssystems transferiert, die von Komponenten unterschiedlicher Hersteller verstanden wird. Versteht ein Partner die neutrale Form des Kommunikationssystems, so entfällt die Darstellungsschicht.

Ein typisches Beispiel für einen Dienst des Presentation Layers ist die Kodierung von Daten auf standardisierte Art. Für die Darstellung von Zeichen werden von Anwendungen unterschiedliche Codes verwendet wie z.B. ASCII, ISO oder Unicode. Ebenso lassen sich Ganz- und Gleitpunktzahlen auf unterschiedliche Art darstellen. Der **Presentation Layer hat die Aufgabe, die entsprechenden Umwandlungen vorzunehmen**.

1.5.8 Application Layer (Verarbeitungsschicht)

Der Application Layer ist das **Bindeglied zur eigentlichen Anwendung** und legt fest, wie die Kommunikationspartner zur Lösung einer Aufgabe zusammenarbeiten.

Typische Protokolle sind:

- File Transfer: File Transfer Protocol (**FTP**)
- E-Mail: Simple Mail Transfer Protocol (**SMTP**)
- World Wide Web: Hyper Text Transport Protocol (**HTTP**)
- Namensauflösung: Domain Name System (**DNS**)

Wie Abbildung 1.4 zeigt, wird bei den Internet-Applikationen je nach Anforderungen ein sicherer oder unsicherer Transport-Dienst verwendet.

1.6 Kritikpunkte zum OSI-Modell

Neben dem OSI-Modell gab es auch einen vollständigen OSI-Protocol-Stack, der sich aber durchsetzen konnte. Gründe waren dessen Kosten, die hohe Komplexität und die Konkurrenz durch die Internet-Protokolle (TCP/IP), die mit wachsender Akzeptanz von Unix lizenziert zur Verfügung standen.

Beide Stacks wurden in einer Zeit konzipiert, in der die Netze praktisch ausschließlich dem Datenaustausch dienten und nur einem relativ kleinen Kreis von Wissenschaftlern und Spezialisten zugänglich waren.

Aus heutiger Sicht fehlen wichtige Themen wie:

- 
- Daten-Sicherheit und -Verschlüsselung
 - Hochverfügbarkeit und Redundanz
 - Netzwerk-Management (Überwachung und Steuerung)
 - Zeitsynchronisation

Diese Punkte lassen sich zwar im OSI-Modell irgendwie unterbringen, wurden ursprünglich jedoch nicht adressiert.

2

Übertragungsmedien

Abbildung 2.1 gibt einen ersten Überblick über die gängigen Übertragungsmedien. Im Folgenden werden paarsymmetrische Kabel, Koaxialkabel und Lichtwellenleiter genauer betrachtet. Funkübertragung ist zur Zeit sehr populär, würde aber den Rahmen dieses Moduls sprengen und kann darum leider nicht behandelt werden.

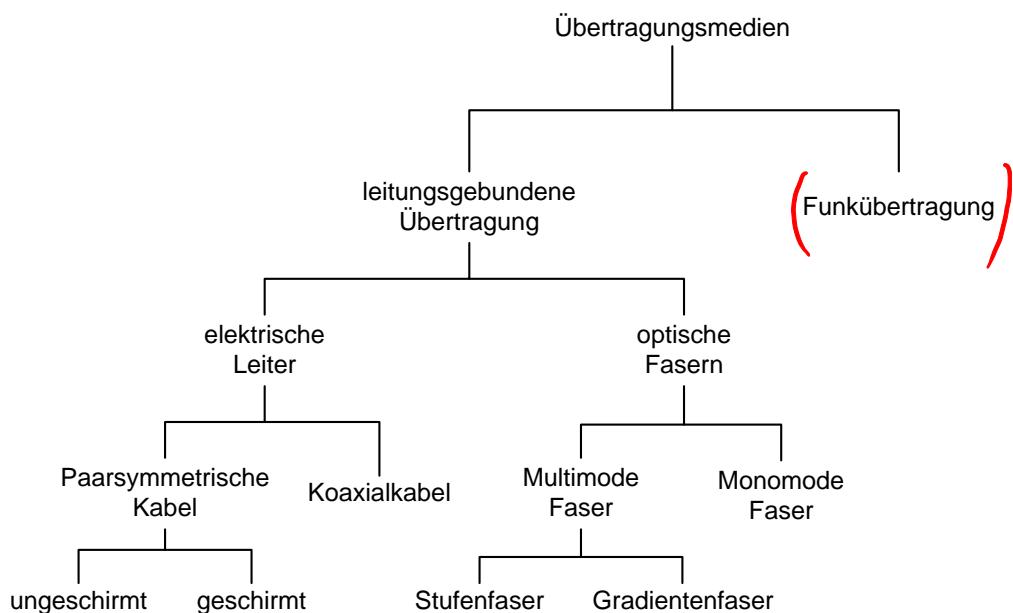


Abbildung 2.1: Überblick der Übertragungsmedien

Für die Auswahl eines Übertragungsmediums sind die erreichbare Distanz und die maximale Datenübertragungsrate die beiden wichtigsten Eigenschaften. Die Übertragungsmedien bilden die physikalische Basis, sind aber nicht alleine für diese Leistungsmerkmale verantwortlich. Durch

(aufwändige) Signalverarbeitung kann viel gewonnen werden, aber das hat seinen Preis. In der Praxis spielen weitere Kriterien wie die elektrische/mechanische Robustheit, die Kosten, einfache Handhabbarkeit und Gewicht eine ebenso wichtige Rolle.

2.1 Ausbreitungsgeschwindigkeit

Es gibt **physikalische Grenzen**, die auch mit noch so grossen Aufwand nicht überschritten werden können. Dazu gehört die **Ausbreitungsgeschwindigkeit der Signale**. Funk- oder Licht-Signale sind elektromagnetische Wellen, die sich im **Vakuum mit Lichtgeschwindigkeit $c_0 = 299'792'458 \text{ m/s}$** ausbreiten. Die Vakuumlichtgeschwindigkeit kann **nicht überschritten** werden. Breitet sich ein Signal in Materie aus (gleichgültig ob Glas oder Metall), so reduziert sich dessen Ausbreitungsgeschwindigkeit in jedem Fall. Das genaue Mass der Reduktion ist abhängig von den Eigenschaften des Übertragungsmediums. Für unsere Überlegungen verwenden wir stets die folgende Näherung:

$$C_{\text{Medium}} = 200'000 \text{ km/s} \approx \frac{2}{3} c_0 \quad (2.1)$$

2.2 Signaldämpfung und Dämpfungsbelag

Die **Signaldämpfung** bezeichnet die Leistungsabnahme eines Signals auf einer Übertragungsstrecke (Engl. Insertion Loss oder Attenuation). Sie ist ein wesentlicher Faktor, der die erreichbare Distanz beschränkt (Abbildung 2.2). Eine Verstärkung beim Empfänger ist zwar möglich, aber nur solange das Signal aus den Störungen extrahiert werden kann¹. Mögliche Ursachen der Störungen sind vielfältig; beispielsweise Übersprechen von anderen Leitungen, Rauschen des Empfängers, elektromagnetische Einstreuungen von anderen Geräten etc.

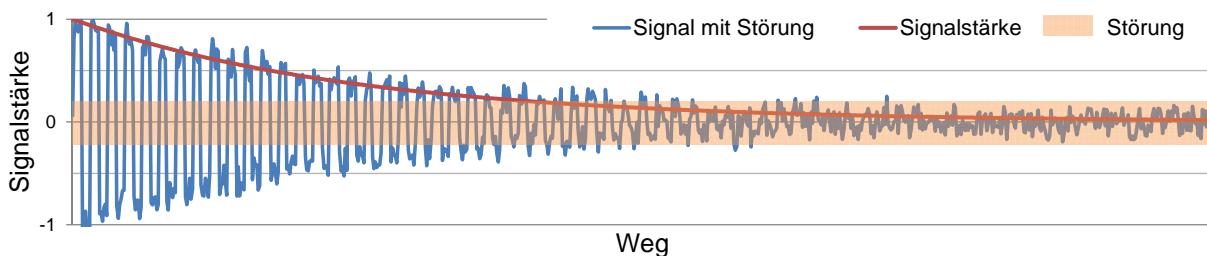


Abbildung 2.2: Signaldämpfung als begrenzender Faktor der erreichbaren Distanz

Signaldämpfung und Resistenz gegen Einstreuungen gehören zu den wichtigsten Eigenschaften des Übertragungsmediums. Die Angabe der **Signaldämpfung** erfolgt in **dB (Dezibel)** als logarithmische Verhältniszahl von der Eingangsleistung P_1 zur Ausgangsleistung P_2 :

$$\text{Signaldämpfung [dB]} = 10 * \log\left(\frac{P_1}{P_2}\right)$$

¹ Abbildung 2.2 suggeriert, dass ab ca. Bildmitte in jedem Fall kein Empfang mehr möglich ist. Dem ist nicht so: Es gibt sogar Verfahren, die ein Signal im Störrauschen verstecken. Dies ist jedoch nicht Thema dieses Fachs.

Berechnet man die Signaldämpfung über die Spannungen, so gilt: $\frac{P_1}{P_2} = \left(\frac{U_1}{U_2}\right)^2$

$$\text{Signaldämpfung [dB]} = 10 * \log\left(\frac{P_1}{P_2}\right) = 10 * \log\left(\left(\frac{U_1}{U_2}\right)^2\right) = 20 * \log\left(\frac{U_1}{U_2}\right) \quad (2.2)$$

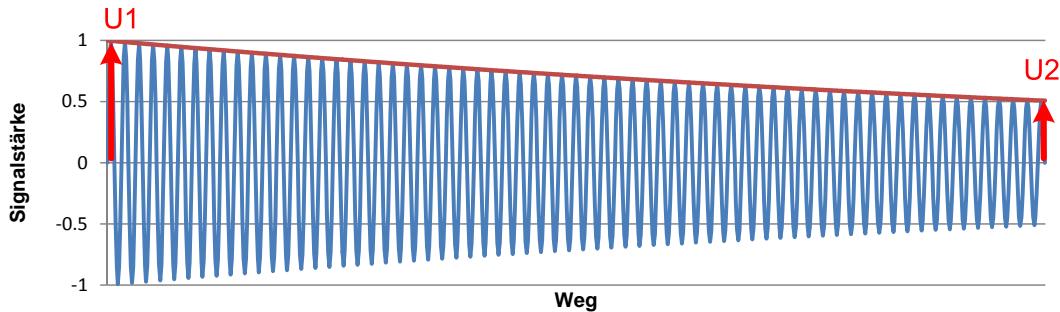


Abbildung 2.3: Beispiel zur Signaldämpfung

Im Beispiel von Abbildung 2.3 gilt:

$$U_1/U_2 = 1/0.5 = 2 \quad \text{Signaldämpfung} = 20 * \log(2) = 6\text{dB}$$

Eine Dämpfung von 6 dB bedeutet also eine Leistungsabnahme um den Faktor 4 oder eine Spannungsabnahme um den Faktor 2.

Für Übertragungsmedien ist (anstelle der absoluten Dämpfung) die Dämpfung pro Distanz aussagekräftiger. Diese charakteristische Größe wird **Dämpfungsbelag** genannt, typischerweise in dB/100 m oder dB/km angegeben. Wie in Abbildung 2.3 ersichtlich, steigt die Signaldämpfung mit der Frequenz an und variiert auch bei gleichartigen Übertragungsmedien (hier verschiedene paarsymmetrische Kabel und ein Koaxialkabel).

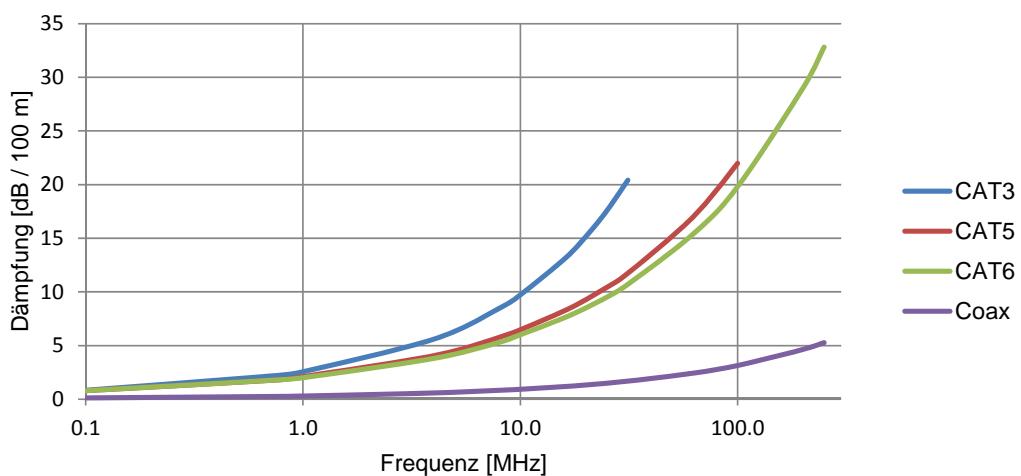


Abbildung 2.4: Dämpfungsbelag als Funktion der Frequenz

2.3 Koaxialkabel

Koaxialkabel eignen sich gut für die Übertragung von hochfrequenten Signalen. Sie haben im Vergleich zu paarsymmetrischen Kabeln einen kleinen Dämpfungsbelag (Abbildung 2.4) und sind auch unempfindlicher gegenüber elektromagnetischen Störungen.

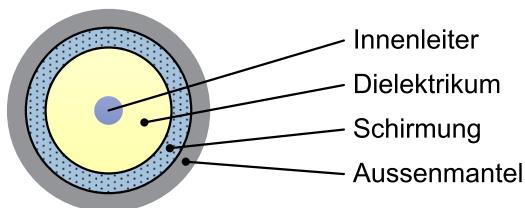


Abbildung 2.5: Querschnitt durch ein Koaxialkabel

Wie Abbildung 2.5 zeigt, bestehen einfache Koaxialkabel aus dem Innenleiter, dem Dielektrikum, der Schirmung und dem Kunststoffaussenmantel.



Abbildung 2.6: Ausbau eines einfachen Koaxialkabels

Dank der **geringen Dämpfung und des hohen Störabstands** können **grössere Distanzen überbrückt werden als mit paarsymmetrischen Kabeln**. Die Handhabung muss hingegen mit grosser Sorgfalt erfolgen. Ein Kabel darf **weder geknickt noch gequetscht** werden und beim Verlegen **sind minimale Radien einzuhalten**.

In den Anfangszeiten der Netzwerke kamen fast ausschliesslich Koaxialkabel zum Einsatz, unter anderem weil sich damit Bus-Topologien einfach realisieren lassen (siehe Ethernet, Abschnitt 5.3). Heute werden Koaxialkabel für Punkt-Punkt-Verbindungen in Hochgeschwindigkeitsnetzen (10, 40 und 100 Gigabit-Ethernet) verwendet. Dabei kommen speziell für diesen Zweck entwickelte Twinax-Kabel (Abbildung 2.7) zum Einsatz. Bei diesem sind mehrere symmetrische Adernpaare einzeln geschirmt und mit einer Gesamtschirmung versehen. Sie bieten damit höchsten Schutz gegen elektromagnetische Beeinflussung und geringe Laufzeitdifferenzen (Skew).

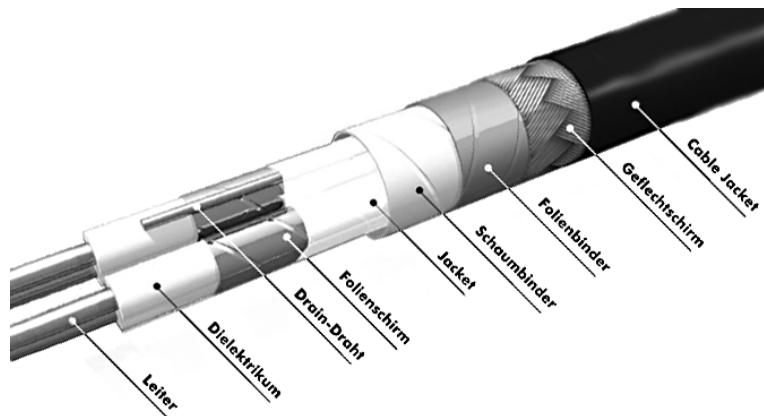


Abbildung 2.7: Aufbau eines Twinax-Kabels (Quelle: <http://www.itwissen.info>)

2.4 Paarsymmetrische Kabel (Twisted Pair)

In der Telefonie werden paarsymmetrische Kabel mit verdrillten Aderpaaren schon lange eingesetzt. Es gibt darum auf der ganzen Welt eine riesige Anzahl von bestehenden Installationen, die alle auf paarsymmetrischen Kabeln basieren.

Fortschritte in der Signalverarbeitung erlauben, Twisted Pair (Abbildung 2.8) auch für die breitbandige Datenübertragung zu nutzen.

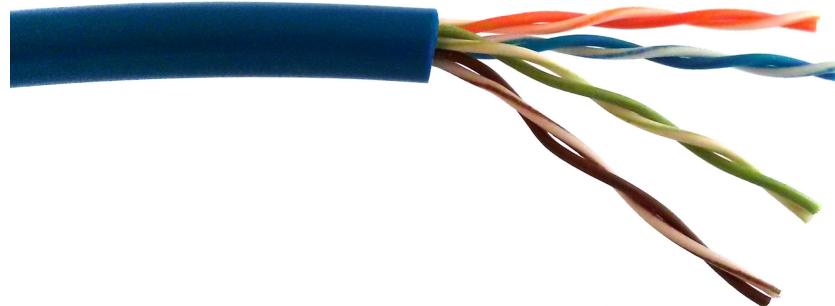


Abbildung 2.8: Unshielded Twisted Pair

Bei paarsymmetrischen Kabeln wird zwischen geschirmten Kabeln (**Shielded Twisted Pair, STP**) und ungeschirmten Kabeln (**Unshielded Twisted Pair, UTP**) unterschieden.

Geschirmte Kabel weisen eine bessere Störsicherheit auf, sind aber auch teurer, schwerer und steifer. Sie werden primär in Europa eingesetzt (vor allem Deutschland, Frankreich und der Schweiz). Der Schirm kann aus Drahtgeflecht, metallisch beschichteter Folie oder beidem bestehen (Abbildung 2.9). Folien schützen besser gegen hochfrequente Störungen; Drahtgeflechte leiten niederfrequente Einstreuungen besser ab.



Abbildung 2.9: Twisted Pair Kabel mit Geflecht und Folie geschirmt

Neben der Gesamtschirmung können (zusätzlich oder anstelle) die Aderpaare einzeln geschirmt sein, was zu einer grossen Anzahl verschiedener Kabeltypen führt. Die ISO/IEC 11801 standardisiert ein Bezeichnungsschema für die verschiedenen geschirmten Twisted-Pair-Kabel der Form:

xx/yTP worin TP für Twisted Pair steht.

xx steht für die Gesamtschirmung:

- U** = ungeschirmt
- F** = Folienschirm
- S** = Geflechtschirm
- SF** = Schirm aus Geflecht und Folie

y steht für die Aderpaarschirmung:

- U** = ungeschirmt
- F** = Folienschirm
- S** = Geflechtschirm

Normgerecht heißen die Kabel von Abbildung 2.8 U/UTP und das von Abbildung 2.9 also SF/UTP.

2.4.1 Funktionsprinzip der paarsymmetrischen Kabel

Im Gegensatz zu Glasfasern und Koaxialkabel sind **paarsymmetrische Kabel anfällig auf elektromagnetische Störungen**. Diese können von parallel geführten Leitungen oder Motoren etc. stammen. Störungen von Datenleitungen werden als **Übersprechen** oder **Nebensprechen** (Crosstalk) bezeichnet. Die Einkopplung der Störung kann induktiv, kapazitiv oder galvanisch (Isolationsschaden) erfolgen.

Zwei parallel geführte Leitungen verhalten sich wie Sender und Empfänger (Abbildung 2.10).

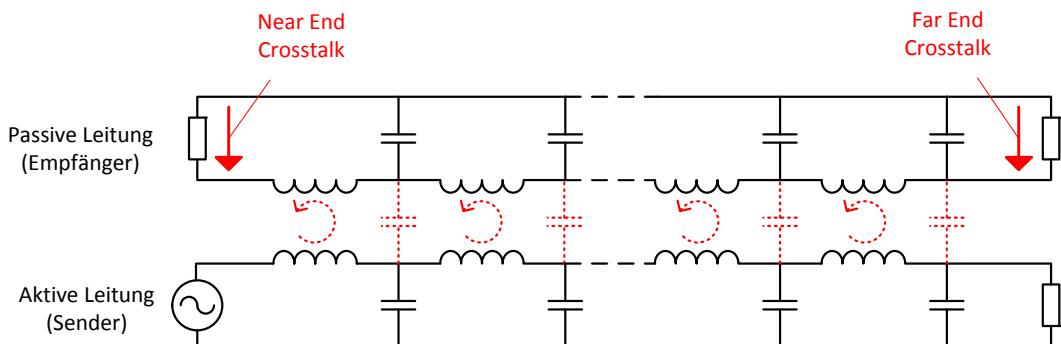


Abbildung 2.10: Prinzip der kapazitiven Kopplung

Kapazitiv eingekoppelte Störungen werden minimiert, indem auf zwei symmetrischen Leitungen ein komplementäres Signal gesendet wird und ein Differenz-Empfänger eingesetzt wird (Abbildung 2.11). Das komplementäre Nutzsignal wird durch den Empfänger addiert. Hingegen wird ein auf beiden Leitungen gleichphasig eingestreutes Störsignal durch die Differenzbildung weitgehend eliminiert.

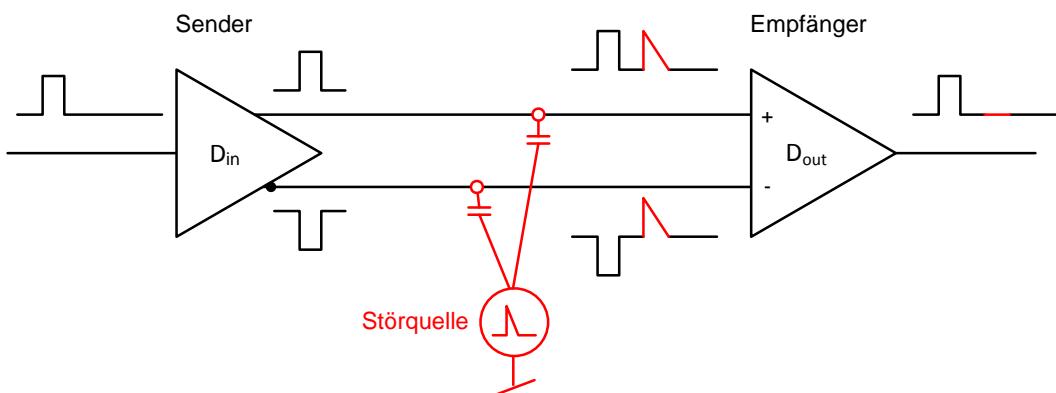


Abbildung 2.11: Funktionsprinzip der Differenzverstärker

Durch die Übertragung eines Komplementärsignals erreicht man zusätzlich, dass die elektromagnetischen Abstrahlungen deutlich reduziert werden, zumal sich diese im Idealfall selber aufheben.

Bei STP-Kabeln bietet der elektrisch leitende Schirm zusätzlichen Schutz, indem er die kapazitiv eingestreuten Signale zu einem grossen Teil ableitet (Abbildung 2.12). Ein Schirm wirkt nur, wenn er gut geerdet ist.

Da der Schirm im Allgemeinen auf beiden Seiten geerdet wird, können Potenzialdifferenzen der Erdung dazu führen, dass hohe Ausgleichsströme über den Schirm fliessen und so wiederum Störungen verursachen (Abbildung 2.13). In modernen, korrekt ausgeführten Hausinstallationen ist dies normalerweise kein Problem; mögliche Potenzialdifferenzen der Erdung zwischen Gebäuden sind jedoch mit ein Grund, warum für deren Vernetzung üblicherweise Glasfasern eingesetzt werden.

Weder Schirme noch Differenzverstärker sind in der Lage, induktiv eingekoppelte Störungen zu

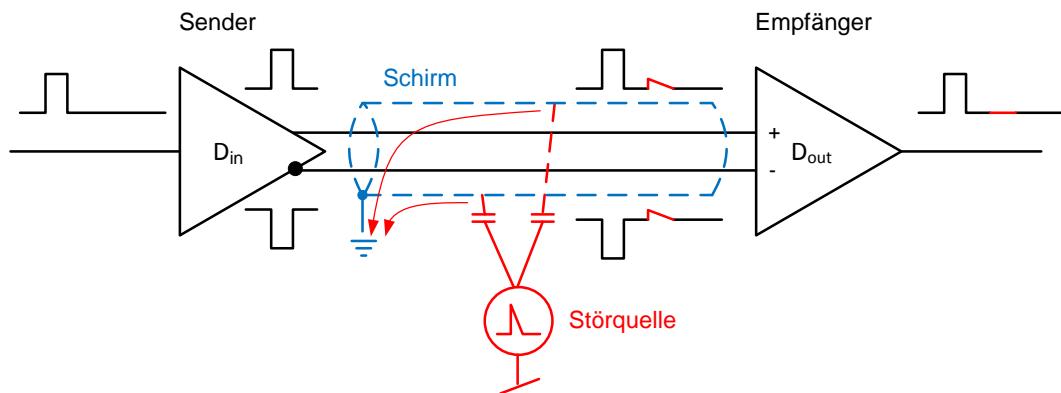


Abbildung 2.12: Funktionsprinzip des Schirms bei STP-Kabeln

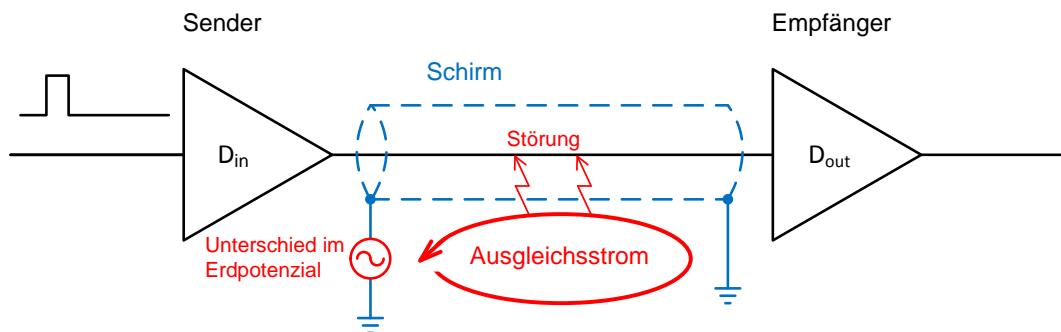


Abbildung 2.13: Unerwünschte Ausgleichsströme bei Potenzialdifferenzen der Erdung

beseitigen. Diese werden von magnetischen Wechselfeldern in der Schleife zwischen Sender und Empfänger induziert und erzeugen auf den beiden Leitungen ein komplementäres Signal (Abbildung 2.14). Schirme, bestehend aus nicht magnetischen Materialien (typischerweise Kupfer oder Aluminium), werden von Magnetfeldern durchdrungen.

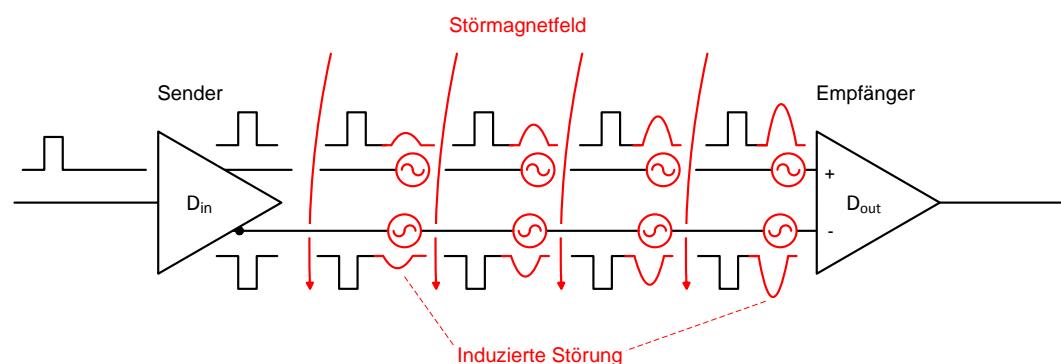


Abbildung 2.14: Induktiv eingekoppelten Störungen

Durch eine Verdrillung der Kabel erreicht man, dass kleinere Schleifen entstehen (Abbildung 2.15 oben). In zwei benachbarten Schleifen werden die Störspannungen nun gegenpolig induziert und heben sich so weitgehend auf. Abbildung 2.15 (unten) zeigt die komplementären Störspannungen auf den gedanklich abgewickelten Leitungen.

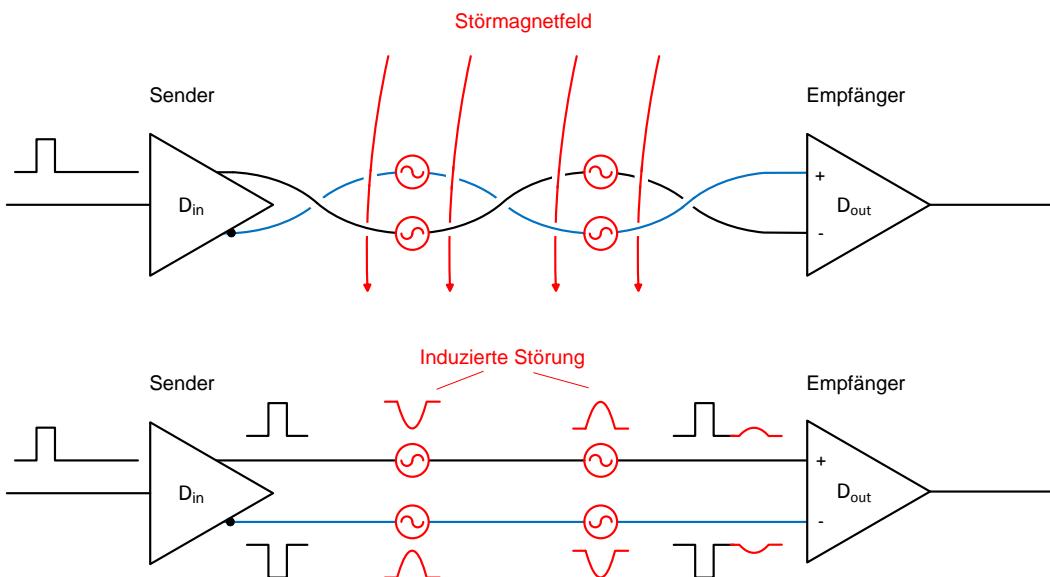


Abbildung 2.15: Funktionsprinzip von paarsymmetrischen Kabeln

2.4.2 Kabelkategorien

Die Kabel werden entsprechend ihrer spezifizierten / nutzbaren Bandbreite in Kategorien eingeteilt. Die einzelnen Kategorien spezifizieren bestimmte Übertragungseigenschaften der TP-Kabel wie die Bandbreite, die Dämpfung, das Nebensprechen und die Impedanz und viele weitere, auf die hier nicht eingegangen wird.

Ohne Anspruch auf Vollständigkeit sind folgende Kategorien standardisiert:

Kategorie 1/2/3/4: Geeignet für Telefon- und Modem-Leitungen oder langsame LAN (z.B. 10Base-T Ethernet) jedoch nicht für moderne LAN-Systeme, da die Bandbreite auf 0.4/4/16/20MHz beschränkt ist.

Kategorie 5: Weit verbreitete und weltweit akzeptierte Norm des amerikanischen EIA/ TIA-Gremiums (Electronic Industries Association / Telecommunication Industry Association), die von internationalen, europäischen und auch von nationalen Gremien übernommen wurde. Die Bandbreite von 100 MHz erlaubt Bitraten bis zu 1000 Mbit/s über eine Entfernung von bis zu 100 m .

Kategorie 6: Der Frequenzbereich dieser Kabel ist bis zu 250 MHz spezifiziert, wodurch sich diese Kabel für Gigabit-Ethernet bestens eignen.

Kategorie 7: Diese Bezeichnung wird für Kabelsysteme verwendet, die für einen Frequenzbereich bis 600 MHz vorgesehen sind und sich damit auch für 10 Gigabit-Ethernet eignet. Die Technik ist aufwändiger als bei allen anderen Kategorien, da die vier Aderpaare sowohl einzeln als auch gemeinsam geschirmt sind (S/FTP Kabel) und auch spezielle Stecker nötig sind.

Für Gebäudeverkabelungen werden tendenziell die höchstwertigen Kabel verwendet, da der Preisunterschied nicht gross ist und primär das Verlegen der Kabel hohe Anforderungen darstellt. Die LAN-Kabel müssen vor Beschädigungen und vor dem Eindringen von Feuchtigkeit geschützt sein. Massnahmen dazu sind (unter anderen):

- Enden mit Abschlusskappen versehen
- Einhaltung der Lagerungs- und Verlegetemperaturen
- Einhalten von zulässigen Biegeradien und Zugkräften
- Kabel nicht verdrehen oder quetschen
- Einsatz geeigneter Werkzeuge

2.4.3 Stecker

Für Netze mit hohen Datenraten sind nicht nur die Kabel sondern auch die Steckverbindungen (Buchsen und Stecker) kritische Punkte. Die **achtadrigen paarsymmetrischen Kabel** werden typischerweise zusammen mit RJ45-Stecker verwendet. Die Nummerierung der Anschlüsse, sowie Anordnung der Aderpaare ist in der Abbildung 2.16 ersichtlich.

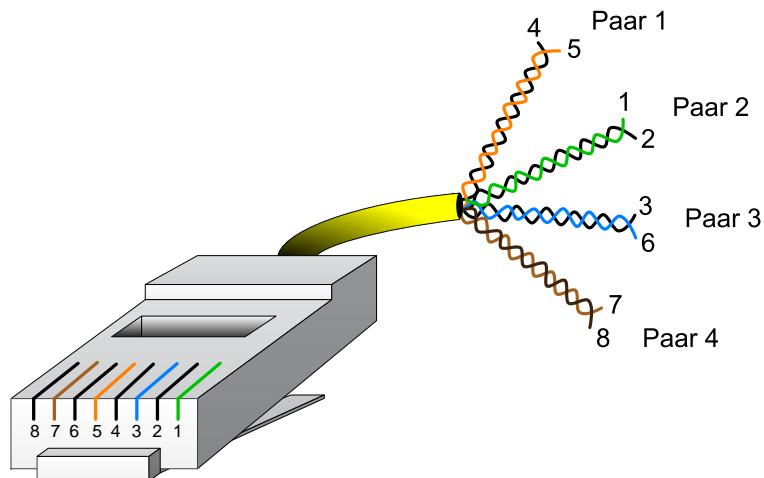


Abbildung 2.16: Pin-Layout des RJ45-Steckers und Bezeichnung der Aderpaare

2.5 Lichtwellenleiter

Lichtwellenleiter (Optical Fiber) bestehen aus Glas oder (selten) aus Kunststoff.

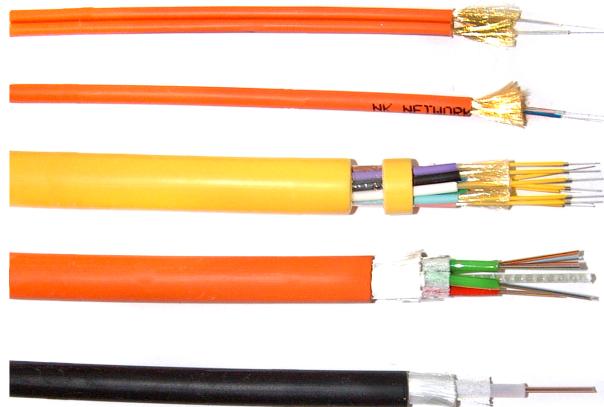


Abbildung 2.17: Lichtwellenleiter und Lichtwellenleiterbündel in verschiedenen Ausführungen

Lichtwellenleiter bieten gegenüber metallischen Leitern folgende Vorteile:

- Vollständige Unempfindlichkeit gegen elektromagnetische Störungen
- Kleine Signaldämpfung und somit grosse Distanzen
- Grosse Bandbreiten und somit grosse Übertragungsraten

Nachteilig ist die teure Umwandlung von elektrischen in optische Signale (Laser oder Dioden) und die aufwendigen Steckverbindungen.

2.5.1 Grundprinzip der Glasfaser

Die Glasfaser besteht aus einer **dünnen, zylindrischen Faser** aus hochreinem Quarzglas (SiO_2), dem sogenannten Kernglas. Um das Kernglas liegt das Mantelglas, das ebenfalls aus Quarzglas besteht. Geschützt wird die Glasfaser durch eine Lackschicht, sowie zusätzliche Ummantelungen und Füllmaterialien (Abbildung 2.17).

Die Führung des Lichtstrahls beruht auf **Totalreflexion** an der Grenze zwischen Kern- und Mantelglas. Die einzelnen Linien in Abbildung 2.18 stellen verschiedene Eigenwellen (Ausbreitungswege der Lichtstrahlen) dar. Diese werden als Moden bezeichnet.

Der Brechungsindex des Kerns n_1 muss leicht höher sein als der Brechungsindex n_2 des Mantels; beispielsweise 1.50 für das Kernglas und 1.48 für das Mantelglas. Die Totalreflexion lässt sich im Wasser gut beobachten, indem man in abgetauchtem Zustand die (ruhige) Wasseroberfläche betrachtet. Abbildung 2.19 zeigt den Effekt der Brechung und die Totalreflexion.

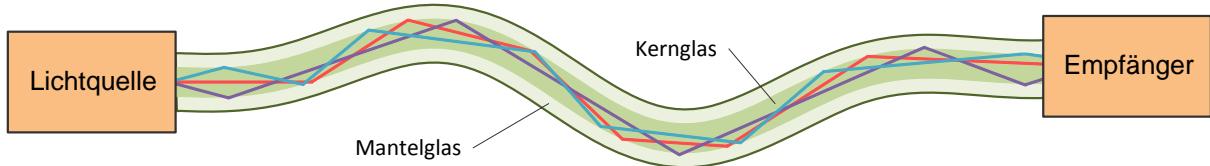


Abbildung 2.18: Totalreflexion im Faserverlauf

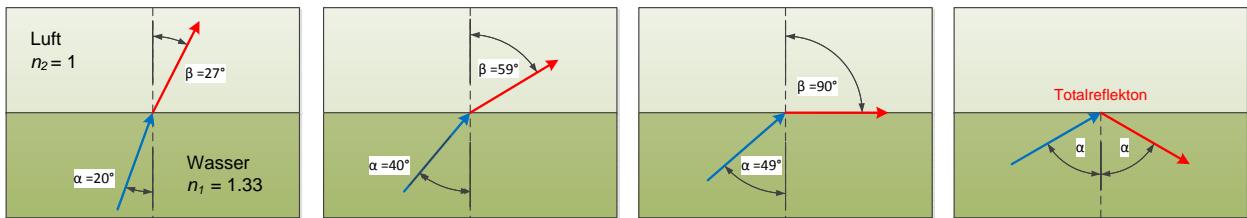


Abbildung 2.19: Brechung und Totalreflexion

Der Zusammenhang zwischen den Ein-/Ausfallwinkeln und Brechungsindizes ist:

$$n_1 * \sin(\alpha) = n_2 * \sin(\beta) \quad (2.3)$$

Vergrößert man den Winkel α bis $\beta \geq 90^\circ$ ist, so tritt eine Totalreflexion ein.

$$\alpha = \arcsin\left(\sin(\beta) * \frac{n_2}{n_1}\right) \quad \text{bei} \quad \beta = 90^\circ \quad \text{gilt} \quad \alpha_t = \arcsin\left(\frac{n_2}{n_1}\right) \quad (2.4)$$

Für die Brechungsindizes von Luft 1 und Wasser 1.33 ergibt sich für die Totalreflexion ein Grenzwinkel α_t von:

$$\alpha_t = \arcsin\left(\frac{n_{Luft}}{n_{Wasser}}\right) = \arcsin\left(\frac{1}{1.33}\right) = 49^\circ$$

Es ist dem Lesenden überlassen den Grenzwinkel α_t für einen Lichtwellenleiter zu bestimmen.

2.5.2 Einteilung der Lichtwellenleiter

Lichtwellenleiter gibt es in verschiedenen Ausprägungen. Je nach Einsatzgebiet werden Monomodefaser, Multimode-Gradientenfasern oder Multimode Stufenfasern eingesetzt. Zwei Faktoren (Dämpfung und Dispersion) begrenzen das Distanz-Bandbreiten-Produkt (Abbildung 2.20).

Die Dämpfung entsteht durch die Absorption im Lichtwellenleiter. Wie Abbildung 2.21 zeigt, variiert sie stark mit der Wellenlänge (Farbe). Bei herkömmlichen Lichtwellenleitern werden drei „Fenster“ (850 nm, 1310 nm und 1550 nm) genutzt. Dazwischen liegende Wellenlängen werden stark von den OH-Ionen absorbiert (hohe Dämpfung).

Mittlerweile sind optimierte „Low Water Peak“-Lichtwellenleiter verfügbar, die bei 1220 nm und 1400 nm einen flacheren Dämpfungsverlauf zeigen.

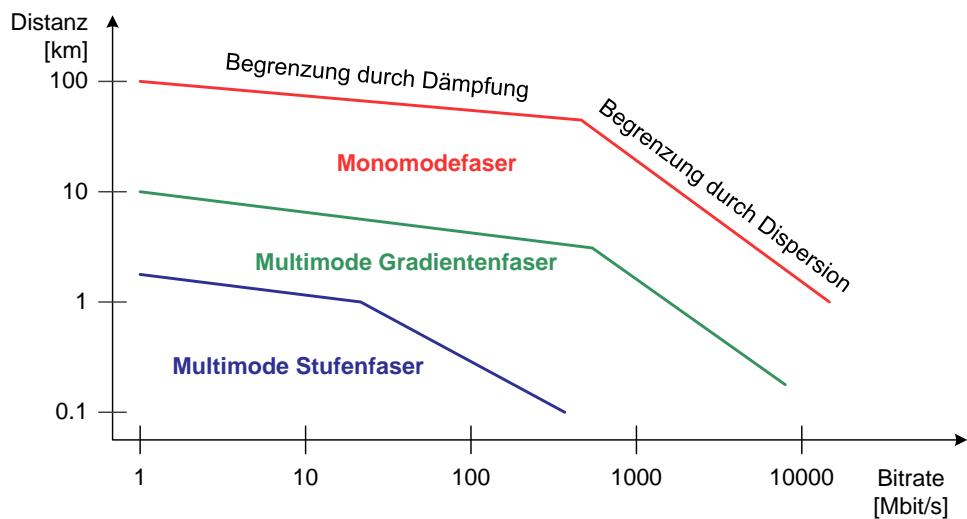


Abbildung 2.20: Einsatzbereiche der Lichtwellenleiter

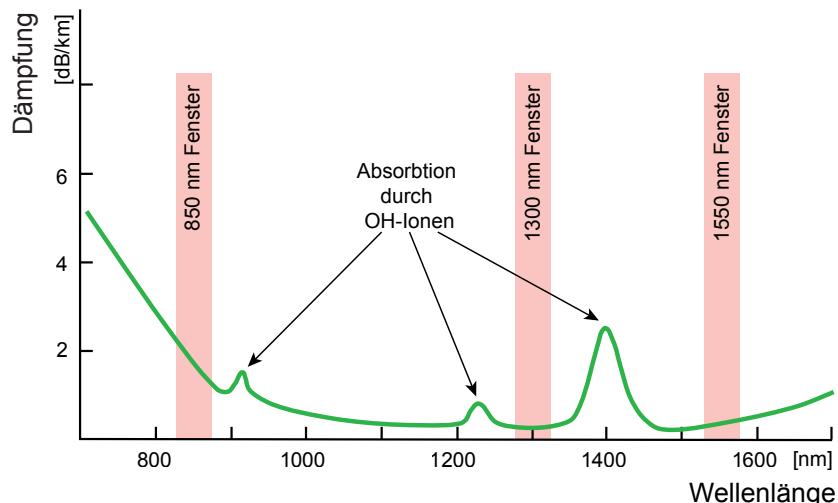


Abbildung 2.21: Dämpfungsverlauf in herkömmlichen Lichtwellenleitern (Quelle www.ciscopress.com)

Mit **Dispersion** wird die **Verzerrung eines Signals** bezeichnet, die letztlich dazu führt, dass diese nicht mehr erkannt werden können. Es gibt verschiedene Arten von Dispersion mit unterschiedlichen Ursachen, deren Erläuterung diesen Rahmen sprengt. Die offensichtlichste ist jedoch die Modendispersion. Sie ist die Folge der in Abbildung 2.18 gezeigten Ausbreitungsmoden mit unterschiedlichen Laufzeiten. Diese Zeitabweichungen werden als Delay Skew bezeichnet. Die Modendispersion führt zu Interferenzen und (wie Abbildung 2.22 zeigt) zu einer Verbreiterung der ausgesandten Pulse bis zu deren Unkenntlichkeit. Es ist einsichtig, dass Distanz und Pulsdichte sich gegenseitig beschränken. In bestimmten Grenzen ist das Produkt aus Bitrate (Pulsdichte) und Übertragungsdistanz konstant.

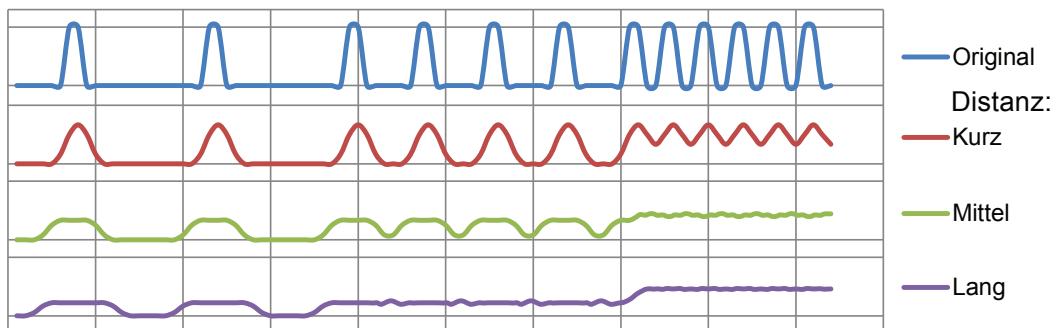


Abbildung 2.22: Modendispersion bei unterschiedlichen Distanzen und Pulsabständen

2.5.3 Multimode Glasfaser

Bei Multimode-Glasfasern wird zwischen Stufen- und Gradientenfasern unterschieden. Im LAN-Bereich (für Distanzen $\leq 2\text{ km}$) werden typischerweise Multimode Stufenfasern (Multimode Fiber, MMF) verwendet. In Europa sind Fasern der Dicke 50/125 üblich; in Nordamerika werden meistens 62,5/125 Fasern verwendet. Diese Zahlen bezeichnen die Durchmesser des Kerns (50) und des Mantelglases (125) in μm . Im Allgemeinen werden Multimode Glasfaser zusammen mit LED oder Vertical Cavity Surface Emitting Lasers (VCSELs) bei einer Wellenlänge von 850 nm resp. 1310 nm betrieben.

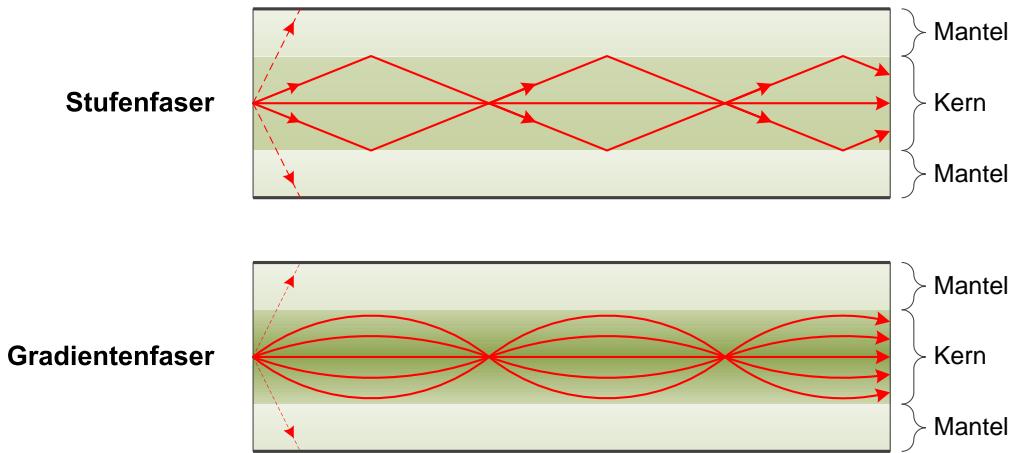


Abbildung 2.23: Lichtausbreitung in Multimode Glasfasern

Stufenfasern haben ein Delay Skew von ca. 50 ns/km und zeigen damit ausgeprägte Modendispersion. Bei der Gradientenfaser werden die Moden mit dem kürzeren, geraden Wegen gebremst, so dass sie gleichzeitig mit den Moden eintreffen, die einen weiteren, wellenförmigen Weg durchlaufen haben (Abbildung 2.23). Dies wird erreicht, indem der Brechungsindex des Kernglases zur Fasermitte kontinuierlich zunimmt (höherer Brechungsindex \rightarrow kleinere Ausbreitungsgeschwindigkeit). Dank diesem Aufbau beträgt der Delay Skew von Gradientenfaser nur ca. 0,5 ns/km.

Die Klassifizierung der Multimodefasern erfolgt mit den Faserkategorien OM1 – OM4 (Abbildung 2.24). Die Faserkategorien OM1 und OM2 sind für LED-basierte Anwendungen konzipiert; OM3 und OM4 dagegen sind für den Einsatz mit VCSELs optimiert.

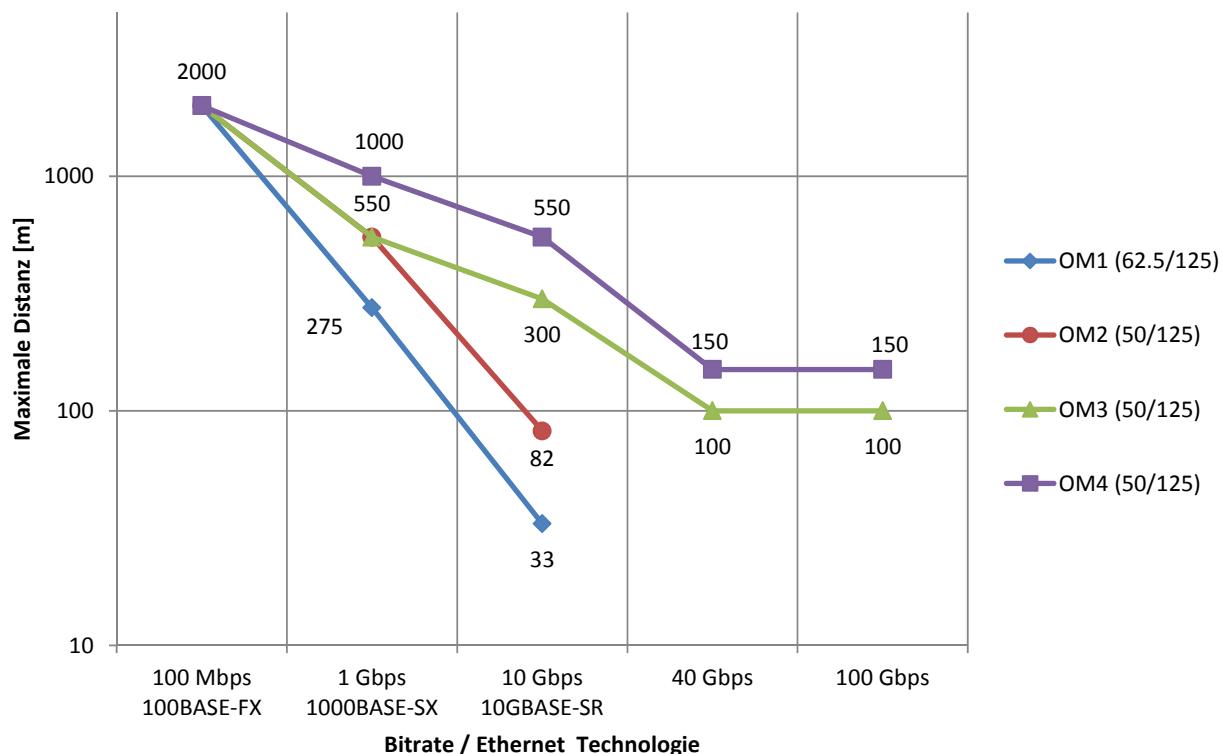


Abbildung 2.24: Einsatzbereiche der Multimodefasern OM1–OM4

2.5.4 Monomode Glasfaser

Monomode Glasfaser (Single Mode Fiber, SMF) haben eine wesentlich grösere Bandbreite und erlauben die Überbrückung grösserer Distanzen als Multimode Glasfasern.

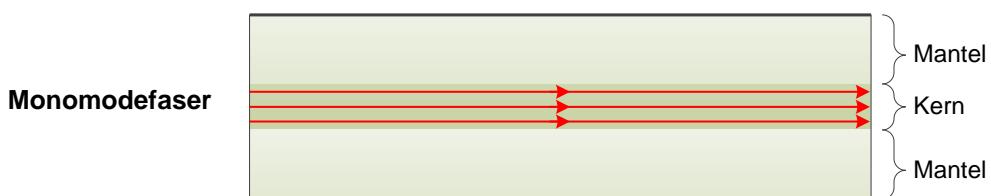


Abbildung 2.25: Lichtausbreitung in einer Monomode Glasfaser

Monomode Glasfaser haben einen sehr kleinen Kern; typischerweise $5\text{--}10 \mu\text{m}$. Da der Kerndurchmesser lediglich einige Vielfache der Wellenlänge des Lichts beträgt, werden praktisch alle Moden bis auf einen unterdrückt (Abbildung 2.25). OS1 sind OS2 standardisierte Monomode Glasfasern mit der Dicke $9/125 \mu\text{m}$. Sie werden mit Laser-Dioden mit Wellenlänge 1310 nm oder 1550 nm betrieben und erlauben Ethernet-Übertragungen mit 10 Gbit/s über Entfernungen bis zu 80 km oder 100 Gbit/s bis 40 km.

2.6 Strukturierte Gebäudeverkabelung

Strukturierte Verkabelungen bilden die Grundlage für eine anwendungsunabhängige und wirtschaftliche Netzwerkinfrastuktur. Neue Verkabelungsinfrastrukturen müssen **Reserven** enthalten, die auch Kommunikationsanforderungen der nächsten 10 bis 15 Jahre berücksichtigen. Darauf basierend wurden Verkabelungsstandards entwickelt. Die in Abbildung 2.26 gezeigte Verkabelungsstruktur ist für eine geographische Ausdehnung von 3000 m, eine Bürofläche von 1 Mio. m² und für 50 bis 50'000 Anwender optimiert.

SV	Standortverteiler:	Primärkabel
GV	Gebäudeverteiler:	Sekundärkabel
EV	Etagenverteiler:	Tertiärkabel
KV	Kabelverteiler (optional):	Tertiärkabel
TA	Telekommunikations Anschlussdose	

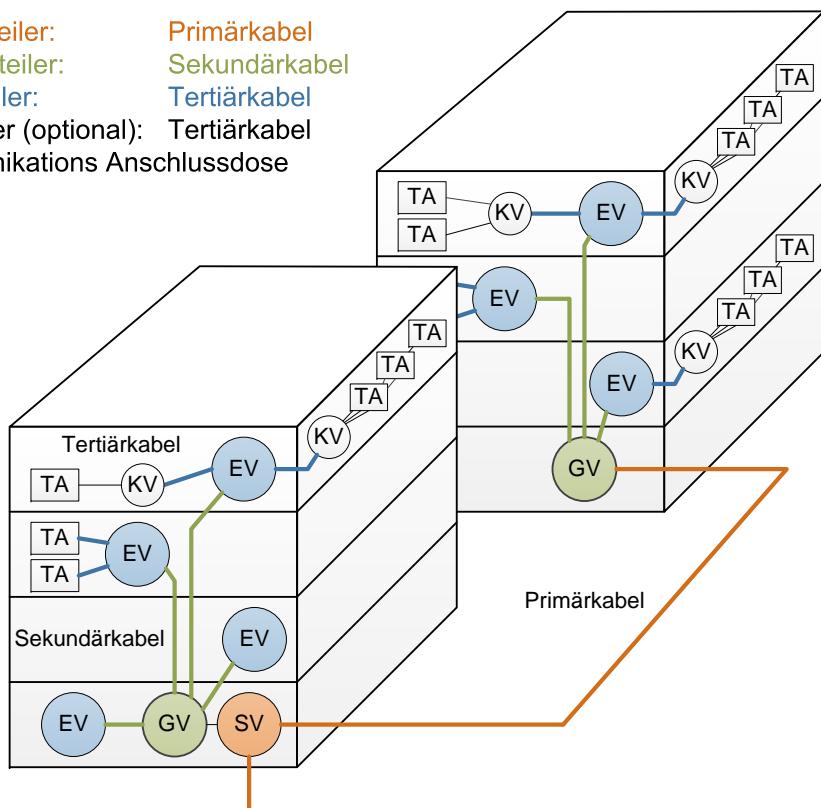


Abbildung 2.26: Verkabelungsstruktur nach ISO/IEC 11801

Die Strukturierung erfolgt innerhalb von Hierarchie-Ebenen, die in horizontale und vertikale Verkabelungsbereiche gegliedert sind: die Geländeverkabelung, die Gebäudeverkabelung mit dem Steigleitungsbereich und die Etagenverkabelung.

Im **Primärbereich** geht der Standard von 62.5/125 µm Glasfasern aus mit einer maximalen Länge von 1500m.

Im **Sekundärbereich**, also in der Gebäudeverkabelung, kommt ebenfalls 62.5/125 µm Glasfaser zum Einsatz. Die Längenrestriktion liegt bei 500 m.

Im **Tertiärbereich**, der Etagenverkabelung, die vom Etagenverteiler bis zur Anschlussdose reicht, werden im Standard TP-Kabel (UTP, STP, 62,5/125 µm Glasfaser) mit einer Länge von 90m festgelegt, zuzüglich 10m Anschlusskabel.

3

Physical Layer (Bitübertragungsschicht)

Der Physical Layer (Bitübertragungsschicht) sorgt für die ungesicherte Übertragung eines Bitstroms zwischen zwei Teilnehmern über das physikalische Medium. Es geht also um Fragen der Anpassung von Endeinrichtungen an die Übertragungseinrichtung und das Übertragungsmedium wie:

- elektrische Eigenschaften (Pegel, Zeiten usw.)
- Codierung (Return to Zero, non return to Zero, AMI, Manchester, FSK usw.)
- mechanische Eigenschaften (Stecker, Pinbelegung usw.)

Das eigentliche Übertragungsmedium (z.B. Kabel) liegt unterhalb des Physical Layers und damit ausserhalb des OSI Referenzmodells. Die physikalische Schicht ist wie jede OSI-Schicht austauschbar, ohne dass die anderen Schichten betroffen sind (im Rahmen der physikalischen Grenzen des Mediums). So kann z.B. eine Kupferleitung, ohne Rückwirkung auf den Data Link Layer, durch eine Glasfaserleitung ersetzt werden, indem der Physical Layer ausgetauscht wird.

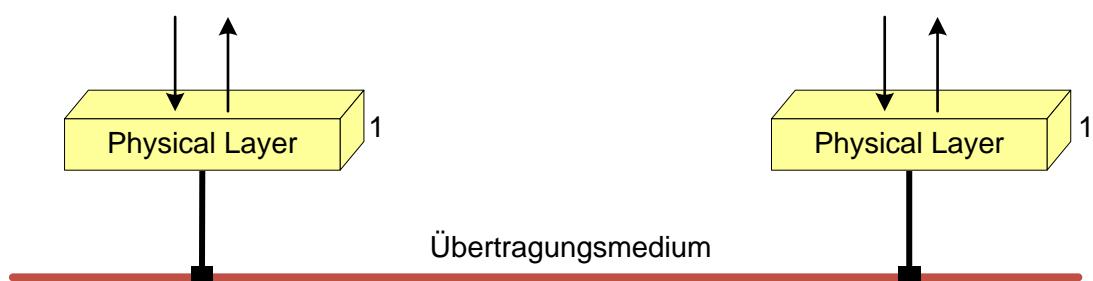


Abbildung 3.1: Physical Layer und Übertragungsmedium

Die Realisierung des Physical Layers wird durch folgende Faktoren wesentlich beeinflusst:

- Übertragungsmedium (siehe Kapitel 2)
- Kopplung der Kommunikationspartner
- Übertragungsverfahren

Die letzten zwei Punkte werden im Folgenden behandelt, wobei auf den Physical Layer nur exemplarisch, soweit für das Grundverständnis notwendig eingegangen wird. Im Kapitel 5 werden einzelne Aspekte anhand der Ethernet-Technologien nochmals aufgegriffen.

3.1 Kopplung der Kommunikationspartner

Abhängig vom Übertragungsmedium gibt es verschiedene Arten, wie Kommunikationspartner gekoppelt werden können. Man unterscheidet die Art der Kommunikation und die Art der Verbindung. Abbildung 3.2 zeigt die möglichen Kombinationen.

Die Arten der Kommunikation sind:

- **Simplex:** es ist nur ein Kanal in eine Richtung vorhanden (Radio, TV).
- **Halbduplex:** es ist nur ein Kanal vorhanden, der abwechselungsweise für die eine oder andere Richtung benutzt wird (einfaches Funkgerät).
- **Vollduplex:** es ist für jede Richtung je ein Kanal vorhanden (Telefon).

Die Trennung der Kommunikationsarten ist nicht scharf; oft hängt sie vom betrachteten Layer ab. Beispielsweise wird beim Zeitmultiplexverfahren technisch eine Halbduplexverbindung realisiert, die aber für den Anwender wie eine Vollduplexverbindung wirkt (weiteres Beispiel siehe Abschnitt 3.3.3).

Die Arten der Verbindung sind:

- Punkt-Punkt: direkte Verbindung zweier Kommunikationspartner; z.B. zwischen PC und lokalem Drucker
- Shared Medium: mehrere Partner verkehren über das gleiche Medium; z.B. Bus oder klassische Local Area Network (LAN)

3.2 Übertragungsverfahren

Daten können seriell oder parallel übertragen werden. Bei der **parallelen Übertragung** werden mehrere Bits gleichzeitig über mehrere (parallele) Leitungen übertragen. Parallel Übertragung kommt nur auf kurzen Distanzen zur Anwendung. Bei der seriellen Übertragung werden die Bits zeitlich nacheinander über eine Leitung übertragen.

Des Weiteren unterscheidet man zwischen **serieller asynchroner** und **serieller synchroner Übertragung**, die in Folgenden genauer betrachtet werden.

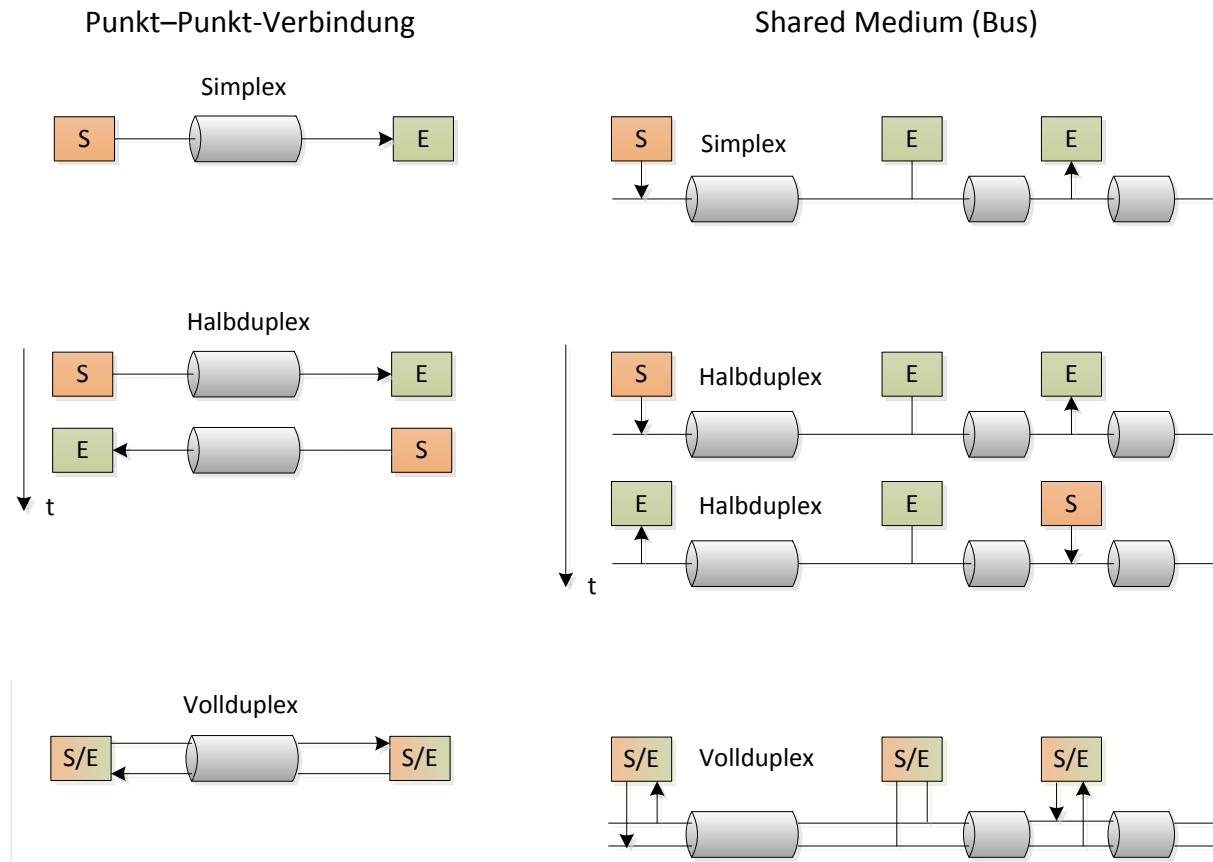


Abbildung 3.2: Kopplung der Kommunikationspartner

3.2.1 Serielle asynchrone Übertragung

Eine Übertragung wird als asynchron bezeichnet, wenn kein Takt für die Bitsynchronisation übertragen wird. **Sender und Empfänger besitzen eigene, unabhängige Taktquellen mit annähernd gleicher Frequenz**. Der Empfänger justiert seinen Übertragungsrahmen bei jedem übertragenen Zeichen von Neuem. Dazwischen arbeiten Sender und Empfänger asynchron mit ihren eigenen Taktgebern.

Zwischen Sender und Empfänger müssen die Bitrate, die Anzahl der n Datenbits (typisch ein Byte, $n = 8$) und Anzahl **Stopbits** (heute typisch 1, früher 2 oder 1,5) abgemacht sein. Für jedes Zeichen sendet der Sender ein **Startbit**, die vereinbarte Anzahl Datenbits und zum Abschluss die Stopbits. Von den Datenbits wird das niedrigwertigste (LSB = Least Significant Bit) zuerst und das höchswertige (MSB = Most Significant Bit) zuletzt übertragen.

Wie Abbildung 3.3 zeigt, ist der Leitungspegel vor der Übertragung 1. Der Empfänger tastet das Signal mit einem Vielfachen der Bitrate ab, bis er den 0-Pegel des Startbits erkennt (fallende Flanke). Jeweils in der Mitte der Bitzeit werden nacheinander die Datenbits eingelesen. Wegen der asynchronen Arbeitsweise dürfen während der Übertragung eines Rahmens die Takte nicht mehr als eine halbe Bitzeit T abweichen. Abschliessend wird geprüft, ob das Stopbit vorhanden ist; andernfalls liegt ein Rahmenfehler vor.

Die asynchrone serielle Schnittstelle ist einfach in der Implementierung und der Handhabung.

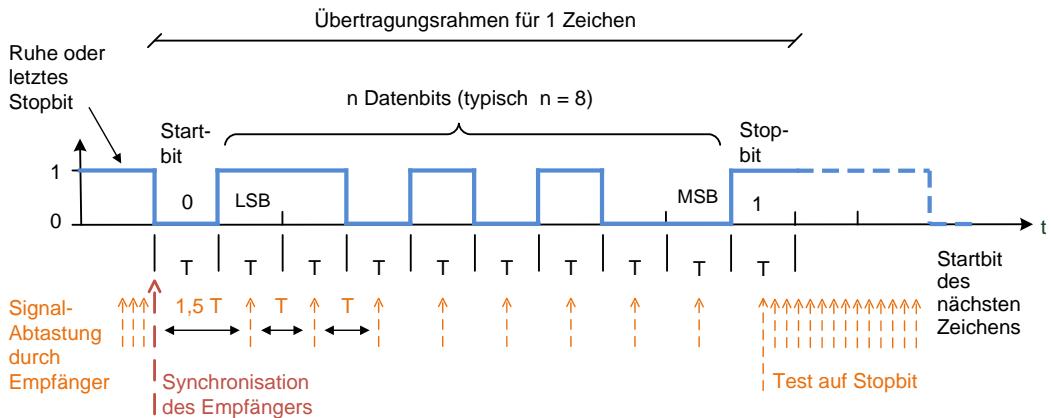


Abbildung 3.3: Prinzip der seriellen asynchronen Übertragung

Nachteilig ist, dass durch die Start- und Stopbits zu Lasten der Bitrate gehen. Beispielsweise resultiert bei 8 Datenbit, einem Stopbit und einer Nettobitrate von 9600 bit/s eine Bruttobitrate von 7680 bit/s. Außerdem ist die Übertragung nicht sehr robust: Durch einen einzelnen Bitfehler kann die Übertragung eines ganzen Datenblocks bis zur nächsten Sendepause aus dem Tritt geraten.

3.2.2 serielle synchrone Übertragung

Bei der seriellen synchronen Übertragung arbeitet der Empfänger synchron zum Sender d.h. mit dessen Takt. Diese bitsynchrone Arbeitsweise von Sender und Empfänger erlaubt hohe Datenraten und ist effizient, da keine Start- und Stopbits benötigt werden. Dafür muss nebst dem Datensignal auch dessen Takt übertragen werden, der angibt, wann das Signal in einem gültigen Zustand ist (Abbildung 3.4).

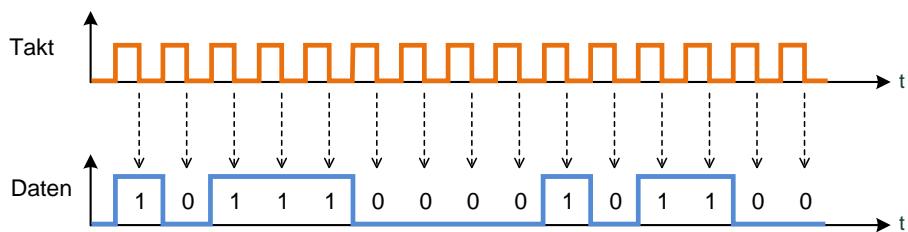


Abbildung 3.4: Serielle synchrone Übertragung mit Takt zur Bitsynchronisation

Mit serieller synchroner Übertragung resultiert auf den Physical Layer ein kontinuierlicher Bitstrom. Es ist Aufgabe des Data Link Layers die Grenzen der einzelnen Bytes sowie den Anfang und das Ende einer Meldung zu ermitteln.

3.3 Leitungscodes

Für die Übertragung des Takts wird entweder eine separate Leitung verwendet (Abbildung 3.5) oder es werden Codierverfahren eingesetzt, die es dem Empfänger erlauben, den Takt aus dem Datensignal zu extrahieren. Unter Codierung versteht man hier die Umsetzung der zu übertragenden Daten auf eine oder mehrere physikalische Größen des Übertragungsmediums.

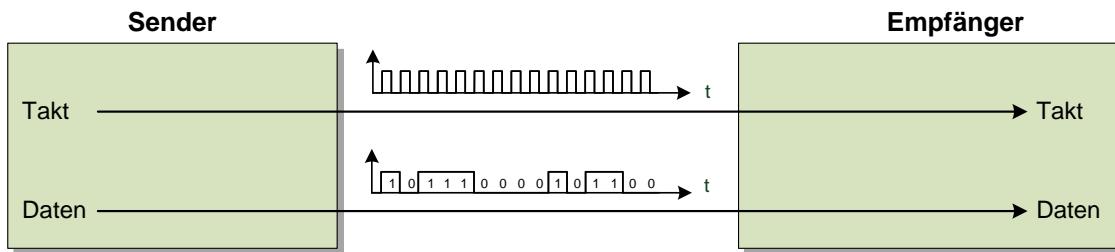


Abbildung 3.5: Serielle synchrone Übertragung mit Takteleitung

Wie Abbildung 3.6 zeigt, kombiniert eine **Leitungseinrichtung** senderseitig die Daten und den Sendetakt in einer für die Übertragung günstigen Form, dem **Leitungscode**. Auf der Empfangsseite extrahiert die Leitungseinrichtung die Daten aus dem Leitungscode und liefert das regenerierte Taktsignal und die Daten.

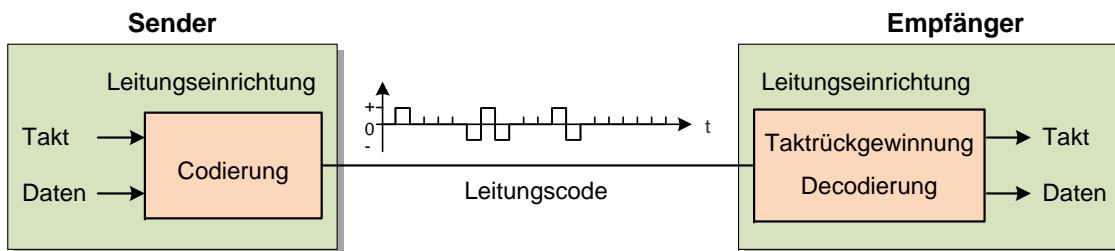


Abbildung 3.6: Serielle synchrone Übertragung mit Leitungseinrichtung

Für die Übertragung benutzte Leitungscodes sollen:

- die vom Übertragungsmedium zur Verfügung gestellte, also physikalisch vorhandene Bandbreite effizient nutzen.
- unabhängig von den übertragenen Daten dem Empfänger eine zuverlässige Taktrückgewinnung ermöglichen.
- möglichst gleichspannungsfrei sein.

Leitungscodes gibt es in sehr grosser Vielfalt. Praktisch alle denkbaren Varianten wurden beschrieben und mit einem Namen versehen. Sie werden hier nur exemplarisch behandelt, um zu zeigen, wie die obigen Forderungen erfüllt werden können. Im Kapitel 5 werden wir einige zusätzliche Leitungscodes betrachten, die bei Ethernet verwendet werden.

3.3.1 Gleichspannungsfreiheit

Die Gleichspannungsfreiheit dient unter anderem dazu, Sender und Empfänger galvanisch trennen zu können. Bei Ethernet wird z.B. eine Trennung von über 1500V verlangt (Abbildung 3.7), was mit günstigen Signaltransformatoren (Magnetics) erreicht werden kann. Diese können tiefe Frequenzen und Gleichspannungen nicht übertragen.

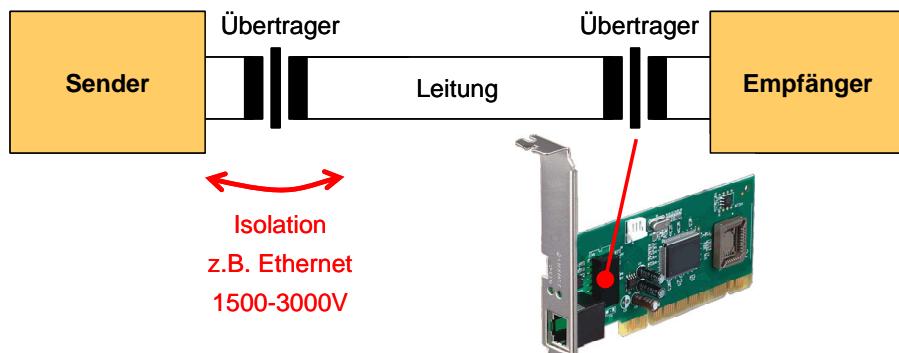


Abbildung 3.7: Übertrager zur galvanischen Trennung

Das binäre Signal in Abbildung 3.8 besitzt wegen der asymmetrischen Pegel 0 und U_+ einen mittleren Gleichspannungsanteil von ca. $\frac{1}{2}U_+$. Aber auch wenn für 0 und 1 erdsymmetrische Pegel U_- und U_+ verwendet würden, wäre die Gleichspannungsfreiheit nicht erfüllt, sofern die Gleichverteilung von 0 und 1 im Datenstrom nicht garantiert ist.

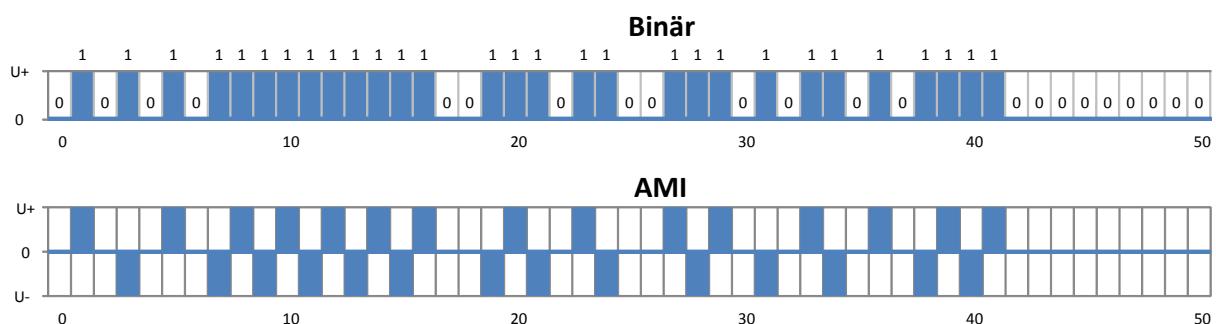


Abbildung 3.8: Beispiel zu Alternate Mark Inversion (AMI)

Alternate Mark Inversion (AMI) (Abbildung 3.8) ist ein Leitungscode, der unabhängig von den Daten ein gleichspannungsfreies Signal garantiert. Der logische Wert 0 wird mit 0 Volt, der logische Wert 1 alternierend mit einer positiven oder negativen Spannung (U_+ , U_-) dargestellt.

Auf der Übertragungsstrecke werden drei Zustände benötigt. In der Informationstechnik wird ein solches Signal als **ternäres Signal** bezeichnet. Gegenüber einem rein binären Signal sind die Anforderungen an das Übertragungsmedium höher.

In der Praxis wird oft ein modifizierter AMI-Code mit invertierte Zuordnung verwendet: Der logische Wert 1 wird mit 0 Volt, der logische Wert 0 alternierend mit einer positiven oder negativen Spannung (U_+ , U_-) dargestellt.

3.3.2 Taktrückgewinnung

Die Grundanforderung für eine Taktrückgewinnung besteht darin, dass auf der Übertragungsstrecke regelmässig Zustandsänderungen auftreten, die es dem Empfänger erlauben, den lokalen Takt nachzuregeln. Bei AMI-Code ist dies gegeben, solange regelmässig Einsen gesendet werden. Im allgemeinen Fall ist die Taktrückgewinnung mit dem AMI-Code nicht möglich, da Nullfolgen mit einem konstanten 0 Volt Signal übertragen werden.

Um bei langen Nullfolgen regelmässige Zustandsänderungen auf der Übertragungsstrecke zu erzwingen, ersetzt der Sender vier hintereinander folgende Nullen durch eine 000V-Sequenz (wertmässig 0001). Damit der Empfänger merkt, dass das V-Bit keine Eins ist, sondern nur ein Puls für die Taktrückgewinnung, wird das V-Bit mit dem gleichen Pegel wie das letzte Datenbit vor der Sequenz gesendet (Abbildung 3.9). Aufgrund dieser Regelverletzung erkennt der Empfänger die 000V-Sequenzen und ersetzt sie durch 4 Nullen.

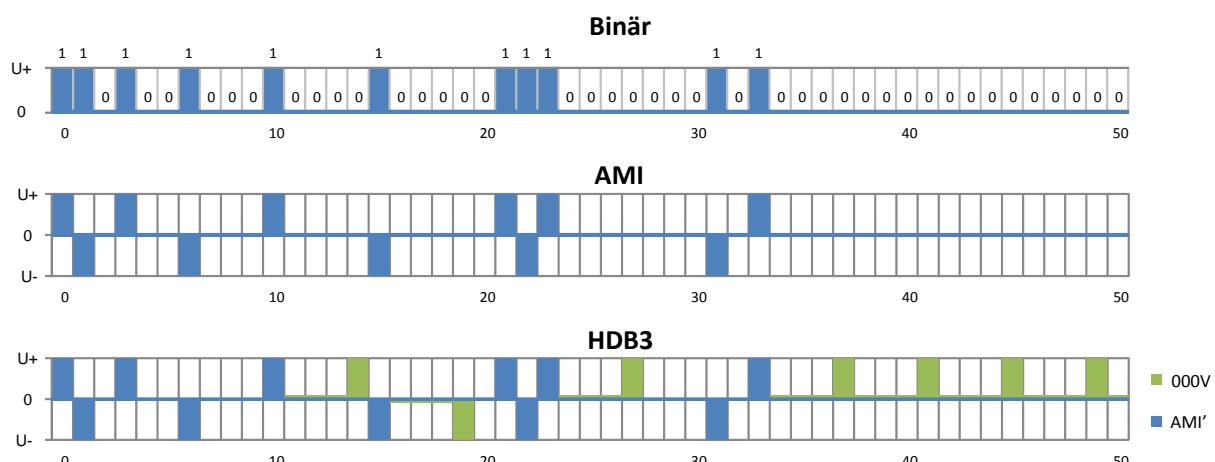


Abbildung 3.9: HDB3-Code mit 000V-Sequenzen verletzt die Gleichspannungsfreiheit

Bei langen Nullfolgen würden mit obiger Regel lauter Pulse derselben Polarität gesendet, was die Gleichspannungsfreiheit verletzen würde. Darum gilt die 000V-Regel nur, wenn die Anzahl Einsen seit der letzten Regelverletzung ungerade ist. Sonst, also wenn die Anzahl Einsen seit der letzten Regelverletzung gerade ist (oder gar keine Eins übertragen wurde), wird eine B00V-Sequenz (wertmässig 1001) angewendet. Das B-Bit wird bezüglich Polarität wie ein Datenbit behandelt; d.h. es hat den *umgekehrten* Pegel wie die benachbarten Einer-Pulse. Damit die B00V-Sequenz trotzdem erkannt werden kann, haben das B- und das V-Bit die gleiche Polarität (Regelverletzung in der Sequenz).

Damit hat man einen Leitungscode, der gleichspannungsfrei ist und regelmässig Pulse liefert, sodass eine Taktrückgewinnung im Empfänger möglich ist. Anstelle von Sequenzen mit 4 Bits können auch andere Längen (HDB N) verwendet werden. Mit „Bipolar 8-Zero Substitution“ (B8ZS) ist ein Code im Einsatz, der auf dem gleichen Prinzip beruht, aber Sequenzen von 8 Bits verwendet.

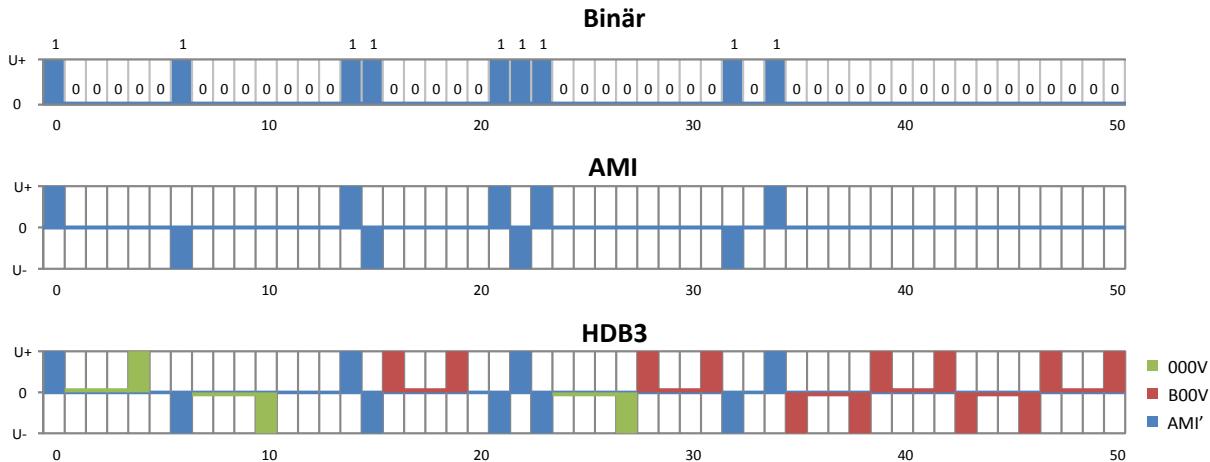


Abbildung 3.10: Gleichspannungsfreier High Density Bipolar Code

3.3.3 Nutzung der Bandbreite

Wie eingangs gefordert, sollen Leitungscodes, die vom Übertragungsmedium zur Verfügung gestellte, also physikalisch verfügbare Bandbreite effizient nutzen. Da stellt sich zunächst die Frage nach dem Zusammenhang zwischen der Bandbreite des Übertragungsmediums und der maximal möglichen Bitrate.

Die Übertragungsrate, die auf einem Übertragungsmedium erreicht werden kann, hängt von folgenden Einflussfaktoren ab:

- der Bandbreite der Übertragungsstrecke
- der Stärke des Signals im Vergleich zu den stets vorhandenen Störungen

a) Symbolrate (Nyquist Rate)

Der Sender setzt die zu übertragenden Daten in eine oder mehrere physikalische Größen des Übertragungsmediums um (Codierung). Beispielsweise übertragen Basisbandkommunikationssysteme Daten, indem sie diese direkt auf Signalspannungen der Leitung abbilden, wogegen bei Breitbandkommunikationssystemen die Daten einem Trägersignal aufgeprägt (moduliert) werden, wofür Amplitude, Frequenz/Phase oder Kombinationen davon als Träger der Information zur Auswahl stehen.

Der Empfänger kann anhand des Zustands des Übertragungsmediums die übertragene Information ermitteln (Dekodierung). Ein solcher Zustand des Übertragungskanals wird in der Informationstheorie als Symbol bezeichnet.

Informationsübertragung bedingt eine zeitliche Abfolge von Symbolen (Abbildung 3.11). Die Anzahl von Zustandsänderungen des Übertragungsmediums pro Zeit beziehungsweise die Anzahl der übertragenen Symbole wird als Symbolrate bezeichnet und in Baud angegeben.¹

¹Baud ist für die Symbolrate reserviert. Symbolrate und Bitrate sind nicht dasselbe; deren Maßeinheit ist bit/s.

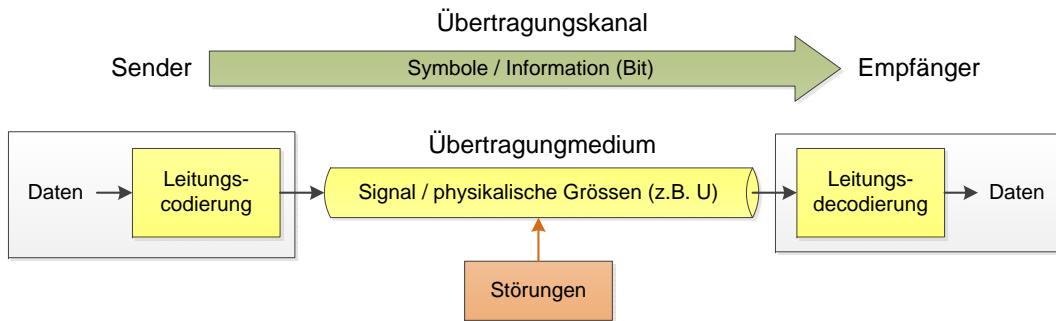


Abbildung 3.11: Übertragungskanal

Die maximale Symbolrate f_s (Baud) ist gemäss Nyquist² direkt abhängig von der nutzbaren Bandbreite B (Hz, Hertz) auf dem Übertragungskanal:

$$f_s \leq 2B \quad (3.1)$$

Die maximale Symbolrate, auch Nyquist Rate genannt, wird in Baud angegeben und ist also maximal das Doppelte der Bandbreite ($2B$) in Hertz.

Dieses Gesetz gilt für Basisband- und Breitbandkommunikationssysteme. Ein Basisbandsystem heisst so, weil es den Frequenzbereich von 0 bis B Hertz belegt, wogegen Breitbandsysteme im Spektrum positionierte Frequenzbänder der Breite B nutzen.

Wie Abbildung 3.12 am Beispiel eines Hausanschlusses zeigt, können auf derselben Übertragungsstrecke ein Basisbandkommunikationssystem (ISDN) und mehrere Breitbandkommunikationskanäle (ADSL Upstream/Downstream) koexistieren. Man sieht auch, dass bei diesen Anwendungen die theoretisch möglichen maximalen Symbolraten (276 kBaud resp. 1656 kBaud) nicht ausgeschöpft werden.

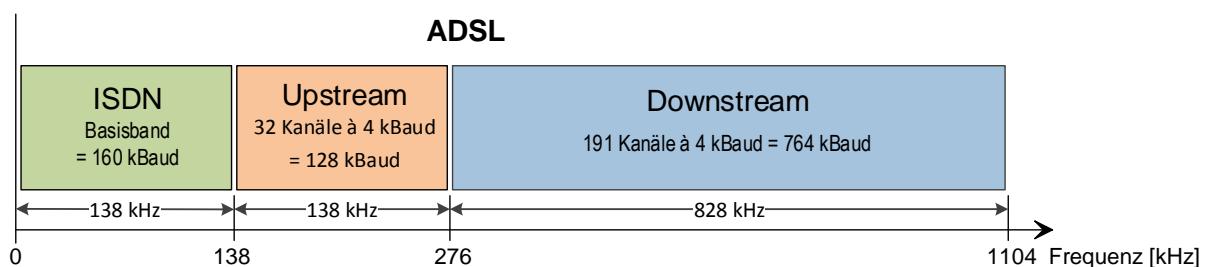


Abbildung 3.12: Basis- und zwei Breitbandkommunikationskanäle

Anmerkung: ISDN und ADSL sind auf den höheren Layern vollduplexfähige Kommunikationssysteme. Während ADSL dafür physikalisch 2 Frequenzbänder (Upstream / Downstream) benötigt, werden bei ISDN (U_{k0} -Schnittstelle) die Signale beider Richtungen im selben Frequenzband übertragen und durch eine sogenannte Gabelschaltung getrennt.

²Nyquist, 1928, "Certain topics in Telegraph Transmission Theory."

b) Maximal erreichbare Bitrate (Hartley's Gesetz)

Die maximal erreichbare Bitrate einer Übertragungsstrecke ist gemäss Hartley³ das Produkt der Symbolrate und dem Informationsgehalt der Symbole.

Informationsgehalt der Symbole hängt davon ab, wie fein der Zustand der Leitung aufgelöst werden kann, so dass der Empfänger die Abstufungen immer noch korrekt erkennen kann. Wie Abbildung 3.13 zeigt, ist die Anzahl M der erkennbaren Zustände:

$$M = 1 + \frac{A}{\Delta V} \quad (3.2)$$

Darin ist A die maximale Grösse des Signals und ΔV die Ungenauigkeit des Empfängers, wobei wir hier alle negativen Effekte berücksichtigen, wie Rauschen des Empfängers, Störungen auf dem Übertragungskanal etc.

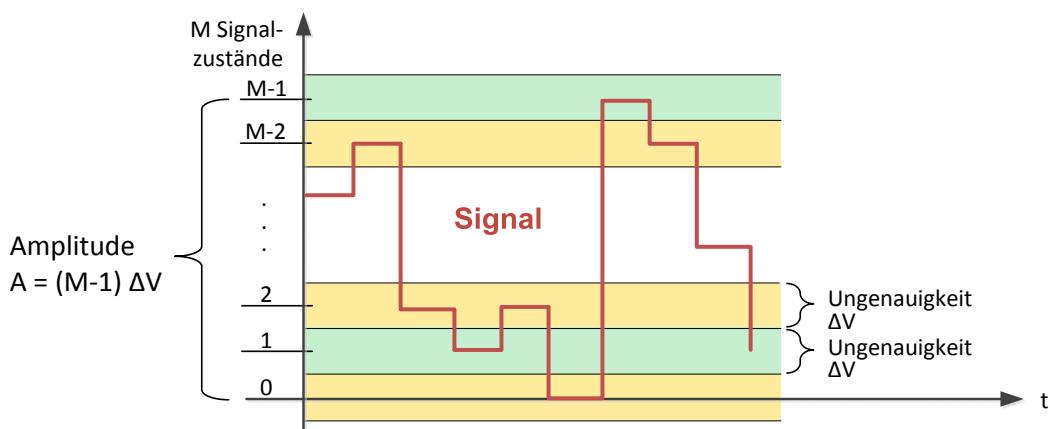


Abbildung 3.13: Anzahl der erkennbaren Zustände eines Signals

Der Informationsgehalt (Bit) eines Symbols I_S ist bekanntlich der Logarithmus zur Basis 2 der Anzahl seiner möglichen Zustände M also $I_S = ld(M)$

Damit gilt für die maximal erreichbare Bitrate R [bit/s]:

$$R \leq 2B * ld(M) = 2B * ld \left(1 + \frac{A}{\Delta V} \right) \quad (3.3)$$

Beispiel: Der AMI-Code ist bekanntlich ein dreiwertiger Code (U_+ , 0, U_-). Mit drei Werten könnten pro Schritt $ld(3) = 1.58$ Bit übertragen werden. Da aber pro Schritt nur 1 Bit übertragen wird, liegt die Effizienz des AMI-Codes also bei $\frac{1}{1.58} = 63\%$.

Offenbar ist die Gleichspannungsfreiheit nicht verlustlos zu erreichen, denn trotz der grossen Vielfalt an Leitungscodes ist keiner bekannt, der den Übertragungskanal optimal (unter voller Ausnutzung der Nyquist-Rate und aller Signalzustände für die Informationsübertragung) ausnutzen könnte. Jedoch lässt sich dank des dreiwertigen Codes die Taktsynchronisation ohne zusätzliche „Kosten“ erreichen (HDB3).

³Hartley, D. A. Bell (1962). Information Theory; and its Engineering Applications (3rd ed.). New York: Pitman.

c) Gesetz von Shannon-Hartley

Basierend auf den obigen Überlegungen von Nyquist und Hartley entwickelte Claude Shannon in den 1940er Jahren eine vollständige Theorie zu Information und deren Übertragung. Gemäss dem Gesetz von Shannon-Hartley gilt für die Kanalkapazität C eines mit weissem Rauschen gestörten Kanals der Bandbreite B und gegebenen Signal- (S) und Rauschleistungen (N):

$$C = B * ld \left(1 + \frac{S}{N} \right) \quad (3.4)$$

Gegenüber der Gleichung (3.3) fällt auf, dass der Faktor 2 fehlt. Der Grund liegt darin, dass S/N ein Verhältnis von Leistungen ist, die quadratisch mit den Spannungen zusammenhängen. Setzt man $M = \sqrt{1 + \frac{S}{N}}$ in Gleichung (3.3) ein, erhält man das Resultat von (3.4):

$$R \leq 2B * ld(M) = 2B * ld \left(\sqrt{1 + \frac{S}{N}} \right) = 2B * \frac{1}{2} ld \left(1 + \frac{S}{N} \right) = B * ld \left(1 + \frac{S}{N} \right)$$

Obwohl die Resultate der Gleichungen formal übereinstimmen, sollte man sich nicht täuschen lassen. Die Theorie von Shannon geht wesentlich weiter als die obigen Überlegungen und erlaubt insbesondere auch Aussagen über Systeme, bei denen das Rauschen grösser als das Signal ist.

4

Data Link Layer (Sicherungsschicht)

Der Data Link Layer (Sicherungsschicht) stellt dem übergeordneten Network Layer eine gesicherte Übertragungsstrecke zwischen zwei direkt miteinander verbundenen Teilnehmern zur Verfügung. Es ist zu beachten, dass im Transport Layer ähnliche Funktionen für netzweite Verbindungen definiert sind, die mit denen des Data Link Layers für direkt miteinander verbundene Teilnehmer vergleichbar sind.

Die Anforderungen an den Data Link Layer sind davon abhängig, ob genau zwei oder mehrere Teilnehmer durch ein Übertragungsmedium miteinander verbunden sind.

Bei einer Punkt-Punkt-Verbindung sind genau zwei Teilnehmer direkt miteinander verbunden.

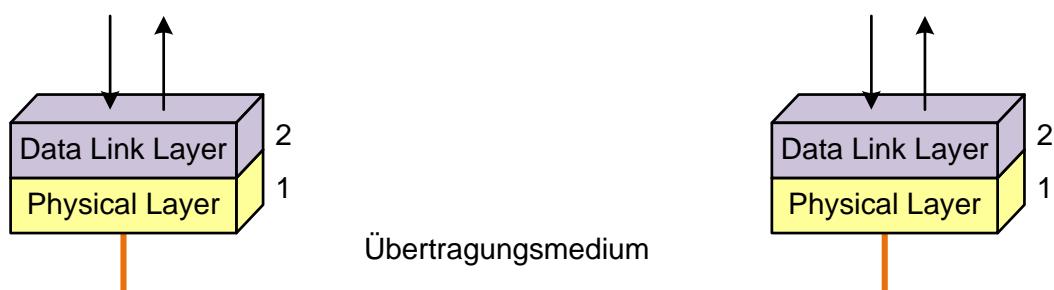


Abbildung 4.1: Punkt-Punkt-Verbindungen

Für Punkt-Punkt-Verbindungen zwischen genau zwei Teilnehmern (Abbildung 4.1) muss der Data Link Layer folgende Aufgaben erfüllen:

- Realisieren einer sicheren Verbindung zwischen zwei direkt miteinander verbundenen Einrichtungen. Dazu werden Massnahmen zur Fehlererkennung oder Fehlerkorrektur notwendig (siehe Abschnitt 4.5).
- Verpacken der vom Network Layer erhaltenen Datenblöcke in Frames für die Übertragung und Auspacken der Datenblöcke aus den empfangenen Frames.
- Frame-Erkennung (siehe Abschnitt 4.1).
- Fluss-Steuerung (Flow Control). Darunter versteht man alle Massnahmen, damit der Sender nicht mehr und schneller Daten sendet, als sie der Empfänger aufnehmen kann.

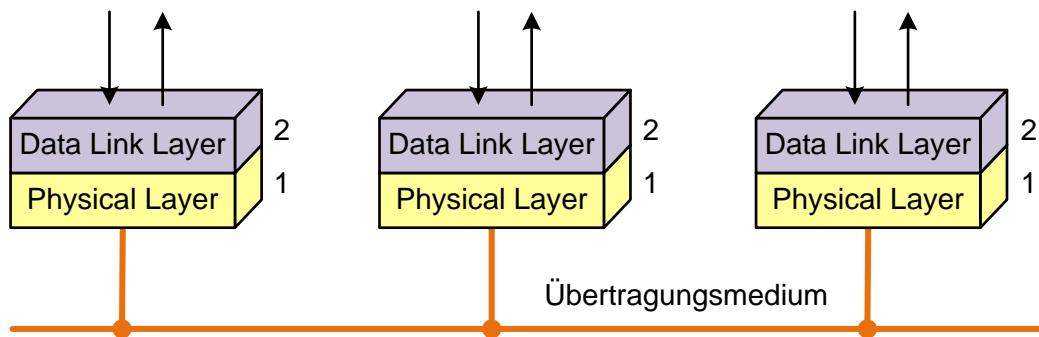


Abbildung 4.2: Konfigurationen mit mehreren Teilnehmern

Abbildung 4.2 zeigt Konfigurationen mit mehreren über ein Übertragungsmedium verbundenen Teilnehmern (z.B. LAN, Funknetz). Dazu muss der Data Link Layer zusätzliche Funktionen bereitstellen:

Addressierung der Teilnehmer: Jeder Teilnehmer muss über eine eindeutige Adresse angesprochen werden können (siehe Kapitel 5).

Medium Zugriff: Da mehrere Teilnehmer ein gemeinsames Übertragungsmedium teilen, muss definiert werden, wie auf den Bus zugegriffen werden darf (siehe Abschnitt “Medium Zugriff”).

4.1 Aufteilen der Layer-3-Daten in Frames

Der Data Link Layer unterteilt die Daten des Network Layers in Blöcke bestimmter Größe, fügt einen eigenen Header dazu und sendet jedes dieser Frames über den ungesicherten Physical Layer. Auf der Empfangsseite müssen diese Frames im ankommenden Bitstrom erkannt und behandelt werden. Bei diesem Vorgang, **Framing** genannt, muss zwischen synchroner und asynchroner Übertragung unterschieden werden.

4.2 Framing bei asynchroner Übertragung

Bei einer asynchronen Übertragung werden nur bei Bedarf Daten übertragen. Stehen keine Daten zur Übertragungsstrecke in Ruhe, so wird die Übertragungsstrecke durch Änderung des Leitungszustands (Startbit des 1. Bytes) den Empfängern angezeigt, dass die Übertragung eines Frames beginnt. Zwischen den Frames muss eine kurze Pause (mindestens eine Zeichendauer) eingehalten werden, um die Synchronisation (z.B. nach einem Bitfehler) sicherzustellen.

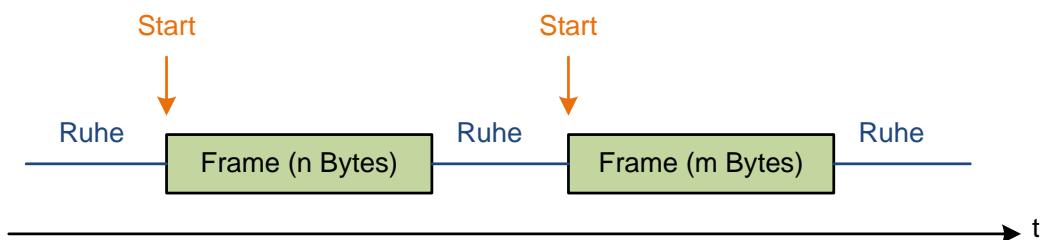


Abbildung 4.3: Frame-Abgrenzung bei asynchroner Übertragung

Ein Frame besteht typischerweise aus einem Header (z.B. Anzahl der Elemente n), dem Datenblock und Prüfbits für die Fehlererkennung (Abbildung 4.4).



Abbildung 4.4: Frame-Aufbau bei asynchroner Übertragung

Anmerkung: Der Data Link Layer fügt die Prüfbits typischerweise am Ende des Frames an, da so die Daten bereits übertragen werden können, während die Prüfbits laufend gebildet werden. Auf den höheren Layern ist es dagegen üblich, die Prüfbits im Header unterzubringen.

4.3 Framing bei synchroner Übertragung

Bei der synchronen Übertragung werden ohne Unterbruch Frames gesendet. Liegen vom Network Layer keine zu sendenden Daten an, so werden Leer-Frames übertragen. Der Data Link Layer des Empfängers muss in diesem kontinuierlichen Bitstrom die Frame-Grenzen erkennen können. Man beachte, dass hier im Gegensatz zur asynchronen Übertragung auch die Byte-Grenzen erkannt werden müssen.

Für das Erkennen von Anfang und Ende eines Frames wird meist das Verfahren mit **Start- und Ende-Flags** eingesetzt. Jedes Frame besteht aus einem Start-Flag, einem Header mit der Anzahl Datenelemente im Datenblock, dem Datenblock, einem Code für die Fehlererkennung und einem Ende-Flag.

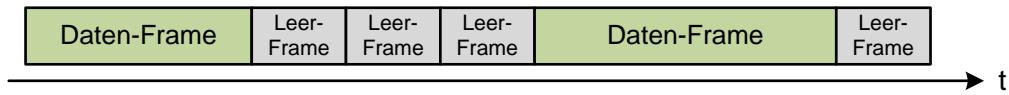


Abbildung 4.5: Framing bei synchroner Übertragung

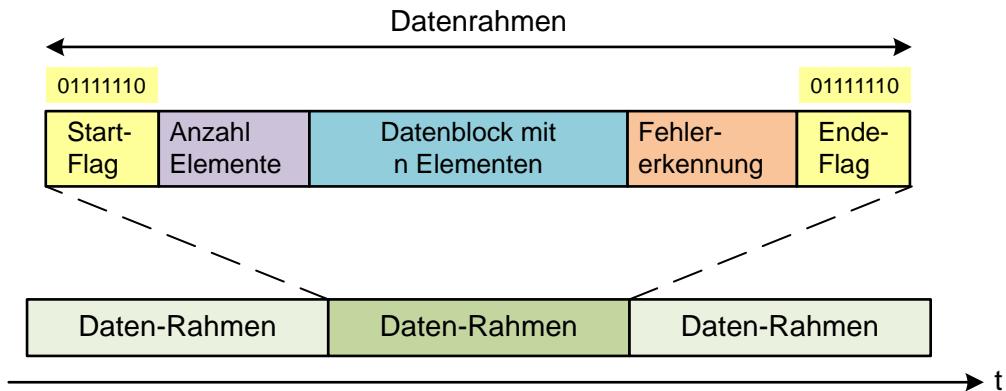


Abbildung 4.6: Frame-Aufbau bei synchroner Übertragung

Als Start- und Ende-Flag wird ein spezielles Bitmuster, meistens 01111110 eingesetzt. Sobald der Empfänger dieses Bitmuster erkennt, betrachtet er dies als Anfang oder Ende eines Frames. Damit darf das Bitmuster der Flags im Frame nicht mehr vorkommen. Dies wird mit folgendem Verfahren, **Bitstopfen** genannt, verhindert (Abbildung 4.7).

Wenn der Data Link Layer des Senders in der zu übertragenden Bitfolge (ausser den Flags!) fünf aufeinanderfolgende Einsen entdeckt, stopft er eine zusätzliche Null in den Bitstrom (Stopfbit). Der Data Link Layer des Empfängers entfernt nach fünf aufeinanderfolgenden Einsen immer ein Bit, da er weiß, dass ein Stopbit eingefügt wurde. Somit können Flags anhand der 6 hintereinander folgenden Einsen eindeutig identifiziert werden. Für die höheren Schichten ist dieser Vorgang nicht sichtbar.

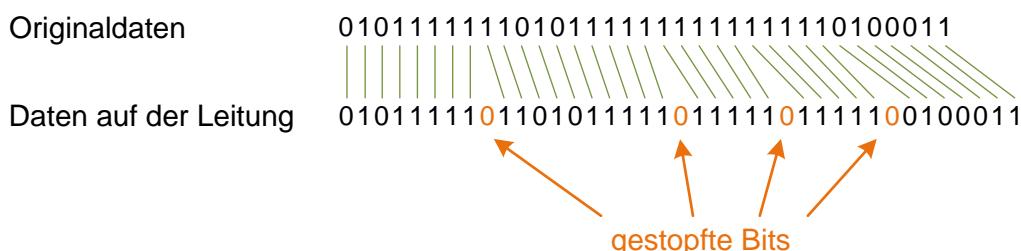


Abbildung 4.7: Bitstopfen fügt nach fünf Einsen immer eine Null ein

4.4 Wahl der Frame-Länge

Beim Framing stellt sich die Frage, wie die Frame-Länge gewählt werden soll. Grundsätzlich bieten lange Frames ein besseres Verhältnis von Nutzdaten zu Header. Bei einer vom Physical Layer gegebenen **Bruttoabitrate** gilt: Je länger die Frames desto besser wird die **Nettobitrate**.

$$\text{Nettobitrate} = \text{Bruttoabitrate} * \frac{\text{Nutzdaten}}{\text{Nutzdaten} + \text{Header}} \quad (4.1)$$

Lange Frames haben aber auch (je nach Applikation entscheidende) Nachteile:

Durchsatz: Die Wahrscheinlichkeit, dass bei einem langen Frame ein Übertragungsfehler auftritt, wird grösser.

Effizienz: Wenn ein Fehler auftritt, ist das Frame verloren; d.h. je länger das Frame desto grösser ist im Fehlerfall der Datenverlust.

Zuverlässigkeit: Die Wahrscheinlichkeit, dass sich ein Fehler einschleicht, der nicht entdeckt wird, wächst mit der Länge des Frames.

Jitter: Darunter versteht man die Variation der Zeitabstände zwischen Frames. Wie im Abschnitt 4.4.1 ausgeführt, erhöhen lange Frames den Jitter.

4.4.1 Einfluss der Frame-Länge auf Jitter

Viele Applikationen erfordern den Austausch von Daten in regelmässigen Intervallen. Man spricht von **isochronen Daten**. Die Zeitvariation der Frames wird als **Jitter** bezeichnet. Während das Datenvolumen oft bescheiden ist, erfordern diese Applikationen eine Begrenzung des Jitters. Typische Anwendungen mit isochronen Daten sind Steuerungs- und Regelungsaufgaben aber auch Data Streaming in Multimedia-Applikationen.

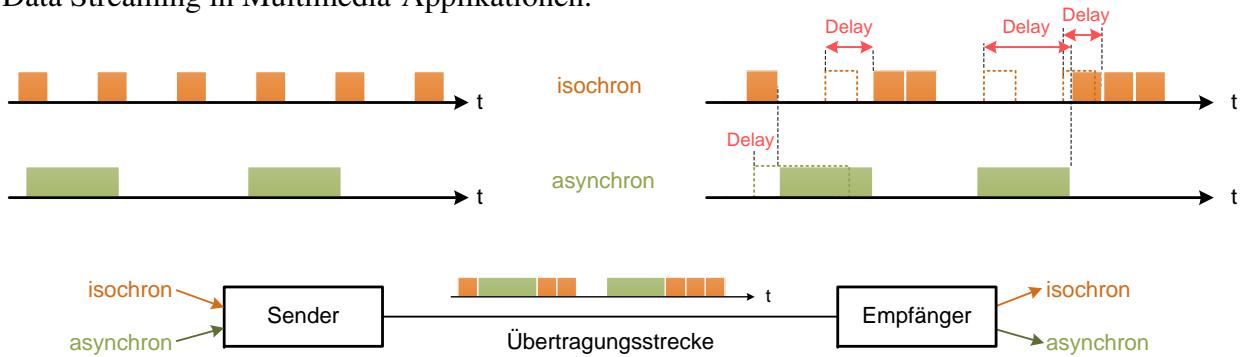


Abbildung 4.8: Entstehung des Jitters bei isochronen Daten

Lange Frames blockieren die Übertragungsstrecke für die Dauer ihrer Übertragung. Wie Abbildung 4.8 zeigt, führt die Blockierung der Übertragungsstrecke durch asynchrone Frames zu einer Verklumpung der isochronen Frames, die natürlich umso ausgeprägter ausfällt, je länger die asynchronen Frames sind.

4.4.2 Einfluss der Frame-Länge auf den Durchsatz

Wie oben erwähnt, wird der Durchsatz von zwei gegenläufigen Größen beeinflusst: der Netto-bitrate und der Frame-Fehlerwahrscheinlichkeit. Beide nehmen zu, wenn größere Frames gesendet werden. Letztere hängt von der **Bitfehlerwahrscheinlichkeit** p_e des Physical Layers ab (**BER – Bit Error Ratio**). Für die Bestimmung der Frame-Fehlerwahrscheinlichkeit bestimmen wir zunächst die Wahrscheinlichkeit, dass ein Frame der Länge N keinen Fehler enthält:

$$\text{Frame-Erfolgswahrscheinlichkeit} = (1 - p_e)^N \quad \text{für } p_e \ll 1 \text{ gilt } \approx (1 - Np_e) \quad (4.2)$$

und somit wird:

$$\text{Frame-Fehlerwahrscheinlichkeit} = 1 - (1 - p_e)^N \quad \text{für } p_e \ll 1 \text{ gilt } \approx Np_e \quad (4.3)$$

Abbildung 4.8 zeigt den Einfluss der beiden Faktoren auf den effektiven Durchsatz. In diesem Beispiel mit einer Header-Länge von 200 Bit und BER $p_e = 10^{-4}$ würde der optimale Durchsatz bei einer Frame-Länge von ca. 1500 Bit erreicht.

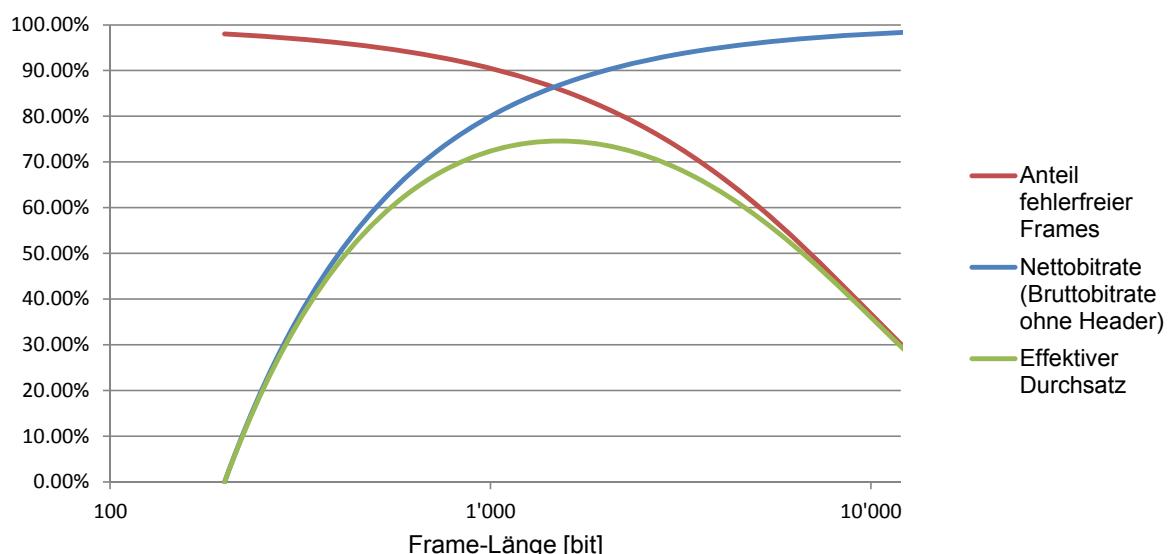


Abbildung 4.9: Durchsatz in Funktion der Frame-Länge

Für eine Bitfehlerwahrscheinlichkeit $p_e \ll 1$ lässt sich die optimale Frame-Länge bei einer Header-Länge H näherungsweise berechnen:

$$\text{Optimale Frame-Länge} \approx \sqrt{\frac{H}{p_e}} \quad (4.4)$$

Für das in Abbildung 4.8 gezeigte Beispiel ergibt die Näherung 1414 Bit.

4.5 Fehlererkennung und Fehlerkorrektur

Wie Eingangs erwähnt, ist es die Aufgabe des Data Link Layers zwischen zwei direkt miteinander verbundenen Einrichtungen eine sichere Verbindung zu realisieren. Dazu müssen die bei der Datenübertragung aufgetretenen Fehler zunächst möglichst zuverlässig erkannt werden (Abbildung 4.10). Falls der Data Link Layer einen zuverlässigen Dienst zur Verfügung stellt, müssen fehlerbehaftete Daten nochmals gesendet werden (**Retransmission**).

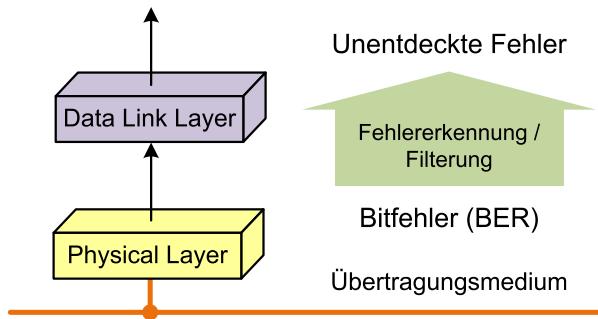


Abbildung 4.10: Fehlererkennung und Filterung als Grundlage einer sicheren Verbindung

Beispiel: Der Standard für das lokale Netz (IEEE 802) fordert vom Physical Layer bei drahtgebundenen Übertragungsmedien die Einhaltung einer $\text{BER} < 10^{-8}$. Dies ist eine Minimalanforderung, denn effektiv liegt bei Ethernet die BER (je nach Technologie) bei 10^{-10} bis 10^{-12} . Der gleiche Standard garantiert den höheren Layern, dass die Wahrscheinlichkeit eines unentdeckten Fehlers in den vom Data Link Layer gelieferten Frames unter $5 * 10^{-14}$ pro Frame-Byte liegt.

Fehlererkennung und Fehlerkorrektur beruhen auf Redundanz; d.h dem bewussten Hinzufügen von Prüfinformation, die nicht der Datenübertragung dient. Man kann folgende Verfahren unterscheiden:

Backward Error Correction: Der Empfänger kann Fehler erkennen, aber nicht korrigieren. Durch geeignete Massnahmen wird der Sender aufgefordert, die Daten noch einmal zu senden (Retransmission). Dieses Verfahren funktioniert bei einer kleinen Bitfehlerwahrscheinlichkeit sehr gut.

Forward Error Correction: Der Empfänger kann den Fehler nicht nur erkennen, sondern auch direkt korrigieren. Dieses Verfahren ist notwendig, wenn kein Rückkanal vorhanden ist oder die Fehlerwahrscheinlichkeit gross ist.

Beispiel: Bei einer $\text{BER} = 10^{-3}$ (jedes tausendste Bit ist falsch) und einer Meldungslänge von 2000 Bit müssten ca. 9 von 10 Meldungen, genauer $1 - (1 - 10^{-3})^{2000} = 86\%$, wiederholt werden und mit derselben Wahrscheinlichkeit wäre die wiederholte Meldung auch wieder fehlerhaft. Offensichtlich wird das Verfahren bei einer hohen BER unbrauchbar.

4.6 Fehlererkennung

Wie Abbildung 4.11 zeigt, erzeugt der Sender aus einer Anzahl Datenbits m eine Anzahl Prüfbits p und überträgt diese mit den Daten. Der Empfänger wertet die Daten- und die Prüfbits aus und ent-

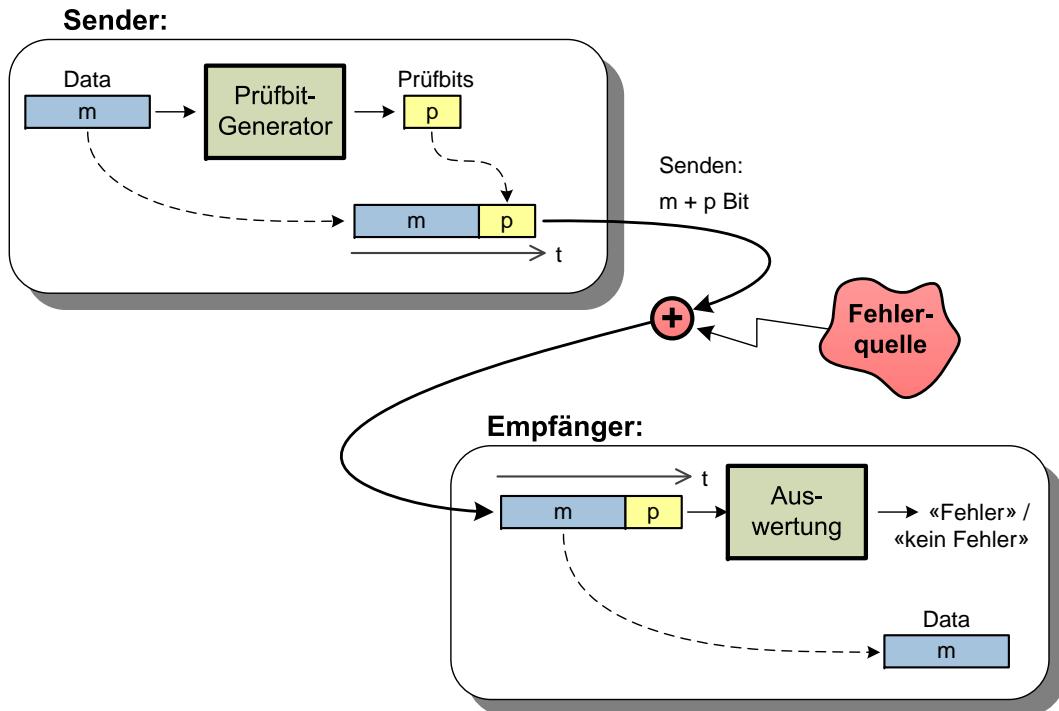


Abbildung 4.11: Grundprinzip der Fehlererkennung bei einer Übertragung

scheidet, ob ein Fehler vorliegt oder nicht. Die Auswertung erfolgt oft, indem der Empfänger aus den empfangenen Daten die Prüfbits nochmals erzeugt und mit denen vergleicht, die vom Sender übertragen wurden. Das Grundproblem besteht in der Konstruktion eines guten Prüfbitgenerators, so dass (bei gegebenen m und p) möglichst viele Fehler anhand der Prüfbits entdeckt werden können.

4.6.1 Hamming-Distanz als Grundlage der Fehlererkennung

Die sogenannte **Hamming-Distanz**¹ ist der Schlüssel zur Fehlererkennung (und Fehlerkorrektur). Die Hamming-Distanz gibt an, wie viele Bits im Bitmuster eines gültigen Codes (mindestens) geändert werden müssen, bis der nächste gültige Code vorliegt. Im Folgenden soll das Prinzip anhand eines 3-Bit-Codes vereinfacht erklärt werden.

Die 3-Stellen des Codes werden je einer Dimension des Raums zugeordnet, wie links in Abbildung 4.12 gezeigt. Die acht möglichen Bitmuster werden so systematisch in je einer Ecke eines Würfels angeordnet.

Hamming-Distanz 1: Im ersten Fall (linker Würfel) gibt es keine Redundanz; alle acht Codes werden genutzt. Die Änderung irgendeines Bits bei einem beliebigen Code führt immer zum nächsten gültigen. Das heisst, bei Hamming-Distanz 1 können keine Fehler erkannt werden.

¹Benannt nach dem amerikanischen Mathematiker Richard Wesley Hamming (1915 – 1998).

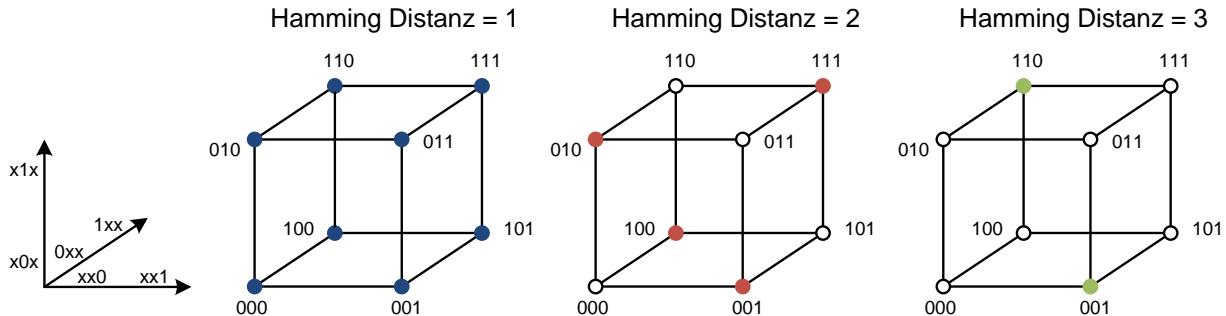


Abbildung 4.12: Hamming-Distanz anhand eines dreistelligen Codes

Hamming-Distanz 2: Im zweiten Fall (mittlerer Würfel) werden nur vier der acht möglichen Bitmuster für die Codierung der Daten verwendet (001, 010, 100, 111). Die anderen vier Bitmuster (000, 011, 101, 110) sind ungültige Codes und zeigen einen Fehler an. Wie man am Würfel leicht sieht, braucht es immer 2 Bitänderungen um zum nächsten gültigen Code zu gelangen. Eine Hamming-Distanz 2 bedeutet also, dass man einen Einbitfehler erkennen kann.

Hamming-Distanz 3: Wenn wir nun (Würfel rechts) nur noch zwei der acht Bitmuster (001, 110) verwenden, beträgt die Hamming-Distanz 3, was erlaubt Ein- und Zweibitfehler zu erkennen. Welche der diagonal gegenüberliegenden Bitmuster-Paare wir verwenden, ist gleichgültig.

Es lassen sich für beliebig lange Datenblöcke Codes finden, die es erlauben, eine geforderte Anzahl Bitfehler zu erkennen. Je grösser die Anzahl der Fehler ist, die man erkennen will, desto grösser muss die Hamming-Distanz sein.

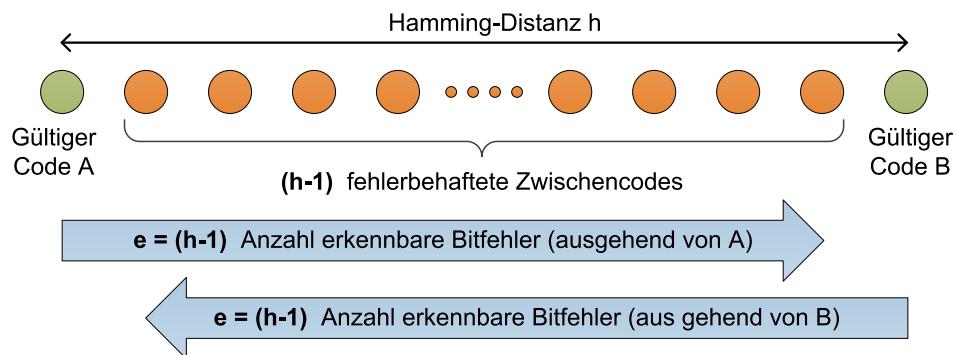


Abbildung 4.13: Zwischen zwei gültigen Codes liegen $h - 1$ fehlerbehaftete Zwischencodes

Wie aus Abbildung 4.13 leicht ersichtlich ist, gibt es bei einer Hamming-Distanz h genau $h - 1$ ungültige Zwischencodes. Damit gilt für die Anzahl der erkennbaren Fehler e :

$$e = h - 1 \quad (4.5)$$

4.7 Fehlererkennung mit Parity

Eine der einfachsten Methoden zur Fehlererkennung geht über die Parität (Parity). Jedem Daten-element wird ein zusätzliches Bit (das sogenannte Parity-Bit) zugefügt, dessen Wert so gewählt wird, dass die Anzahl der Einsen inklusiv Parity-Bit entweder gerade (Even Parity) oder ungerade (Odd Parity) ist. Die Wahl ob Even Parity oder Odd Parity ist willkürlich; beide sind gleichwertig.

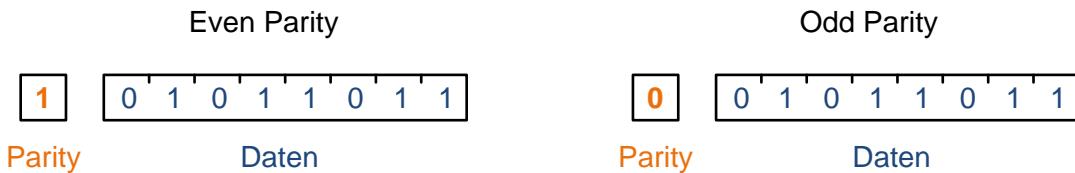


Abbildung 4.14: Even und Odd Parity über 8 Datenbits

Man sieht in Abbildung 4.14, dass die Änderung eines beliebigen Bits immer zu einer ungültigen Parität führt. Es müssen immer 2 Bits geändert werden, um wieder einen gültigen Code zu erhalten. Dies entspricht einer Hamming-Distanz 2, womit man mit dieser Methode genau einen Bitfehler erkennen kann. Ein einziges Parity-Bit genügt, um in einer beliebig langen Bitsequenz einen Bitfehler erkennen zu können.

Die Codes des mittleren Würfels in Abbildung 4.12 können auch als 2 Datenbits mit einem Parity-Bit interpretiert werden (Odd Parity), wobei es gleichgültig ist, welches Bit wir als Paritybit interpretieren. Die ungültigen 4 Codes haben alle Even Parity. Die Wahl, welche der Codegruppen wir als gültig oder ungültig betrachten, entspricht also der Wahl zwischen Odd oder Even Parity.

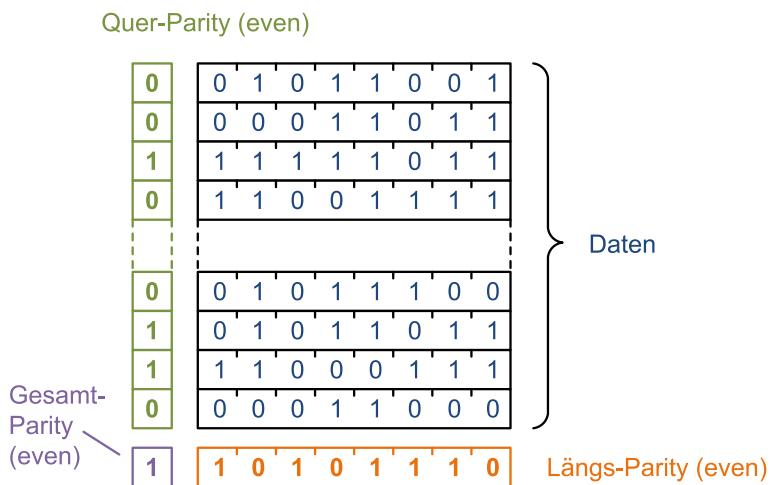


Abbildung 4.15: Datenblock mit Quer- und Längs-Parity

In einem mit Quer- und Längs-Parity gesicherten Datenblock (Abbildung 4.15) können 3 Fehler sicher erkannt werden. Bei einem fehlerhaften Bit im Datenblock zeigen die Quer-Parity und die Längs-Parity je einen Fehler an und diese sind zusätzlich über eine Gesamt-Parity gesichert. Also braucht es total 4 Bitänderungen um den nächsten gültigen Code zu erreichen. Somit beträgt die Hamming-Distanz 4.

4.8 Prüfsumme

Eine oft eingesetzte Methode zur Fehlererkennung ist eine Prüfsumme (**Checksum**). Dabei wird über eine Anzahl Datenelemente n eine Summe der Breite m gebildet (Abbildung 4.16).

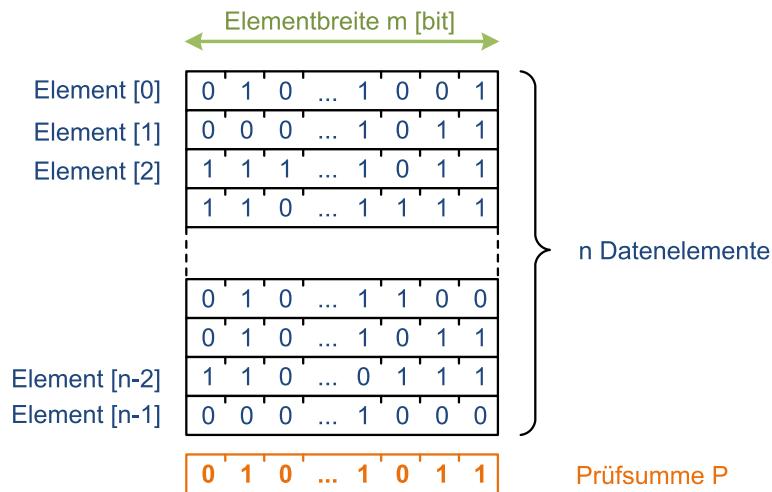


Abbildung 4.16: Prüfsumme über einen Datenblock

Die Prüfsumme P kann durch einfache Addition berechnet werden:

$$P = \left(\sum_{i=0}^{n-1} \text{Element}[i] \right) \mod 2^m$$

In der Praxis wird der Begriff „Prüfsumme“ oft für irgendwelche Prüfbits verwendet, die anders als oben beschrieben erzeugt wurden (z.B. Längs-Parity) oder die Prüfsumme wird noch „veredelt“, was aber kaum Einfluss auf die Qualität der Fehlererkennung hat.

Beispiel: Bei TCP und UDP wird über das gesamte Frame eine Prüfsumme der Breite $m = 16$ gebildet. Anschliessend wird die Prüfsumme P invertiert zu P_F . Mathematisch betrachtet ist $P_F = -P - 1$. Die Prüfsumme P_F wird mit den Daten übertragen. Der Empfänger addiert die Elemente und die Prüfsumme P_F . Falls das Resultat nicht -1 ist, liegt ein Fehler vor.

Das Hauptproblem von einfachen Prüfsummen und Längs-Parity besteht in ihrer Anfälligkeit auf Mehrbitfehler. Bei beiden kann nur ein Fehler sicher erkannt werden. Der Vorteil der Prüfsumme besteht darin, dass ein Burst von gleichartigen Fehlern (z.B. mehrere Bits auf 1) zu einem Übertrag auf die höherwertigeren Stellen führen kann. Damit können Fehler sich nicht mehr so einfach gegenseitig verdecken.

4.9 Cyclic Redundancy Check

Mit der CRC-Technik (Cyclic Redundancy Check) können mit bescheidenem Aufwand Mehrfach- und Burstfehler in Datenblöcken mit grosser Sicherheit erkannt werden. Das Verfahren beruht auf

der Idee, dass sich ein Fehler auf möglichst viele Prüfbits auswirken soll. Durch dieses Verschmieren können nachfolgende Fehler die Signatur der vorgängigen nicht mehr so leicht aufheben.

Das CRC-Verfahren wird mathematisch als Polynomdivision beschrieben, wobei der Divisionsrest die Prüfbits bildet. Der Divisor wird durch ein sogenanntes Generatorpolynom gebildet, das maßgeblich für die Güte der Fehlererkennung verantwortlich ist. Tatsächlich ist die Hamming-Distanz nur abhängig von der Wahl des Generatorpolynoms und der Länge der Daten. Generatorpolynome werden in der folgenden Form beschrieben:

$X^4 + X + 1$ bedeutet $1 * X^4 + 0 * X^3 + 0 * X^2 + 1 * X^1 + 1 * X^0$ und entspricht 10011₂

Wie Abbildung 4.17 zeigt, wird der gesamte Datenbitstrom durch das Generatorpolynom dividiert, wobei spezielle Rechenregeln angewandt werden. (Die Subtraktion wird z.B. als bitweise XOR-Verknüpfung ausgeführt). Man erkennt, dass sich das Verfahren auf einen beliebig langen Bitstrom anwenden lässt.

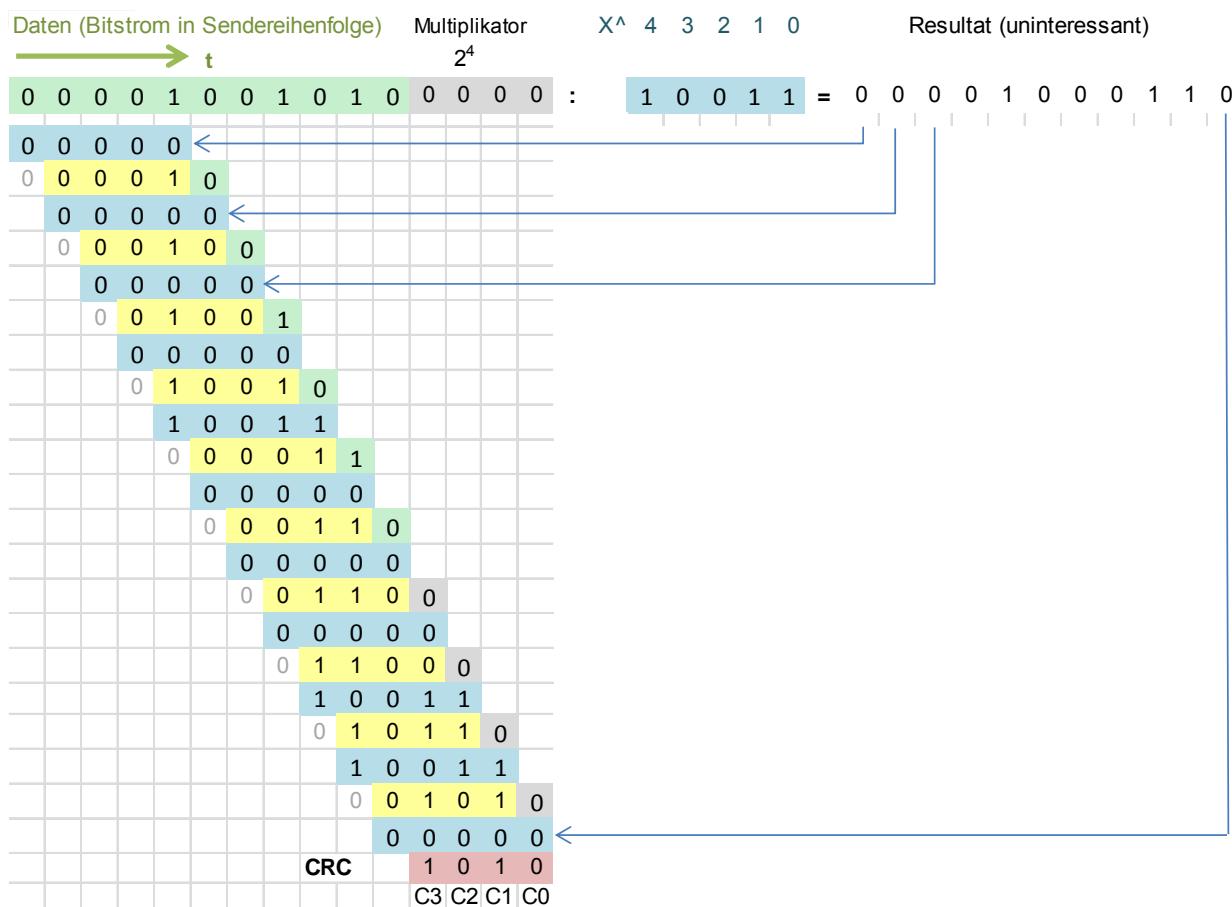


Abbildung 4.17: CRC Bildung durch Division mit dem Polynom $X^4 + X + 1$

CRC-Generator und -Prüfer lässt sich sehr einfach in Hardware mittels rückgekoppelter Schieberegister realisieren. Wie Abbildung 4.18 zeigt, verwenden Sender und Empfänger dieselbe Schaltung. Der Sender überträgt nach den Daten den CRC. Man beachte, dass dieser mit dem Divisionsrest in Abbildung 4.17 übereinstimmt.

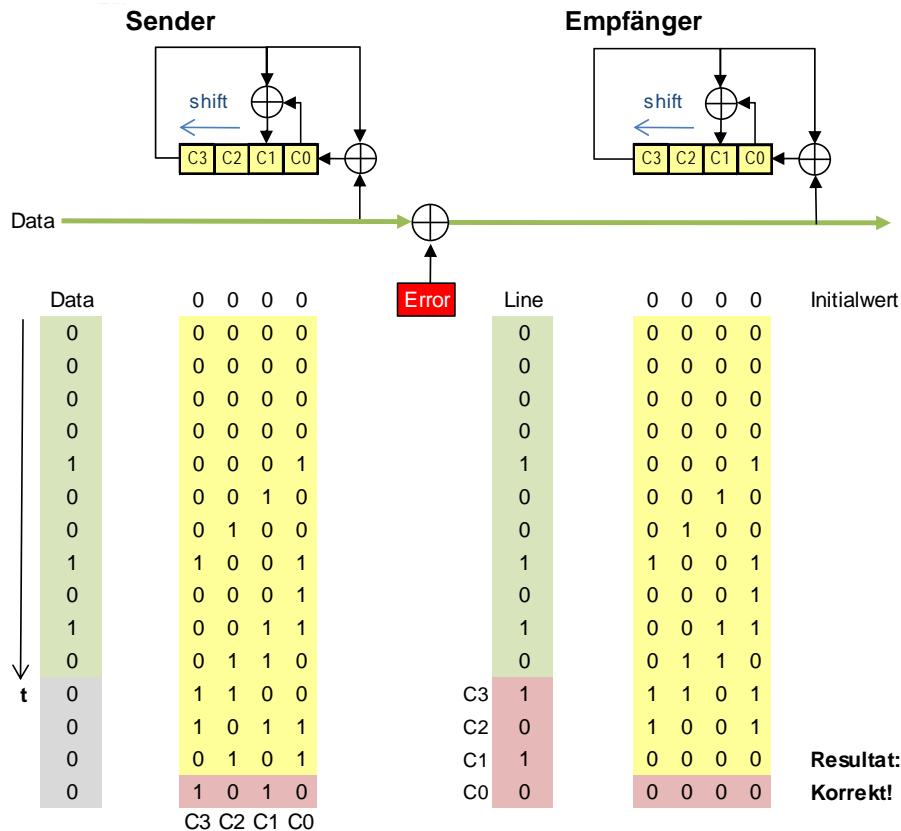


Abbildung 4.18: CRC Bildung und Überprüfung in Hardware

Der Empfänger behandelt die CRC-Bits gleich wie Datenbits. Ein Resultat ungleich 0 bedeutet, dass ein Fehler vorliegt (Abbildung 4.19).

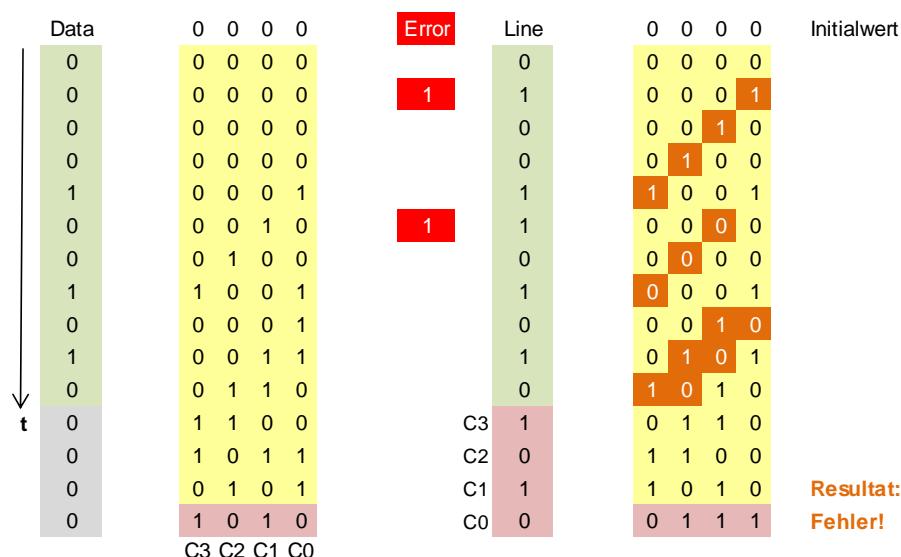


Abbildung 4.19: CRC Bildung und Überprüfung in Hardware

4.10 Fehlerkorrektur

4.10.1 Backward Error Correction

Die Fehlerkorrektur auf dem Data Link Layer wird als Backward Error Correction oder Automatic Repeat-reQuest (ARQ) bezeichnet. Es basiert auf einer wiederholten Übertragung von Frames durch den Sender. Der Empfänger muss dabei lediglich in der Lage sein, Frames auf Fehler zu prüfen.

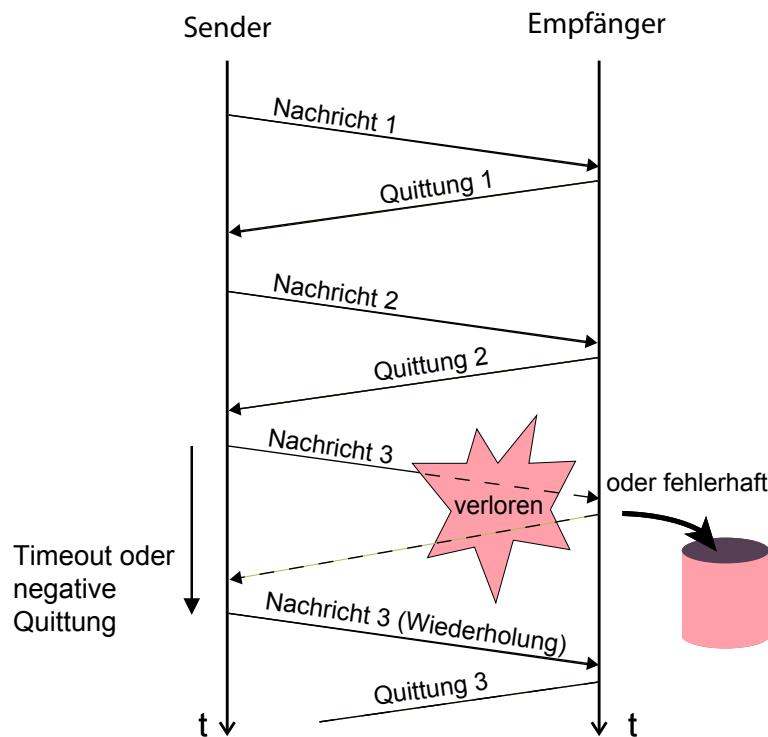


Abbildung 4.20: Prinzip des Verfahrens Automatic Repeat Request

Wie Abbildung 4.20 zeigt, sendet der Empfänger eine Quittung, sobald ein Frame korrekt empfangen wurde. Geht das Frame oder die Quittung verloren, so wird der Sender keine Quittung erhalten und wird nach einer gewissen Zeit (Timeout) das Frame automatisch nochmals senden. Dieser Vorgang wird solange wiederholt, bis beim Sender eine Quittung für das Frame eintrifft.

Im einfachsten Fall muss der Empfänger einen Fehler im Frame lediglich erkennen und das Frame vernichten. Es gibt verschiedene Varianten, beispielsweise kann der Empfänger eine negative Quittung verschicken und so den Sender zur sofortigen Wiederholung des Frames auffordern. Die hier beschriebene gilt als die einfachste Variante des ARQ-Protokolls (Stop-and-Wait) und ist in Bezug auf den Durchsatz nicht optimal. Im Kapitel 8 werden wir bei der Behandlung des TCP Protokolls auf das Thema zurückkommen.

4.10.2 Forward Error Correction

Die Anwendung der Backward Error Correction hat ihre Grenzen. Ist die Bitfehlerwahrscheinlichkeit hoch, so dass in praktisch jedem Frame ein Fehler auftritt, wird das Verfahren extrem ineffizient oder gänzlich unbrauchbar.

In diesem Fall kommt die Forward Error Correction zum Tragen. Darunter versteht man die Reduktion der Bitfehlerwahrscheinlichkeit durch fehlerkorrigierende Leitungscode. Wie bereits früher erwähnt, ist dieses Thema nicht Bestandteil dieses Moduls. Ziel der folgenden Ausführungen ist darum den Zusammenhang von Fehlererkennung und Fehlerkorrektur aufzuzeigen und nicht die Behandlung von fehlerkorrigierenden Codes.

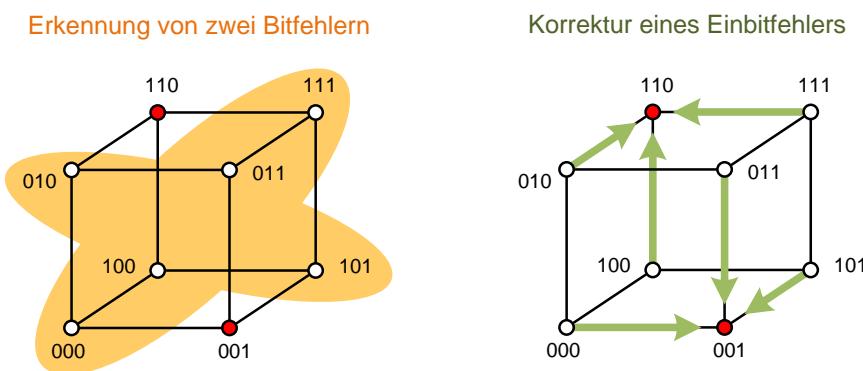


Abbildung 4.21: Prinzip der Fehlerkorrektur bei Hamming-Distanz 3

Wir haben gesehen, dass ein Code mit einer Hamming-Distanz von 3 die Erkennung von 1 oder 2 Bitfehlern erlaubt. Abbildung 4.21 zeigt, dass derselbe Code alternativ verwendet werden kann, um einen Einbitfehler zu korrigieren. Wenn wir Einbitfehler korrigieren, so können keine Zweibitfehler mehr erkannt werden. Treten Mehrbitfehler auf, so werden sie zum nächstliegenden Code „korrigiert“. Wie man leicht sieht, ist die Anzahl der korrigierbaren Bitfehler k kleiner als die Hälfte der Hamming Distanz h :

$$k \leq \frac{h-1}{2} \quad (4.6)$$

Abbildung 4.22 zeigt den Zusammenhang zwischen Hamming-Distanz h , der Anzahl der korrigierbaren Bitfehler k und der Anzahl der erkennbaren Bitfehler e .

Es gibt einen Spielraum zwischen der Anzahl der korrigierbaren und der erkennbaren Bitfehler:

$$k + e = (h - 1) \quad \text{wobei gilt: } k \leq \frac{h-1}{2} \quad (4.7)$$

Beispiel: Eine Blocksicherung mit Längs- und Quer-Parity hat eine Hamming Distanz von vier. Verwenden wir die Zeilen und Spalten der Parity-Fehler als Koordinaten, so kann ein einzelner Bitfehler lokalisiert und damit korrigiert werden (Abbildung 4.23). Treten 2 Bitfehler auf, so können diese im allgemeinen Fall nicht mehr lokalisiert und somit auch nicht korrigiert werden. Wegen der Hamming-Distanz von 4 liegt der Zweibitfehler zwischen den korrigierbaren Einbitfehlern, was bedeutet, dass Zweibitfehler weiterhin zuverlässig erkannt werden können.

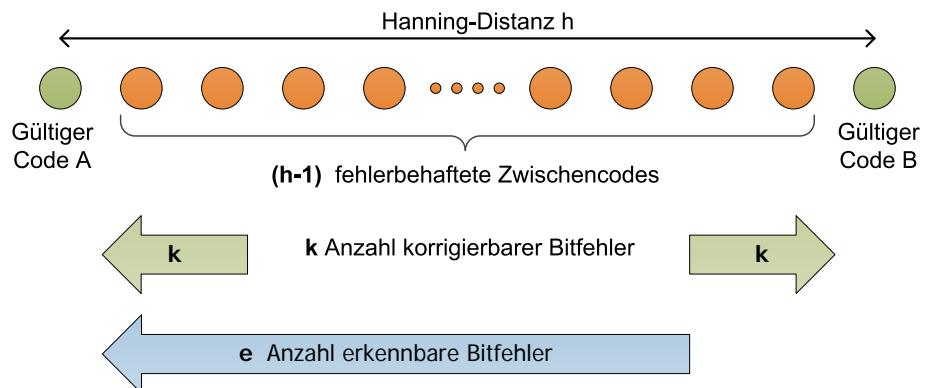


Abbildung 4.22: Anzahl der korrigierbaren und erkennbaren Bitfehler

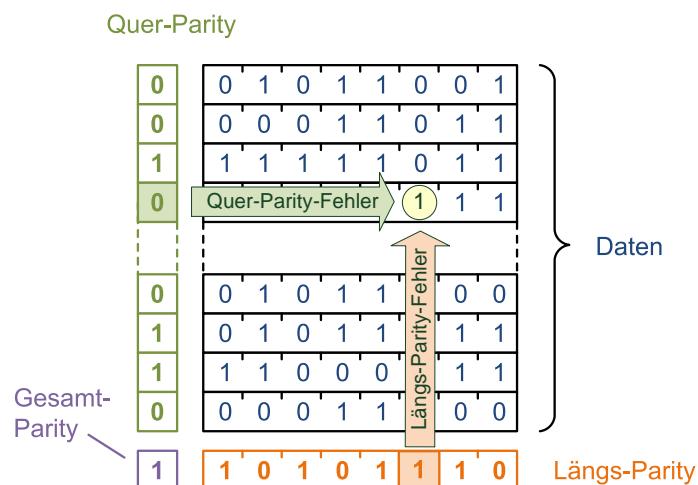


Abbildung 4.23: Fehlerkorrektur mit Längs- und Quer-Parity

4.11 Flusssteuerung

Überlastprobleme können beim Empfänger auftreten, wenn ein Sender über eine gewisse Zeit mehr Daten produziert, als der Empfänger verarbeiten kann. Geht dem Empfänger der Speicher aus, so bleibt ihm nichts anderes übrig, als Daten zu verwerten. Um solche Situationen zu verhindern, wird eine Flusssteuerung (**Flow Control**) benötigt, die es dem Empfänger erlaubt, den Sender temporär zu stoppen (Abbildung 4.24). Erreicht der Empfangs-Buffer des Empfängers eine obere Limite, so schickt dieser eine Stop-Meldung. Und umgekehrt: Wenn im Empfangs-Buffer eine untere Limite erreicht ist, wird der Sendevorgang mit einer Startmeldung wieder freigegeben. Die Stop-/Start-Meldungen können über separate Steuerleitungen oder Meldungen im Datenrückkanal erfolgen.

Wenn für die Fehlerkorrektur ein Stop-and-Wait-Protokoll verwendet wird, so ist eine implizite Flusssteuerung vorhanden (Abbildung 4.25). Der Empfänger kann einfach die Quittierung verzögern, bis er wieder bereit zur Aufnahme von Daten ist. Im einfachsten Fall wird das Problem so gelöst, dass der Empfänger die Quittierung jedes einzelnen Frames verzögert, bis er dessen Daten verarbeitet hat. Eine solche einfache Fluss-Steuerung kann die Kapazität des Übertragungskanals nicht gut nutzen. In der Praxis werden darum oft sogenannte Schiebefens-terprotokolle eingesetzt (siehe Kapitel 8).

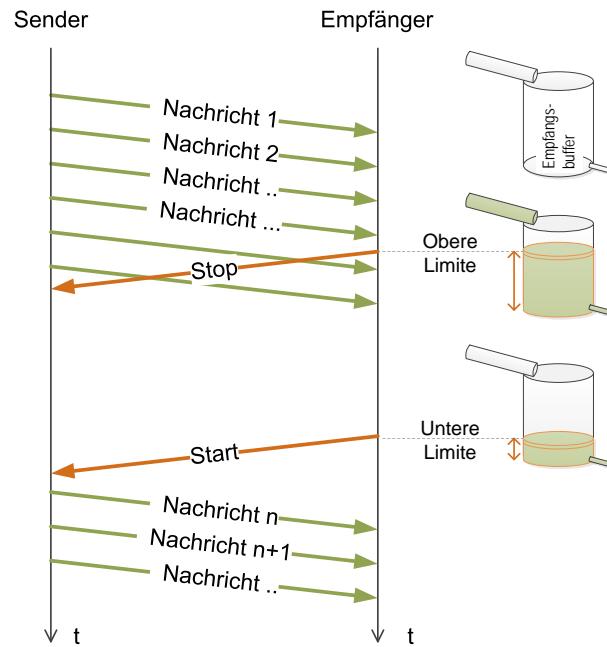


Abbildung 4.24: Flusssteuerung mit Start-Stop-Meldungen

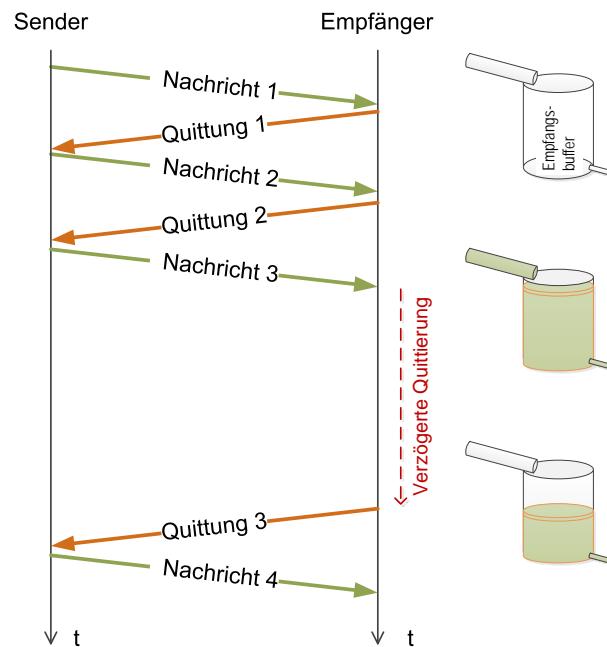


Abbildung 4.25: Flusssteuerung mit Stop-and-Wait-Protokoll

Um Überlastprobleme auf Stufe des Data Link Layers ursächlich beheben zu können, müssen diese über die höheren Schichten an die Applikation gemeldet werden, da dort die Daten letztendlich generiert werden.

4.12 Medium Zugriff

Einer der wichtigsten Dienste des Data Link Layers bei Kopplungen von mehreren Teilnehmern ist die Regelung des Zugriffs auf das gemeinsame Übertragungsmedium (Shared Medium).

Die Verfahren zur Regelung des Zugriffs auf das Übertragungsmedium (**Media Access Control**) lassen sich grob unterteilen in deterministische und nicht-deterministische Verfahren. Im Kapitel 5 wird ein undeterministisches Verfahren detailliert behandelt. Im Folgenden beschränken wir uns darauf, grobe Lösungsansätze aufzuzeigen.

4.12.1 Deterministische Verfahren

a) Master/Slave-Verfahren

Bei diesem einfachen Zugriffsverfahren fragt der Master zyklisch jeden Slave an. Der angesprochene Slave antwortet sofort und liefert vorhandene Daten an den Master. Der Master kann so mit einfach ein Abbild aller Slave-Daten abspeichern. Das Verfahren hat den Vorteil, dass die Anschaltung der Slaves sehr einfach ist und dass die Zeit zwischen zwei Slave-Zugriffen vor- aussehbar ist. Damit ist das Master/Slave-Verfahren deterministisch. Nachteilig ist die grösere Transferzeit beim Datenaustausch zwischen zwei Slaves, da die Daten immer über den Master transferiert werden. Kritisch ist ein Ausfall des Masters, da dies automatisch zum Systemausfall führt. Das Master/Slave-Verfahren wird bei den meisten Feldbussen im Bereich Automatisierungs- technik eingesetzt.

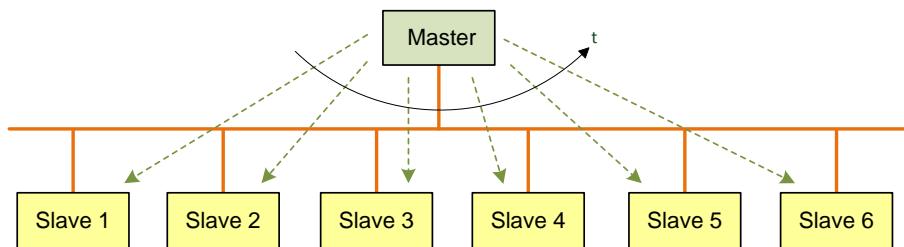
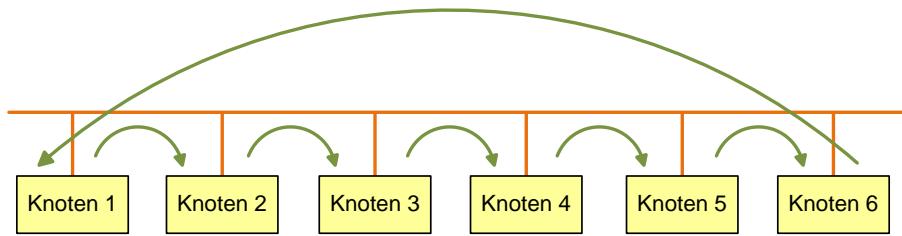


Abbildung 4.26: Prinzip des Master/Slave-Verfahrens

b) Token Passing

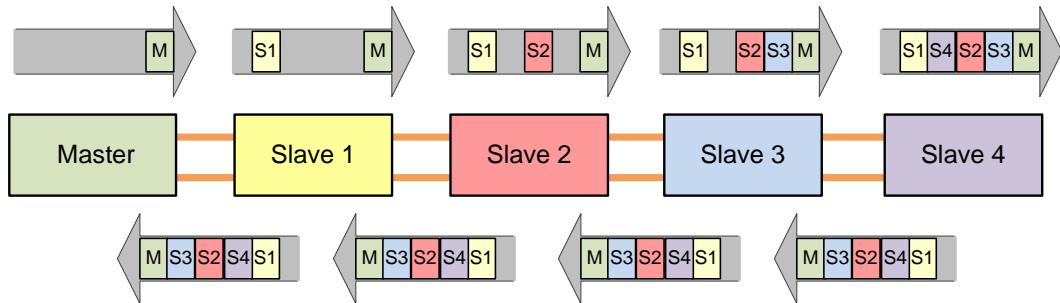
Beim Token Passing wird zwischen gleichberechtigten Knoten in einer festgelegten Reihenfolge eine Sendeberechtigung (Token) weitergereicht. Die Netzwerk-Geräte senden nur, wenn sie ein Token erhalten haben.

Das Token Passing ist wiederum ein deterministischer Zugriffsmechanismus. Man weiss ähnlich wie beim Master/Slave-Verfahren, dass jeder Knoten innerhalb eines bestimmten Zeitintervalls einmal sendeberechtigt ist. Der Vorteil ist allerdings, dass die Kommunikation Peer-to-Peer, also direkt zwischen den Knoten, erfolgt. Nachteilig wirkt sich aus, dass das Verfahren beim Startup und im Betrieb (Management) relativ aufwändig ist. Geht beispielsweise ein Token verloren, so muss ein Mechanismus vorhanden sein, um dieses zu regenerieren.

**Abbildung 4.27:** Token Passing

Historische Netze, die den Medium-Zugriff per Token Passing regeln, sind z.B. Token Ring/IEEE 802.5 und Fiber Distributed Data Interface (FDDI). Heute wird das Verfahren beispielsweise in Multi-Master-Applikationen zur Festlegung des aktiven Masters eingesetzt.

Als Variante kann ein Master anstelle eines Tokens ein Leer-Frame verschicken (Abbildung 4.28). Die Knoten fügen ihre Daten an vorbestimmten Positionen in das Frame ein. Typische Anwendungen dieses Verfahrens sind die ISDN/B-Kanäle und die Feldbusse Interbus und Ethercat.

**Abbildung 4.28:** Frame Passing

c) Zeitgesteuerter Zugriff

In zeitgesteuerten Netzen wird der Datenverkehr geplant; analog zum Taktfahrplan im Bahnhof. Der Zugriff auf das Übertragungsmedium erfolgt zeitgesteuert. Dadurch wird es möglich, den Netzbetrieb auf Auslastung oder Durchsatz zu optimieren. Nachteile sind die erforderliche Planung, die bei jeder Änderung der Verkehrssituation wiederholt werden muss. Zudem muss in allen Netzknoten eine genaue Zeit und die Kommunikationstabellen (Fahrplan) vorhanden sein.

Dieses Verfahren wird bei PROFINET/IRT und IEEE 802.1 für Time Sensitive Networks eingesetzt.

4.12.2 Undeterministische Verfahren

a) Carrier Sense Multiple Access

Beim CSMA-Verfahren ist jede Station gleichberechtigt und darf jederzeit auf den Bus zugreifen, nachdem kontrolliert wurde, dass dieser frei ist (Carrier Sense). Ist der Bus besetzt, so wird gewartet und etwas später wieder versucht die Meldung abzusetzen (Multiple Access).

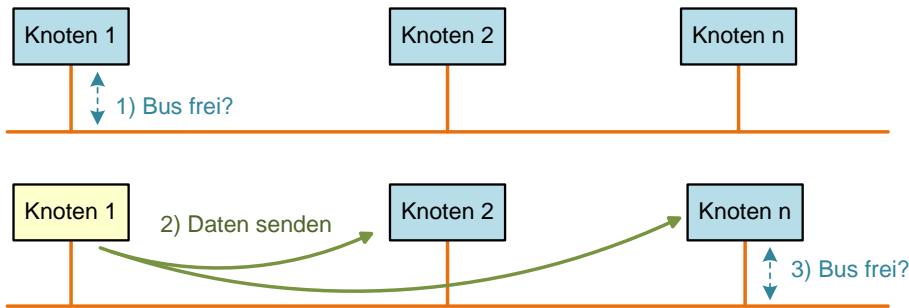


Abbildung 4.29: Carrier Sense Multiple Access

Der Vorteil dieses Verfahrens ist, dass es keinen Master benötigt sowie ohne Konfiguration und Planung auskommt. Nachteilig ist hingegen, dass nicht vorausgesagt werden kann, wann ein Knoten den Bus für einen Sendevorgang bekommt; d.h. das Verfahren ist nicht deterministisch im Gegensatz zu den oben vorgestellten. Das undeterministische Verhalten zeigt sich besonders ausgeprägt, wenn mehrere Knoten auf die Freigabe des Übertragungsmediums warten und nach Abschluss der laufenden Übertragung alle gleichzeitig mit dem Senden beginnen. Für die Behandlung dieser so genannten Kollision gibt es verschiedene Möglichkeiten:

b) Carrier Sense Multiple Access / Collision Detection (CSMA/CD)

Bei CSMA/CD muss jeder beteiligte Knoten eine Kollision feststellen. In diesem Fall bricht er die Übertragung ab und wartet eine zufällige Zeit, bis er erneut einen Sendevorschlag startet. Dieses Verfahren wird im nächsten Kapitel 5 noch ausführlich betrachtet.

c) Carrier Sense Multiple Access with Collision Resolution (CSMA/CR)

CSMA/CR erfordert eine Form von Hardware-unterstützter Arbitrierung; d.h. einer der Signalpegel (1 oder 0) wird als dominant definiert und gewinnt im Konfliktfall. Nur der Verlierer kann eine solche Kollision feststellen und bricht seine Übertragung ab. Der Gewinner fährt mit der Übertragung seiner Daten fort. In Abbildung 4.30 ist der 0-Pegel dominant. Knoten 1 und 2 senden zunächst die gleiche Bitfolge, was ohne Probleme möglich ist. Dann versucht Knoten 2 als sechstes Bit eine 1 zu senden, während Knoten 1 eine 0 sendet. Die 0 ist dominant und somit gewinnt Knoten 1 die Ausmarchierung. Ein Empfänger des Frames (Knoten n) wird von dieser Kollision nichts merken.

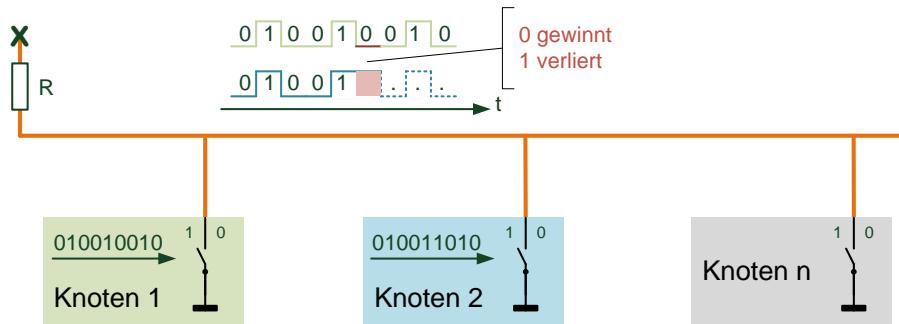


Abbildung 4.30: Collision Resolution

Dieses Verfahren wird beispielsweise beim CAN-Bus und beim ISDN/D-Kanal eingesetzt. Die Arbitrierung kann passiv (wie in Abbildung 4.30) oder aktiv durch einen Busmaster erfolgen.

Der Vorteil dieses Verfahrens liegt darin, dass durch die Kollision keine Übertragungskapazität verloren geht. Nachteilig ist, dass die Arbitrierung innerhalb einer Bitzeit erfolgen muss. Damit ist die maximale Ausdehnung des Netzwerksegments auf ca. eine halbe Bitlänge beschränkt (Ausdehnung eines Bitpulses auf dem Übertragungsmedium).

4.13 Wichtige Layer-2-Protokolle

Folgende Layer-2-Protokolle werden eingesetzt:

- High Level Data Link Control (HDLC)

Von ISO und ITU genormtes Protokoll, das in vielen Anwendungen anzutreffen ist.
- Point to Point Protocol (PPP = HDLC-Variante)

Protokoll zur Anbindung von abgesetzten Teilnehmern an einen Internet Provider.

5

Lokale Netzwerke (Local Area Network)

Ein LAN (Local Area Network) ist ein räumlich begrenztes Netzwerk, in dem Daten mit hoher Geschwindigkeit übertragen werden. Während LANs früher praktisch ausschliesslich in IT-Umgebungen (Server, Workstations, Personalcomputer, Drucker etc.) eingesetzt wurden, sind sie heute auch anderen Anwendungsbereichen wie der Automatisierung, der Telekommunikation und der Unterhaltungselektronik präsent.

5.1 Topologie

Die Netzwerk-Struktur oder Topologie bestimmt, wie die einzelnen Stationen miteinander verbunden werden. Die einzelnen Stationen werden im folgenden Knoten genannt. Netzwerke werden aufgrund ihrer Struktur in Bus-, Linien-, Ring-, Stern- und Baumtopologien eingeteilt. In der Praxis stehen heute Baumtopologien im Vordergrund.

5.1.1 Bustopologie

Bei der Bustopologie sind die Knoten passiv an das Übertragungsmedium angeschlossen; sie werden also nur im Sendefall aktiv. Eine Erweiterung der Busstruktur ist durch die Verbindung einzelner Bussegmente via Verstärker möglich.

5.1.2 Linientopologie

Bei der Linientopologie sind jeweils zwei benachbarte Knoten mit Hilfe des Übertragungsmediums verbunden. Die Linientopologie ist speziell in der Automation sehr beliebt.

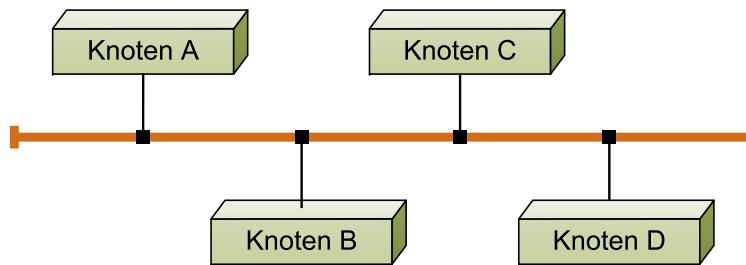


Abbildung 5.1: Netzwerk mit Bustopologie

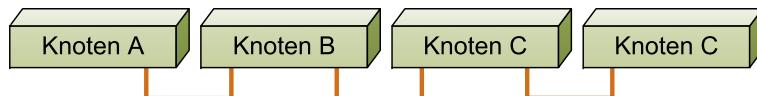


Abbildung 5.2: Netzwerk mit Linientopologie

Typischerweise erfordert die Weiterleitung der Nachrichten in jedem Knoten eine aktive Komponente. Diese empfangen, regenerieren die durchlaufenden Nachrichten und leiten sie weiter. Ohne spezielle Vorkehrungen kann der **Ausfall einer einzelnen Komponente** zu einem **Unterbruch** im Netzwerk führen oder legt im Extremfall das ganze **Netzwerk lahm**.

5.1.3 Ringtopologien

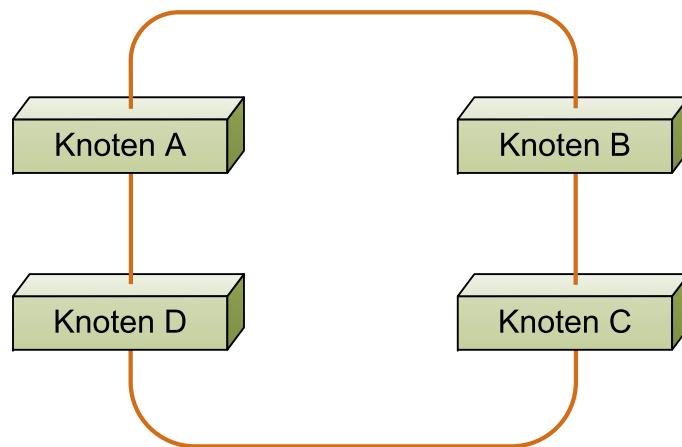


Abbildung 5.3: Netzwerk mit Ringtopologie

Schliesst man bei der Linientopologie die offenen Enden, so führt dies zur Ringtopologie. Da jeder Knoten auf zwei Wegen erreicht werden kann, wird die Ringtopologie gerne für **redundante, ausfallsichere Netze eingesetzt**. Natürlich muss sichergestellt werden, dass Nachrichten nicht endlos im Kreis herum geschickt werden.

Durch die Verwendung einer Doppel-Ringtopologie kann die Ausfallsicherheit gegen Unterbrüche weiter erhöht werden. Durch Umleitung der Nachrichten wird aus dem Doppelring ein einfacher

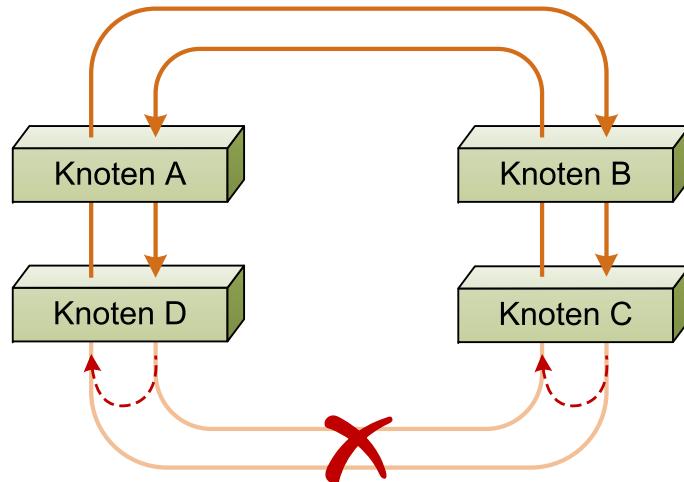


Abbildung 5.4: Netzwerk mit Doppel-Ringtopologie

(gefalteter).

5.1.4 Sterntopologie

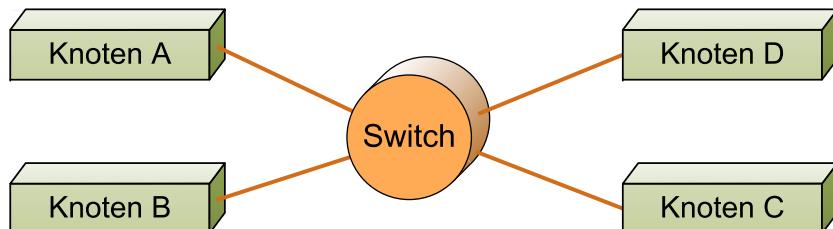


Abbildung 5.5: Netzwerk mit Sterntopologie

Eine Sterntopologie entsteht, indem man das Medium der Bustopologie in einem zentralen Verteiler konzentriert. Dieser Verteiler (der sogenannte **Hub** oder **Switch**) ist für die Weiterleitung der Nachrichten verantwortlich. Meist ist dieser aktiv, entkoppelt die Knoten (z.B. gegen Störungen) und erlaubt üblicherweise eine **flexible Kontrolle der Datenflüsse**. Andererseits legt ein **Ausfall des zentralen Verteilers alle angeschlossenen Knoten lahm.**

5.1.5 Baumtopologie

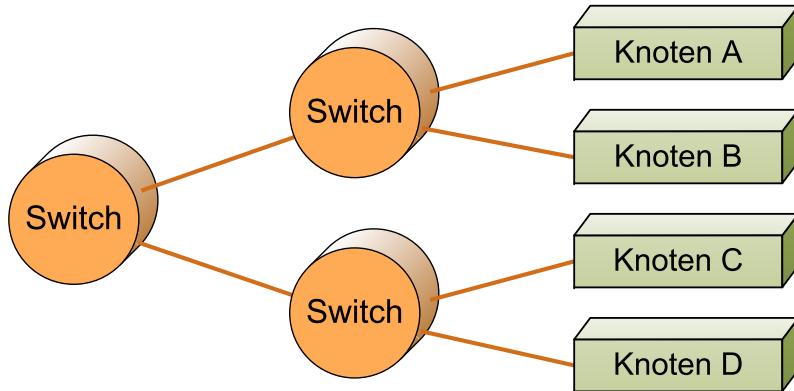


Abbildung 5.6: Netzwerk mit Baumtopologie

Die Baumtopologie entsteht aus einer hierarchischen Kombination der obigen Sternkopologie.

5.2 Übertragungsarten

Die Übertragung von Daten in einem LAN kann in drei Gruppen unterteilt werden: an **einzelne Stationen (Unicast)**, an **eine Gruppe von Stationen (Multicast)** und an **alle Stationen (Broadcast)**. Dabei wird immer nur ein einzelnes Paket betrachtet.

Unicast-Übertragung bezeichnet einen Transfer von einer Quelle zu genau einem Ziel im Netzwerk. Der Sender versieht das Paket mit der Adresse des Empfängers. Im Idealfall übertragen Netzwerke mit Stern- oder Baumtopologie Unicast-Pakete nur genau zum Empfänger. Dies ist bei anderen Topologien (z.B. Bus) im Allgemeinen nicht möglich. Nicht adressierte Knoten werden durch Unicast-Verkehr nicht belastet.

Broadcast-Übertragung bezeichnet einen Transfer an alle Knoten eines Netzwerks. In diesem Fall versieht der Sender das Paket mit einer definierten Broadcast-Adresse. Das Paket wird vom Netzwerk an alle Knoten übertragen und von allen Knoten empfangen. Broadcast-Verkehr belastet dadurch alle Knoten, auch wenn diese die Pakete nicht nutzen.

Multicast-Übertragung bezeichnet einen Transfer an eine bestimmte Gruppe von Knoten im Netzwerk. Der Server versieht das Paket mit der Multicast-Adresse für die Gruppe. Im Idealfall überträgt das Netzwerk die Pakete nur an die Knoten der Gruppe. Das Management der Gruppe erfordert jedoch ein Hilfsprotokoll, mit dem sich Knoten z.B. in einer Gruppe ein- und austragen können. Alternativ werden die Pakete an alle Knoten übertragen (wie bei einem Broadcast) und von den Knoten ausgefiltert.

5.3 Normung für LAN und MAN

Die Entwicklung und Normierung von LANs und MANs wird durch das IEEE¹ dominiert. Das IEEE hat in der Norm 802 eine Reihe von Standards für LAN und MAN aufgestellt. Von den sieben Schichten des OSI-Referenz-Modells wird mit der Normenreihe 802 nur der Physical- und der Data Link Layer, also die untersten zwei Schichten, festgelegt. Wie in Abbildung 5.7 dargestellt, orientieren sich die Standards nicht exakt an den Grenzen der OSI-Layer.

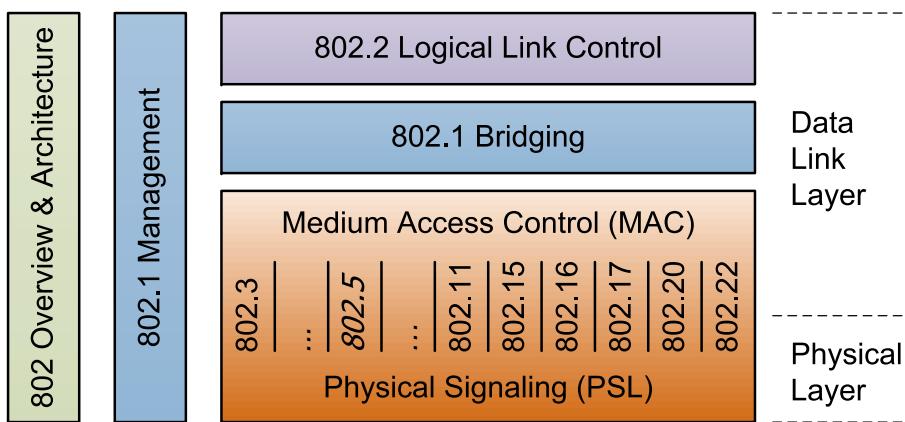


Abbildung 5.7: Übersicht der IEEE 802 Standards

Das Dokument *802: IEEE Standard for Local and Metropolitan Area Networks* gibt eine Übersicht über die LAN- und MAN-Architektur. Beispielsweise stammen die im Abschnitt 4.5 verwendeten Fehlerwahrscheinlichkeiten aus diesem Dokument. Der 802.1 Standard besteht aus einer ganzen Reihe von Substandards (z.B. 802.1D), die verschiedene der LAN/MAN-Komponenten und deren Zusammenspiel beschreiben. Im Abschnitt 6.1 werden wir einige davon betrachten. Der Standard 802.2 (Logical Link Control) gehört zum Layer 2, hat jedoch keine grosse Bedeutung erlangt; ausser von Apple (in AppleTalk) wurde er kaum je umgesetzt.

In den Standards ab IEEE 802.3 sind verschiedene LAN/MAN-Technologien mit den entsprechenden Methoden für Medium Access Control (MAC) und Physical Signaling definiert. Die aktiven und damit wichtigsten IEEE 802-Standards sind:

ANSI/IEEE	Beschreibung
802.3	Ethernet
802.11	Wireless LANs (WiFi)
802.15	Wireless PANs (Bluetooth, ZigBee etc.)
802.16	Broadband Wireless MANs (WiMAX)
802.17	Resilient Packet Rings (MAN)
802.20	Mobile Broadband Wireless Access
802.22	Wireless Regional Area Networks

¹Das IEEE (Institute of Electrical and Electronics Engineers) ist ein weltweiter Ingenieurverband für Elektro- und Informationstechnik. Mit mehr als 400'000 Mitgliedern in über 160 Ländern ist IEEE der grösste technische Berufsverband der Welt, wobei knapp die Hälfte der Mitglieder aus den USA stammt.

Die Standards von IEEE 802 können 6 Monate nach Verabschiedung unter <http://standards.ieee.org/getieee802/> kostenlos eingesehen werden. (Die Drafts sind kostenpflichtig). Nach einiger Zeit werden die IEEE Standards jeweils von ISO/IEC übernommen. Darum existieren unter ähnlichen Namen auch entsprechende ISO/IEC-Normen: IEEE 802.3 entspricht z.B. ISO/IEC 8802-3. Diese sind jedoch kostenpflichtig.

Viele IEEE 802 Standards sind veraltet (z.B. 802.5 / Token Ring). Die neueren aktiven Standards behandeln MAN oder Wireless-Netze. Für drahtgebundene LANs ist nur noch Ethernet also IEEE 802.3 aktiv.

Unter dem Begriff Ethernet wird eine ganze Familie von Netzwerktechnologien mit Bitraten zwischen 10Mbit/s bis 100 Gbit/s zusammengefasst. Alle werden im Standard IEEE 802.3 beschrieben. Jede Variante hat einen dreiteiligen Namen, der ihre Eigenschaften wiedergibt (Abbildung 5.8).

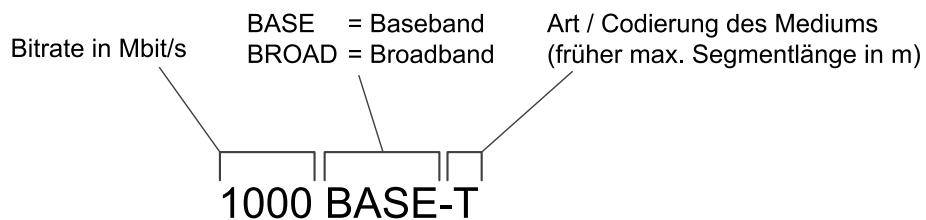


Abbildung 5.8: Namenskonventionen der IEEE-802.3 Protokolle

Die Bezeichnung 1000BASE-T bedeutet „Ethernet mit Basisband-Kanalcodierung und einer Bitrate von 1Gbit/s unter Einsatz von Twisted Pair-Kabeln“. Die ersten Standards gaben anstelle des Mediums die Distanz an (Beispiel 10BASE5: 10Mbit/s Basisband-Ethernet mit max. 500m Segmentlänge).

Die heutigen drahtgebundenen LANs basieren also praktisch alle auf Ethernet. Es mag darum erstaunen, dass viele Experten Ethernet anfänglich als nicht-skalierbare Technologie ohne Zukunft bezeichnet haben. Immer wieder wurden andere Technologien als besserer Ersatz angepriesen. Trotzdem hat einzig Ethernet überlebt und sich in fast allen Bereichen durchgesetzt. Eine Stärke von Ethernet ist sicherlich, dass es flexibel, relativ leicht zu implementieren und zu verstehen ist. Der Haupterfolg beruht jedoch darauf, dass es gelungen ist, durch technologische Neuerungen immer wieder neue Leistungsklassen zu erschliessen.

Nach Einführung von Ethernet wurden zunächst einige Varianten entwickelt, die auf eine Verbilligung der Verkabelung und eine Vereinfachung der Installation abzielten. Später stand die Steigerung der Bitrate im Vordergrund. Einzelne der in Abbildung 5.9 gezeigten Technologien werden später noch ausgeführt (siehe Abschnitt 6.2).

In Abbildung 5.9 fällt auf, dass praktisch alle Verfahren das Basisband nutzen. In der gesamten Entwicklung gab es nur eine Breitbandvariante 10BROAD36, deren Ziel es war, die Infrastruktur des Kabelfernsehens (CATV) nutzen zu können (10 Mbit/s über 3600 m). Die Technologie hatte sich jedoch auf dem Markt nicht etablieren können.

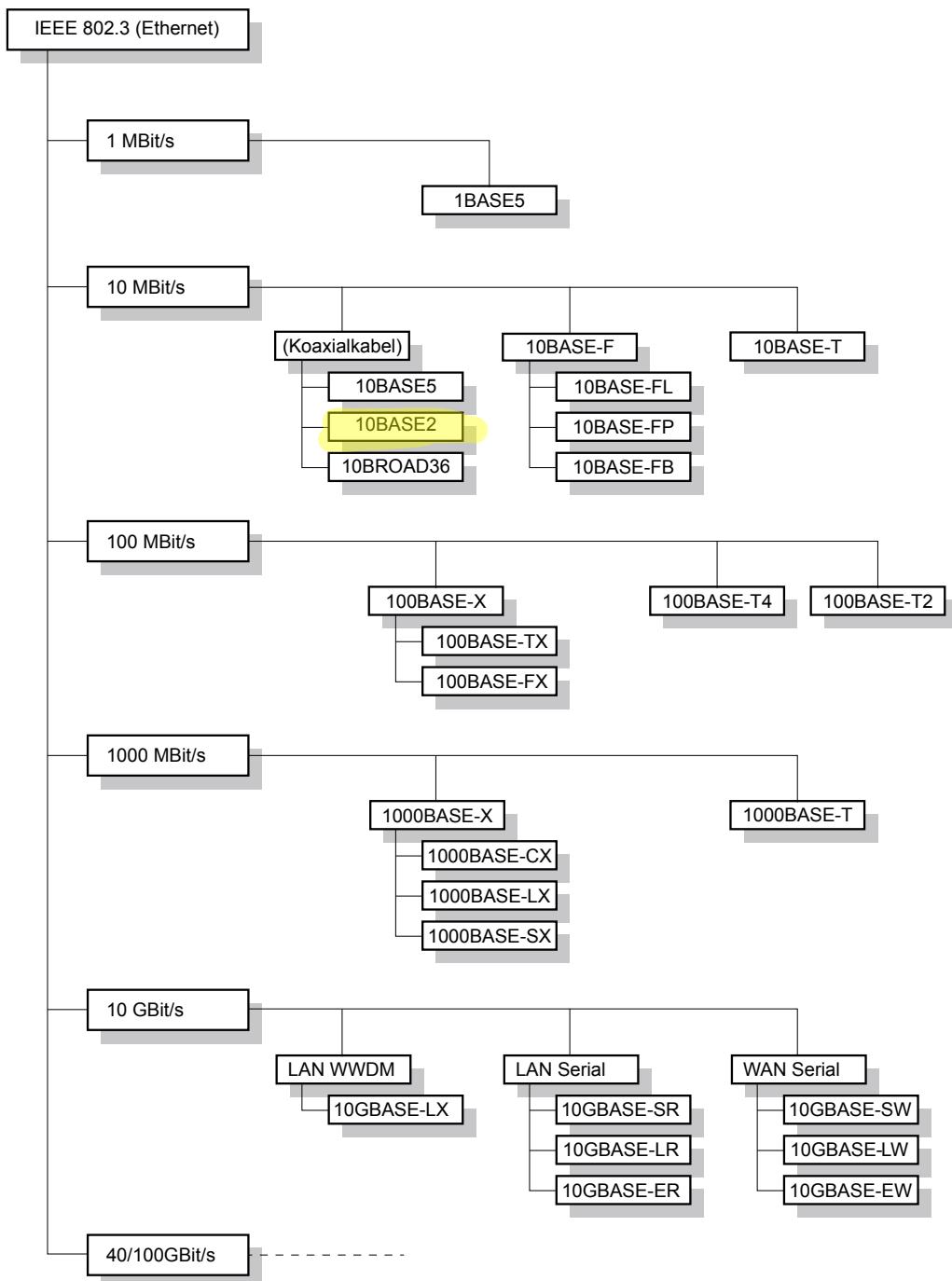


Abbildung 5.9: Übersicht der Ethernet-Technologien

5.4 Shared Ethernet

In diesem Abschnitt beschränken wir uns zunächst auf Ethernet mit Bus-Topologie, da diese dem ursprünglichen Prinzip entspricht.

5.4.1 Entstehungsgeschichte

Als der Informatikstudent Robert Metcalfe 1972 bei einem Kollegen übernachtete, konnte er auf dem unbequemen Wohnzimmersofa nicht einschlafen. So las der 24 jährige in einer herumliegenden Fachzeitschrift einen Beitrag der Universität Hawaii über ein paketvermitteltes Funknetz mit einer Übertragungsrate von 4.8 Kbit/s. Monate später entwarf er in seiner Abschlussarbeit am MIT ein verbessertes Verfahren mit Namen **Carrier Sense Multiple Access with Collision Detection (CSMA/CD)**.

In einem Forschungszentrum von Xerox bekam Metcalfe später den Auftrag, mehrere PCs zu vernetzen und entwickelte mit seinem Kollegen David Boggsim Kommunikationscontroller, mit denen knapp 3 Mbit/s übertragen werden konnten. Später tauften sie ihre Erfindung „Ethernet“, in Anlehnung an den „Lichtäther“, von dem man im 19. Jahrhundert annahm, dass sich elektromagnetische Strahlen darin fortpflanzten. 1976 wurde Ethernet erstmals einer breiten Öffentlichkeit vorgestellt (Abbildung 5.10).

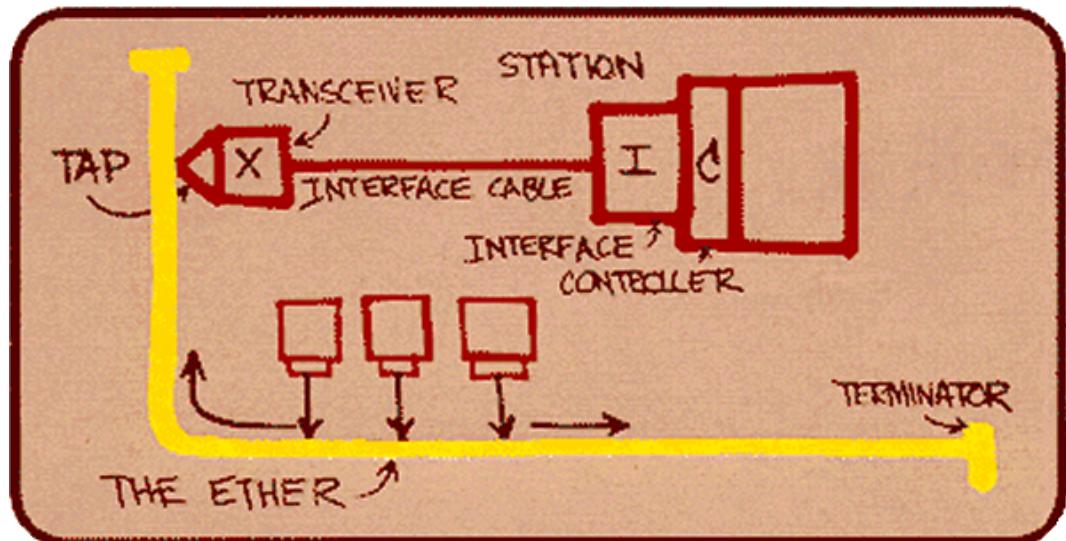


Abbildung 5.10: Skizze von Robert Metcalfe für die Präsentation im Jahre 1976

Es dauerte drei weitere Jahre, bis die Firmen Xerox, Intel und Digital Equipment Corporation das System gemeinsam weiterentwickelten und eine Übertragungsgeschwindigkeit von 10 Mbit/s festlegten. 1982 wurden die vorgeschlagenen Spezifikationen fast unverändert übernommen und zum Standard IEEE 802.3 erklärt.

5.4.2 Physical Layer 10BASE5 und 10BASE2

Beim ersten Ethernet wurde ein spezielles Koaxialkabel² eingesetzt. Wie Abbildung 5.11 zeigt, konnte ein sogenanntes Segment maximal 500 m lang sein und bis zu 100 Knoten umfassen.

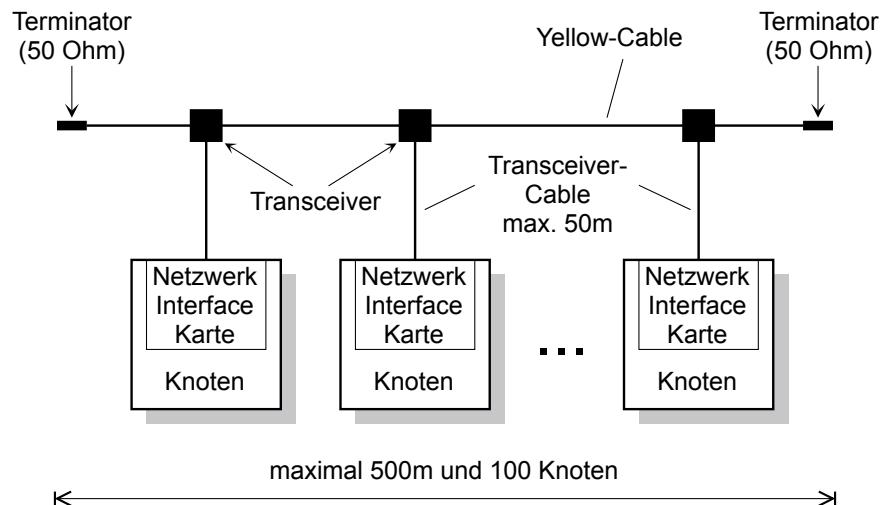


Abbildung 5.11: 10BASE5 Netzwerk-Konfiguration

Die Besonderheit war, dass das Kabel zum Anschliessen von Geräten angebohrt werden musste. Dazu wurde ein dickes Kabel (Durchmesser ca. 10 mm) mit einem festen Innenleiter (2 mm Drahtdurchmesser) benötigt. Die Technologie wurde darum zu Recht auch Thick-Wire-Ethernet-Kabel genannt.

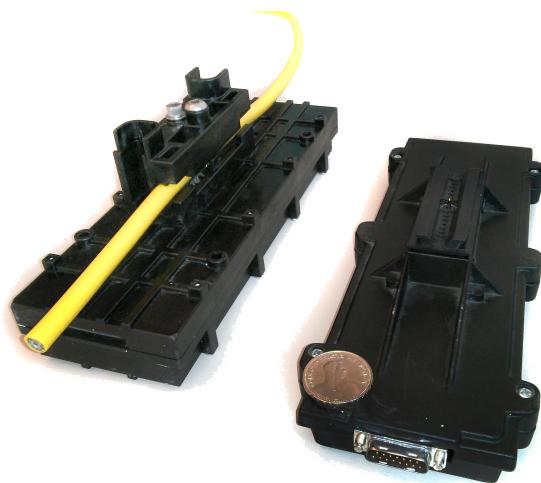


Abbildung 5.12: Transceiver mit Thick-Wire-Ethernet-Kabel

Wie Abbildung 5.12 zeigt, wurde der sogenannte **Transceiver** (Transmitter/Receiver) direkt auf das Koaxialkabel montiert. Von jedem Transceiver führt ein Verbindungskabel (Transceiver-Cable) zu einem Knoten.

²Dieses wurde als „Yellow Cable“ bezeichnet, da es üblicherweise mit einem gelben Mantel geliefert wurde.

Später wurde das sogenannte **Thin-Wire-Ethernet (10BASE2)** als flexiblere und vor allem billigere Alternative eingeführt. Es verwendet die normalen Koaxialkabel (Typ RG58) mit einem Durchmesser von ca. 5mm, um die Knoten direkt miteinander zu verbinden. Wie Abbildung 5.13 zeigt, werden die Geräte über T-Stücke mit den üblichen BNC-Steckern angeschlossen und die Leitung mit BNC-Terminatoren abgeschlossen.

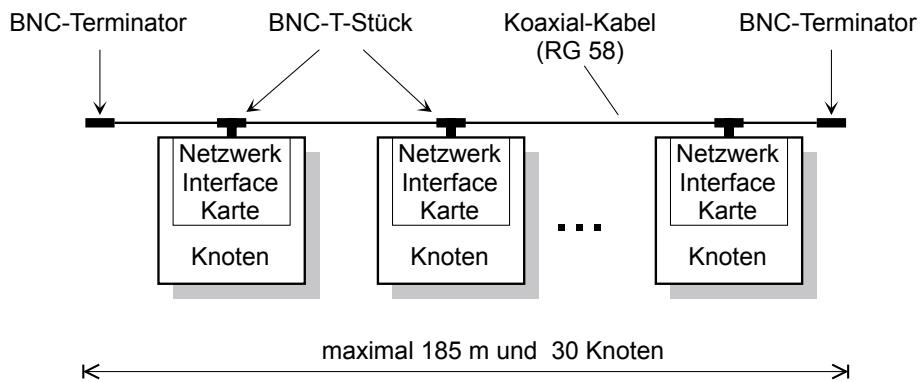


Abbildung 5.13: 10BASE2 Netzwerk-Konfiguration

In der Praxis hatte sich 10BASE2 nicht bewährt. Da die BNC-Stecker sehr empfindlich sind, kam es immer wieder zu mechanischen Problemen, was zu schwer lokalisierbaren (sporadischen) Fehlern im Netz führte.

5.4.3 Leitungscodierung

Dauer des Interpacket Gaps

Die Übertragung bei 10BASE2 und 10BASE5 erfolgt bekanntlich mit einer Bitrate von 10 Mbit/s im Basisband. Das bedeutet, dass keine Trägerfrequenz verwendet wird und alle 100 ns ein Symbol übertragen wird. Als **Leitungscode** wird ein **Manchester-Code** eingesetzt. Wie Abbildung 5.14 zeigt, wird dabei eine **1 als positive** und eine **0 als negative Flanke** übertragen.

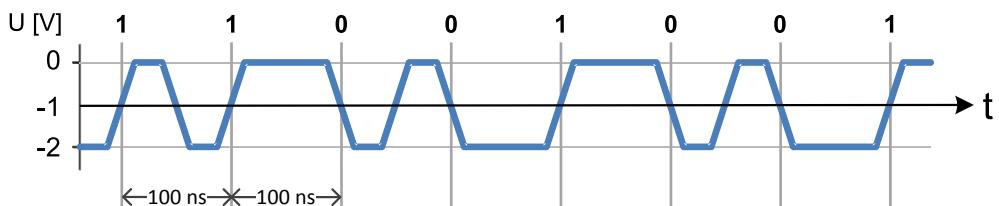


Abbildung 5.14: Übertragungssignal einer manchester-codierten Bitfolge (10BASE2)

Der Manchester-Code hat den Vorteil, dass er auf einfache Weise die Taktrückgewinnung erlaubt. Allerdings wird für **10 Mbit/s eine Bandbreite von 10 MHz benötigt**; also das **Doppelte vom theoretischen Minimum**. 10BASE2 verwendet **asymmetrische Signalpegel von 0V und -2V**. Mit symmetrischen Pegeln, wie sie bei 10BASE-T (Abschnitt 6.1.2) verwendet werden, ist der Manchester-Code gleichstromfrei.

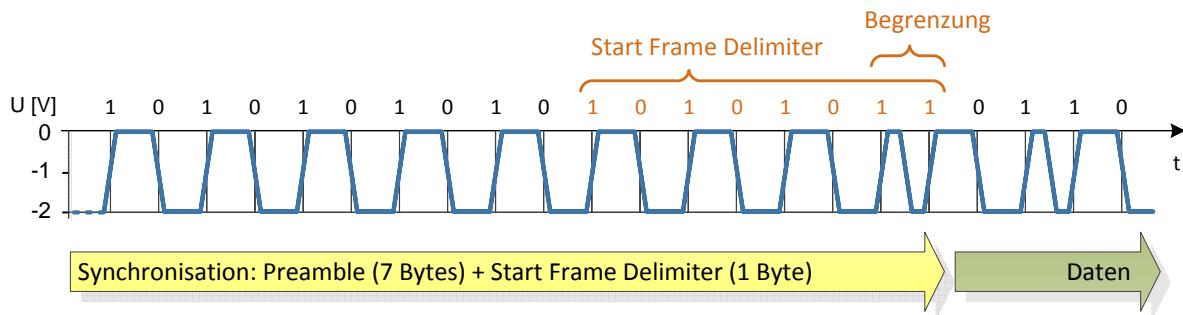


Abbildung 5.15: Bitsynchronisation mit Hilfe von Preamble und Start Frame Delimiter

Der Code ist bei reinen 0- oder 1-Folgen nicht eindeutig, weshalb ein Verfahren zur Bitsynchronisation erforderlich ist. Die Synchronisation des Empfängers erfolgt durch eine Bitfolge von 0 und 1. Wie Abbildung 5.15 zeigt, sind bei dieser Bitfolge alle Flanken des manchester-codierten Signals an gültigen Bitpositionen. Das Ende der Synchronisationssequenz wird durch eine 1-1-Bitfolge angezeigt. Bei Ethernet werden 8 Bytes für die Synchronisation verwendet, wobei die ersten 7 Bytes als Preamble und das letzte Byte als Start Frame Delimiter bezeichnet wird.

5.4.4 Datenübertragung und Zugriffskontrolle

Ethernet verwendete das CSMA/CD Zugriffsverfahren. Ein wichtiger Vorteil dieser Technik besteht darin, dass keine zentralen Kontrollelemente für die Übertragung via Ethernet-Kabel notwendig sind, sondern dass vielmehr jeder Knoten autonom ist. Im Folgenden wird das Verfahren genauer beschrieben.

a) Senden (ohne Kollision)

Bevor ein Knoten mit dem Senden eines Frames beginnt, wartet er so lange, bis der Übertragungskanal frei ist. Erst dann wird das Frame an den Physical-Layer übergeben. Bevor der Physical-Layer das Frame überträgt, sendet er die Preamble und den Start Frame Delimiter (siehe Abschnitt 5.4.3), damit die Empfänger sich synchronisieren können.

Der Physical-Layer kontrolliert während der gesamten Sendezeit den Signalpegel auf dem Koaxialkabel. Findet eine Kollision statt - d.h. versucht ein anderer Knoten gleichzeitig zu senden - so überlagern sich die Signale U_1 und U_2 der beiden Knoten (Abbildung 5.16). Der resultierende höhere Signalpegel U_D wird erkannt und vom Physical-Layer an den Datalink-Layer (Collision Detected Signal) gemeldet.

Der Datalink-Layer meldet die Beendigung einer kollisionsfreien Übertragung den darüber liegenden Schichten und erwartet die Daten für das nächste Frame.

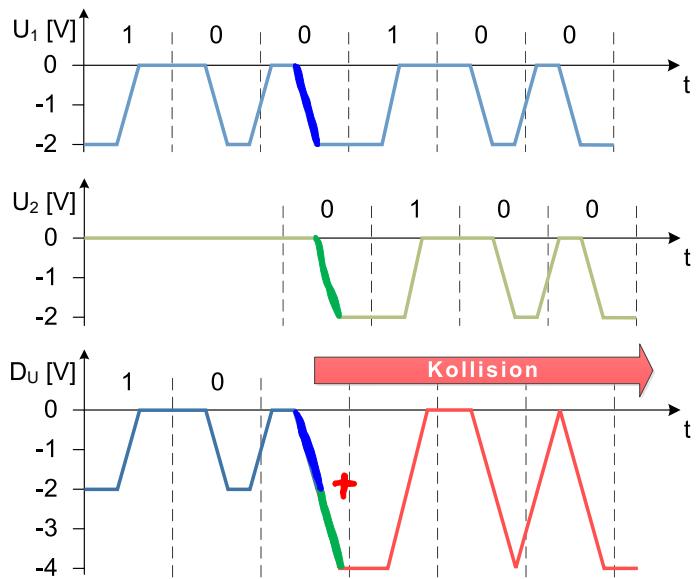


Abbildung 5.16: Überlagerung der Sendesignale bei einer Kollision

b) Empfang (ohne Kollision)

Sobald ein Knoten zu senden beginnt, stellen die Physical-Layer aller anderen Knoten dies fest und melden das an ihre Datalink-Layer, damit diese ihrerseits keine Sendeversuche unternehmen. Jetzt werden alle Knoten des Ethernets zu Empfangsknoten, synchronisieren ihre Empfangseinrichtungen auf die oben erwähnte Präambel und übersetzen die anschliessend empfangene Zieladresse vom Manchester- in den Binärcode. Die Zieladresse wird vom Physical-Layer an den Datalink-Layer übermittelt. Beim adressierten Knoten wird nun der Rest des Frames vom Physical- zum Datalink-Layer übermittelt und die Daten von dort weiter zu den darüber liegenden Schichten. Die durch die Zieladresse nicht angesprochenen Knoten übergeben dem Datalink-Layer keine weiteren Bytes.

c) Senden mit Kollision

Es ist möglich, dass zwei (oder mehr) Knoten feststellen, dass der Übertragungskanal frei ist und ungefähr gleichzeitig zu senden beginnen. Eine solche Kollision kann also offensichtlich nur zu Beginn einer Übertragung auftreten. Eine Kollision meldet der Physical-Layer dem Datalink-Layer mittels eines speziellen Signals (collision detect signal).

Um sicher zu gehen, dass alle Knoten die stattgefundene Kollision registriert haben, wird ein Jam-Signal (32 Bit mit beliebigem Inhalt) ausgegeben. Daraufhin wird die Sendung eingestellt. Der Datalink-Layer der Empfänger ist in der Lage, ein gültiges Frame von einem durch Kollision verstümmelten zu unterscheiden, weil die Prüfsequenz am Ende des Frames falsch ist.

Eine neuerliche Übertragung wird wieder nach einer gewissen, variablen Zeit versucht. Diese Zeitspanne ist pseudo-zufällig und wird durch einen Binary Exponential Backoff genannten Algorithmus bestimmt. Die Wartezeit nach einer Kollision beträgt immer ein Vielfaches der sogenannten Slot-Zeit t_S , die 512 Bitzeiten entspricht (10 Mbit/s Ethernet $t_S = 51.2\mu\text{s}$; 100 Mbit/s Ethernet

$t_S = 5.12\mu s$). Bei der ersten Kollision wird zufällig $0 * t_S$ oder $1 * t_S$ gewartet. Bei jeder direkt folgenden Kollision wird der Zeitbereich verdoppelt. Bei der zweiten Kollision gibt es also vier Möglichkeiten $0, 1, 2$ oder $3 * t_S$; bei der dritten Kollision 8 etc. Nach insgesamt 16 erfolglosen Versuchen wird die Übertragung abgebrochen.

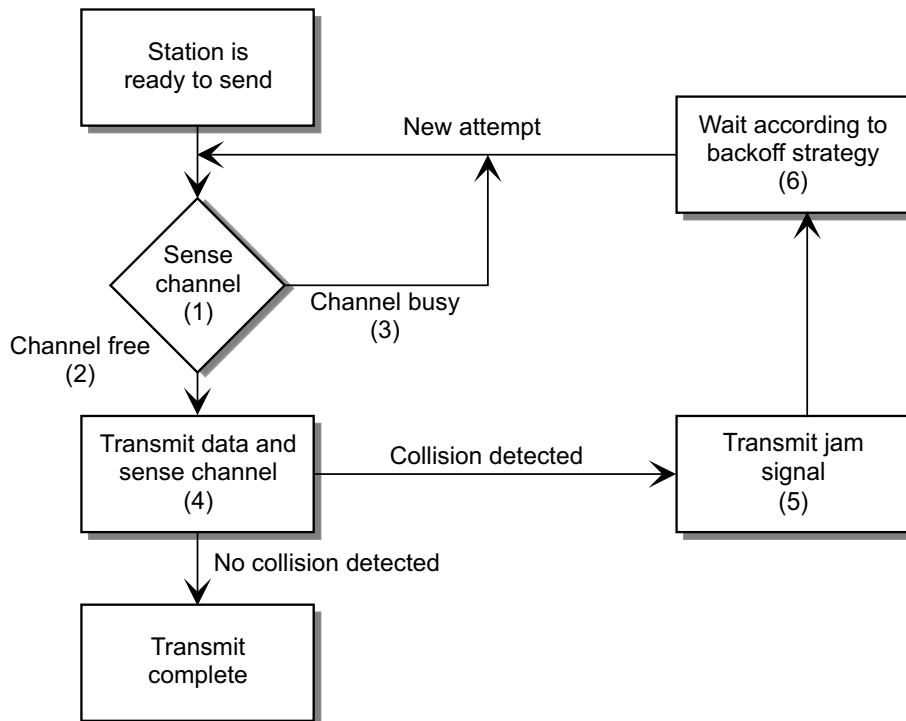


Abbildung 5.17: Ablauf des CSMA/CD-Verfahrens

Das Flussdiagramm in Abbildung 5.17 illustriert das Verfahren nochmals. Das CSMA/CD Zu-griffsverfahren wird vollständig vom MAC-Layer ausgeführt. In der Praxis ist es in Hardware implementiert; d.h. die CPU wird durch Kollisionen und Retransmissions nicht belastet.

d) Durchsatzbetrachtungen von CSMA/CD

Bei grossem Verkehrsaufkommen nimmt die Wahrscheinlichkeit von Kollisionen zu, was zu einer Reduktion der effektiven Verkehrsleistung eines Ethernet-LANs führt.

Zu CSMA/CD und anderen Verfahren wurden in den Anfangszeiten umfangreiche theoretische Analysen erstellt. Einige dieser frühen Arbeiten zeigten, dass die Antwortzeiten bei einer mittleren Netzbelastung von $> 50\%$ dramatisch ansteigen und praktisch zu einem Netzzusammenbruch führen. Diese Untersuchungen waren ein Hauptargument für andere Verfahren, wie Token-Passing.

Spätere Untersuchungen mit echten Daten zeigen jedoch, dass die theoretischen Studien praktisch alle auf falschen Grundannahmen beruhen (Poisson-Verteilung des Verkehrs).

5.5 Aufbau des Ethernet-Frames

Der in Abbildung 5.18 gezeigte Aufbau des Ethernet-Frames ist fast das einzige, das vom ursprünglichen Ethernet die Jahre und Technologiewechsel überdauert hat.

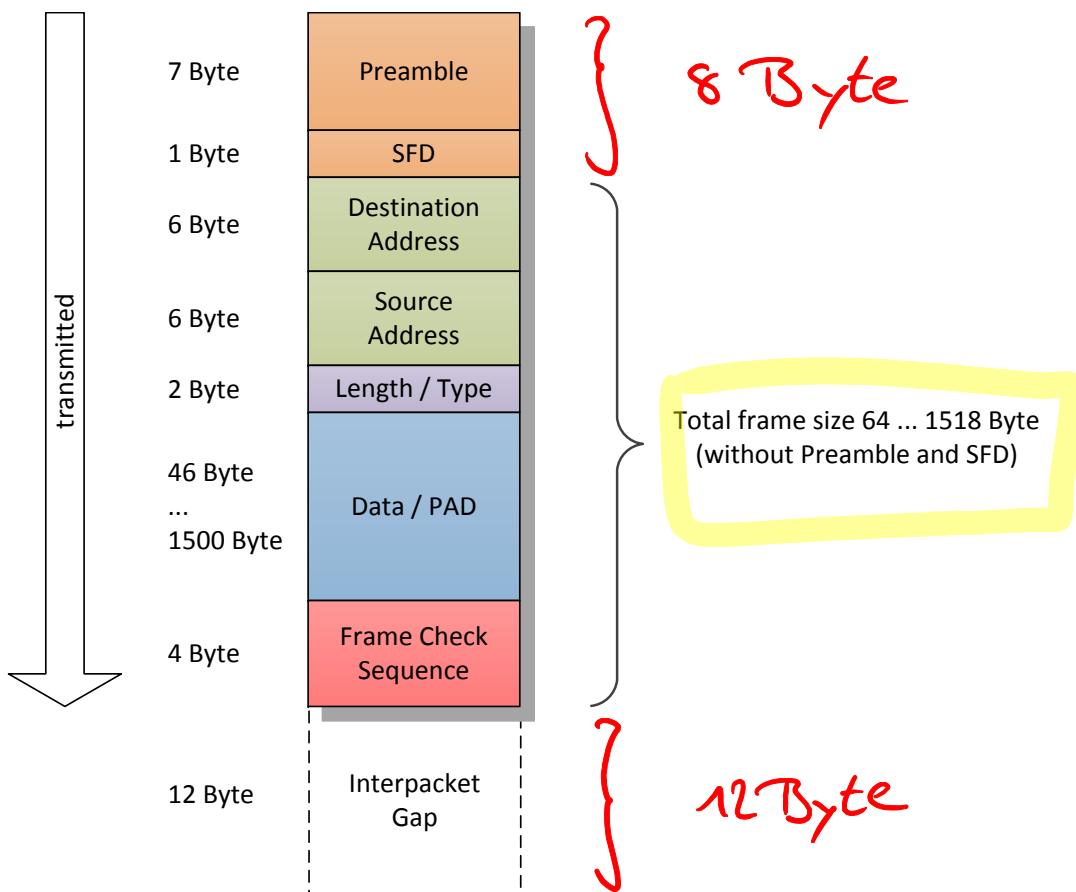


Abbildung 5.18: Felder des Ethernet-Frames

Von den einzelnen Bytes wird immer das **niederwertigste Bit (LSB)** zuerst, das **höchstwertigste Bit (MSB)** zuletzt übertragen.

Preamble und Start Frame Delimiter (SFD) gehören zum Physical Layer. Wie im Abschnitt 5.4.3 gezeigt, dienen sie gemeinsam der **Synchronisation des Empfängers**.

Die Preamble ist ein 7 Byte-Feld mit dem Bitmuster (10101010...). Dieses ist so gewählt, dass für jedes Bit eine Flanke im manchester-codierten Signal entsteht. Der Start Frame Delimiter (SFD) signalisiert mit einer 11-Bitfolge das Ende der Synchronisation und den Anfang eines Frames.

Destination Address bezeichnet die Adresse des Ziel- bzw. Empfangsknotens. Jedes Ethernet-Gerät hat eine weltweit eindeutige 6-Byte-Adresse. Sie wird üblicherweise als sechs zweistellige hexadezimale Zahlen dargestellt (siehe Abbildung 5.19). (Die Bindestriche dienen nur der besseren Lesbarkeit.) Die hexadezimalen Ziffernpaare repräsentieren die binären

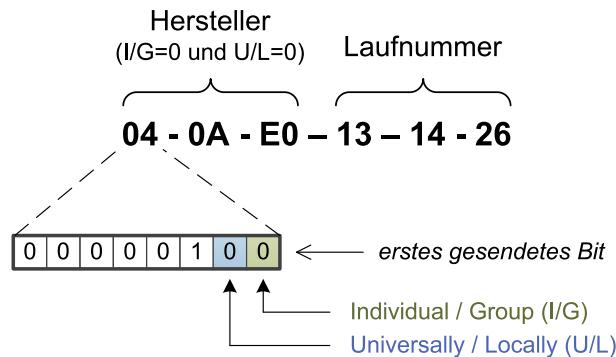


Abbildung 5.19: Aufbau einer ANSI/IEEE-Adresse

Werte der Adressbytes. Die Reihenfolge (links nach rechts) der hexadezimalen Ziffernpaare entspricht der beim Übertragen.

Die ersten drei Bytes (links in der hexadezimalen Darstellung) werden als **Organizationally Unique Identifier (OUI)** bezeichnet und sind üblicherweise Hersteller-spezifisch. Der OUI wird durch das IEEE (im Namen der ISO) an Firmen vergeben, die Ethernet-Geräte (z.B. LAN-Controller-Karten für PC) produzieren (Kosten pro OUI 1250\$). Die Hersteller vergeben dann die restlichen 24 Bits einmalig an die von ihnen produzierten Geräte. Jede solche **Universally Administered Address** sollte weltweit nur einmal vorkommen.

Anhand einer MAC-Adresse lässt sich aufgrund des Herstellercodes feststellen, um was für ein Controller-Produkt es sich handelt. Im RFC-1700 (oder aktueller unter <http://standards.ieee.org/regauth/oui/oui.txt>) können die meisten Herstellercodes eingesehen werden.

Beispiele von OUI sind:

00-00-0C	Cisco
00-00-0E	Fujitsu
00-00-AA	Xerox
00-15-12	Zurich University of Applied Sciences
08-00-09	Hewlett-Packard
08-00-11	Tektronix, Inc.
08-00-20	Sun
08-00-46	Sony
08-00-5A	IBM

Ausnahmen: Daneben gibt es Multicast- oder Broadcast-Adressen (z.B. FF-FF-FF-FF-FF-FF). Dies sind Gruppenadressen, die der Adressierung mehrerer oder sogar aller Empfangsknoten dienen. Das tiefstwertigste Bit (I/G) des ersten Adressbytes zeigt eine solche Gruppenadresse an.

I/G = 0 → Individual Address

I/G = 1 → Group Address

Weiter existieren noch lokal administrierte Adressen. Diese werden durch den Netzwerkadministrator definiert und durch die Software gesetzt.

Das zweite Bit (U/L) unterscheidet, ob es sich um eine global (Universal) administrierte oder lokal (Locally) zugewiesene Adresse handelt.

U/L = 0 → Universally Administrated Address

U/L = 1 → Locally Administrated Address

Knoten mit einer Locally Administrated Address haben in der Regel zwei: eine global und eine lokal administrierte Adresse.

Source Address enthält die Adresse des Quell- resp. Sendeknotens. Der Aufbau ist analog dem der Destination Address. Das I/G-Bit der Adresse *muss* hier 0 sein (Individual Address).

Length/Type besteht aus zwei Bytes, wovon das höherwertige Byte zuerst übertragen wird. Es hat in Abhängigkeit von seinem Wert zwei Bedeutungen, wobei heute praktisch ausschliesslich die Type-Bedeutung eingesetzt wird:

Length: Ein Wert ≤ 1500 gibt die effektive Anzahl von Bytes im folgenden Datenfeld „Data“ an. Effektive Anzahl meint ohne allfällige Fülldaten (siehe PAD-Feld).

Type: Ein Wert ≥ 1536 gibt an, was für ein höheres Protokoll im Datenfeld enthalten ist.

Data / Pad bezeichnet das Datenfeld. Es enthält 0 bis 1500 Datenbytes (Nutzinformation).

Falls die Anzahl der Bytes im Data-Feld kleiner als 46 ist, wird mit Nullbytes auf Bytes aufgefüllt (Padding). Dadurch wird eine minimale Frame-Länge von 64 Bytes garantiert.

Frame Check Sequence (FCS) dient zur Feststellung von Übertragungsfehler bei der Übermittlung eines Frames. Es wird ein **CRC32-Algorithmus** mit dem folgenden Generatorpolynom angewandt (siehe Abschnitt 4.9):

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Interpacket Gap (IPG) ist kein Feld des Frames, sondern bezeichnet den minimalen zeitlichen Abstand zwischen zwei sich folgenden Frames. Er beträgt mindestens 96 Bit-Zeiten und gehört zum Framing des Physical Layers.

5.6 LAN-Erweiterungen mit Repeater

Die Länge der Ethernet Bus-Segmente ist im Wesentlichen durch die Dämpfung limitiert. Durch bidirektionale Signalverstärker (**Repeater**) können Segmente gekoppelt werden. Repeater arbeiten auf dem Physical Layer des OSI-Modells (siehe Abbildung 5.20). Neben der Amplitude werden auch die Synchronisationsbits (Preamble), die Flanken und kleinere zeitliche Verschiebungen (Jitter) restauriert. Sobald ein Repeater an einem seiner Ports ein Signal empfängt, schickt er es mit möglichst kleiner Zeitverzögerung (einige Bitzeiten) ungesiehten weiter. d.h. Repeater beachten keine Frame-Inhalte wie Adressen oder Prüfsequenzen.

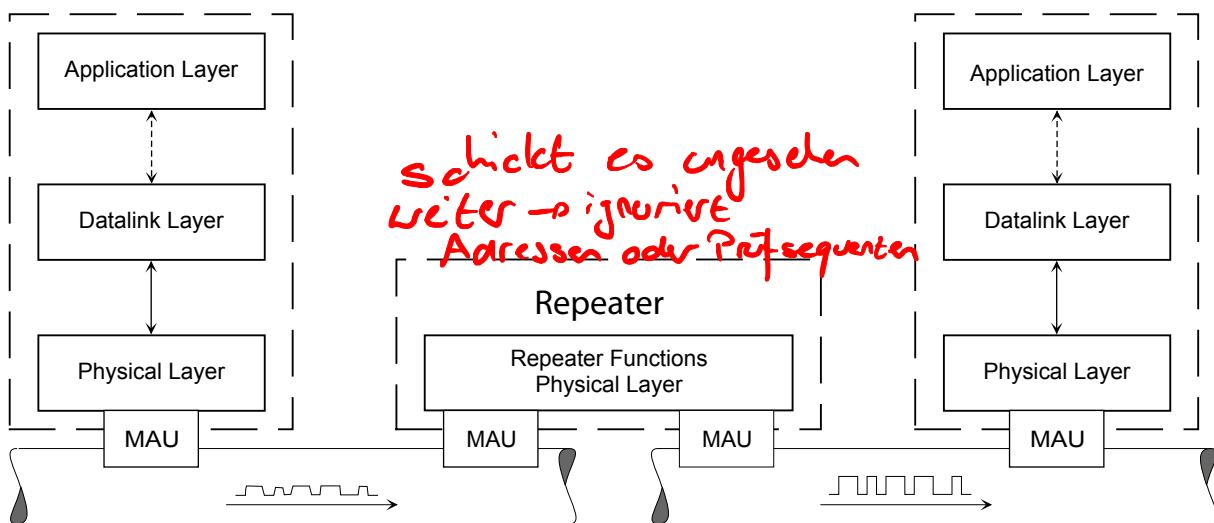


Abbildung 5.20: Repeater arbeiten auf dem Physical Layer des OSI-Modells

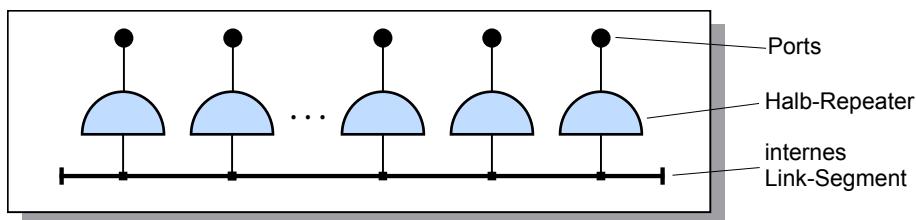


Abbildung 5.21: Ersatzschaltbild für einen Multiport-Repeater

Multiport Repeater mit typischerweise 8, 16 oder 32 Ports werden als **Hub** (Mittelpunkt, Radnabe) bezeichnet. Wie Abbildung 5.21 zeigt, verhält sich ein Hub wie ein „Ethernet in the Box“. Alle Ports sind gleichberechtigt.

Hubs können mehrere Netzsegmente koppeln und erlauben den Aufbau von Stern- und Baumtopologien. Neben der Signalverstärkung haben sie noch weitere Aufgaben:

- Erkennt ein Hub auf einem seiner Ports eine **Kollision**, gibt er auf **allen Segmenten** ein **Jam-Signal** (mindestens 32 Bits) aus.
- Wenn auf einem der Ports **länger als 5 ms ohne Unterbruch** gesendet wird, muss der Hub das **Signal unterdrücken**. Diese Funktion wird als **Jabber Suppression** bezeichnet (Geplapperunterdrückung).

5.6.1 Konfigurationsregeln für Repeater

Es gibt Begrenzungen bezüglich der Anzahl von Repeatern und Kabelsegmenten, die zwischen zwei beliebigen Knoten eines Netzwerks liegen können. Zwei Größen bestimmen die maximale Anzahl der Repeater:

Round Trip Delay: Ein beliebiger sendernder Knoten muss eine Kollision erkennen können, so lange er noch am Senden ist.

Interpacket Gap Shrinkage: Der Interpacket Gap (IPG) bezeichnet den zeitlichen Abstand zwischen zwei sich folgenden Frames. Der **minimale Interpacket Gap** ist 96 Bit-Zeiten. Beim Durchlaufen eines Repeaters werden verlorene Bits der Preamble regeneriert. Dadurch wird die Preamble länger und der Interpacket Gap kleiner.

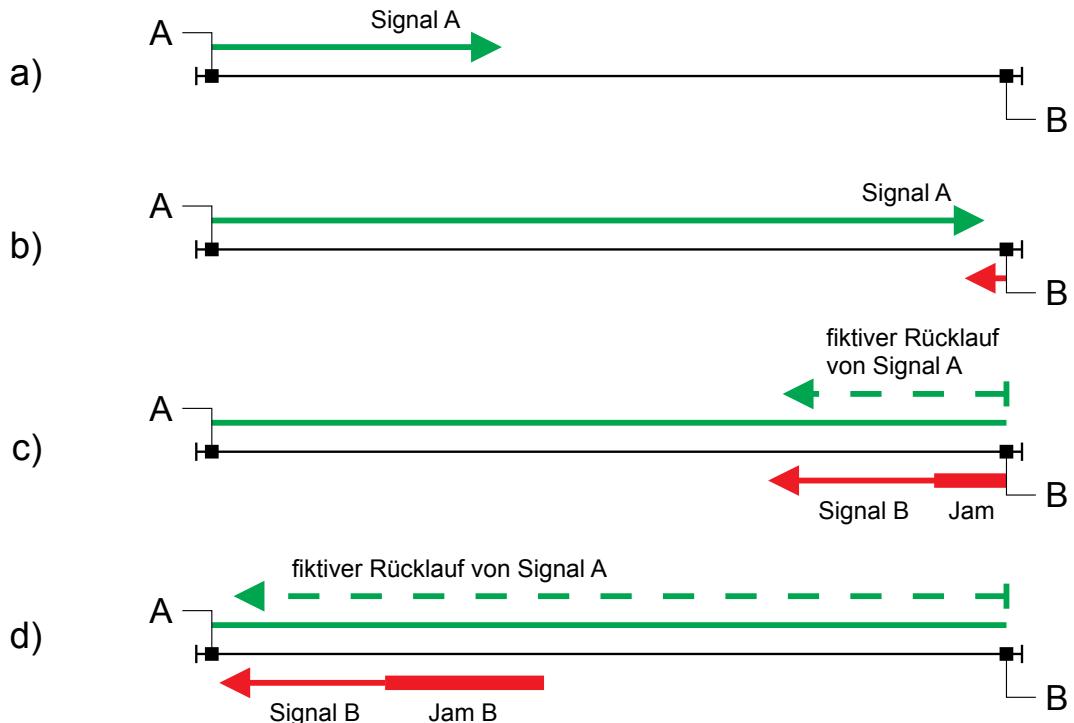


Abbildung 5.22: Worst-Case-Betrachtung für die Kollisionserkennung

In den weitaus meisten Konfigurationen sind die Limiten durch den Round Trip Delay gegeben. Die Ursache für die Begrenzung ist im CSMA/CD-Protokoll zu finden. Jeder Knoten, der sendet, muss noch während er am Senden ist, feststellen, dass eine Kollision stattgefunden hat. Der Bereich des Netzwerks, in dem eine Kollision erkannt werden kann, heisst **Collision Domain**. Der zu betrachtende schlimmste Fall ist in Abbildung 5.22 gezeigt. Dabei sind zwei Knoten A und B an den entgegengesetzten Enden eines Segments angeschlossen.

- Der Knoten A beginnt zu senden.
- Der Knoten B beginnt im letzten Moment zu senden, wo er noch darf. Das ist kurz bevor das Signal vom Knoten A eintrifft.
- Kurz darauf wird der Knoten B eine Kollision feststellen und das JAM-Signal senden.
- Das Signal vom Knoten B muss nun beim Knoten A eintreffen, solange dieser noch am Senden ist.

Das heisst, dass eine **Collision Domain** nur *halb so gross* sein darf, wie die **Ausdehnung des kürzesten Frames**. Dabei sind die verschiedenen Verzögerungszeiten zu berücksichtigen. Falls das

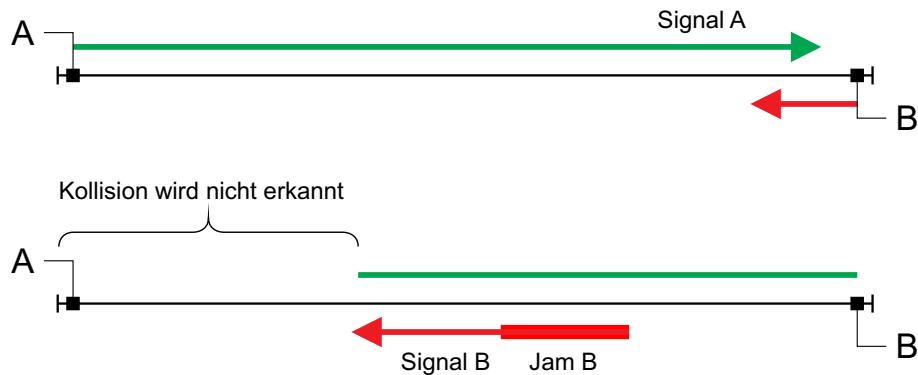


Abbildung 5.23: Das kollidierende Signal vom Knoten B trifft zu spät beim Knoten A ein.

Netz grösser als die doppelte Ausdehnung des Signals ist, wird der Knoten A (und solche in seiner Nähe) die Kollision des Signals A *nicht* bemerken, was bedeutet, dass das Frame nicht nochmals versandt wird. Man spricht dann von einer **Late Collision** (siehe Abbildung 5.23). Vom Anwender werden Late Collisions nur indirekt bemerkt, da die Transport Layer Protokolle fehlende Frames nochmals übertragen. Allerdings leidet die Performance und die Prozessoren werden zusätzlich belastet.

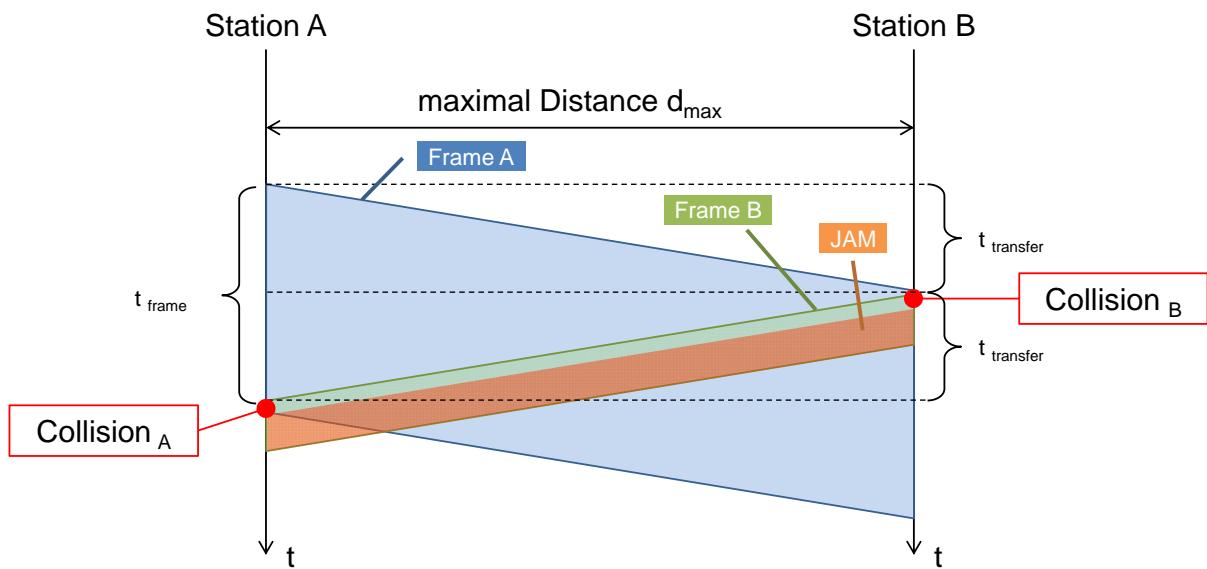


Abbildung 5.24: Maximale Ausdehnung eines Segments

Abbildung 5.24 zeigt den selben Sachverhalt in einem Weg/Zeit-Diagramm.

Die Bedingung für die **Erkennung einer Kollision** durch Knoten A ist:

$$t_{frame} > 2 * t_{transfer} \quad (5.1)$$

Will man die **maximale Ausdehnung eines Segments** bestimmen, so setzt man

$t_{frame} = \frac{\text{Framesize}_{min}}{\text{Bitrate}}$ sowie $t_{transfer} = \frac{d_{max}}{C_{Medium}}$ und löst nach d_{max} auf:

$$d_{max} < \frac{1}{2} \frac{\text{Framesize}_{min}}{\text{Bitrate}} C_{Medium} \quad (5.2)$$

Abbildung 5.25 zeigt ein Weg/Zeit-Diagramm mit einem Repeater. Wie man sieht, addiert sich die Weiterleitungszeit $t_{forwarding}$ des Repeaters zur Übertragungszeit $t_{transfer}$. Damit Knoten A die Kollision erkennen kann, muss nun gelten:

$$t_{frame} > 2 \left(\sum t_{transfer} + \sum t_{forwarding} \right) \quad (5.3)$$

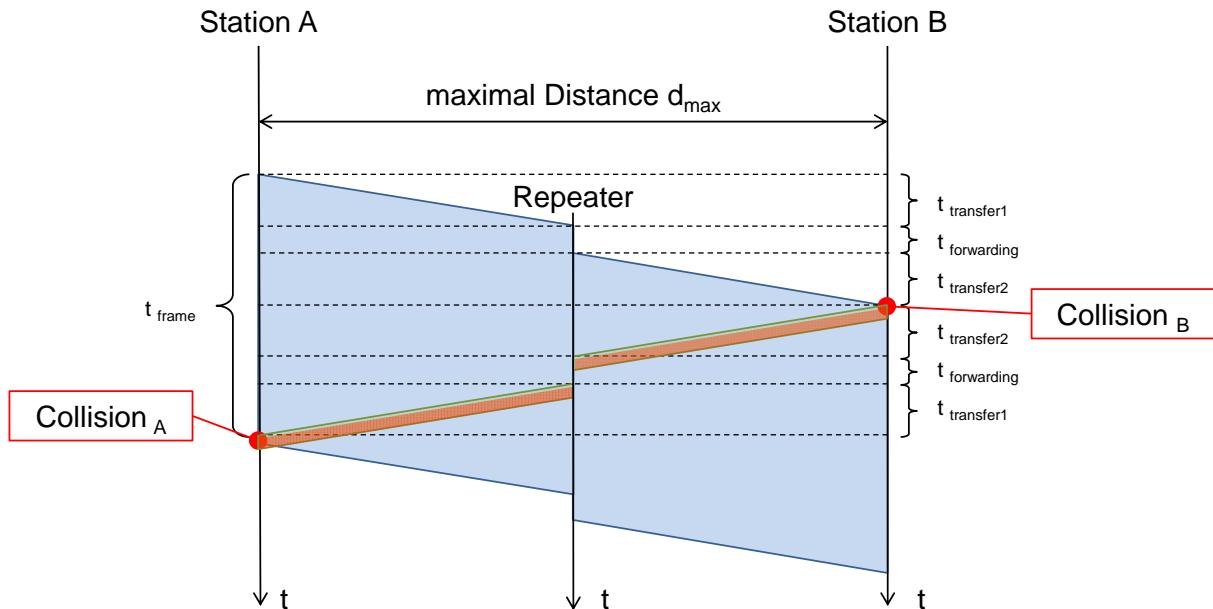


Abbildung 5.25: Maximale Ausdehnung eines Segments mit einem Repeater

5.7 Übungen

mögliche Kriterien: Kapazität, Ausfallsicherheit, (Einfachheit Verkabelung)

- a) Man nenne je einen *typischen* Vorteil der folgenden Topologien und gebe an, welche heute allgemein eingesetzt wird.
- Bus-Struktur **Thick-Ethernet: 10Base2; Thin Ethernet 10Base5**
Vorteil: einfach & kostengünstig Nachteil: Kapazitätbegrenzung -> SharedMedium
 - Baum-Struktur
Vorteil: Kapazität, kann sich gut verteilen Nachteil: von einem zum anderen Knoten muss jenachdem über einen root kehren; Single point of failure
 - Ring-Struktur
Vorteil: Ausfallsicherheit Nachteil: Komplexität / Logik
- b) Man nenne je zwei *typische* Nach- oder Vorteile von CSMA/CD.
Vorteil: -keine zentrale Logik / Kontroller -Verfahren bei Collision ist geregt
Nachteil: -Fairness nicht gewährleistet -nicht Echtzeitfähig
- c) Das Flussdiagramm in Abbildung 5.26 soll das CSMA/CD-Verfahren beschreiben. Man ergänze die Pfade (Pfeile) und gebe die Bedingungen an.

Zugriffverfahren MAC:

- TDM (Time Division Management =
Zeitschlitzverfahren)

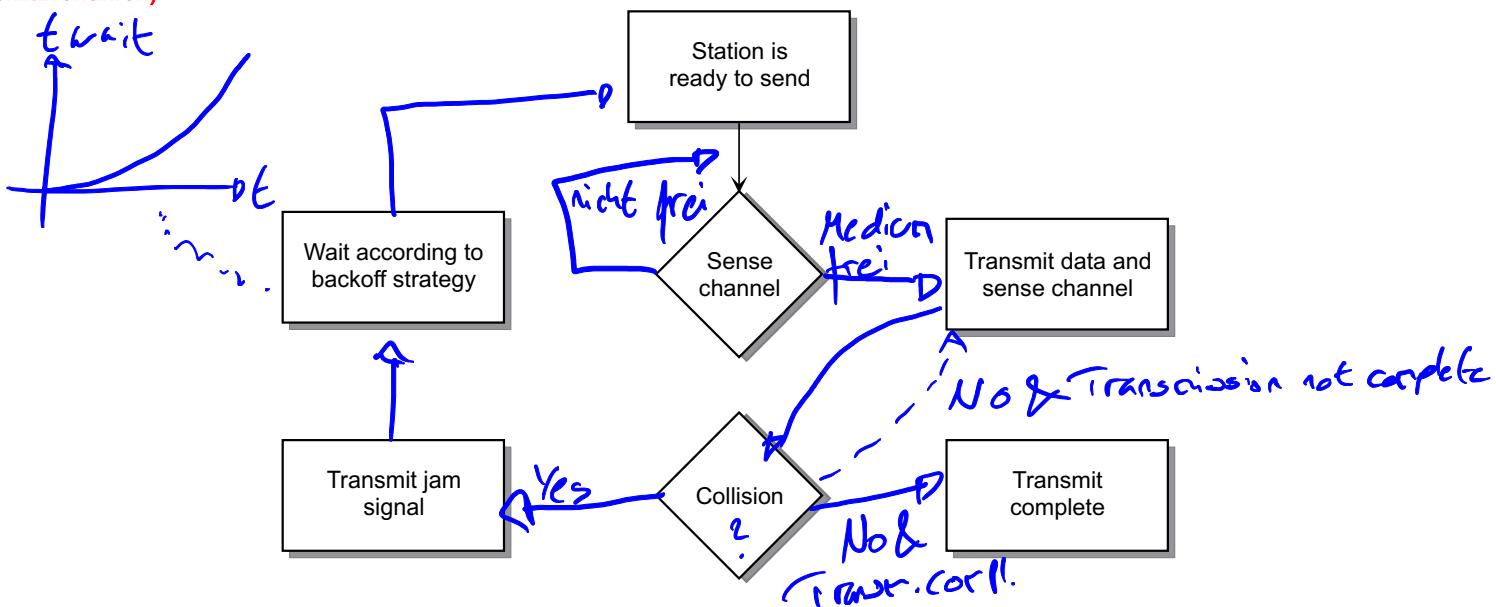


Abbildung 5.26: Unvollständiges Flussdiagramm

- d) Abbildung 5.27 zeigt ein Ethernet-Signal (0.5μs/Division). Eingezeichnet sind die letzten 4 Bits des Start-Frame-Delimiter.
- Man ergänze die Zeichnung, so dass anschliessend die Bits 0–0–1–0 folgen.
 - Man zeichne links eine mögliche Signalform bei einer Kollision ein.
 - Wie nennt man diese Codierung? *Manchester*

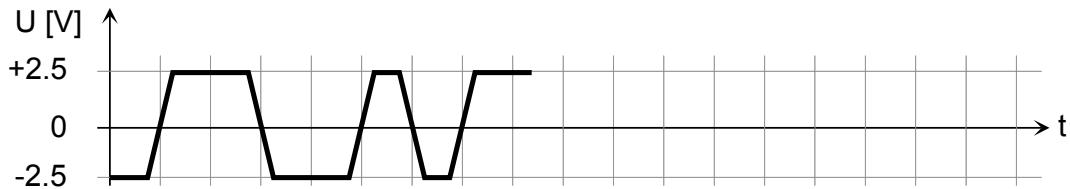


Abbildung 5.27: Die letzten 4 Bits des Start-Frame-Delimiters

Manchester-Codierung

- Was ist ihr Vorteil (= Was erlaubt sie?)

Taktrückgewinnung

- e) Das zuerst gesendete Bit (I/G) einer Ethernet „Destination Address“ hat die Bedeutung „Individual Address/Group Address“.

- Wie wird eine „Group Address“ (oder das zugehörige Verfahren) üblicherweise genannt?

multicast

- Warum hat das Bit I/G in der „Source Address“ keine Bedeutung?

Da es Unicast ist

- Man gebe in hexadezimaler Darstellung eine typische „Group Address“ an.

01:00:0c:cc:cc:cc

- Was für ein Bit der Ethernet-Adresse hat sonst noch eine spezielle Funktion? (Welche?)

- f) Mit Hilfe des LAN/Ethernet-Analyzers stellt man fest, dass ein Gerät dauernd fehlerhafte Frames sendet. Man weiss jedoch nicht, welches Gerät dies sein könnte.

- Woran lässt sich erkennen, dass ein 802.3-Frame fehlerhaft ist (2 Antworten)?

- CRC

- Typefield (sollte IPv4 sein)

- Anhand von welcher (MAC-)Information lässt sich die Suche einschränken?

Source

- Wie könnte man konkret vorgehen, um den Übeltäter (in einem grossen Netz) zu finden?

als erstes im Wireshark nachschauen oder Port für Port abschalten oder LearningTable betrachten

6

Switched LAN und Ethernet-Technologien

In diesem Kapitel werden LAN-Erweiterungen mit den benötigten Mechanismen erläutert, sowie moderne Ethernet-Technologien eingeführt, die sich speziell für **Baum-Topologien** eignen. In der Praxis müssen Netzwerk-Segmente fast immer erweitert werden. Mit Repeatern oder Hubs lassen sich zwar mehrere Segmente zusammenschalten, um jedoch die Beschränkungen des CSMA/CD-Verfahrens zu durchbrechen, werden **Bridges** und **Switches** benötigt.

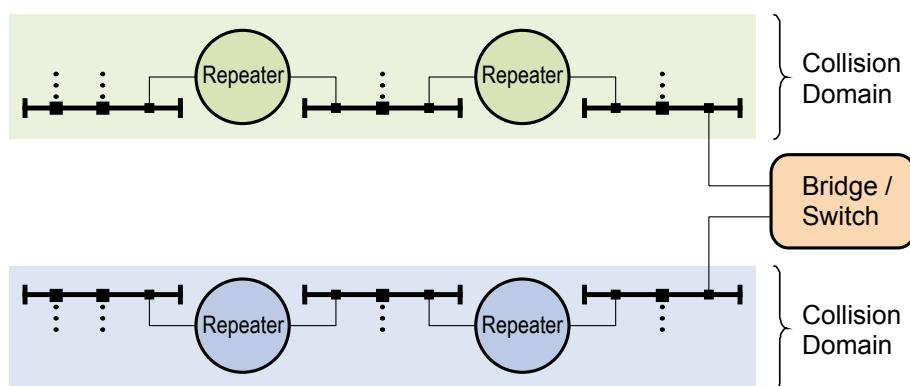


Abbildung 6.1: Bridge (ent)koppelt Collision Domains

6.1 Bridges

Die über **Repeaters** (siehe Abschnitt 5.6) zusammengeschlossenen Ethernet-Segmente bilden eine so genannte **Collision Domain** (Abbildung 6.1). Innerhalb einer Collision Domain muss jede Collision von allen angeschlossenen Knoten erkannt werden.

Eine Bridge erlaubt eine Collision Domain aufzuteilen (Abbildung 6.1). Oder anders betrachtet: Eine Bridge koppelt mehrere Collision Domain miteinander, ohne dass diese sich in Bezug auf das CSMA/CD-Verfahren beeinflussen. Jede Collision Domain arbeitet also grundsätzlich unabhängig von den anderen und in jeder gelten weiterhin die durch die Kollisionserkennung gegebenen Limiten.

Ein Switch ist ein Marketing-Ausdruck für eine Bridge mit mehreren Ports. Wir verwenden im Folgenden den Begriff Bridge, weil dies der Standardterminologie entspricht.

6.1.1 Funktionsweise

a) Transparent Bridging

Eine Bridge arbeitet auf dem Data Link Layer (Abbildung 6.2). Sie koppelt die angeschlossenen Segmente, indem sie die an einen Port empfangenen Frames prüft und basierend auf der Zieladresse an andere Ports weiterleitet. Die grundsätzlichen Bridge-Funktionen sind beschrieben in *IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges*.

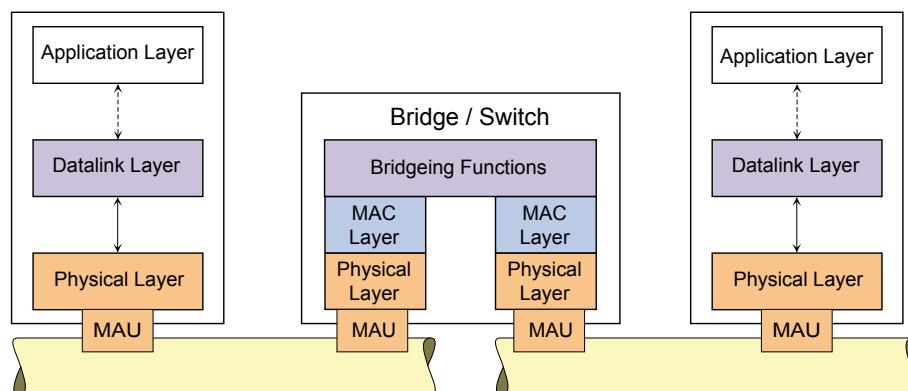


Abbildung 6.2: Arbeiten auf dem Datalink Layer

Sende- und Empfangsknoten sollen nicht merken, dass sich eine Bridge dazwischen befindet; d.h. am Inhalt der Frames darf sich nichts ändern. Da Bridges aus Sicht des Netzwerks unsichtbar sind, wird ihre Funktion als Transparent Bridging bezeichnet.

Anders als normale Knoten, die nur an sie adressierte Frames akzeptieren, nehmen die Bridge-Ports sämtliche Frames entgegen. Sie arbeiten im so genannten **Promiscuous Mode**¹. Obwohl alle Bridge-Ports eigene Mac-Adressen besitzen, erfolgt die Weiterleitung mit der Adresse des ursprünglichen Absenders.

¹Grundsätzlich lässt sich jedes Ethernet Interface in den Promiscuous Mode bringen. Diese Funktion wird beispielsweise von Wireshark für die Netzwerkanalyse benutzt.

b) Address Learning

Das Management und der Einsatz von Bridges soll möglichst einfach sein. Bridges funktionieren ohne vorherige Konfiguration und ohne Änderung an den Einstellungen von Netzwerknoten. Dazu verfügen die Bridges über einen Mechanismus zum Erlernen von Adressen. Eine Bridge hört den Verkehr von allen Ports ab und merkt sich die Sender-Adressen aus den empfangenen Frames in der sogenannten **Forwarding Database** (Abbildung 6.3). Diese beinhaltet im Wesentlichen für jede bekannte Mac-Adresse das Bridge-Port, über welches der zugehörige Knoten erreichbar ist. Unbenutzte Einträge in der Forwarding Database werden nach einer gewissen Zeit² automatisch gelöscht.

c) Weiterleitungs- und Filter-Funktion

Um das Netzwerk nicht unnötig zu belasten, sollen Bridges den Verkehr möglichst gezielt verteilen. Die Weiterleitung basiert auf der Forwarding Database. Dafür durchsucht die Bridge für jedes empfangene Frame die Forwarding Database nach dessen Zieladresse und leitet es an das in der Forwarding Database registrierte Port weiter.

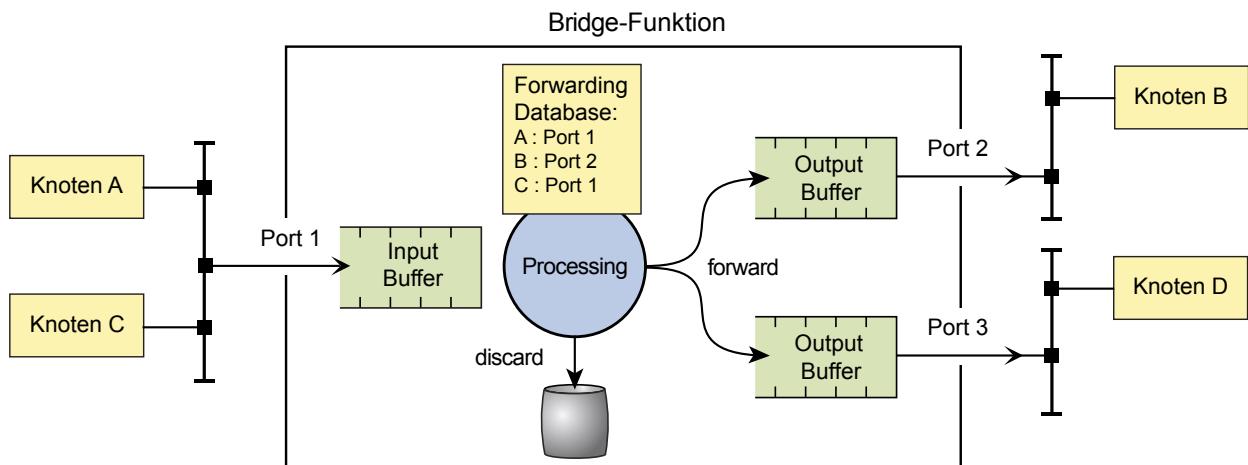


Abbildung 6.3: Bridge-Funktion (nur eine Richtung)

Abbildung 6.3 zeigt die Funktion einer Bridge, wobei nur eine Transferrichtung dargestellt ist. Sendet beispielsweise Knoten A im links gezeichneten LAN ein Frame an den Knoten B, so wird dieses von der Bridge von Port 1 empfangen und im Input-Buffer abgelegt. Falls das Frame (z.B. wegen einer Kollision) unvollständig oder fehlerhaft empfangen wurde, so wird es gelöscht. Bei korrekten Frames wird die Destination-Adresse in der Forwarding Database gesucht und so das Ausgabe-Port ermittelt. Im Fall von Knoten B wäre dies Port 2. Das Frame würde in den Output-Buffer von Port 2 übertragen (forward) und gesendet.

Hat der adressierte Empfänger (wie Knoten C) die **selbe Port-Nummer wie der Absender (Knoten A)**, so wird das **Frame nicht** in den Output-Buffer übertragen und gelöscht (discarded).

²Diese Zeit ist bei managed Switches einstellbar. Die Standardeinstellung beträgt 600 s.

Wird die Destination-Adresse in der Forwarding Database (z.B. Knoten D) nicht gefunden, so wird das Frame auf *allen* Ports ausgegeben (**Frame Flooding**), ausser dem Eingangs-Port. Zum Glück sind die meisten höheren Protokolle so aufgebaut, dass Empfänger Bestätigungen schicken. Damit kann die Bridge die Source-Adresse in die Forwarding Database aufnehmen und das nächste Frame bereits gezielt weiterleiten.

Eine **Spezialität** sind die **Broadcast-Frames**. Diese sind per Definition an **alle Knoten** gerichtet und dürfen von der Bridge nicht aus gefiltert werden. Alle mit Bridges gekoppelten LAN-Segmente bezeichnet man als **Broadcast-Domain** und meint damit den Bereich, in dem Broadcast-Frames verteilt werden. Auch Broadcast-Frames werden von den Bridges am Eingangs-Port *nicht* ausgegeben, um Frame-Duplikate zu vermeiden. Moderne Bridges unterstützen auch Multicast-Verkehr. Solche Frames werden nur an bestimmten Ports ausgegeben. Die Festlegung der Multicast-Ports erfolgt über spezielle Protokolle oder durch manuelle Konfiguration.

d) Anwendung zur Verkehrstrennung

Da eine **Bridge** einen Teil der Frames ausfiltert, **kann sie die Last in Teilnetzen reduzieren**.

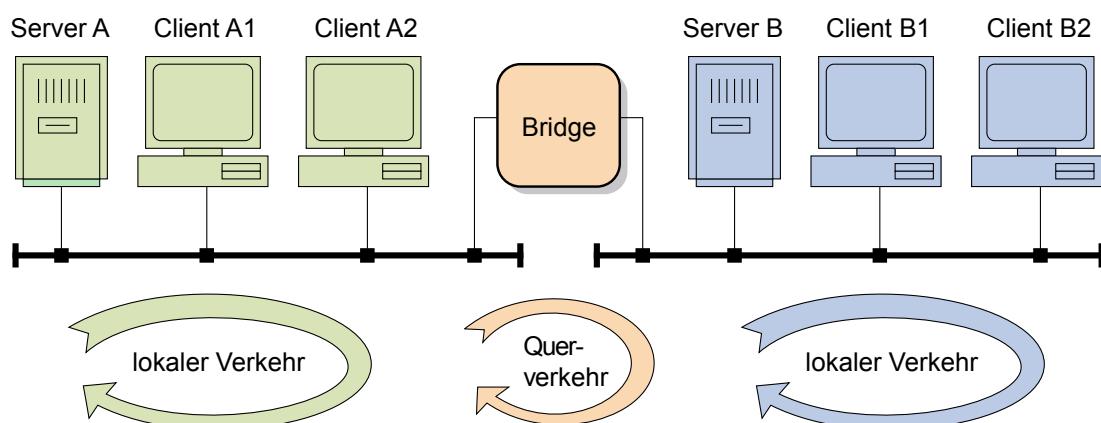


Abbildung 6.4: Ist der Hauptverkehr lokal, kann eine Bridge eine Lastreduktion bewirken

Werden beispielsweise in Abbildung 6.4 auf dem linken LAN vom Server A zum Client A1 Frames übertragen, so erscheinen diese nicht auf dem rechten LAN; es könnte also *gleichzeitig* ein Datentransfer (z.B. vom Client B1 zum Server B) stattfinden.

Eine Bridge dient also nicht nur zur Erweiterung eines bestehenden LAN, sondern auch zur Trennung des Verkehrs und zur Vergrösserung der Sicherheit gegenüber Totalausfällen. Dies gilt allerdings *nur*, wenn der Verkehr zum grössten Teil lokal ist, das heisst, innerhalb der Collision-Domain bleibt. Falls der grösste Teil des Verkehrs über die Bridge geht (wie in Abbildung 6.5), wird der Verkehr nicht aufgeteilt. Wegen der durch die Bridge entstehenden Verzögerungen kann der Durchsatz sogar abnehmen.

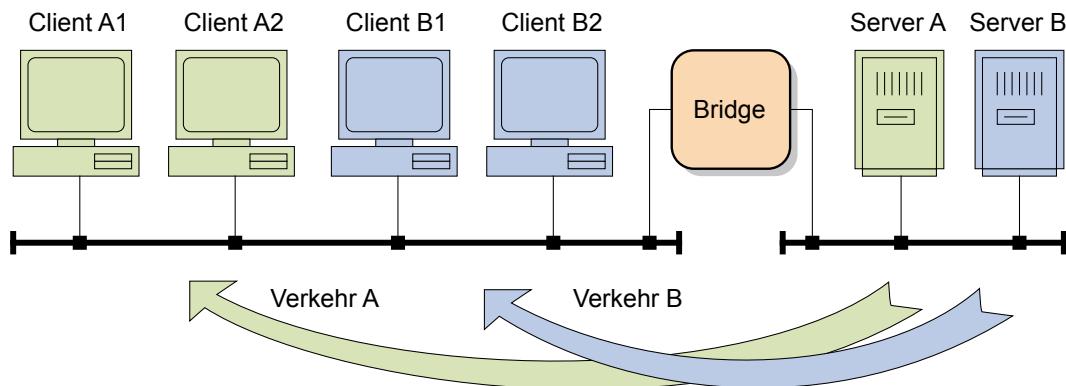


Abbildung 6.5: Ungünstige Anordnung einer Bridge

6.1.2 Vollduplex Betrieb

Mit **Vollduplex** kann der **Durchsatz verdoppelt** werden (zumindest theoretisch), da **gleichzeitig gesendet und empfangen** werden kann. Dies bedingt, dass zum Senden und Empfangen getrennte oder bidirektionale Übertragungskanäle existieren, wie dies heute mit Twisted Pair-Verkabelungen praktisch immer der Fall ist (siehe Abschnitt 6.2.1).



Vollduplex auf einer Punkt-Punkt-Verbindung erlaubt über die durch die Kollisionserkennung limitierte Distanz hinauszugehen. In einem vollständig vollduplex-fähigen geswitchten Netz (ohne Repeater) treten *keine* Kollisionen mehr auf, da jede Punkt-Punkt-Verbindung ein unabhängiges LAN mit genau einem Sender und einem Empfänger pro Verbindung darstellt. Man spricht dann von einem **mikrosegmentierten LAN**. Anstelle von Kollisionen und Wiederholungen müssen die Frames in den Switches so lange zwischengespeichert werden, bis das benötigte Ausgabe-Port frei ist. Das CSMA/CD-Verfahren, das Ethernet ursprünglich ausgezeichnet hat, kommt bei einem mikrosegmentierten LAN nicht mehr vor. Es erfolgt kein Carrier Sense, gibt keinen Multiple Access, damit keine Kollisionen und also auch keine Collision Detection.

6.1.3 Remote- und Multiport-Bridges

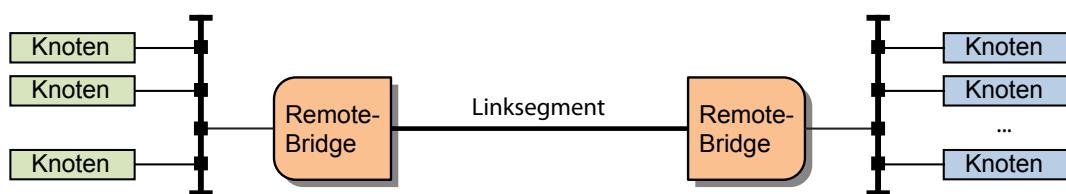


Abbildung 6.6: Prinzip einer Remote-Bridge

Remote Bridges erlauben die Verbindung zweier LAN über grössere Strecken (Abbildung 6.6). Sie werden paarweise eingesetzt, wobei sie je über einen normalen LAN-Port und einen speziellen Anschluss (z.B. für Glasfaserstrecken, optische Links etc.) verfügen. Bei Vollduplexbetrieb ist die maximale Distanz nur durch die Eigenschaften der Übertragungsstrecke beschränkt.

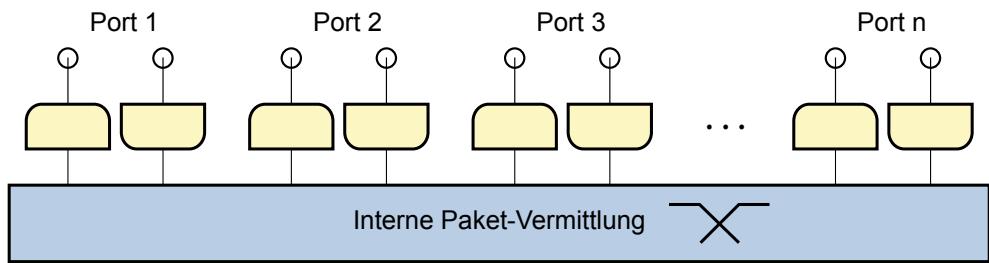


Abbildung 6.7: Aufbau einer Multiport-Bridge (Prinzip)

Multiport-Bridges (üblicherweise **Switch** genannt) sind die heute üblichsten Netzwerkkomponenten (siehe Abbildung 6.7). Sie erlauben die problemlose Koppelung von zwei und mehr Segmenten. Die Anzahl der Ports variiert typisch zwischen 4 und 64. Multiport-Bridges müssen eine **sehr schnelle interne Paketvermittlung** besitzen, da die vermittelte Bitrate mit der Port-Anzahl steigt.

a) Typische Anwendung von Remote Bridges

Aus sicherheitstechnischen, organisatorischen und auch psychologischen Gründen werden Server nur ungern dezentralisiert. Darum werden Server oft in so genannten Server-Farmen an einem zentralen Standort zusammengefasst. Dies wird durch die „Switch“-Technik unterstützt, da die unbeteiligten Segmente nicht mehr von fremden Frames belastet werden.

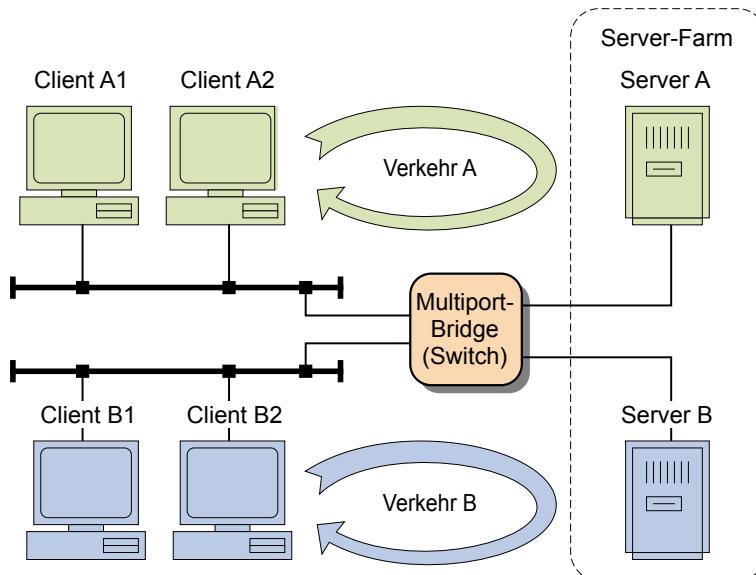


Abbildung 6.8: Switch zur Last-Entkoppelung

In örtlich verteilten Organisationen wird heute sehr oft eine Kombination von Multiport- und Remote-Bridges – so genannte **Multiport-Remote-Bridge** oder **Remote-Switch** – eingesetzt. Die letzte Variante in der Abbildung 6.9 zeigt den prinzipiellen Aufbau. Diese erlauben mehrere Segmente über grössere Distanzen zu koppeln. Zum Beispiel sind die ZHAW-Standorte in Winterthur

so verbunden. Wegen der hohen Anforderungen an den Backbone-Link (Server-Farmen etc.) werden in Firmennetzen typischerweise Verbindungen mit hoher Bitrate eingesetzt.

b) Bauformen von Switches

Es werden auch modulare Systeme angeboten (Stackable Switch). Diese werden entweder über ein Bündel von normalen Ports, einen Backbone-Port oder einen internen Bus gekoppelt (siehe Abbildung 6.9).

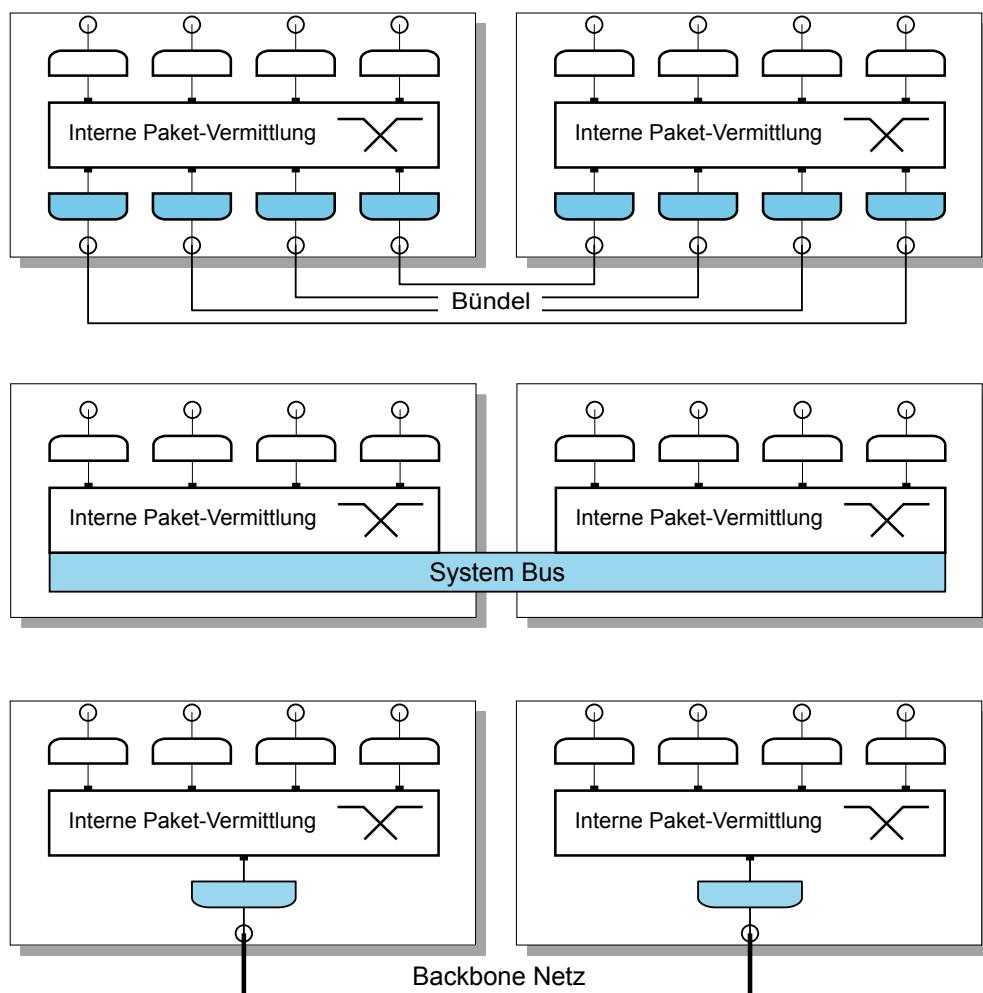


Abbildung 6.9: Modulare Multiport-Bridge (Prinzip)

Die Bündelung von mehreren **physischen LAN-Ports** zu einem schnellen Kanal wird als **Link Aggregation** oder **Port Trunking** bezeichnet und ist mit IEEE 802.1AX-2008 standardisiert worden. Das Verfahren erlaubt nicht nur Switches über einen schnelleren Kanal zu koppeln, sondern auch Knoten, die einen Anschluss mit höherer Bitrate benötigen. Ein Vorteil des Verfahrens ist, dass es von jedem Switch unterstützt werden kann. Nachteilig ist, dass Ports verloren gehen; beispielsweise werden durch die in Abbildung 6.9 oben gezeigte Konfiguration keine zusätzlichen Ports gewonnen.

6.1.4 Virtuelle LAN

In grossen Netzen mit vielen Knoten stellt der Broadcast-Verkehr ein Problem dar. Dieser lässt sich kaum vermeiden, da wichtige Protokolle, wie z.B. das Address Resolution Protocol (Abschnitt 7.6.3) darauf basieren. Mit Hilfe von **virtuellen LAN** kann ein grosses Netz in unabhängige logische Netze aufgeteilt werden (Abbildung 6.10).

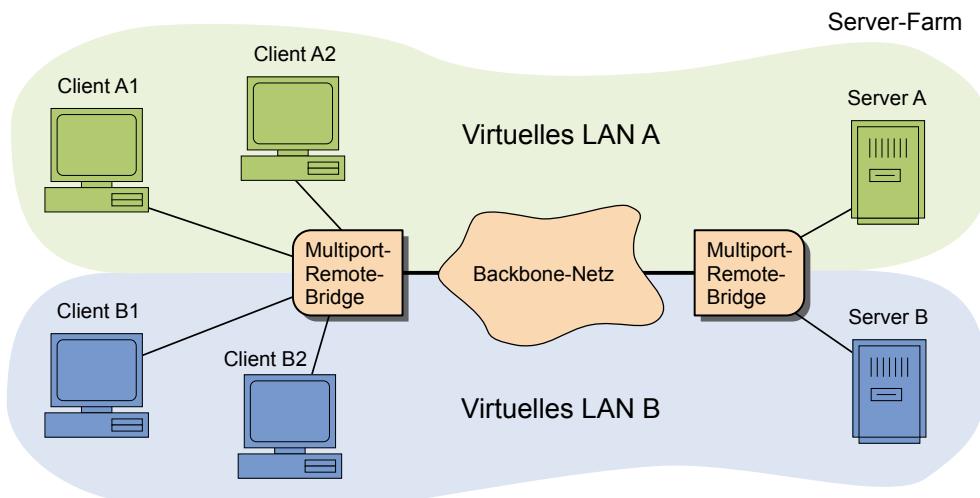


Abbildung 6.10: Virtuelles LAN zur logischen Unterteilung des Netzes

Durch das Netzwerk-Management kann jedes Switch-Port einem beliebigen virtuellen LAN zugeordnet werden. Ein **virtuelles LAN** bildet eine **Broadcast Domain**, das heisst die Grenze für die Verteilung der Broadcast-Frames.

Wie können in Abbildung 6.11 die Frames im Backbone-Netz dem richtigen VLAN zugeordnet werden? Dazu wird das Ethernet-Frame mit zusätzlichen Feldern versehen, die **VLAN-Tag** oder 802.1Q-Header genannt werden. Mit Hilfe des darin enthaltenen VLAN-Identifiers kann der Empfänger-Switch das Frame dem richtigen VLAN zuordnen. Im einfachsten Fall wird der VLAN-Tag vom Switch hinzugefügt und auch wieder entfernt. Zusätzlich erlaubt das Priorität-Feld „user priority“ eine Bevorzugung von bestimmten Applikationen, Knoten oder Netzen. Beispielsweise wäre es möglich, dem LAN A (Staff) in Abbildung 6.10 eine höhere Priorität als dem LAN B (Edu) zu geben. Falls es dann im Backbone zu einem Engpass käme, würden die Frames vom LAN A immer noch übertragen, die vom LAN B jedoch verzögert und allenfalls verworfen.

Wie erwähnt, wird im einfachsten Fall der VLAN-Tag von den Switches eingefügt und entfernt. Endknoten werden also nicht merken, dass sie an einem VLAN angeschlossen sind. Damit Applikationen die Priorität der Frames kontrollieren können, müssen die VLAN-Tags bis zu den Endknoten weitergeleitet werden. Dazu müssen die Switch-Ports entsprechend konfiguriert werden, und die Endknoten müssen über eine VLAN fähige Netzwerkkarte mit Protokoll-Software verfügen. Damit können beispielsweise zeitkritische (Voice-)Frames gegenüber unkritischen (Datenbackup-)Frames bevorzugt werden.

Bei hohem Verkehrsaufkommen kann dem Switch der Bufferspeicher ausgehen. Dann müssen Frames verworfen werden und gehen damit verloren, ohne dass der ursprüngliche Sender dies bemerkt. Für solche Situationen wurde die **Priorisierung** von Frames eingeführt.

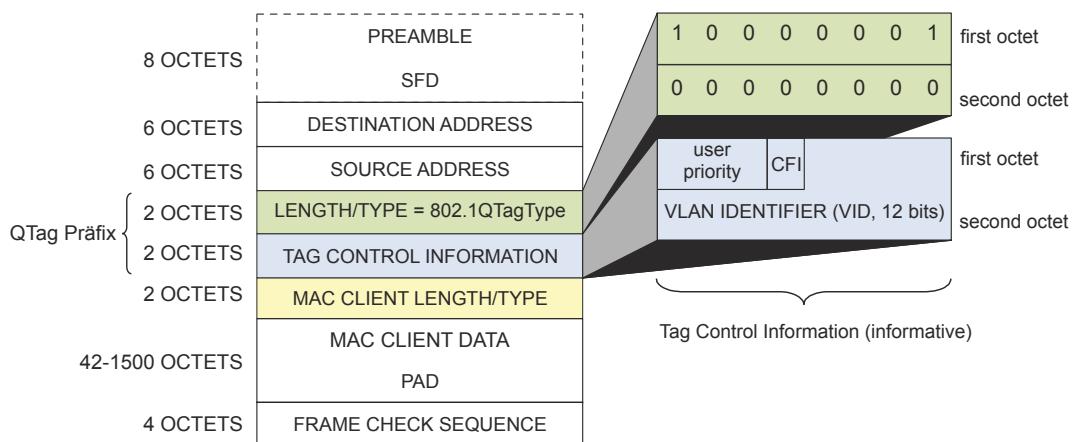


Abbildung 6.11: Tagged MAC Frame Format (Quelle: IEEE 802.3)

6.1.5 Redundanz Protokolle

Zur Verbesserung der Netzwerkuverlässigkeit können mit Bridges redundante Pfade gebildet werden. Allerdings muss vermieden werden, dass Frames endlos in Maschen des Netzwerks herumkreisen. Dies ist Aufgabe der Redundanzprotokolle.

a) (Rapid) Spanning Tree Protocol

Beim Spanning Tree Protocol besteht der Ansatz darin, von den redundanten Pfaden alle bis auf einen zu sperren. Im Fehlerfall wird der Prozess wiederholt und (falls möglich) der ausgefallene Pfad durch einen anderen ersetzt. Wie der Name Spanning Tree Protocol sagt, wird vom Algorithmus die Wurzel des Baums (Root-Bridge) bestimmt und ausgehend von dieser ein Baum aufgespannt, der alle Knoten genau einmal verbindet. Die Verbindungen die zu Schleifen führen würden, werden gesperrt. In Abbildung 6.12 bilden die ausgezogenen Verbindungen den Spanning Tree; die gestrichelten sind unterbrochen.

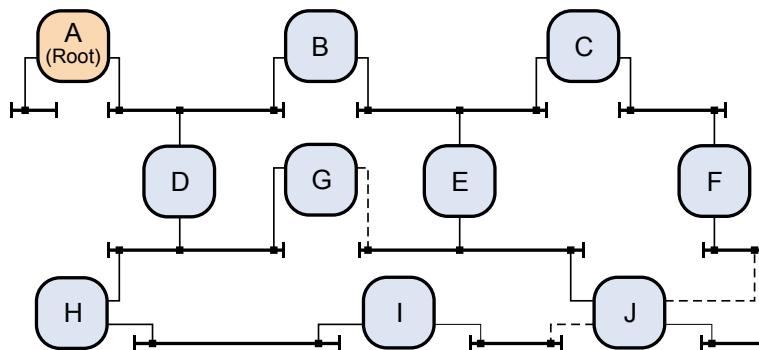


Abbildung 6.12: Spanning Tree bei einem vermaschten Netz

Der Algorithmus basiert auf dem Austausch von speziellen Meldungen (Bridge Protocol Data Unit, BPDU) zwischen den Switches. Die BPDU verwenden spezielle Multicast-Adressen (Bridge

Group Adresse), die von den Switches nicht weitergeleitet werden. Es gibt 16 solcher Adressen im Bereich von 01-80-C2-00-00-00 bis 01-80-C2-00-00-0F.

In den BPDU teilt jeder Knoten seinen Nachbarn mit, welches (seines Wissens) die beste Root-Bridge ist und zu welchen Kosten er diese erreichen kann. Die Auswahl der Root-Bridge beruht auf einem Bridge Identifier, der aus einer wählbaren Priorität und der MAC-Adresse besteht. Die Bridge mit dem tiefsten Identifier wird die Root-Bridge. Die Pfadkosten werden ausgehend von der Root aufaddiert, wobei Kosten einer Verbindung ebenfalls einstellbar sind (default-mässig umso höher je kleiner die Bitrate).

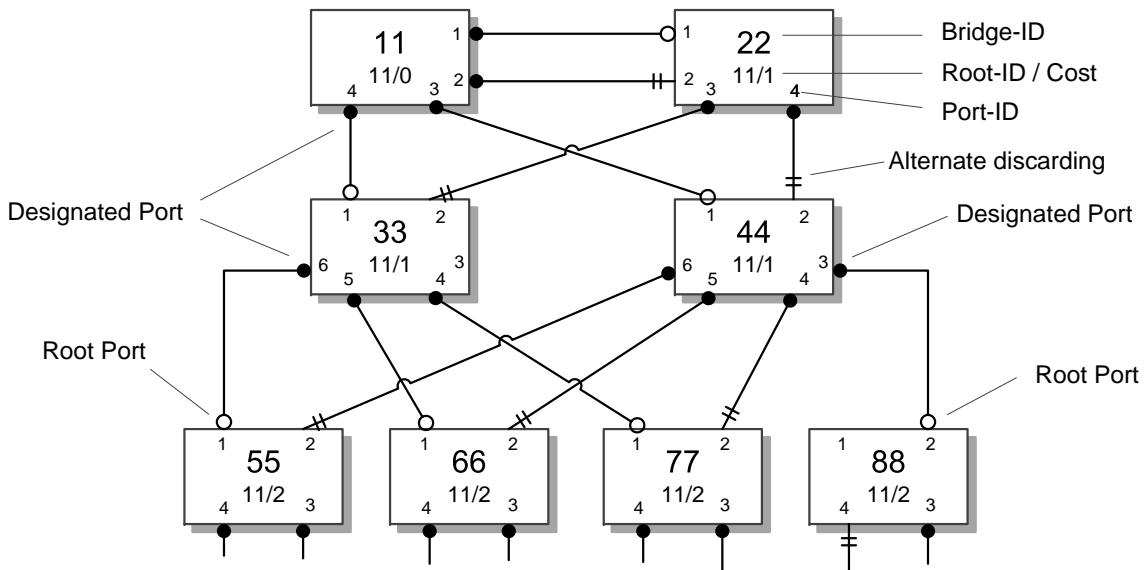


Abbildung 6.13: Vermaschtes Netz nach Anwendung des Spanning Tree Algorithmus

Nach dem Einschalten wird jeder Knoten sich selber als Root-Bridge mit Kosten 0 anpreisen. Erkennt ein Knoten, dass ein anderer eine bessere Bridge-ID hat, so wird dieser diese als Root-Identifier übernehmen und diesen zusammen mit den entsprechenden Pfadkosten seinen Nachbarn mitteilen. Nach wenigen Schritten werden alle Bridges die wirkliche Root-Bridge kennen. Bestehen redundante Pfade, so wird jeder Switch das Port mit den tiefsten Pfadkosten (zur Root) als sogenanntes **Root-Port** bestimmen. Das Port der nächsten Bridge, das von den Root-Port zur Root führt, heißt Designated Port. Falls es im Auswahlprozess Mehrdeutigkeiten gibt, so wird stets die Bridge oder das Port mit der tiefsten Adresse verwendet. Alle Ports außer den Root- und Designated-Ports werden gesperrt (Abbildung 6.13).

Im Betrieb versenden die Bridges alle 2 Sekunden BPDU. Bleibt eine solche Meldung aus, wird das Netzwerk rekonfiguriert. Nach der Rekonfiguration ist die Forwarding Database teilweise nicht mehr gültig und muss neu aufgebaut werden. Beim ursprünglichen Spanning Tree Protokoll wurden während der Rekonfiguration alle Ports für den normalen Datenverkehr gesperrt, was zu einem bis 30 Sekunden dauernden Unterbruch führte. Aus diesem Grund wurde das **Rapid Spanning Tree Protocol (RSTP)** definiert. Die Idee hinter RSTP ist, dass bei einer Topologieänderung der Verkehr (soweit möglich) auf den bisherigen Pfaden weitergeleitet und parallel dazu die Alternativpfade bestimmt werden. Erst wenn der neue Baum bestimmt ist, wird umgeschaltet. Die Unterbruchszeit kann mit RSTP auf unter eine Sekunde reduziert werden.

b) Alternative Redundanzprotokolle

Für viele Anwendungen ist auch dies noch zu lange, weshalb in IEC 61784 spezielle Hochverfügbarkeitsprotokolle definiert wurden, die eine völlig unterbrechungsfreie Kommunikation ermöglichen. High Availability Seamless Redundancy (HSR) und Parallel Redundancy Protocol (PRP) verbinden die hochverfügbaren Knoten (DANH/DANP) über zwei Ports und schicken die Frames über zwei unabhängige Pfade (Abbildung 6.14). In IEEE 802.1CB sind Anfangs 2014 ebenfalls entsprechende Normierungsbestrebungen angelaufen.

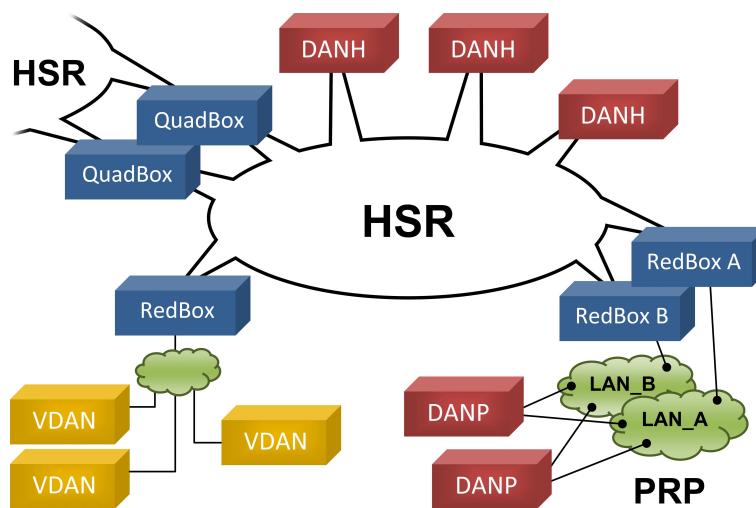


Abbildung 6.14: Beispiel: Knoten mit Netzredundanz bei HSR/PRP

Ein weiterer Nachteil des Spanning Tree Protocols ist, dass die blockierten Pfade im Normalfall ungenutzt bleiben. Diese Ressourcen werden schlecht genutzt, da auch in Überlastsituationen darüber kein Ausgleich erfolgen kann. Dafür wurde das Shortest Path Bridging (IEEE 802.1aq) entwickelt. Es erlaubt, mehrere Pfade aktiv zu nutzen und unterstützt zudem grössere und flexiblere Layer-2 Topologien.

6.1.6 Weitere Leistungsmerkmale von Switches

Bei den Switches gibt es eine Vielzahl von Leistungsmerkmalen:

Grösse der Adressstabelle: Sie gibt an, wie viele Zieladressen von Knoten gespeichert werden können. Überläuft die Adressstabelle, so werden gewisse Adressen nicht gefunden und die Frames darum an alle verschickt.

Grösse des Buffers: Sie gibt an, wie viel Speicher der Switch Total für alle Ports zur Verfügung hat, um Frames zwischenspeichern.

Bit Rate: Sie gibt an, wie viele Bits pro Sekunde der Switch maximal weiterleiten kann. Falls ein Switch in der Lage ist, die Bitrate zu bewältigen, die entsteht, wenn alle Ports mit der maximalen Bit Rate arbeiten, so sagt man, er sei blockierungsfrei (non-blocking).

Forwarding Rate: Sie gibt an, wie viel Frames pro Sekunde die Bridge auf die andere Seite weiterleiten kann.

Weiterleitungsverzögerung (Latency): Sie gibt an, wie viel Zeit die interne Verarbeitung zwischen dem Empfang und der Weiterleitung eines Frames benötigt.

Switching Mechanismus: Norm-konforme Bridges werden auch **Store-and-Forward-Bridges** genannt, weil sie zuerst das ganze Frame einlesen, zwischenspeichern und es erst dann weiterleiten. Bei langen Frames (1500 Byte Daten) und 10 Mbit/s beträgt die Verzögerung mindestens 12 ms.

Beim **Cut-Through-** oder **On-the-Fly-Switching** beginnt die Weiterleitung des ankommenen Frames bereits nach Empfang der 6-Byte-Destination-Adresse. Da nicht das gesamte Frame empfangen werden muss, tritt bei 10 MBit/s nur eine Zeitverzögerung von ca. 40 μ s ein. Sollte das Zielsegment bei der Übertragung gerade belegt sein, wird das Frame im Switch zwischengespeichert.

Switches sollen fehlerhafte Frames ausfiltern. Beim Cut-Through-Switching ist dies nicht möglich. Da die CRC-Prüfung erst bei einem vollständig gelesenen Frame durchgeführt werden kann, werden *auch fehlerhafte* Frames auf das andere Segment übertragen.

Solange der Prozentsatz von fehlerhaften Frames im Netz gering ist, entstehen dadurch keine Probleme. Sobald aber (z.B. aufgrund fehlerhafter Hardware oder extrem hoher Netzlast) der Prozentsatz der fehlerhaften Frames steigt, kann die Leistung des Gesamtnetzes deutlich sinken. Es gibt darum auch **adaptive Cut-Through-Bridges**, die bei hoher Fehlerzahl automatisch auf Store-and-Forward-Bridging umschalten.

Management Obwohl Switches möglichst Plug and Play sein sollen, erfordern sie (vor allem in grösseren Netzen) Managementfunktionen zur Konfiguration und Überwachung. Die meisten Switches unterstützen die Konfiguration über die Kommandozeile, eine Weboberfläche oder über ein spezielles Protokoll (SNMP). Das Link Layer DiscoveryProtocol (LLDP) erlaubt die Nachbarschaftserkennung und in Zusammenarbeit mit SNMP eine automatische Topologieerkennung.

Zu den Managementfunktionen gehören auch Redundanzprotokolle (siehe Abschnitt a)).

Echtzeitfähigkeit und Zeitsynchronisation Im Rahmen von IEEE 1588 und IEEE 802.1AS sind verschiedene Mechanismen und Verfahren definiert worden, die eine präzise Zeit Synchronisation und deterministischen Datentransfer erlauben. Diese Verfahren müssen von den Switches unterstützt werden; d.h. die Leitungsverzögerungszeiten müssen bestimmt, Frames müssen bei ihrer Ankunft mit einem Zeitstempel versehen und die Verweilzeiten im Switch müssen gemessen und weitergeleitet werden.

6.2 Moderne Ethernet-Systeme

6.2.1 Twisted-Pair-Ethernet 10BASE-T

Neben den früher behandelten Bus-orientierten Standards (10BASE5 und 10BASE2) wurde eine zweite Gruppe von Ethernet-Technologien definiert, die wie 10BASE-T auf Punkt-zu-Punkt-Verbindungen basieren. Um ein Netzwerk von mehr als zwei Knoten zu bilden, werden darum Switches/Hubs benötigt (Stern- oder Baum-Topologie).

Bezeichnung / Norm	Medium	Max. Distanz (Segmentlänge)	Topologie	Bemerkungen
10BASE5	50 Ohm Koax	500 m	Bus	Thick Ethernet
10BASE2	50 Ohm Koax	185 m	Bus	Cheapernet
10Broad36	75 Ohm Koax	3600 m	Bus	CATV-Technik
10BASE-T	2 Paar UTP Cat. 3	100 m	Linie	
10BASE-FL	2 MMF (62.5 μ m)	2000 m	Linie	
10BASE-FP	2 MMF (62.5 μ m)	500 m	Stern	passiver Hub
10BASE-FB	2 MMF (62.5 μ m)	2000 m	Linie	

10BASE-T verwendet *paarweise* verdrillte Adern (Twisted Pair). Der Aufbau des LAN erfolgt hier in einer sternförmigen Topologie, wobei die Koppelung über Hubs erfolgt (siehe Abbildung 6.15).

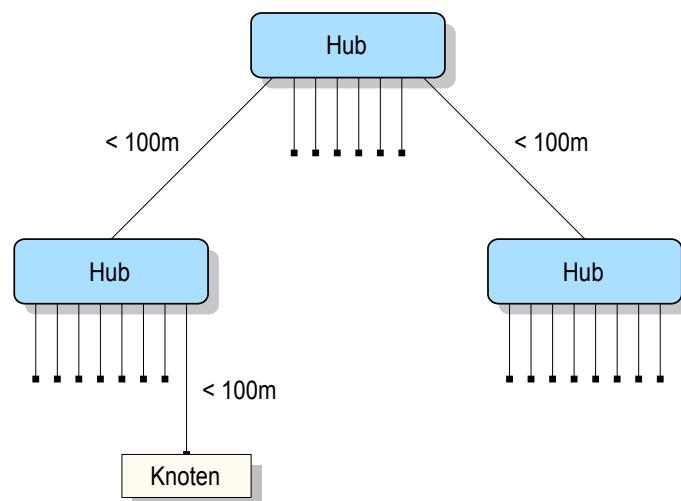


Abbildung 6.15: 10BASE-T Netzwerk-Konfiguration

Als Stecker werden achtpolige RJ-45-Stecker eingesetzt. Die folgende Tabelle zeigt die Anschlussbelegung eines Endgeräts.

Contact	MDI Signal	Contact	MDI Signal
1	TD+ (Transmit Data)	5	Not used by 10BASE-T
2	TD- (Transmit Data)	6	RD- (Receive Data)
3	RD+ (Receive Data)	7	Not used by 10BASE-T
4	Not used by 10BASE-T	8	Not used by 10BASE-T

Werden zwei 10BASE-T-Geräte miteinander verbunden, so müssen die Sendeleitungen des einen Geräts mit den Empfangsleitungen des anderen verbunden werden und umgekehrt. Bei normalen Knoten (z.B. PC mit Netzkarten) können die Leitungen, wie in Abbildung 6.16 gezeigt, extern mit Hilfe von Cross-Over-Kabeln ausgetauscht werden.

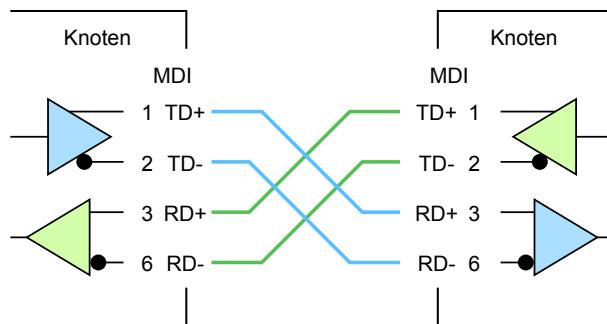


Abbildung 6.16: Verbindung von 10BASE-T-Geräten

Bei Switches/Hubs sind die Leitungen intern am Port ausgetauscht (**MDI-X**), wie in Abbildung 6.17 gezeigt. Damit können Endgeräte mit „geraden“ Kabeln an den Repeater-Ports angeschlossen werden.

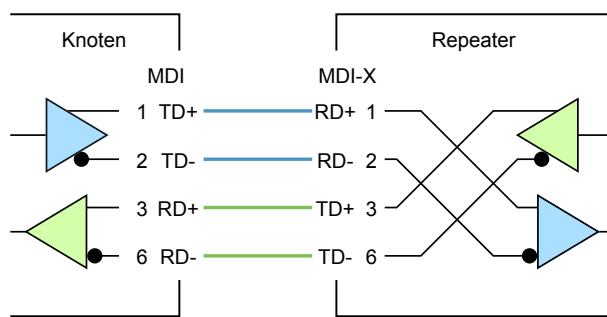


Abbildung 6.17: Verbindung eines 10BASE-T-Geräts mit einem Switch/Hub

Anmerkung: Moderne Ethernet-Systeme verfügen über eine **Auto-Crossover**-Funktion, womit sie selber erkennen können, ob die Leitungen ausgetauscht werden müssen oder nicht und diese

dann auch entsprechend schalten. **Auto-Polarity** erlaubt zudem vertauschte Drähte eines Aderpaars zu korrigieren.

a) Halbduplex Betrieb

Da bei 10BASE-T für das Senden und Empfangen separate Aderpaare verwendet werden, entstehen bei Kollisionen natürlich keine Spannungsüberhöhungen. Wie in Abbildung 6.18 gezeigt, werden Kollisionen erkannt, indem die DI- und DO-Signale überwacht werden. Sind beide gleichzeitig aktiv, so wird eine Kollision gemeldet (Collision Detected).

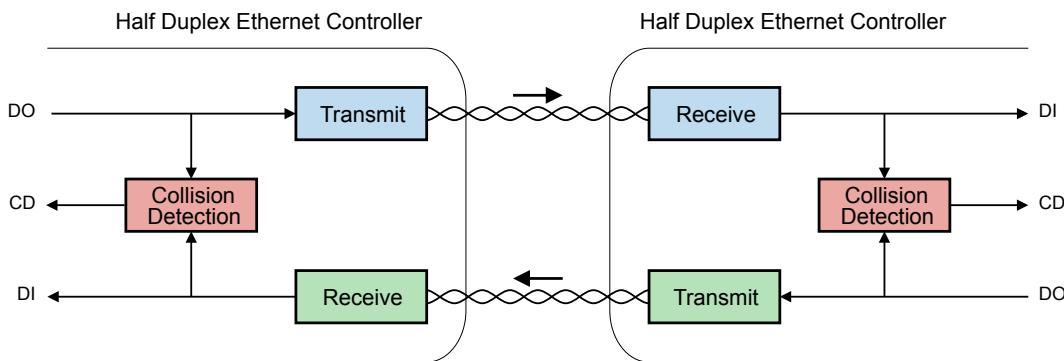


Abbildung 6.18: Prinzipschaltbild eines Halbduplex Ethernet Controllers

b) Vollduplex Betrieb

Ein mit Switches aufgebautes Netz besteht eigentlich aus lauter Punkt-Punkt-Verbindungen. Da bei 10BASE-T Strecken für die Sende- und Empfangsrichtung unterschiedliche Aderpaare verwenden, ist auch ein Vollduplex-Betrieb möglich. Wie in Abbildung 6.19 gezeigt, ist die Schaltung sogar wesentlich einfacher, da keine Kollisionsüberwachung mehr erforderlich ist. Gleichzeitig wird auch die Loopback-Funktion gesperrt.

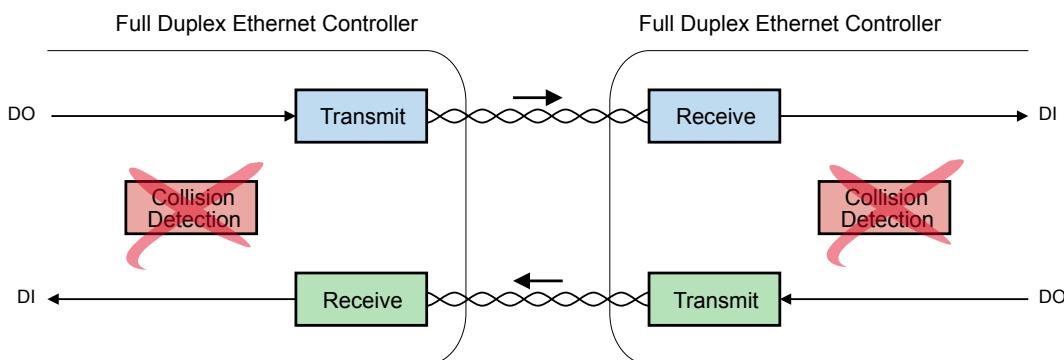


Abbildung 6.19: Prinzipschaltbild eines Vollduplex Ethernet Controllers

6.2.2 100 MBit/s-Ethernet-Systeme (Fast-Ethernet)

Fast-Ethernet ist die 100-MBit/s-Ethernet Spezifikation aus IEEE 802.3 und unterstützt theoretisch drei Typen von Übertragungsmedien:

Bezeichnung / Norm	Medium	Max. Distanz (Segmentlänge)	Bemerkungen
100BASE-TX	2 Paar UTP Cat. 5	100 m	
100BASE-FX	2 MMF (62.5 μm)	2000 m	
100BASE-T4	4 Paar UTP Cat. 3	100 m	Keine Produkte mehr
100BASE-T2	2 Paar UTP Cat. 3	100 m	Keine Produkte

Dafür wurden zwei unterschiedliche Leitungskodierungen definiert: 100BASE-X für 100BASE-TX und 100BASE-FX, sowie eine separate Kodierung für 100BASE-T4 und 100BASE-T2.

Die 100BASE-X-Kodierung basierte auf einer erprobten Technologie und war darum sehr früh am Markt erhältlich. Als die Spezifikationen 100BASE-T4 und 100BASE-T2 fertig wurden, waren bereits viele Geräte auf dem Markt. Da 100BASE-T4 und 100BASE-T2 nicht mit 100BASE-TX kompatibel sind, konnten sie sich nicht durchsetzen.

a) Abwärtskompatibilität von 100BASE-T

Wie erwähnt, war es für den Erfolg von Ethernet ausschlaggebend, dass neue Technologien stets abwärtskompatibel zu den bestehenden Installationen gestaltet wurden. Aus diesem Grund wurde bei Fast-Ethernet das IEEE-802.3 Frame-Format, die Frame-Grösse und die Art der Fehlererkennung beibehalten. Damit unterstützte es die vorhandene Software.

In den Anfangszeiten wurden 10 und 100 MBit/s-Ethernet-Systeme gemischt eingesetzt. Fast-Ethernet-Ports müssen darum in der Lage sein, mit beiden Übertragungsgeschwindigkeiten zu arbeiten.

Abbildung 6.20 zeigt den Aufbau des Fast-Ethernet-Stacks mit seinen Sub-Layern. Die Aufwärtskompatibilität wird auf verschiedenen Ebenen sichergestellt:

Das Medium Dependent Interface (MDI) ist die mechanische und elektrische Schnittstelle zwischen dem Übertragungsmedium und Physical Layer des Knotens. Für Übertragungsmedien, die von 10 und 100 Mbit/s-Systemen unterstützt werden, wird die Interoperabilität gewährleistet.

Die Autonegotiation ist ein zusätzlicher Sub-Layer, der es den Ports erlaubt, automatisch zwischen 10 oder 100 MBit/s-Betrieb und anderen Leistungsmerkmalen zu wählen, je nachdem welche davon die Gegenseite unterstützt. Diese Funktion wird im folgenden Abschnitt b) genauer beschrieben.

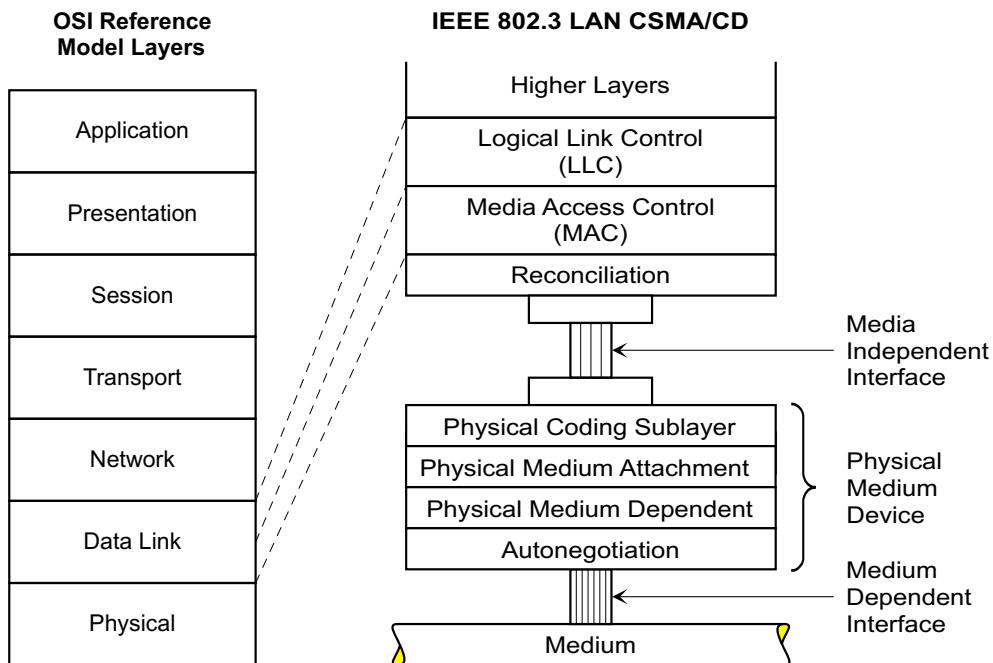


Abbildung 6.20: Fast-Ethernet-Architektur

Das Media Independent Interface (MII) unterstützt Datenraten von 10 MBit/s und 100 MBit/s über zwei je vier-Bit-breite Sende- und Empfangskanäle und ist unabhängig (wie der Name sagt) vom darunter liegenden Übertragungsmedium, kann aber nicht gleich sein wie bei 10 MBit/s Systemen.

In der Praxis ist das Media Independent Interface nur als physische Schnittstelle zwischen dem PHY- und dem MAC-Chip zu finden.

Der Reconciliation Sublayer passt die Signale des MII an den Media Access Control Sublayer (MAC) an. Die Schnittstelle zwischen der MAC-Schicht und dem Reconciliation Layer ist genau die gleiche wie die von 10 MBit/s-Systemen.

b) Autonegotiation

An Fast-Ethernet unterstützen alle Systeme eine (optionale) Funktion zur automatischen Abstimmung (Autonegotiation). Damit sind zwei Netzwerkkomponenten in der Lage, ihre Angaben zu ihren Leistungsmerkmalen auszutauschen und die beste Betriebsart zu ermitteln.

Die Autonegotiation unterstützt eine Vielzahl von Eigenschaften. Abbildung 6.21 zeigt, welche Leistungsmerkmale bei Fast Ethernet ausgehandelt werden können. Die wichtigsten Parameter sind die Abstimmung der Übertragungsgeschwindigkeit sowie des Voll- oder Halb-Duplex-Betriebs.

Wie Abbildung 6.22 zeigt, erfolgt der automatische Abstimmungsprozess zwischen den Ports mit Hilfe spezieller Signale, die als **Fast-Link Pulses (FLP)** bezeichnet werden. Diese werden zu Beginn des Verbindungsaufbaus (beim Einsticken des Mediums oder beim Einschalten) und vor dem eigentlichen Beginn des Datenaustauschs auf die Leitung gegeben.

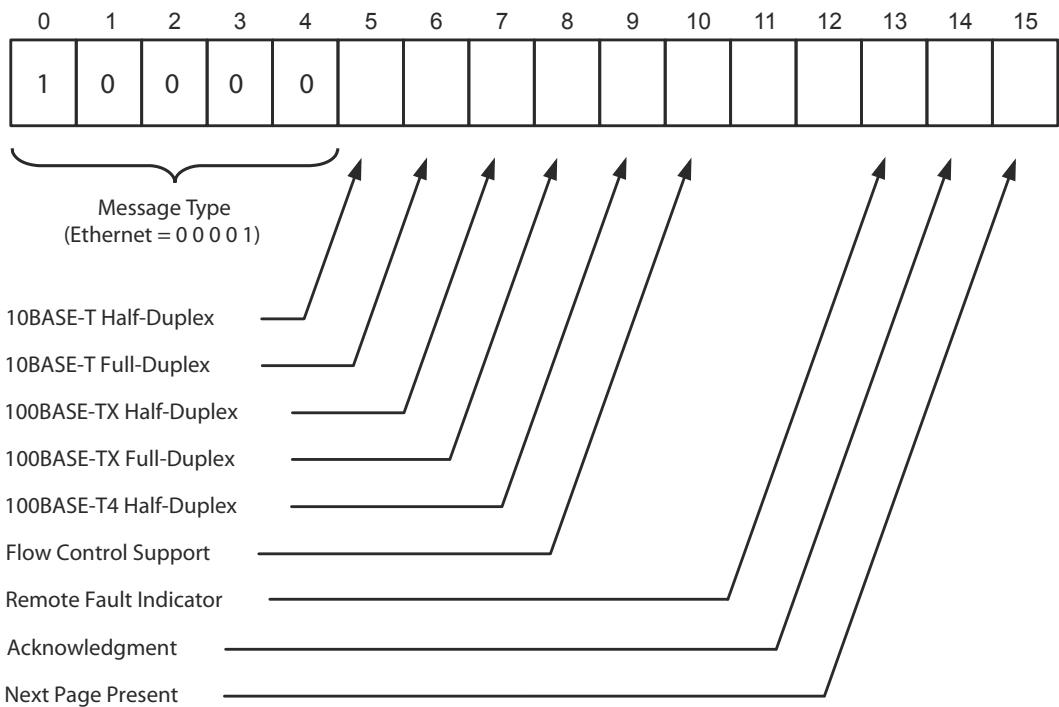


Abbildung 6.21: Informationen der Auto-Negotiation Base Page

Auch diese Funktion ist (auf- und abwärts kompatibel ausgestaltet. Schon bei 10BASE-T wird im Ruhezustand die Verbindungsintegrität überprüft. Diese Signale werden als **Normal-Link Pulses (NLP)** bezeichnet (Abbildung 6.22). Mit Next Page Present kann die Base Page für neue Technologien und weitere Optionen erweitert werden.

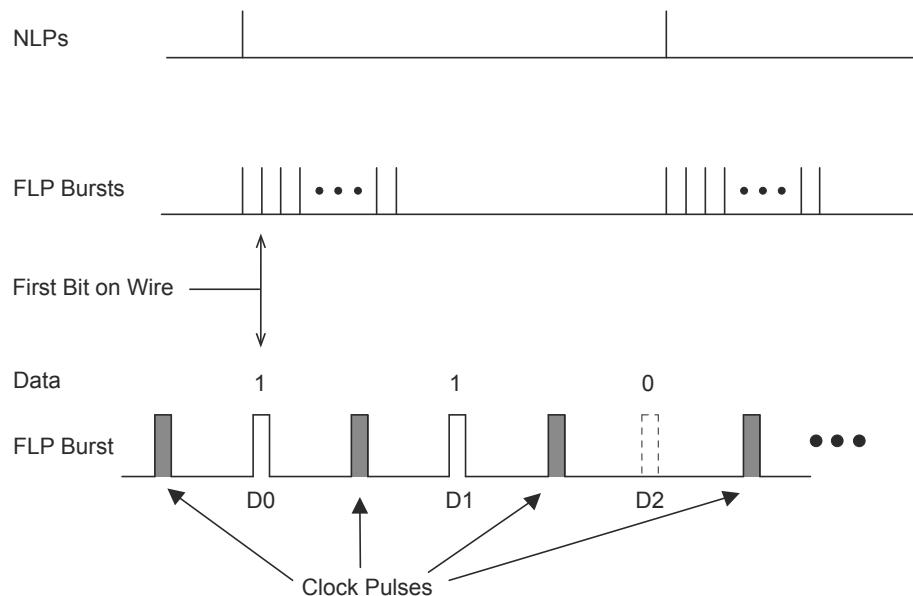


Abbildung 6.22: Vergleich der Fast-Link- und der Normal-Link-Pulse

Der Aufbau ist so, dass immer jeweils ein Clock-Puls und Daten-Puls sich abwechseln (Abbildung 6.23).

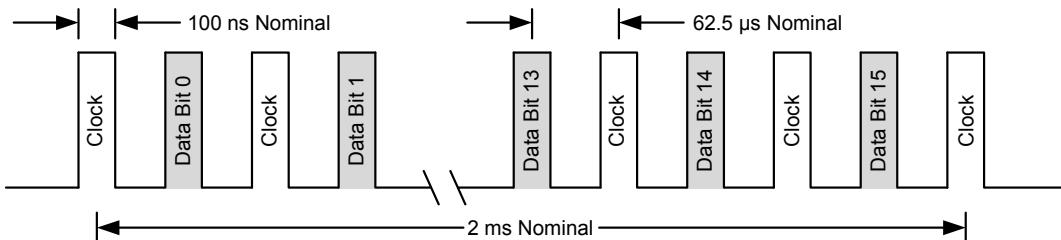


Abbildung 6.23: Auto-Negotiation-Signalverlauf

Es gilt zu beachten:

- Die früher erwähnten Auto-Crossover- und Auto-Plarity-Funktionen sind nur verfügbar, wenn Auto-Negotiation eingeschaltet ist.
- Fix konfigurierte Ports senden keine Fast-Link-Pulse³. Darum müssen entweder beide Ports mit Auto-Negotiation oder beide mit einer identischen Einstellung fix konfiguriert werden. Falls ein Port fix mit Vollduplex konfiguriert ist und das andere mit Auto-Negotiation arbeitet, so wird letzteres auf Halbduplex eingestellt, weil dies die Default-Einstellung ist.

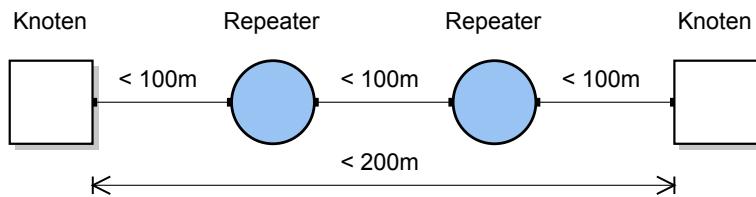
c) Fast-Ethernet Halbduplex- und Vollduplex-Betrieb

Fast-Ethernet setzt das gleiche Frame-Format und die gleichen Verfahren für den MAC-Zugriff und die Fehlererkennung ein wie das 10MBit/s-Ethernet.

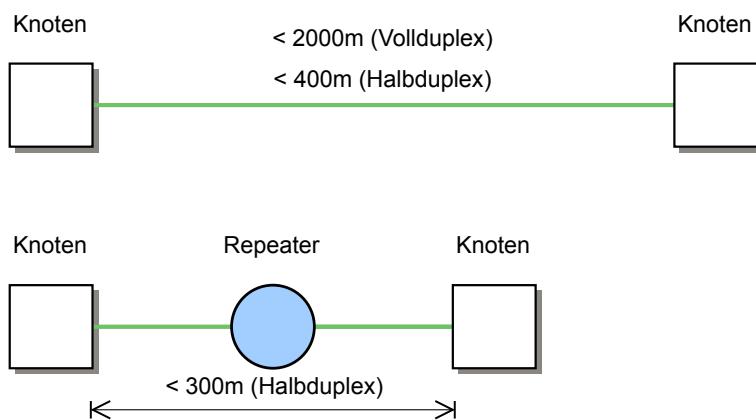
Der eigentliche Unterschied (ausser dem offensichtlichen Geschwindigkeitsunterschied) ist der maximale Netzwerkdurchmesser (Collision Domain). Für Fast-Ethernet ist er um den Faktor 10 kleiner als beim 10 MBit/s-Ethernet. Der Grund für die Verkleinerung des Netzwerkdurchmessers bei Fast-Ethernet liegt im CSMA/CD-Verfahren, das unverändert übernommen wurde. Wegen der zehnmal höheren Bitrate muss die Grösse der Kollisionsdomäne verkleinert werden (Herleitung siehe Abschnitt 5.4.4).

100BASE-TX wird von praktisch allen aktuellen Ethernet-Komponenten unterstützt. Die Spezifikation erlaubt ungeschirmte Twisted-Pair-Kabel und geschirmte Twisted-Pair-Kabel. 100BASE-T erlaubt Segmentlängen bis zu 100 m. Mit zwei möglichen Repeatern darf die maximale Grösse der Collision Domain trotzdem 200 Metern nicht überschreiten (Abbildung 6.24).

³Im Standard gibt es eine Empfehlung, dass auch fix konfigurierte Ports Fast-Link Pulse senden sollen, in der Praxis hat sich diese Empfehlung nicht durchgesetzt.

**Abbildung 6.24:** Maximale Distanzen im 100BASE-TX-Ethernet

100BASE-FX verwendet zwei Multi-Mode-Lichtwellenleiter (MMF); je einen pro Richtung. Die Spezifikation ermöglicht Halbduplex-Verbindungen von bis zu 400 m oder mit einem Repeater ca. 300 Metern Länge. Mit Vollduplex können 2000 m erreicht werden. Abbildung 6.25 illustriert diese Konfigurationsrichtlinien.

**Abbildung 6.25:** Maximale Distanzen im 100BASE-FX-Ethernet

d) 100BASE-X Leitungscodierung

100BASE-X, die Leitungscodierung für 100BASE-TX (2 Twisted-Pair, Cat. 5) und 100BASE-FX (2 Multimode-Glasfasern), verwendet die seit den 80er Jahren verfügbare und erprobte Technologie von FDDI/CDDI⁴.

Für die Übertragung werden die zu sendenden Bytes in jeweils 2 Nibble aufgeteilt und über das Medium Independent Interface (MII) geschickt. Im Physical Coding Sublayer (PCS) wird dann jedes Nibble (4 Bit) mit einem 5-Bit-Symbol (4B/5B Code) codiert. Von den 32 theoretisch möglichen Symbolen werden 16 für die Datencodierung verwendet (Abbildung 6.27).

⁴Der Physical Layer von 100BASE-TX ist zum grössten Teil im Standard ANSI X3.263-1995 *Fibre Distributed Data Interface (FDDI) - Token Ring Twisted Pair Physical Layer Medium Dependent* definiert. FDDI (Fibre Distributed Data Interface) verfügte über eine Übertragungsr率e von 100 Mbit/s, unterstützte Lichtwellenleiter und Twisted-Pair-Kabel CDDI (Copper Distributed Data Interface) und wurde praktisch ausschliesslich im Backbone-Bereich eingesetzt.

PCS code-group 4 3 2 1 0	Name	MII (TXD/RXD) 3 2 1 0	Interpretation
1 1 1 1 0	0	0 0 0 0	Data 0
0 1 0 0 1	1	0 0 0 1	Data 1
1 0 1 0 0	2	0 0 1 0	Data 2
1 0 1 0 1	3	0 0 1 1	Data 3
0 1 0 1 0	4	0 1 0 0	Data 4
0 1 0 1 1	5	0 1 0 1	Data 5
0 1 1 1 0	6	0 1 1 0	Data 6
0 1 1 1 1	7	0 1 1 1	Data 7
1 0 0 1 0	8	1 0 0 0	Data 8
1 0 0 1 1	9	1 0 0 1	Data 9
1 0 1 1 0	A	1 0 1 0	Data A
1 0 1 1 1	B	1 0 1 1	Data B
1 1 0 1 0	C	1 1 0 0	Data C
1 1 0 1 1	D	1 1 0 1	Data D
1 1 1 0 0	E	1 1 1 0	Data E
1 1 1 0 1	F	1 1 1 1	Data F

Abbildung 6.26: 4B/5B Code-Gruppen für Daten

Sechs Symbole werden für die Signalisierung verwendet. Abbildung 6.27 zeigt diese Symbole.

PCS code-group 4 3 2 1 0	Name	MII (TXD/RXD) 3 2 1 0	Interpretation
1 1 1 1 1	I	undefined	IDLE; used as inter-stream fill code
1 1 0 0 0	J	0 1 0 1	Start-of-Stream Delimiter, Part 1 of 2; always used in pairs with K
1 0 0 0 1	K	0 1 0 1	Start-of-Stream Delimiter, Part 2 of 2; always used in pairs with J
0 1 1 0 1	T	undefined	End-of-Stream Delimiter, Part 1 of 2; always used in pairs with R
0 0 1 1 1	R	undefined	End-of-Stream Delimiter, Part 2 of 2; always used in pairs with T
0 0 1 0 0	H	undefined	Transmit Error; used to force signaling errors

Abbildung 6.27: 4B/5B Code-Gruppen für die Signalisierung

Um die Abstrahlung zu verringern (Energie noch etwas gleichmässiger zu verteilen), werden diese Symbole vom Sender in einem Scrambler noch zusätzlich mit einer 11-Bit-Pseudozufallszahl (LFSR, Linear Feedback Shift Register) verknüpft. Der Empfänger muss dies rückgängig machen. Da der Zufallszahlengenerator des Senders freilaufend ist, muss der Sender seinen Zufallszahlen-generator mit Hilfe einer Idle-Sequenz synchronisieren.

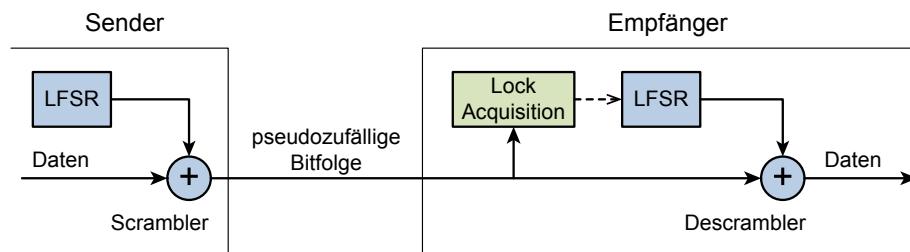


Abbildung 6.28: Zusammenspiel von Scrambler und Descrambler

Die effektive Übertragungsrate beträgt so 125 Mbit/s. Mit dem Manchester-Code würde eine Bandbreite von 125 MHz benötigt, was mit einem Cat. 5 Kabel nicht übertragen werden kann. Es wird darum eine NRZI-Codierung (Non Return to Zero, Invert on ones) verwendet: Bei einer Null bleibt der Ausgangspegel wie er ist, bei einer Eins erfolgt ein Pegelwechsel. Um den Gleichstromanteil zu eliminieren, wird zwischen positiven und negativen Pulsen (ca. +/- 1 Volt) abgewechselt.

Die Abbildung 6.29 zeigt eine gemessene 100BASE-TX-Verbindung im Ruhezustand (IDLE). Eine senkrechte Teilung entspricht einem Bit (8ns) des „verwursteten“ 4B/5B-Codes.

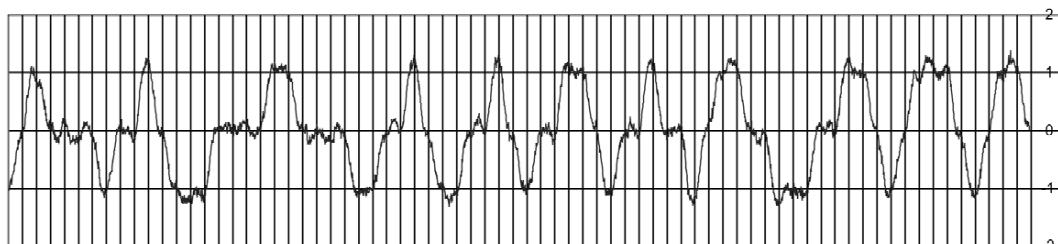


Abbildung 6.29: Dreiwertiger NRZI-Code

6.2.3 1000 MBit/s-Ethernet-Systeme (Gigabit Ethernet)

Bezeichnung / Norm	Medium	Max. Distanz (Segmentlänge)	Bemerkungen
1000BASE-CX	2 Paar STP	25m	
1000BASE-SX	2 MMF (62.5µm)	275m	Kurzwellige Laser
	2 MMF (55µm)	550m	770 – 860nm
1000BASE-LX	2 MMF (62.5µm)	550m	Langwellige Laser
	2 MMF (50 µm)	550m	1270 – 1355nm
	2 SMF (10 µm)	5000m	
1000BASE-LX10	2 SMF	10km	
1000BASE-BX10	1 SMF	10km	Bidirektional
1000BASE-PX10	2 SMF	10km	Point-to-multipoint
1000BASE-PX20	2 SMF	20km	Point-to-multipoint
1000BASE-T	4 Paar Cat. 5	100m	

Gigabit Ethernet hat eine Bitrate von 1000 MBit/s, wobei dank Autonegotiation die Kompatibilität zu bestehenden Ethernet- und Fast-Ethernet-Geräten gewahrt bleibt. Das heisst, Gigabit Ethernet ist abwärtskompatibel zu 10BASE-T und 100BASE-T, verwendet das gleiche Frame-Format und die gleiche Frame-Grösse, wie vorhandene IEEE 802.3-Netze. Praktisch alle aktuellen IT-Geräte unterstützen Gigabit Ethernet (1000BASE-T).

Gigabit Ethernet unterstützt Multi-Modus-Glasfaser-Verbindung mit einer maximalen Länge von 500 Metern; Single-Modus-Glasfaser-Verbindung mit einer maximalen Länge von 2 km und STP-Kupferverkabelung mit maximal 25 Metern sowie UTP-Kupferverkabelung mit maximal 100 Metern.

Gigabit Ethernet wird üblicherweise im Voll-Duplex-Betrieb mit Switches eingesetzt. Halb-Duplex-Betrieb mit einem Shared Medium ist möglich, wobei die gleichen Regeln und Verfahren gelten wie bei 100BASE-T.

a) Gigabit-Ethernet-Spezifikation

Bei den optischen Transmittern wird für die Serialisierung und Deserialisierung eine 8B/10B-Codierung eingesetzt.

Um für die künftige Entwicklung gewappnet zu sein, wurde wieder eine logische, Medium-unabhängige Schnittstelle zwischen den Schichten MAC und PHY spezifiziert, das Gigabit Media Independent Interface (GMII). Diese erlaubt Codierungen für UTP-Verkabelung unabhängig von der Fiber-Channel-Codierung zu implementieren.

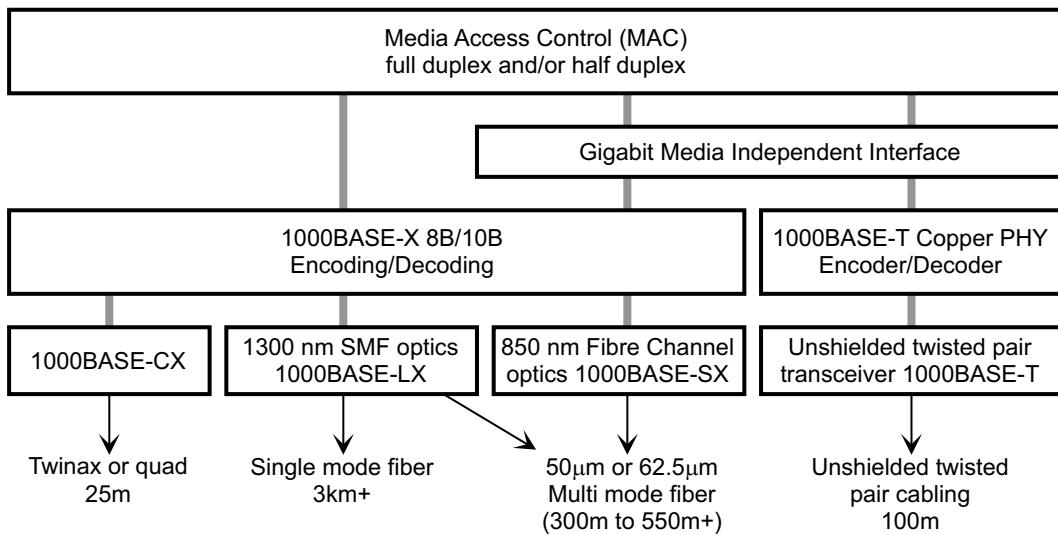


Abbildung 6.30: Layer des Gigabit-Ethernet

b) Gigabit-Ethernet über UTP-Verkabelung (1000BASE-T)

Dieser Abschnitt beschreibt grob die technischen Grundlagen von 1000BASE-T.

Kategorie 5 Kabel, wie im Standard ANSI/TIA/EIA-568-A definiert, sind gegenwärtig die am häufigsten eingesetzten (in den USA 72%). Da 1000BASE-T auf dieser bestehenden Infrastruktur einsetzbar sein sollte, musste ein neues Übertragungsverfahren für 1000Mbit/s entwickelt werden.

- Es werden alle 4 Aderpaare des Cat. 5 Kabels gleichzeitig verwendet. Dies erlaubt, die Symbolrate pro Leitung zu vieren.
- Es wird ein fünfwertiger Leitungs-Code (PAM-5) eingesetzt um die Informationsmenge pro Symbol zu erhöhen. Dies wird aber mit einem schlechteren Störabstand erkauft. Pro Symbol werden somit 2.3 Bit übertragen. Davon werden 2 Bit für die Information und 0.3 Bit für die Fehlerkorrektur eingesetzt.
- Mit Hilfe einer Fehlerkorrektur (4D 8-State Trellis Forward Error Correction) wird der Einfluss von Rauschen und Übersprechen kompensiert.
- Eine ausgereifte Technik (analog und digital) zur Puls-Formung dient dazu, einen möglichst guten Störabstand des Signals zu erzielen und gleichzeitig die Störabstrahlung klein zu halten. Das Spektrum von 1000BASE-T entspricht weitgehend dem vom 100BASE-TX.
- Durch digitale Signalverarbeitung können die störenden Einflüsse (Noise, Echo, Crosstalk) soweit reduziert werden, dass eine Bitfehlerrate von unter 10^{-10} eingehalten werden kann.

Durch die Verwendung der 4 Aderpaare und den fünfwertigen Code kann die Signalisierungsrate auf einer Leitung (Symbole pro Sekunde) geachtet werden. Sie ist damit gleich wie bei 100BASE-TX, nämlich 125 Mbaud.

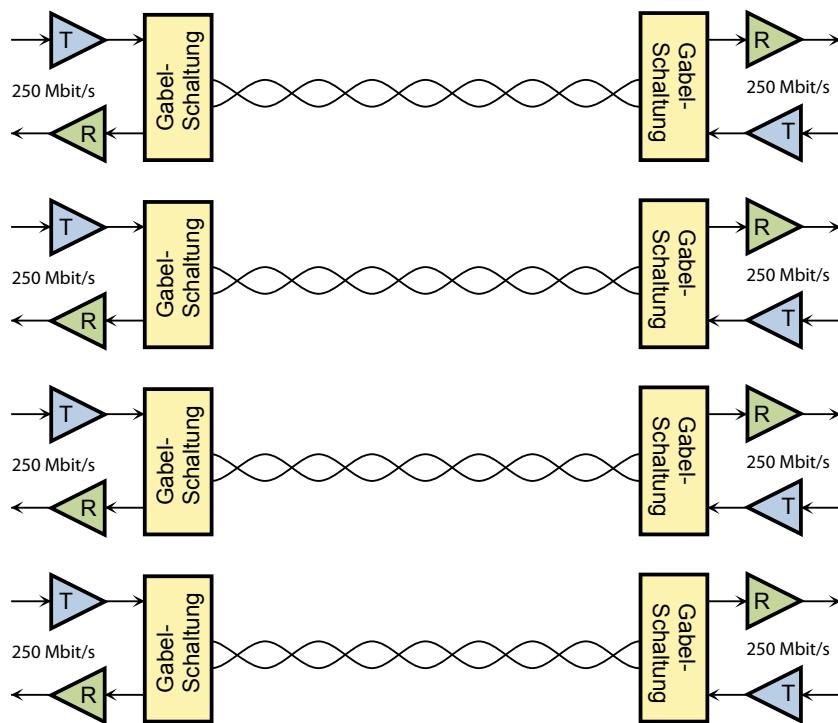


Abbildung 6.31: 1000BASE-T überträgt bidirektional und gleichzeitig auf 4 Aderpaaren

c) Halbduplex Betrieb

Mit Gigabit Ethernet ist auch ein Halb-Duplex-Betrieb mit einem Shared Medium möglich, wobei die gleichen Regeln und Verfahren gelten wie bei 100BASE-T.

Durch die zehnmal höhere Übertragungsrate von Gigabit Ethernet würde aber die Ausdehnung einer Collision Domain auf 25m schrumpfen. Um die geforderten 100m ohne Late Collisions zu erhalten, musste die minimale Frame-Grösse von 64 Byte auf 512 Byte erhöht werden. Dies erfolgt für den Anwender transparent mit einer so genannten **Extra Carrier Extension**. Frames, die grösser als 511 Byte sind, werden nicht erweitert.

Besteht der Datenverkehr aus lauter kleinen Frames, so wird durch die Carrier Extension der Durchsatz verschlechtert. Um dem entgegenzuwirken, wurde das **Frame Bursting** Verfahren eingeführt. Damit lassen sich mehrere kleine Frames von einem Absender zu einem Sender in einem Frame zusammen verschicken, womit die Frame Extension kleiner wird oder ganz wegfällt.

Im Vollduplex-Betrieb ist die Extra Carrier Extension nicht erforderlich, da keine Kollisionen auftreten. Da es kaum Gigabit-Ethernet-Installationen geben dürfte, die nicht mit vollduplexfähigen Switches betrieben werden, hat die Carrier Extension wohl keine praktische Bedeutung.

6.2.4 10 GBit/s-Ethernet-Systeme (Gigabit Ethernet)

Bezeichnung / Norm	Medium	Max. Distanz (Segmentlänge)	Bemerkungen
10GBASE-LX4	MMF (1310nm)	300m	Wide Wave Division Multiplexing (WWDM)
	SMF (1310nm)	10km	
10GBASE-SR	MMF (850nm)	65m	Serielles LAN
	10GbE MMF	300m	
10GBASE-LR	SMF (1310nm)	10km	Serielles LAN
10GBASE-ER	SMF (1550nm)	40km	Serielles LAN
10GBASE-SW	MMF (850nm)	65m	Serielles WAN
	10GbE MMF	300m	
10GBASE-LW	SMF (1350nm)	10km	Serielles WAN
10GBASE-EW	SMF (1550nm)	40km	Serielles WAN
10GBASE-CX4	8fach Koax	15m	
10GBASE-T	UTP/STP	55-100m	Cat. 5e bis Cat. 7

Der 10 Gigabit Ethernet Standard (IEEE 802.3ae) wurde im Juli 2002 verabschiedet. 10 Gigabit Ethernet hat eine Bitrate von 10'000 MBit/s. Dabei bleibt wie üblich die Durchgängigkeit zu bestehenden Ethernet-Standards gewahrt. Das heisst, 10 Gigabit verwendet das gleiche Frame-Format und die gleiche Frame-Grösse, wie vorhandene IEEE 802.3-Netze. 10 Gigabit Ethernet unterstützt aber nur noch Voll-Duplex-Betrieb.

Zusätzlich wird aber auch eine Schnittstelle zu den Metropolitan Area Networks (MAN) und Wide Area Networks (WAN) geschaffen. Der 10 Gigabit Ethernet Standard definiert dazu 2 PHY-Layer. Der LAN-PHY-Layer arbeitet mit exakt 10 Gbit/s. Der WAN-PHY-Layer arbeitet mit 9,58464 GBit/s und ist damit kompatibel mit dem wichtigsten Transportprotokoll für WAN-Backbones (SONET OC-192c und SDH VDC-4-64c). Unter dem Namen 10GBASE-CX4 ist ein Verfahren standardisiert, das eine 10Gbit/s Übertragung über ein 15m langes 8 paariges Twinax-Kabel erlaubt. Seit Juni 2006 ist unter der Bezeichnung 10GBASE-T auch ein 10 Gigabit Ethernet für diverse Twisted Pair Verkabelung spezifiziert.

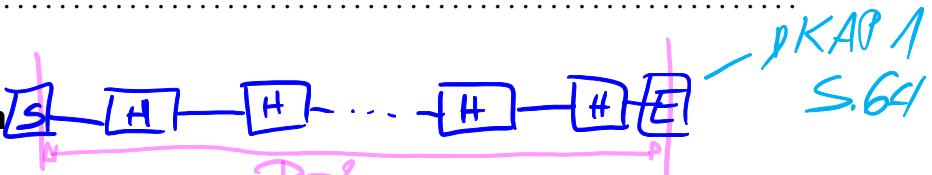
6.3 Übungen

6.3.1 802.3-Normierung

Was bedeuten in den Bezeichnungen 10BASE2 resp. 10BASE-T die Teile:

- 10
nominelle Bitrate
- BASE
Basisband Modulation -> **BROAD = Broadband Modulation hat sich nie durchgesetzt**
- 2
10Base2 -> Thin Ethernet 10Base5 -> ThickEthernet
- -T
10Base-T -> Twisted Pair

6.3.2 LAN-Erweiterungen



- $t_{frame} = \frac{64 \cdot 8 \cdot 10^6}{100 \cdot 10^6} = 512 \mu s$
- a) Mit Hilfe von Hubs (des KT-Labors) soll ein möglichst grosses Netz entworfen werden, ohne dass Late Collisions auftreten. Als Basis dienen die Messungen im Labor. Zudem verwendet man: Segmentlänge $\leq 100m$ und nehme an, dass die Ethernet-Interfaces der Knoten keine Verzögerung haben. $t_{frame} \geq 2 * t_{transfer}$

- b) Worin besteht der Unterschied zwischen einem Up-Link-Port eines 10BASE-T-Repeater

- und einem „normalen“ Repeater-Port?

- und einem 10BASE-T-Port eines Knotens (PC-Interface-Karte)?

- c) Welche Netzelemente behandeln Broadcast-Frames anders als Unicast-Frames?

- | | |
|---|--|
| <input type="radio"/> Sender-Netzinterface(-Karte) | <input type="radio"/> Hubs |
| <input type="radio"/> Empfänger-Netzinterface(-Karte) | <input type="radio"/> Store-and-Forward-Bridge |
| <input type="radio"/> Repeater | <input type="radio"/> On-the-Fly-Switch |

d) Man gebe die Aufgaben an von Normal- und Fast-Link-Puls.

-
-

e) Bei 100BASE-TX ist die maximale Ausdehnung einer Collision-Domain 200m (max. Distanz zwischen zwei Knoten). Warum ist es dann möglich, (z.B. bei 100BASE-FX) im Full-Duplex-Mode 2km zwischen zwei Knoten zu haben?

.....
.....

f) Wie gross ist bei einer On-The-Fly-Bridge theoretisch die minimale Frame-Verzögerungszeit (gemessen mit Preamble). Begründung?

.....
.....

g) Wie viele Frames pro Sekunde können auf einem 100 Base-TX Ethernet mit bei Full Duplex pro Richtung übertragen werden, wenn die Framelänge ohne Präambel und SFD 1000 Byte beträgt?

7

Internet Protokolle des Network Layers

In diesem Kapitel werden Konzepte und Protokolle des Network Layers anhand der „Mutter aller Netze“, dem weltweiten **Internet**, vorgestellt.

7.1 Network Layer

Der Network Layer stellt einen universellen Dienst für den Datenaustausch über mehrere heterogene Netze bereit (siehe OSI-Modell Abschnitt 1.5.4). Ziel ist ein nahtlos ineinander greifendes Kommunikationssystem, an das beliebige Knoten angeschlossen werden können. Jedem Knoten wird eine (hardware-unabhängige) Adresse zugewiesen, und jeder soll in der Lage sein, Pakete mit den anderen auszutauschen.

Da die einzelnen Netzwerktechnologien untereinander schlecht verträglich sind, mussten Konzepte entwickelt werden, um die heterogene Netze zu verbinden. Die Verbindung der verschiedenen Netze erfolgt hardwaremäßig mittels spezieller Rechner, die **Router** oder **Gateway**¹ genannt werden.

Die Protokolle des Network Layers verbergen die technischen Details der zugrunde liegenden physischen Netze, Verbindungsstrecken und Router (siehe Abbildung 7.1). Das resultierende Network Layer-Kommunikationssystem wird darum auch als **virtuelles Netz** oder **Internet**² bezeichnet, weil es die Illusion eines einheitlichen Universalnetzes schafft, das in Wirklichkeit nicht existiert. Dafür sind technische Spezialitäten der Teilnetze (Verkehrspriorisierung etc.) nur noch bedingt nutzbar.

¹In den RFC (Internet Normen) werden Router generell als Gateways bezeichnet. In der üblichen Terminologie decken Gateways aber alle sieben Schichten des OSI-Modells ab und dienen der Protokollumsetzung zwischen zwei Applikationen.

²Das Internet-Konzept soll nicht mit dem weltweiten Netz gleichgesetzt werden, auch wenn dieses seinen Namen sicher daher bezogen hat.

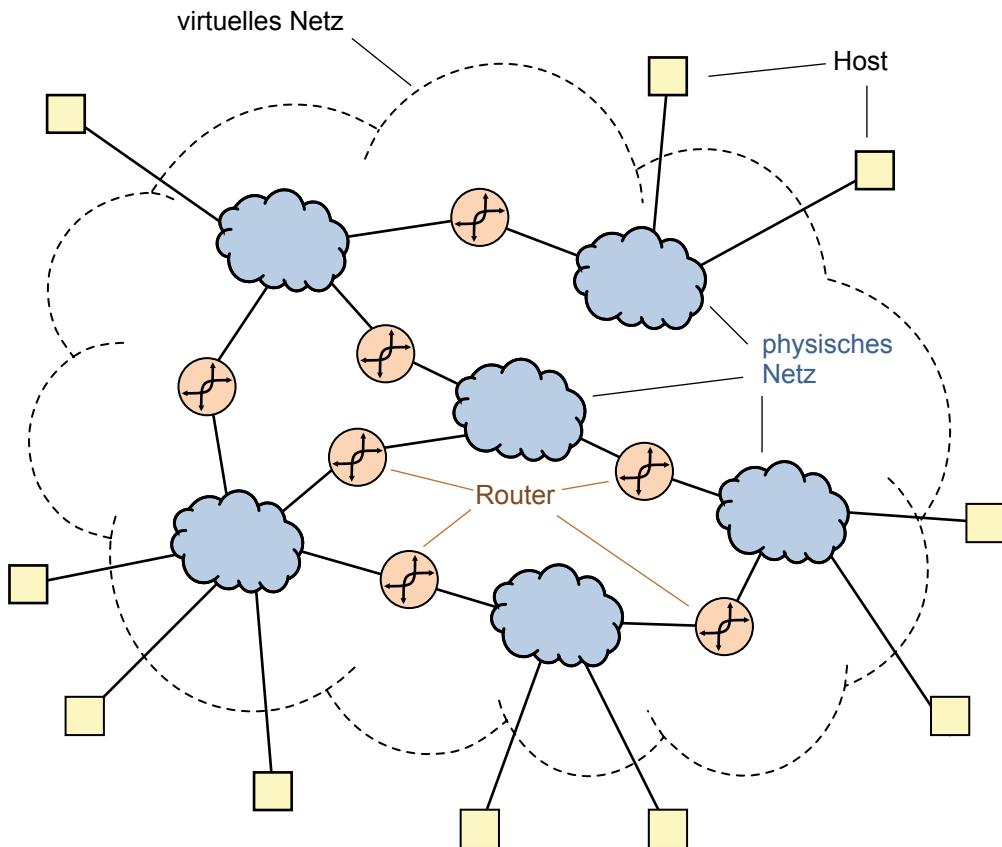


Abbildung 7.1: Das Internet erzeugt die Illusion eines einzigen Netzes

7.1.1 Adressierungsschema

Ein **flacher Adressraum**, wie er auf dem Data Link Layer üblich ist, beschränkt die Ausbaufähigkeit eines Netzes. Die MAC-Adresse, die bei Ethernet verwendet wird, identifiziert ein Gerät zwar eindeutig, sagt aber nichts über dessen Position im Netz aus. Für grosse Netze ist dieses flache Adressierungsschema nicht geeignet, denn soll eine Bridge richtig funktionieren, muss sie von allen Knoten wissen, über welches Port sie erreichbar sind. In grossen Netzen führt dies zu entsprechend grossen Adresstabellen in *jeder* Bridge, was technisch aufwendig und entsprechend teuer wäre.

Ein **hierarchischer Adressraum** ist dafür geeigneter, da er aus einer beschränkten Anzahl von Subnetzen besteht, die ihrerseits wiederum aus Subsubnetzen bestehen usw.. Eine typische hierarchische Adresse ist die Postanschrift. Die Postverteilung ist relativ einfach, da bekanntlich die erste Zeile (von unten nach oben) das Land angibt, die nächste die Ortschaft, die dritte Straße/Hausnummer und die letzte die Person. Es ist leicht einzusehen, dass eine Postverteilung aufgrund einer flachen Adresse, wie z.B. der AHV-Nummer, wesentlich aufwendiger und nur für kleine Gruppen möglich wäre (Abbildung 7.2).

Die hierarchische Adressierung vereinfacht das Routing, stellt aber auch einen gravierenden Nachteil dar. Wechselt ein Host das Netzwerk, muss er eine neue Adresse bekommen.

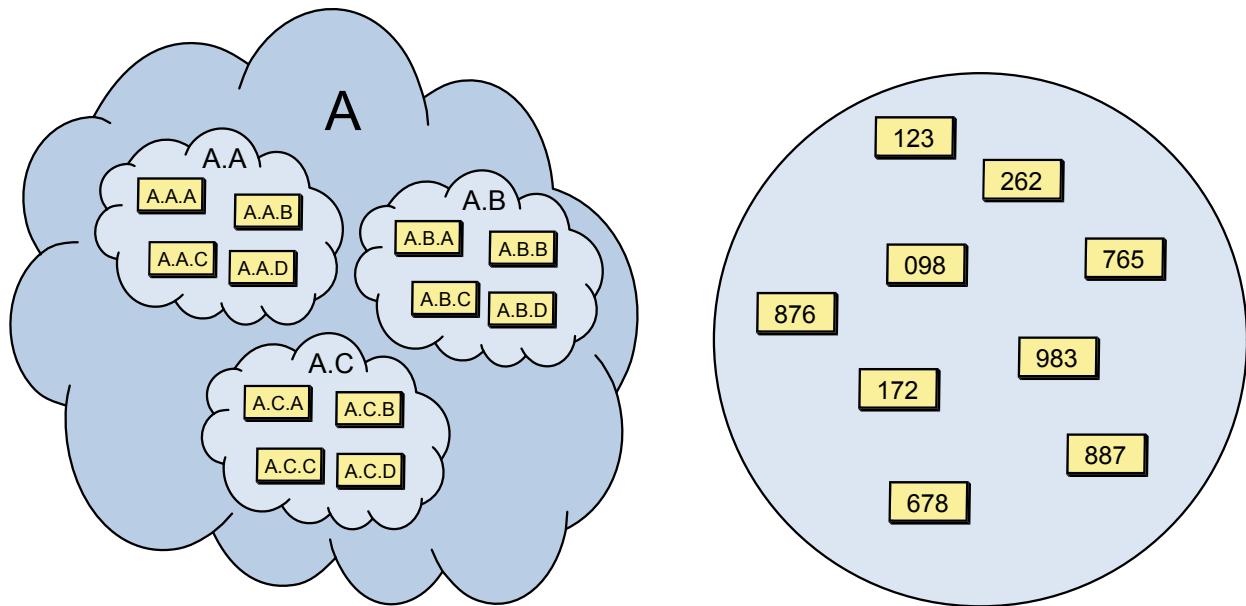


Abbildung 7.2: Hierarchischer Adressraum und flacher Adressraum

7.1.2 Router

Router sind Komponenten, die es erlauben, Subnetze miteinander zu verbinden. In diesem Abschnitt werden nur grob die Eigenschaften von Routern erklärt. Im Abschnitt 7 wird anhand des Internet-Protokolls (IP) das Funktionsprinzip detailliert erläutert.

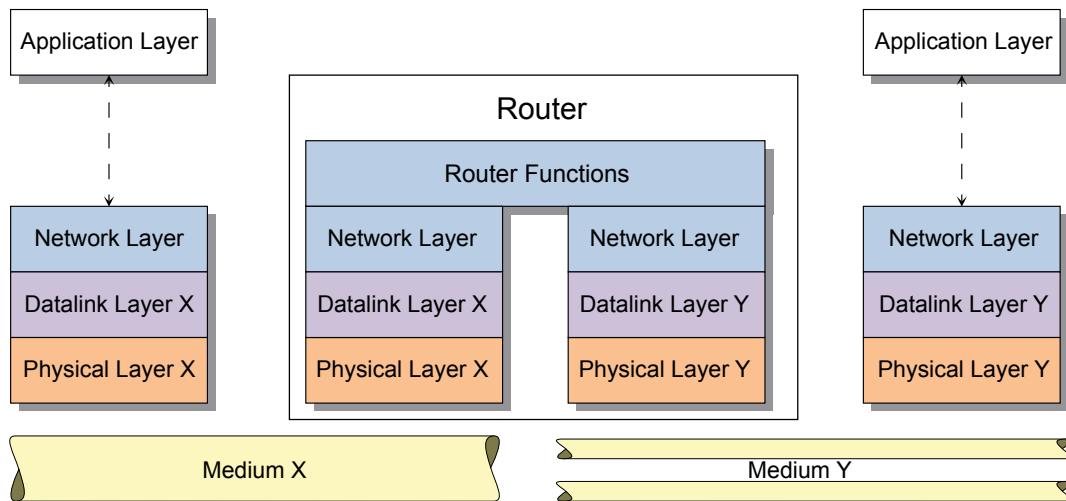


Abbildung 7.3: Router arbeiten auf dem Network Layer

Router haben eine ähnliche Funktion wie Bridges, allerdings arbeiten sie auf dem Network Layer (Abbildung 7.3). Im Gegensatz zu Transparent-Bridges empfangen Router jedoch nur Pakete, die (auf dem Datalink Layer) direkt an sie adressiert sind; d.h. sie müssen beispielsweise an den Router mit dessen MAC-Adresse geschickt werden.

Die Weiterleitung der Datenpakete erfolgt anhand einer Network Layer Adresse und nicht aufgrund einer Data Link Layer Adresse (z.B. MAC-Adresse). Damit kann eine globale, vom Übertragungssystem unabhängige Adressierung eingeführt werden.

Sollen Netze unterschiedlicher Technologien, also mit verschiedenen Medien, Adressschemata oder Rahmenformaten gekoppelt werden, muss der Router in jedem Netz ein Interface und den Protokoll-Stack der entsprechenden Technologie besitzen. In Abbildung 7.3 könnte beispielsweise der linke Pfad X Ethernet sein und der rechte Pfad Y ein XDSL-Zugangsnetz (Abbildung 7.4).

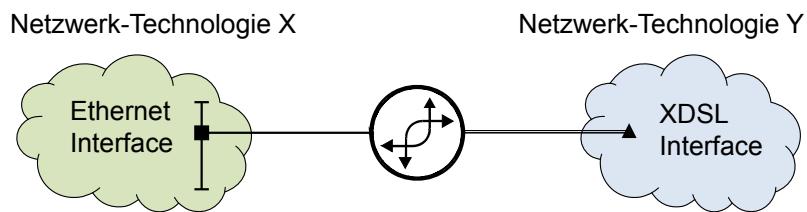


Abbildung 7.4: Router können heterogene Netze koppeln

a) Gegenüberstellung von Bridging und Routing

Gegenüber Bridges haben Router folgende Vorteile:

- Router benutzen immer den optimalen Pfad für die Verbindung zwischen zwei beliebigen Punkten im Netz. In verzweigten Netzen, insbesondere in WANs, führen sie Daten effektiver zum Ziel.
- Mit Hilfe von Routern können verschiedene Netzbereiche logisch und auch verkehrsmässig sehr gut voneinander getrennt werden. Dies ist vor allem bei grossen Netzen sehr wichtig.
- Die Anzahl der Stationen, die mit Hilfe von Routern vernetzt werden können, ist praktisch unbeschränkt.
- Das Trennen verschiedener Netzbereiche erhöht die Sicherheit.
- Das Aufteilen grosser Netze in Unterbereiche verbessert die Übersicht und vereinfacht die Verwaltung.
- Router wirken als Barriere für die auf Schicht 2 häufig verwendeten Broadcast-Meldungen. Dies verhindert wirkungsvoll so genannte Broadcast-Storms (lawinenartige Anhäufungen von Broadcast-Meldungen).

Als Nachteile der Router können aufgeführt werden:

- Leistungsfähige Router sind meist teurer als Bridges.
- Router sind oftmals nicht einfach zu konfigurieren. Im Gegensatz dazu müssen Bridges kaum konfiguriert werden. Einmal angeschlossen, erledigen sie sofort ihre Aufgaben.
- Es gibt Protokolle, die sich nicht routen lassen, da sie über keine Schicht 3 verfügen. Solche „Nonroutable Protocols“ sind beispielsweise die meisten Automatisierungs-Protokolle. Für den Transport dieser Protokolle sind Bridges oder Hubs erforderlich.

b) Routing

Router haben zwei Hauptaufgaben, die getrennt betrachtet werden können. Das **Routing** besteht darin, den optimalen Weg (Route) im Netz zu bestimmen. Das **Forwarding** ist die Weiterleitung der Datenpakete entlang dieser Routen.

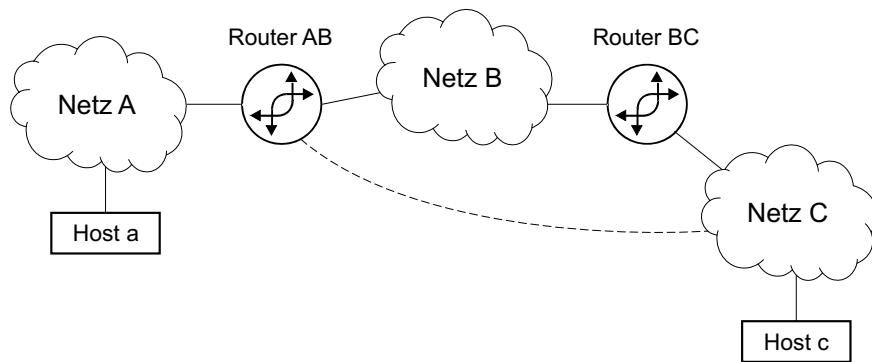


Abbildung 7.5: Router müssen die Netztopologie kennen

Um die besten Routen bestimmen zu können, müssen Router die Topologie des Netzes kennen. Will zum Beispiel (in Abbildung 7.5) der Host a dem Host c ein Paket schicken, so wird er es an den Router AB adressieren, wie er es mit allen Paketen tut, die das Netz A verlassen. Der Router AB muss wissen, dass das Netz C über den Router BC erreichbar ist, der am Netz B angeschlossen ist. Hätte der Router AB die (gestrichelt gezeichnete) direkte Verbindung zum Netz C, so wäre diese vorzuziehen.

Routen können statisch angegeben oder dynamisch bestimmt werden. Um dynamisch immer die besten Routen bestimmen zu können, müssen die verschiedenen Router miteinander kommunizieren. Diese Kommunikation geschieht über so genannte **Routing Protocols**³. Für einzelne Netzwerkprotokolle (Routed Protocols) existieren im Allgemeinen verschiedene Routing-Protokolle. Damit Router verschiedener Hersteller miteinander kommunizieren können, wurden Anstrengungen unternommen, die Routing-Protokolle zu normieren.

Normierte Routing-Protokolle für IP-Netzprotokolle:

- Routing Information Protocol (RIP)
- Exterior Gateway Protocol (EGP)
- Open Shortest Path First (OSPF)
- Border Gateway Protocol (BGP)

Von den oben erwähnten IP-Protokollen ist das RIP sehr verbreitet. Das RIP wird verwendet, um innerhalb eines Subnetzes zu routen, während das EGP für das Routing zwischen einzelnen Subnetzen eingesetzt wird. Das OSPF ist das neueste und leistungsfähigste IP-Routing-Protokoll. Es

³Der Begriff Routing Protocol darf nicht mit **Routed Protocol** verwechselt werden. Ein Routed Protocol ist das Netzwerkprotokoll, das die Nutzdaten transportiert (also z.B. das Internet Protokoll).

wurde für den Einsatz in sehr grossen Netzen entwickelt und ist in der Lage, in einem hierarchischen Netzverbund zu arbeiten.

Normierte Routing-Protokolle für OSI-Netzprotokolle:

- Intermediate System - End System (IS-ES)
- Intermediate System - Intermediate System (IS-IS)

Der Unterschied zwischen den beiden OSI-Routing-Protokollen ist, dass das IS-IS zwischen Knoten routet, die selber weiterrouten können, während das IS-ES zu einem Endgerät hin routet, von dem aus nicht mehr weitergeroutet wird.

Trotz den normierten Protokollen verwenden immer noch viele Hersteller ihre proprietären Routing-Protokolle, die nicht mit Routern fremder Anbieter zusammenarbeiten können.

7.1.3 Brouter und Layer 3 Switches

Zur Verbesserung der Flexibilität haben heute alle namhaften Hersteller von Router-Systemen Bridging-Funktionen in ihre Router eingebaut. Ebenso gehen Bridge-Hersteller dazu über, Systeme anzubieten, welche in der Lage sind, einzelne Protokolle zu routen. Man spricht dann vom so genannten **Brouter (Bridging Router)** oder **Layer-3-Switches**.

7.2 Die Geschichte der TCP/IP-Protokollfamilie

1957 startete Russland (UDSSR) den Sputnik - den ersten künstlichen Satelliten. Dieser Schock veranlasste die Amerikaner zur Gründung der Advanced Research Projects Agency (ARPA) als Teil des US-Verteidigungsministeriums (DoD - Department of Defense).

Im Laufe der sechziger Jahre wurde von der ARPA ein Netzwerk aufgebaut, das im Jahr 1972 auf 37 Computer angewachsen war und ARPANET genannt wurde. Das Forschungsnetz wurde weiterhin vom DoD gefördert und verband zunächst Hunderte von Universitäten und Regierungsbehörden über Mietleitungen.

Im Jahr 1973 wurde durch die U.S. Defense Advanced Research Projects Agency (DARPA) ein Forschungsprojekt gestartet, um Technologien und Techniken zur Verbindung von verschiedenen Paket-basierten Datennetzen zu erforschen. Grund war die Besorgnis des US-Verteidigungsministeriums. Es galt darum, das Netz gegen die Zerstörung von Teilnetzen überlebensfähig zu machen. Eine Verbindung sollte intakt bleiben, solange die Quell- und Zielknoten funktionierten, auch falls einige der dazwischenliegenden Router oder Übertragungsleitungen zusammenbrechen sollten.

Die Protokollfamilie, die im Folgenden aus dem „Internetworking Project“ entstand, wurde nach den beiden ursprünglichen Protokollen „Transmission Control Protocol“ (TCP) und „Internet Protocol“ (IP), als TCP/IP-Protokoll-Stack benannt; das System wurde unter der Bezeichnung „Internet“ bekannt.

Die bisherige Denkweise der Telefongesellschaften basierte im Wesentlichen auf kontrollierten Umgebungen mit einer möglichst zentraler Steuerung und garantierten Servicequalitäten. Für das Internet mussten somit eine ganze Reihe von neuen Konzepten und Lösungen gefunden werden.

Das TCP/IP-Konzept wurde erstmals 1974 von Vint Cerf und Bob Kahn beschrieben. Dabei spielten die folgenden vier Grundsätze eine entscheidende Rolle:

- Jedes Netzwerk soll für sich selber funktionsfähig sein und für den Anschluss ans Internet sollen keine (Netz-)internen Änderungen notwendig sein.
- Die Kommunikation erfolgt auf der Basis von „best effort“, d.h. so gut es eben geht. Wenn ein Paket es nicht schafft, sein endgültiges Ziel zu erreichen, soll es nach kurzer Zeit von der Quelle nochmals übertragen werden.
- Die Verbindung der Netze erfolgt über „Black boxes“, die später als Router oder Gateways bezeichnet wurden. In den Routern soll über die einzelnen Paketflüsse keine Information gesammelt werden, damit die Router einfach bleiben und sich von den möglichen Fehlerzuständen rasch wieder erholen können.
- Es soll keine zentrale Funktionssteuerung benötigt werden.

Diese Anforderungen führten letztlich zur Wahl eines paketvermittelten Netzes auf der Grundlage von vier Layern (Abbildung 7.6).

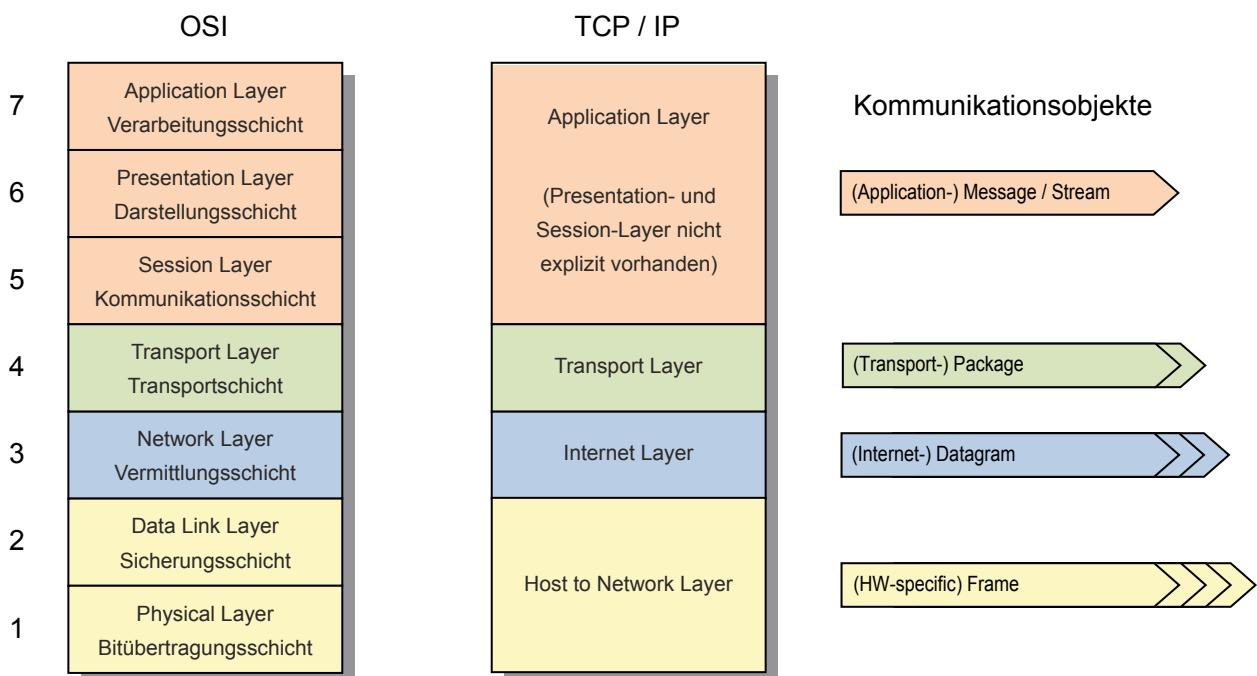


Abbildung 7.6: IP-Layer und Schnittstellenobjekte

Eine Einordnung des TCP/IP-Protokoll-Stacks in das OSI-Referenzmodell war nicht ganz eindeutig, da das TCP/IP-Protokoll vor dem OSI-Referenzmodell entwickelt wurde und letzteren andere Vorstellungen und Zielsetzungen zugrunde lagen. Heute ist das Internet-Protokoll (IP) das Network Layer Protokoll schlechthin. Mit seinem verbindungslosen Dienst bildet es den Kitt, der die gesamte Architektur zusammenhält.

Oberhalb des Internet-Layers befindet sich der Transport-Layer mit den zugehörigen Protokollen UDP (User Datagram Protocol) und TCP (Transmission Control Protocol), die im Kapitel 8 behandelt werden.

Unter der Internet-Schicht herrscht im TCP/IP-Referenzmodell grosse Leere. Abgesehen von dem Hinweis, dass sich ein Host über ein bestimmtes Protokoll am Netz anschliessen muss, um IP-Pakete darüber versenden zu können, wird nicht viel darüber ausgesagt, was dort passiert. In der Praxis bedeutet dies, dass für die relevanten Technologien die Protokolle und das Interface zu Internet Layer definiert werden muss. Dazu gibt es eine *Vielzahl* von entsprechenden RFCs.

7.3 IP Adressierung

Kommunikationsquellen und -senken werden im TCP/IP Jargon als **Hosts** bezeichnet. Diese können beliebige netzwerkfähige Geräte wie Rechner (PC, Workstation oder Grossrechner), Terminal-/Printer-Server oder Router sein.

Wie im Abschnitt 7.1 erklärt wurde, stellt das Internet für den Benutzer ein virtuelles Netz (Virtual Network) dar, wobei weder der Benutzer noch die Applikationsprogramme die zugrunde liegenden Netze und Router kennen müssen. Die Basis dafür ist die weltweite, einheitliche Adressierung, welche die Details der unterlagerten, physischen Netze verbirgt. Konzeptionell besitzt jeder Host mindestens eine Internet-Adresse und jede Internet-Adresse ist einem Host zugeordnet; genauer gesagt einem Interface.

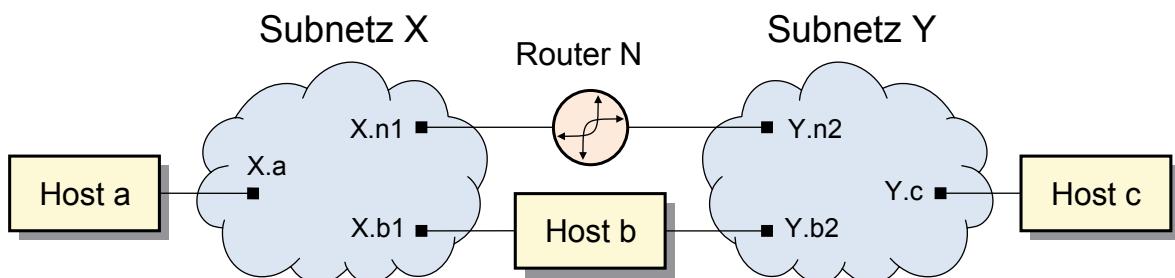


Abbildung 7.7: Eine IP-Adresse steht für ein IP-Interface

Es gibt auch Hosts (z.B. Router und Server) mit mehreren Interfaces, die in mehreren Subnetzen angeschlossen sein können. Solche Hosts werden als **Multi-homed Hosts** bezeichnet. Der Router N in der Abbildung 7.7 muss zwei Interfaces mit je einem Anschluss im Subnetz X und im Subnetz Y besitzen. Beide Anschlüsse benötigen je eine Adresse (X.n1 und Y.n2) vom angeschlossenen Subnetz. Das gleiche gilt für den Host b, der z.B. zur Verbesserung der Performance oder der Verfügbarkeit in beiden Subnetzen angeschlossen sein könnte.

Eine IP-Adresse bezeichnet also ein Netzwerk-Interface, und Multi-homed Hosts benötigen mehrere IP-Adressen.

7.3.1 Darstellung und Aufbau der Adresse

Die IP-Protokolle verwenden generell die Big-Endian-Konvention (höchstwertigstes Byte wird zuerst übertragen). Die Reihenfolge der Bit-Übertragung ist nicht definiert, da von der verwendeten Technologie abhängig. In den Darstellungen ist das LSB an seinem gewohnten Platz (rechts), das MSB links.

MSB								LSB							
1. Byte (Oktett)		2. Byte (Oktett)		3. Byte (Oktett)		4. Byte (Oktett)		1. Byte (Oktett)		2. Byte (Oktett)		3. Byte (Oktett)		4. Byte (Oktett)	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1								2							
									3						
										4					
5								6							
									7						
9								10							
									11						
										12					

Abbildung 7.8: Byte-Wertigkeit im IP-Umfeld

Etwas ungewohnt sind die Darstellung und die Nummerierung der Bits (siehe Abbildung 7.8): Es werden immer 4 Bytes auf eine Zeile geschrieben. Die 32 Bits werden aber von links nach rechts nummeriert, so dass das MSB die Nummer 0 erhält und das LSB die Nummer 31.

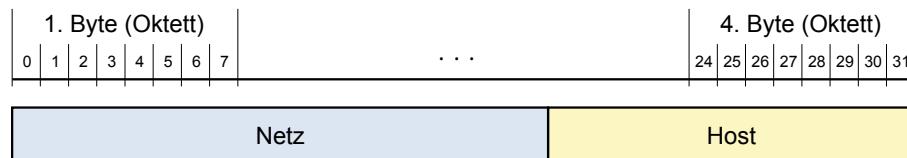


Abbildung 7.9: Aufbau einer IP-Adresse

Eine IP-Adresse ist eine eindeutige 32 Bit grosse Zahl. Aufbau der Adresse und Verwendung der Nummern sind im RFC 1700 beschrieben. Jede Adresse besteht aus einer Netz- und einer Host-Nummer (Abbildung 7.9). Alle Hosts im gleichen Netz haben die gleiche Netz-Nummer aber unterschiedliche Host-Nummern. In unterschiedlichen Netzen kann die Host-Nummer jedoch gleich sein.

1. Byte (Oktett)		2. Byte (Oktett)		3. Byte (Oktett)		4. Byte (Oktett)	
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
1	0	0	0	0	0	0	0
1	0	0	1	1	0	1	0
0	0	0	1	1	0	1	0
1	1	1	1	1	0	1	1

128 . 154 . 26 . 246

Abbildung 7.10: Beispiel zur Schreibweise der IP-Adresse

IP-Adressen werden typischerweise in der Punkt-Schreibweise dargestellt. Wie Abbildung 7.10 zeigt, werden dabei die vier Bytes der Adresse als vorzeichenlose Dezimalzahlen mit Punkten abgetrennt dargestellt.

7.3.2 Subnetze und Subnetzmasken

Aus technischen Gründen wurden ursprünglich **Netzklassen** mit fixen, definierten Größen eingeführt. Auf dieses so genannte **Classful-Routing** wird im Abschnitt 7.3.4 eingegangen.

Das heute übliche **Classless-Routing** mit Netzmasken erlaubt eine flexible Zuteilung von Adressräumen in Zweierpotenzschritten. Um die Größen von Subnetzen flexibel einzustellen, benötigt es dafür einen Parameter, die **Subnetzmaske**, welche die Grenze zwischen Netz- und Host-Adresse festlegt.

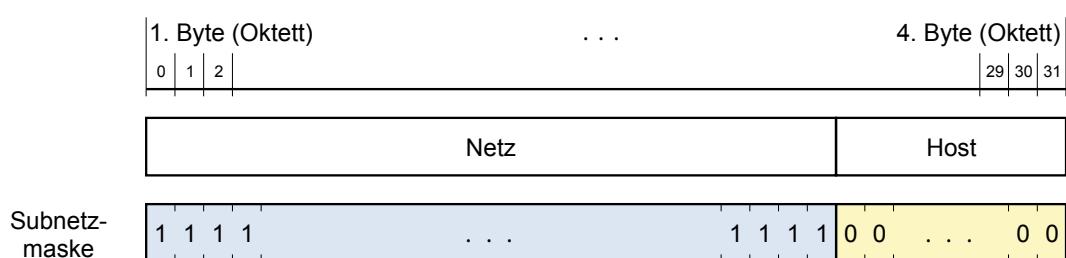


Abbildung 7.11: Darstellung von Subnetzmasken

Die Subnetzmaske gibt mit 1 an, welche Bits für die Netz-Adresse und mit 0 die Bits, welche für die Host-Adresse verwendet werden. Wie das Beispiel in der Abbildung 7.12 zeigt, wird die Subnetzmaske analog zur IP-Adresse in der Punkt-Notation (vier Bytes als vorzeichenlose Dezimalzahlen mit Punkten abgetrennt) dargestellt.

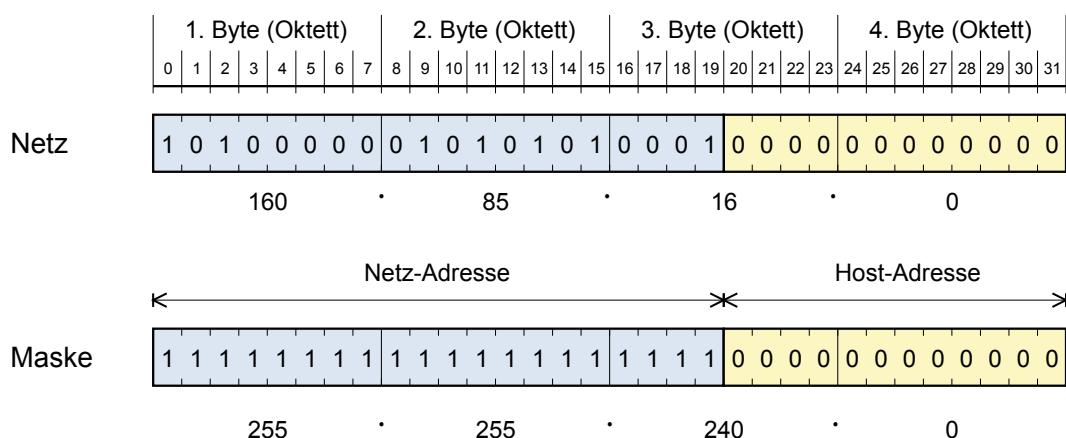


Abbildung 7.12: Subnetz des KT-Labors: Adresse und Subnetzmase

Alternativ zur Maske in der Punkt-Notation wird die Bit-Anzahl der Netzadresse angegeben. Das Subnetz von Abbildung 7.12 mit der Netzadresse 160.85.16.0 und der Subnetzmaske 255.255.240.0 würde damit als 160.85.16.0/20 geschrieben.

Da die Subnetzmaske keine Lücken aufweisen kann (1 stehen immer links, 0 immer rechts), können die Bytes einer Subnetzmaske nur die folgenden 9 Werte annehmen:

255 (1111'1111)	254 (1111'1110)	252 (1111'1100)	248 (1111'1000)
240 (1111'0000)	224 (1110'0000)	192 (1100'0000)	128 (1000'0000)
0 (0000'0000)			

7.3.3 Netz- und Broadcast-Adresse

Die **Netzwerk-Adresse** ist die tiefste Adresse eines Netzes. Die Host-Nummer (Abbildung 7.9) besteht (binär betrachtet) aus lauter Nullen. Die Netzwerk-Adresse ist reserviert und darf *nicht* an Knoten vergeben werden.

Mit Hilfe der **Broadcast-Adresse** können alle Knoten eines Netzes gemeinsam adressiert werden. Heute ist die Broadcast-Adresse als die höchste Adresse eines Netzes definiert. Die Host-Nummer (Abbildung 7.9) besteht also (binär betrachtet) aus lauter Einsen⁴ (All Ones Broadcast).

Mit Hilfe der Subnetzmaske können die Netz- und Broadcast-Adressen effizient bestimmt werden.

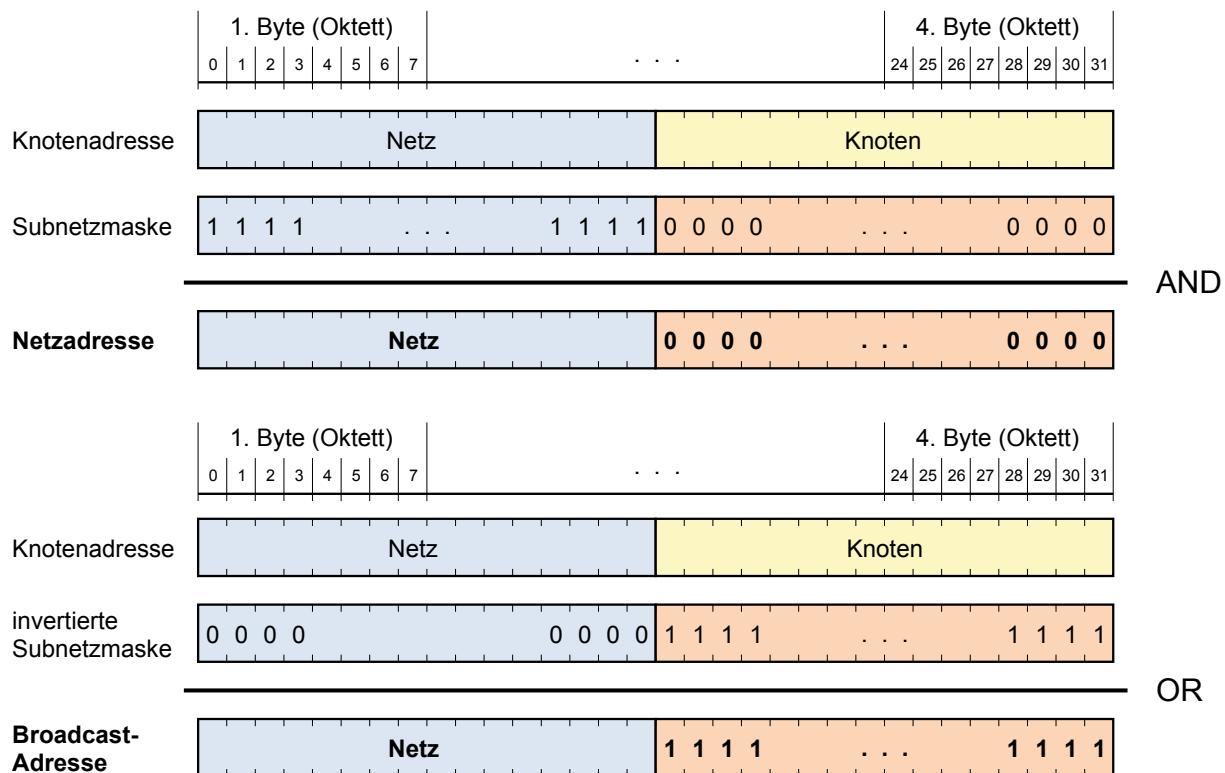


Abbildung 7.13: Anwendung der Subnetzmaske

Abbildung 7.13 zeigt, dass die AND-Verknüpfung einer beliebigen Knotenadresse mit der Subnetzmaske die Netz-Adresse ergibt. Durch eine OR-Verknüpfung mit der invertierten Subnetzmaske findet man die Broadcast-Adresse.

⁴ Teilweise wurde die heutige Netzwerk-Adresse als Broadcast-Adresse verwendet (All Zero Broadcast). Es ist möglich, dass einzelne Hosts diese Adresse immer noch als Broadcast-Adresse interpretieren.

7.3.4 Classful-Routing

Aus technischen Gründen wurden ursprünglich verschiedene **Netzklassen** mit fixen, definierten Größen eingeführt. Der Vorteil des **Classful-Routing** ist, dass anhand der ersten Adress-Bits die Klasse und damit Größe eines Netzes bestimmt werden. Es wird also keine Netzmaske benötigt, was ein effizientes Routing wesentlich erleichtert.

Abbildung 7.14 zeigt, wie mit Hilfe der höchstwertigsten Bits der Adresse die Größe des Netzes einfach bestimmt werden kann.

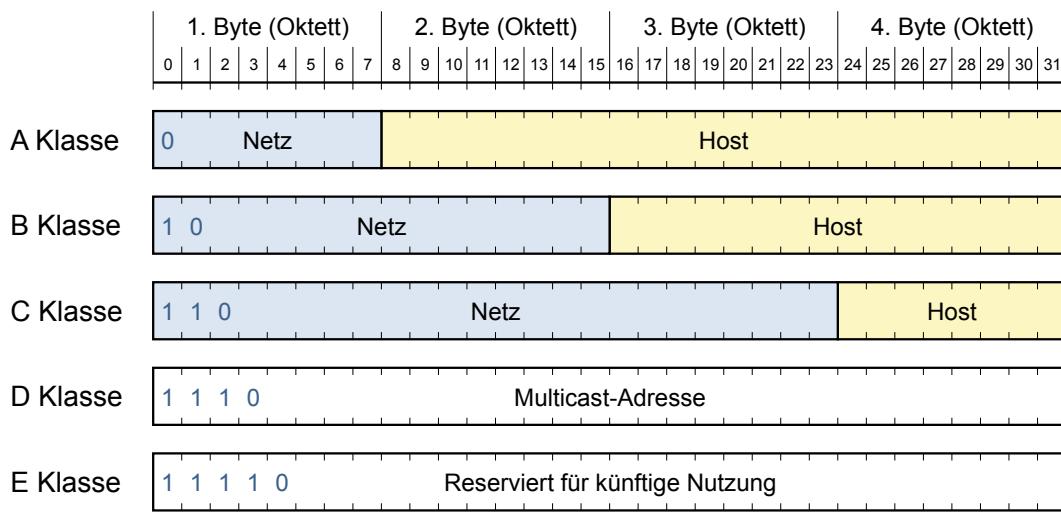


Abbildung 7.14: Binäre Betrachtungsweise der IP-Adressformate

Jede Klasse umfasst eine Zweierpotenz von Adressen (A-Klasse 2^{24} , B-Klasse 2^{16} , C-Klasse 2^8), wovon die tiefste (Netzwerkadresse) und die höchste (Broadcast-Adresse) für Knoten *nicht* verwendet werden können. Die folgende Tabelle zeigt die Eigenschaften der verschiedenen Netzklassen.

Klasse	Adressbereich	Anzahl Netze	Anzahl Knoten	Netz-Bit Wert	Anz. Potenz
A	1.0.0.0 – 127.255.255.255	128	16'777'214	0	2^{24}
B	128.0.0.0 – 191.255.255.255	16'384	65'534	10	2^{16}
C	192.0.0.0 – 223.255.255.255	2'097'152	254	110	2^8
D	224.0.0.0 – 239.255.255.255	Multicast-Adressen		1110	
E	240.0.0.0 – 247.255.255.255	Reserviert für künftige Nutzung		11110	

Multicast-Adressen (D-Klasse) erlauben eine applikationsspezifische Adressierung von Gruppen (limited Broadcast; wird im Weiteren nicht mehr behandelt.)

Classful-Routing ist effizient und die Adressen sind leicht zu lesen, da die Grenze zwischen Netz- und Host-Adresse in Byte-Schritten verschoben wird. Die Abbildung 7.15 zeigt beispielhaft verschiedene zusammengeschlossene Netze der Klassen A, B und C, die durch unterschiedliche Größen angedeutet werden, sowie einzelne angeschlossene Knoten.

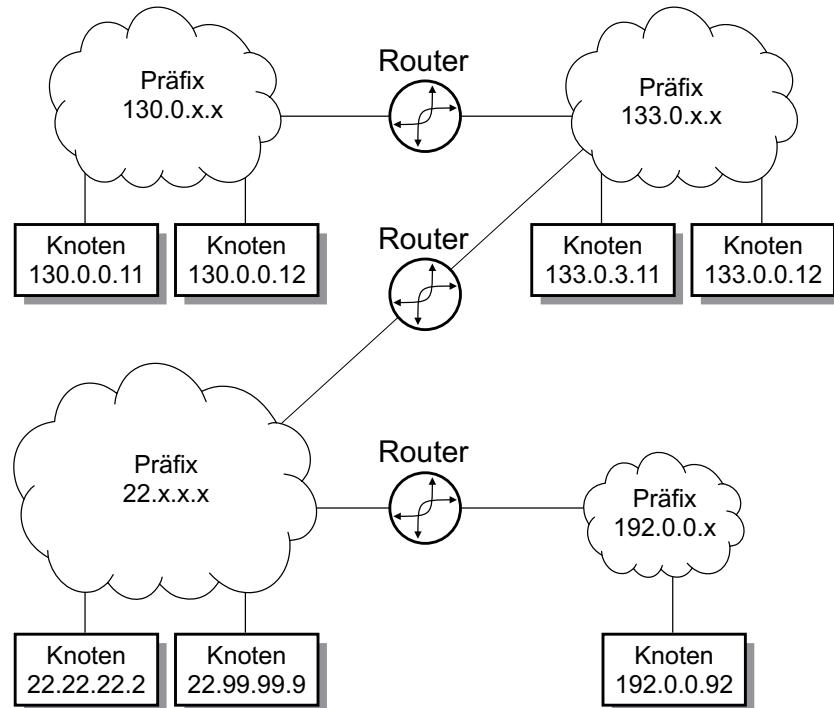


Abbildung 7.15: Beispiel für Classful-Routing

Durch die starke Zunahme der Internet-Knoten in den letzten Jahren ist das Classful-Routing definitiv an seinen Limiten angelangt. Besonders im Bereich der B-Klasse stehen nicht genügend Netznummern zur Verfügung.

Das im Abschnitt 7.3.2 beschriebene, flexible Classless-Routing hat das Problem für einige Zeit entschärft, da für mittelgrosse Organisationen ein C-Netz oft zu klein, ein B-Netz aber eigentlich zu gross ist. Mit Classless-Routing ist es möglich, anstelle von B-Netzen zusammenhängende Blöcke von C-Netzen zu vergeben. Andererseits kann eine Organisation, die mehrere Standorte und ein grosses Netz besitzt, dieses in Subnetze aufteilen.

7.3.5 Verwaltung der Adressen

a) Globale Netzadressen

Wie erwähnt, muss eine IP-Adresse weltweit eindeutig sein. Um Konflikte zu vermeiden, wurden früher Netzadressen von einer zentralen Stelle direkt vergeben.

Heute erfolgt die Vergabe über eine komplizierte, hierarchische Organisation. Zuoberst steht die IANA (Internet Assigned Numbers Authority), die Adressblöcke den fünf regionalen Vergabestellen, den Regional Internet Registries (RIR), zuteilt. Für die europäische und nahöstliche Region (auch die Schweiz) ist dies das RIPE-NCC (Réseaux IP Européens Network Coordination - Network Coordination Centre). Die Netzadressen werden dann an Local Internet Registries zugeteilt (z.B. Switch), die sie an ihre Kunden weitergeben.

b) Private Netzadressen

Für private Netze ohne direkten Internet-Zugang wurden private Netzadressen reserviert, die nicht registriert werden müssen. Pakete mit privaten Adressen werden von der Netzwerk-Software normal behandelt und verschickt. Dies sind aus der Klasse A ein Netz, aus der Klasse B 16 Netze und aus der Klasse C 256 Netze (siehe RFC 1918):

Klasse	Netzadresse	Subnetzmaske
A	10.0.0.0	255.0.0.0
B	172.16.0.0 ... 172.31.0.0	255.255.0.0
C	192.168.0.0 ... 192.168.255.0	255.255.255.0

Loopback-Adressen ermöglichen, die Kommunikation von Programmen auf einem Rechner auszuführen, ohne über ein Netzwerk zu gehen. So können z.B. Programme getestet werden ohne andere Knoten zu stören und auch auf Knoten, die gar keinen Netzzugang haben.

Die Adressen des A-Klassen-Netzes mit der Nummer 127 wurden für diesen Zweck, den so genannten Loopback-Test, reserviert. Im Gegensatz zu den privaten Adressen werden Pakete mit Loopback-Adressen (IP-Adressen der Form 127.x.x.x) nicht auf das Netz gesendet, sondern von einem (emulierten) Loopback-Gerät (unter Linux „/dev/lo“) behandelt.

7.4 Routing

7.4.1 Funktion und Routing-Tabelle

Routing ist eine der wichtigsten Funktionen von IP. Es bezeichnet genau genommen nur die **Suche** eines geeigneten Wegs für ein Datagramm; die eigentliche Weiterleitung wird als **Forwarding** bezeichnet. Umgangssprachlich werden beide Aufgaben unter Routing zusammengefasst.

Im Folgenden wird mit dem Begriff „Router“ ein Host bezeichnet, der Routing und Forwarding beherrscht. Die meisten modernen Betriebssysteme haben diese Fähigkeiten implementiert, und sie können üblicherweise mit einer Betriebssystem-Option ein- oder ausgeschaltet werden. Das Routing wurde bis vor kurzen fast ausschliesslich software-mässig realisiert. Heute gibt es aber auch sehr schnelle Router, welche die Datagramme hardware-mässig behandeln.

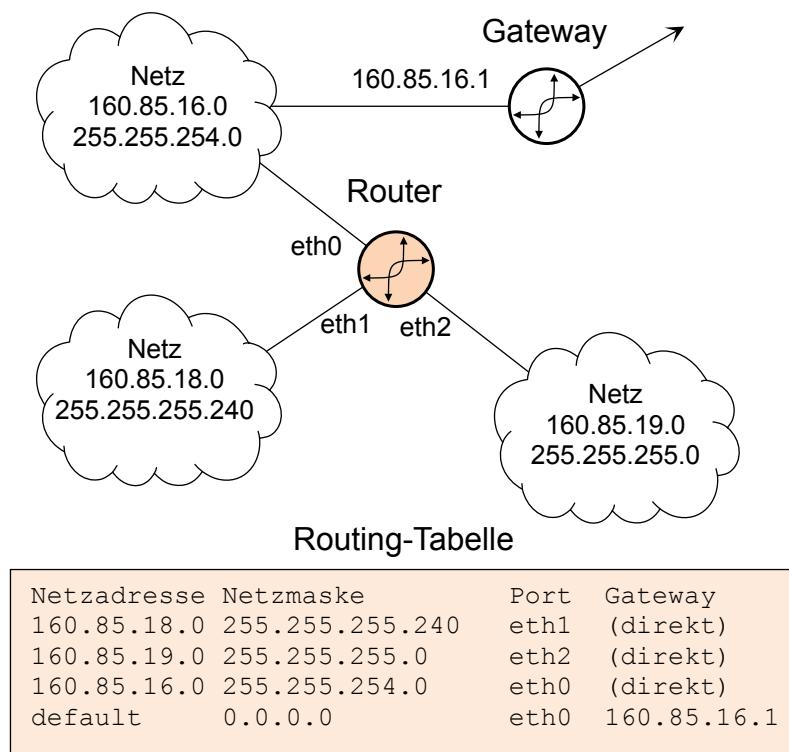


Abbildung 7.16: Beispiel einer Routing-Tabelle

Basis für das Routing ist die **Routing-Tabelle**. Sie bestimmt, über welchen Pfad ein Datagramm weitergeleitet werden soll. Die einzelnen Einträge werden nach der Länge der Netzmaske sortiert (Abbildung 7.16).

Für die Weiterleitung eines Datagramms durchsucht der Router seine Routing-Tabelle von oben nach unten. Die Zieladresse des Datagramms wird mit der Subnetzmaske des Eintrags AND-verknüpft. Stimmt das Resultat mit der Netzadresse überein, wird das Datagramm über das eingetragene Port (Interface) an den entsprechenden Host oder Gateway weitergeleitet.

7.4.2 Route-Befehl zur Anzeige und Manipulationen der Routing-Tabelle

Die Routing-Tabelle kann auf Unix-Systemen mit den Befehlen `route -n` oder `netstat -rn` sowie unter NT und Windows95 mit `route -print` angezeigt werden. (Für Details zu den Spalten Metric, Ref und Use sei auf die Man-Pages verwiesen.)

```
# route -n
Kernel IP routing table
Destination      Gateway        Genmask        Flags Metric Ref    Use Iface
160.85.17.0      0.0.0.0        255.255.255.0  U     0      0          5 eth0
160.85.18.0      0.0.0.0        255.255.255.0  U     0      0          5 eth1
160.85.19.0      0.0.0.0        255.255.255.0  U     0      0          5 eth2
127.0.0.0        0.0.0.0        255.0.0.0      U     0      0          1 lo
default         160.85.17.1    0.0.0.0        UG    0      0          4 eth0
```

Zum Beispiel bedeutet der erste Eintrag in der obigen Tabelle. Das Subnetz 160.85.17.0 ist

- direkt erreichbar (Gateway 0.0.0.0)
- ein Netz mit 256 Adressen (Genmask 255.255.255.0).
- in Betrieb (Flag U = Up)
- über das erste Ethernet-Interface (Iface eth0) erreichbar.

Der zweitletzte Eintrag 127.0.0.0 ist für das interne Loopback-Device, das immer die Bezeichnung lo trägt (siehe Abschnitt 7.3.5).

Der letzte Eintrag in der obigen Tabelle ist so zu lesen, dass alle Datagramme, die nicht vorher eine Route gefunden haben, über das Interface eth0 dem Router 160.85.17.1 geschickt werden sollen. Das Flag G zeigt zusätzlich an, dass es eine Route zu einem Gateway (Router) ist.

Mit dem `route`-Befehl kann die Routing-Tabelle auch verändert werden. Der folgende Befehl definiert, dass das C-Netz 160.85.19.0 direkt über das Interface eth2 erreichbar ist.

```
# route add -net 160.85.19.0 netmask 255.255.255.0 dev eth2
```

Der letzte (Default-Route-)Eintrag wurde erstellt mit:

```
# route add default gw 160.85.17.1
```

Das Device muss hier nicht angegeben werden, da durch einen vorherigen Route-Eintrag klar ist, dass das Netz 160.85.17.1 über das Port eth0 erreichbar ist.

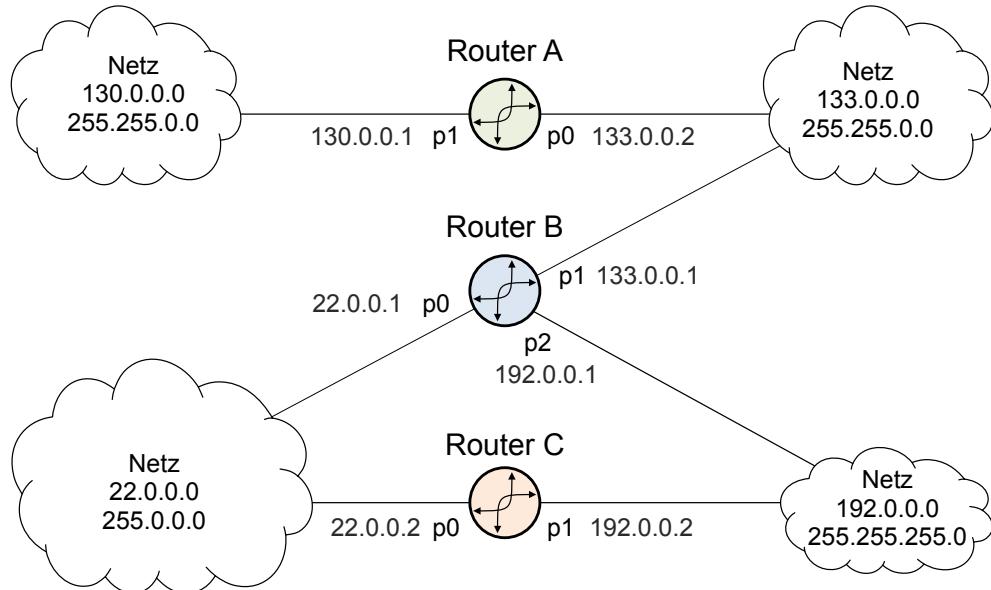
Ein Eintrag kann mit `route del` wieder gelöscht werden.

```
# route del -net 160.85.19.0 netmask 255.255.255.0 dev eth2
# route del -net 160.85.18.0
```

7.4.3 Flaches und hierarchisches Routing

Abhängig von Topologie und Philosophie kann entweder hierarchisch oder flach geroutet werden.

Flaches Routing wird vorzugsweise in den zentralen Teilen stark vermaschter Netze verwendet (siehe Abbildung 7.17). Im Extremfall kennt jeder Router alle möglichen Wege zum Zielnetz und leitet die Datagramme auf dem momentan besten Weg weiter.



Routing-Tabellen

	Netzadresse	Netzmaske	Port	Gateway
Router A	192.0.0.0	255.255.255.0	p0	133.0.0.1
	130.0.0.0	255.255.0.0	p1	(direkt)
	133.0.0.0	255.255.0.0	p0	(direkt)
	22.0.0.0	255.0.0.0	p0	133.0.0.1
Router B	192.0.0.0	255.255.255.0	p2	(direkt)
	192.0.0.0	255.255.255.0	p0	22.0.0.2
	130.0.0.0	255.255.0.0	p1	133.0.0.2
	133.0.0.0	255.255.0.0	p1	(direkt)
	22.0.0.0	255.0.0.0	p0	(direkt)
Router C	192.0.0.0	255.255.255.0	p1	(direkt)
	130.0.0.0	255.255.0.0	p0	22.0.0.1
	130.0.0.0	255.255.0.0	p1	192.0.0.1
	133.0.0.0	255.255.0.0	p0	22.0.0.1
	133.0.0.0	255.255.0.0	p1	192.0.0.1
	22.0.0.0	255.0.0.0	p0	(direkt)
	22.0.0.0	255.0.0.0	p1	192.0.0.1

Abbildung 7.17: Beispiel für flaches Routing

Falls zum Beispiel beim Router B das Port p2 ausfällt, können die Datagramme indirekt über das Port p0 zum Router C geschickt werden. Der Kommunikationsaufwand um diese Routing-Informationen aktuell zu halten, kann allerdings gewaltig sein.

Hierarchisches Routing oder **Default-Routing** wird typischerweise in der Netz-Peripherie eingesetzt. Es erlaubt, die Routing-Tabellen klein zu halten. Beim reinen hierarchischen Routing hat jeder Host nur zwei Einträge in der Routing-Tabelle: einen für das eigene Subnetz und einen Default-Eintrag für „seinen“ Router (siehe Abbildung 7.18).

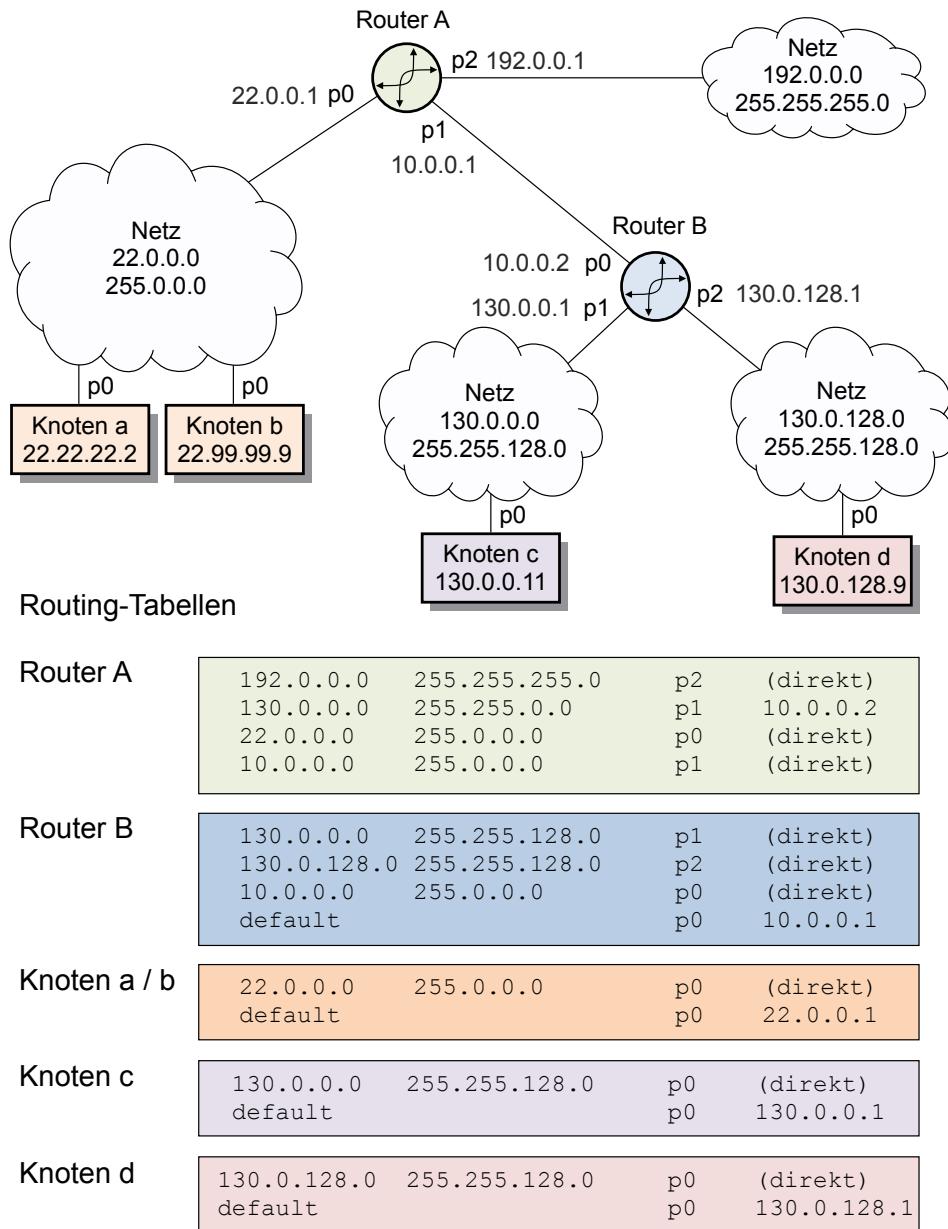


Abbildung 7.18: Beispiel für Default-Routing

Ein Host, der ein Datagramm senden muss, dessen Ziel nicht im selben Subnetz ist, schickt dieses einfach seinem Default-Router. Dieser kennt auch nur die direkt angeschlossenen Subnetze und verfährt analog.

7.5 IP Protokoll

7.5.1 IP-Protokoll-Header

Ein IP-Datagramm besteht aus einem Header und den eigentlichen Nutzdaten. Wie in Abbildung 7.19 gezeigt, besteht der Header seinerseits aus einem 20 Byte grossen Teil mit fester Grösse und einem optionalen Teil mit variabler Grösse.

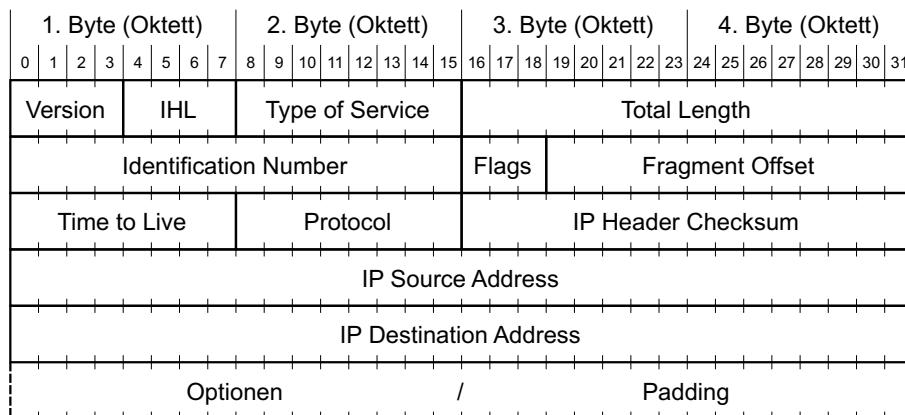


Abbildung 7.19: IP-Protokoll-Header

Im Folgenden werden die Felder des IP-Protokoll-Headers kurz beschrieben.

Version (4 Bit) gibt die Version des IP-Headers an. Damit ist es möglich, mehrere IP-Versionen (z.B. IP-Version 4 und IP-Version 6) parallel zu betreiben.

Internet Header Length (IHL) (4 Bit) gibt die Länge des IP-Headers inklusive dem optionalen Teil in Double Words (32 Bit) an. Die Länge bezeichnet also die Stelle, wo im Datagramm die Nutzdaten beginnen.

Für einen gültigen Header ist die IHL mindestens 5 (keine Optionen) und maximal 15 ($2^4 - 1$), was einer Header-Länge von 60 Byte entspricht. Da der fixe Teil des IP-Headers 20 Byte beträgt, kann der Option-Teil also maximal 40 Byte gross sein.

Type of Service (8 Bit) umfasste historisch eine Reihe von Parametern, die Hinweise zur gewünschten Dienstgüte (**Quality of Service**) geben sollten.

Feld	Position	Wert	Funktion	Erklärung
Precedence	Bits 0-2:	0-7	Low-High Precedence	Vorrangigkeit
D	Bit 3:	0/1	Normal/Low Delay	Verzögerungen
T	Bits 4:	0/1	Normal/High Throughput	Durchsatz
R	Bits 5:	0/1	Normal/High Reliability	Zuverlässigkeit
	Bit 6-7:		Reserved for Future Use	Reserviert

Die Verbesserung eines dieser Parameter kann zur Verschlechterung der anderen führen. So kann z.B. durch das Weiterleiten über eine Satelliten-Verbindung anstelle einer Standleitung die Bandbreite verbessert werden; allerdings auf Kosten der Verzögerungszeit und evtl. auch der Zuverlässigkeit. Die „Type of Service“ Parameter wurden von den Netzanbietern nie durchgängig unterstützt.

Im Zusammenhang mit der Übertragung von zeitkritischen Bild und Ton- Informationen (Audio-/Video-Streaming) gewinnt die Quality of Service zunehmend an Bedeutung. Dieses Feld wurde neu definiert, um diese Applikationen besser unterstützen zu können.

Feld	Position	Funktion	Definition
DSCP	Bit 0-5:	Differentiated Services Codepoints	RFC 2474
ECN	Bits 6-7:	Explicit Congestion Notification	RFC 3168

Total Length (16 Bit) bezeichnet die gesamte Länge des Datagramms in Byte (inklusive Header und Nutzdaten). Die Feldgrösse begrenzt die maximale Grösse eines Datagramms auf $2^{16} - 1 = 65'535$ Bytes.

Für das Internet wird eine Länge von maximal 576 Bytes empfohlen. Längere Pakete sollen nur verwendet werden, wenn sichergestellt ist, dass der empfangende Host genügend grosse Buffer zur Verfügung stellt.

Identification (16 Bit) Der vom Sender zugeteilte Wert erlaubt fragmentierte Datagramme wieder zusammenzustellen, da alle Fragmente den gleichen Wert haben. Die Fragmentierung wird im Abschnitt 7.5.2 erklärt.

Flags (3 Bit) beinhaltet zwei Kontrollflags für die Fragmentierung, die im Abschnitt 7.5.2 erklärt wird.

Feld	Position	Wert	Funktion	Erklärung
	Bit 0:		reserved, must be zero	Reserviert, immer Null
DF	Bit 1:	0/1	May / Don't Fragment	keine Fragmentierung
MF	Bit 2:	0/1	Last / More Fragments	Folgefragmente

Fragment Offset (13 Bit) gibt an, wo innerhalb des Datagramms ein Fragment hingehört. Der Fragment-Offset wird in 8-Byte-Einheiten (64 bits) angegeben, wobei das erste Fragment einen Offset von Null hat.

Aufgrund der Feldgrösse von 13 Bit könnte ein Datagramm theoretisch in maximal $2^{13} = 8192$ Fragmente zerlegt werden. Die Fragmentierung wird im Abschnitt 7.5.2 erklärt.

Time to Live (TTL) (8 Bit) gibt die verbleibende Zeit in Sekunden an, die das Datagramm noch im Internet-System verbleiben darf. Wenn der Wert Null ist, muss das Datagramm verworfen werden. Damit soll verhindert werden, dass Datagramme (z.B. wegen fehlerhafter Routing-Tabellen) endlos im Kreis herumgeschickt werden.

Da es in der Praxis schwierig ist, diese Zeit zu bestimmen, wird im Normalfall das Feld einfach von jedem Router um Eins dekrementiert, auch dann, wenn das Datagramm weniger als eine Sekunde unterwegs war.

Protocol (8 Bit) gibt das übergeordnete Protokoll an, das im Netzdatenteil des Datagramms verwendet wird. Die Protokollwerte sind im RFC 1700 „Assigned Numbers“ definiert.

Einige Beispiele von Protokollen sind:

Protocol	Keyword	Protokollbezeichnung
1	ICMP	Internet Control Message
6	TCP	Transmission Control
17	UDP	User Datagram

Header Checksum (16 Bit) ist – wie der Name sagt – eine Prüfsumme, die nur über den IP-Header gebildet wird. Da einige Felder durch die Router geändert werden, muss die Prüfsumme in jedem Router geprüft und nach dem folgenden Algorithmus neu berechnet werden.

Zur Bildung der Prüfsumme wird Header in Worte (16 Bit) zerlegt. Diese werden invertiert (Einerkomplement) und dann addiert, wobei das zu berechnende Feld „Header Checksum“ als Null angenommen wird. Die nochmals invertierte Summe (16-Bit) ergibt die Prüfsumme.

Source Address (32 Bit) ist die IP-Adresse des Hosts, der das Datagramm ursprünglich abgesandt hat. Die IP-Adresse ist im Abschnitt 7.3 beschrieben.

Destination Address (32 Bit) ist die IP-Adresse des Hosts, der das Datagramm schlussendlich erhalten soll. Die IP-Adresse ist im Abschnitt 7.3 beschrieben.

Options und Padding (variable) wurde vorgesehen, um am IP-Protokoll Erweiterungen vornehmen zu können, ohne das Design ändern zu müssen.

Es hat eine variable Länge und kann auch ganz fehlen. Es ist in sich auch wiederum strukturiert und kann mehrere Optionen enthalten. Zurzeit sind im RFC 791 fünf Optionen definiert, die aber von keinem kommerziellen Produkt unterstützt werden; allerdings wird das Options-Feld von einigen Tools zum Netzwerk-Testen und -Debuggen eingesetzt.

Das Padding-Feld wird verwendet, um den Internet-Header auf ein ganzzahliges Vielfaches von 32 Bit zu füllen. Dazu werden Nullen eingefügt.

7.5.2 Fragmentation and Reassembly

Die maximal übertragbare Paketgrösse ist von der eingesetzten Netz-Technologie auf dem Data Link Layer abhängig und wird zum Beispiel durch die in der Hardware und Software bereitgestellten Buffer limitiert. Diese **maximale Übertragungseinheit** wird in der Literatur als **Maximum Transfer Unit (MTU)** bezeichnet. Für die Übertragung müssen IP-Datagramme aufgeteilt und wieder zusammengesetzt werden.

Beispiel: Im Abschnitt 5.3 haben wir gesehen, dass bei einem Ethernet-Frame der Datenanteil maximal 1500 Byte beträgt. Da IP-Datagramme eine maximale Grösse von 64K Byte haben, könnten bis zu 44 Frames für die Übertragung eines IP-Datagramms benötigt werden.

a) Aufteilung eines IP-Datagramms in Fragmente

Ist die Länge eines Datagramms grösser als die Maximum Transfer Unit (MTU), muss es für die Übertragung in kleinere IP-Datagramme, so genannte **Fragmente**, zerlegt werden. Der Vorgang wird als **Fragmentierung (Fragmentation)** bezeichnet. Jedes der Fragmente bekommt dabei einen eigenen IP-Header (IH), wird separat gekapselt und gesendet.

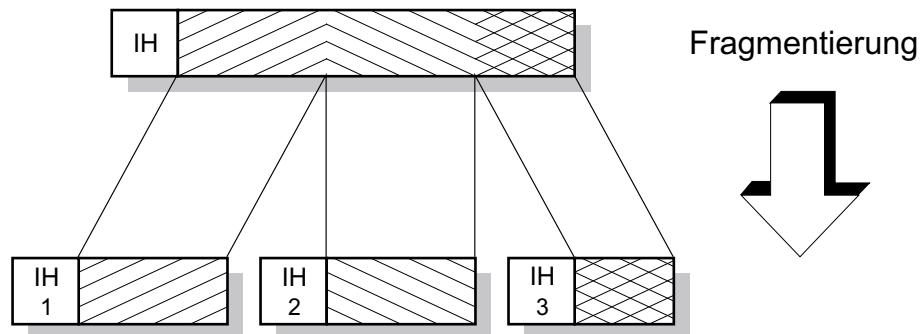


Abbildung 7.20: Aufteilung eines IP-Datagramms in drei Fragmente

Falls alle Teilnetze die gleiche oder grössere MTU als das Netz des Senders haben, erfolgt keine Fragmentierung. Falls aber eine Teilstrecke eine kleinere MTU hat, müssen die Fragmente unterwegs (falls nötig mehrmals) fragmentiert werden. Zum Beispiel (Abbildung 7.21) kann die Ethernet-MTU von 1500 Byte für Internet-Verbindungen zu gross sein. Dort ist nur eine Datagrammlänge von mindestens 576 Bytes gefordert (RFC 791).

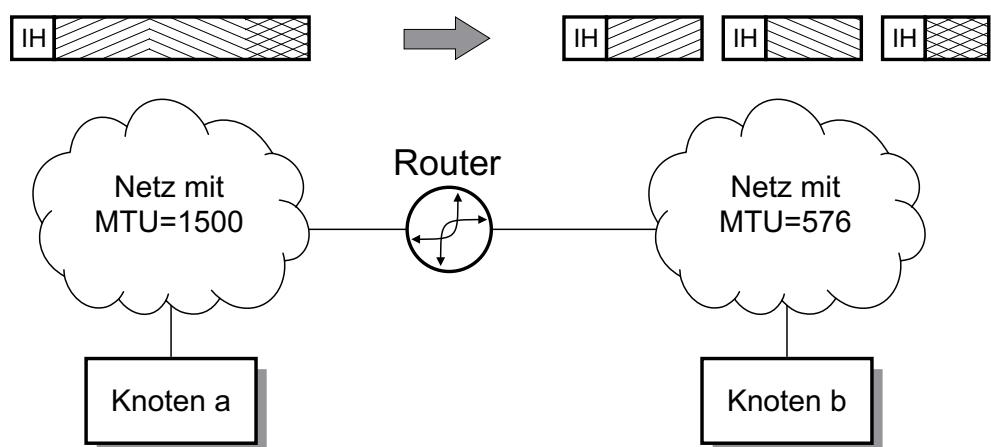


Abbildung 7.21: Fragmentierung beim Übergang zu einem Netz mit kleinerer MTU

b) Wiedervereinigung von Fragmenten

Nach der Übertragung der einzelnen Fragmente müssen diese wieder zu einem Datagramm zusammengesetzt werden. Diese Wiedervereinigung von Fragmenten wird als **Reassembly** bezeichnet. In der Abbildung 7.22 könnte dies schon an der frühesten Stelle (bei Router v) geschehen oder erst bei endgültigem Ziel dem Knoten b.

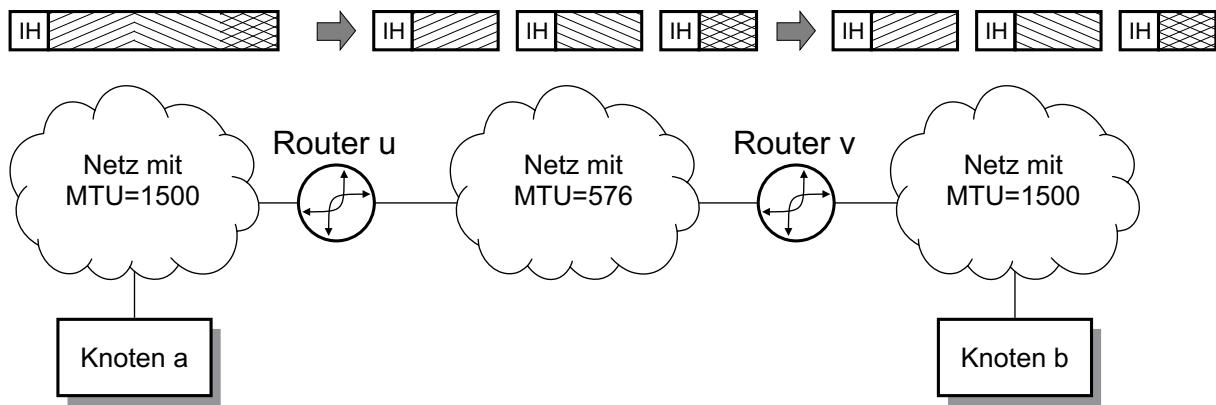


Abbildung 7.22: Fragmentierung durch Netze mit unterschiedlichen MTU

Der IP-Standard legt fest, dass die Fragmente erst beim *Zielhost* zusammengesetzt werden. Dieses Verfahren hat entscheidende Vorteile:

- Die Router werden entlastet, da diese viel weniger Statusinformationen zu den Fragmenten speichern müssen.
- Routen können dynamisch geändert werden. Das heisst, die Fragmente können auf unterschiedlichen Wegen zum Zielhost gelangen. Würden die Fragmente durch dazwischen liegende Router zusammengesetzt, müssten *alle* über diesen geleitet werden.

Der Empfänger kann anhand des Feldes „Identification“ fragmentierte Datagramme wieder zusammenstellen, da alle Fragmente des Datagramms den gleichen Identifikationswert haben. Dazu werden die folgenden Felder benötigt, die schon im Abschnitt 7.5.1 beschrieben wurden.

- Fragment Offset (FO)
- Last / More Fragments (MF)
- Total Length (TL) und Internet Header Length (IHL)

Zunächst wird geprüft, ob das empfangene Datagramm vollständig ist, also nicht fragmentiert wurde. Dies ist der Fall, wenn die Felder „Fragment Offset“ und „More Fragments“ beide Null sind. Das Datagramm wird dann direkt an den übergeordneten Layer weitergegeben.

Sonst wird eine Buffer-Identifikation bestimmt, die sich aus Source- und Destination-Adresse sowie den Feldern „Protocol“ und „Identification“ zusammensetzt. Falls noch kein Buffer mit dieser Identifikation besteht, wird jetzt eine Datenstruktur (Abbildung 7.23) zur Sammlung der Fragmente angelegt (reserviert).

Die Datenstruktur für das Reassembly besteht aus einem Daten-Buffer, einem Header-Buffer, einer Fragment-Block-Bit-Tabelle, einem Total-Data-Length-Feld und einem Timer (Abbildung 7.23). Jedes empfangene Fragment wird entsprechend seinem Fragment-Offset und seinem Längen-Feld in den Buffer eingefüllt. Außerdem werden die entsprechenden Bits in der Fragment-Block-Bit-Tabelle gesetzt.

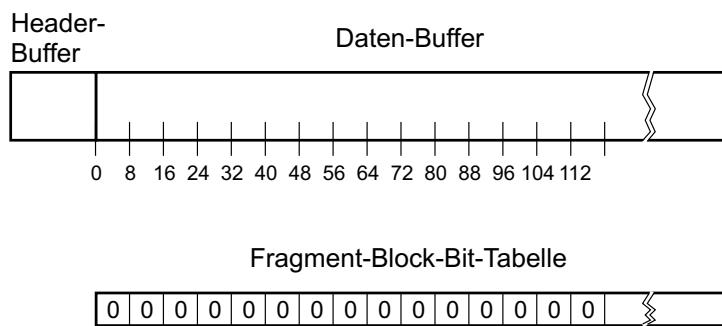


Abbildung 7.23: Datenstruktur für die Wiedervereinigung von Fragmenten

Bei Reassembly müssen folgende Fälle unterschieden werden:

- Falls der „Fragment Offset“ Null ist (erstes Fragment im Buffer), wird der Header des Fragments in den Header-Buffer kopiert (siehe Beispiel in Abbildung 7.24).

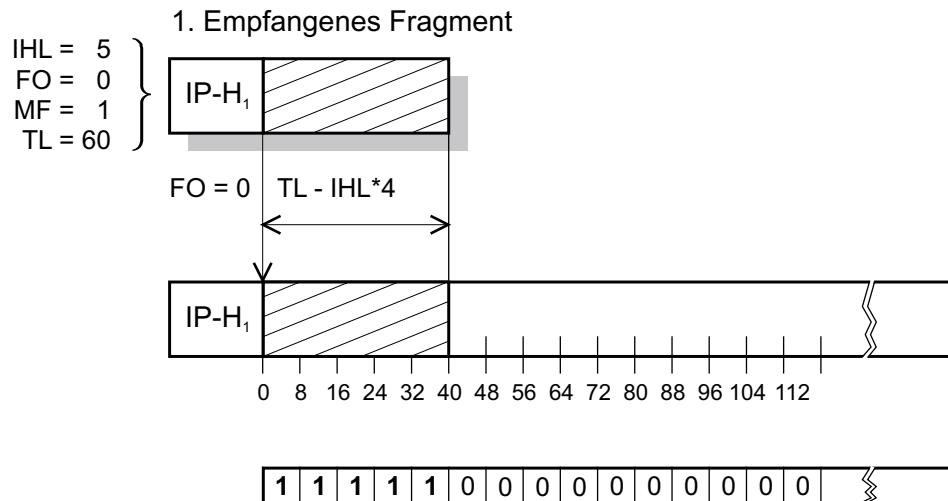


Abbildung 7.24: Beispiel: Reassembly des vordersten Fragments

- Falls das „More Fragments“ Feld Null ist, so ist dies das hinterste Fragment. Damit kann aus dem „Fragment Offset“ und der „Total Length“ der Wert des Total-Data-Length-Felds berechnet werden (siehe Beispiel in Abbildung 7.25).

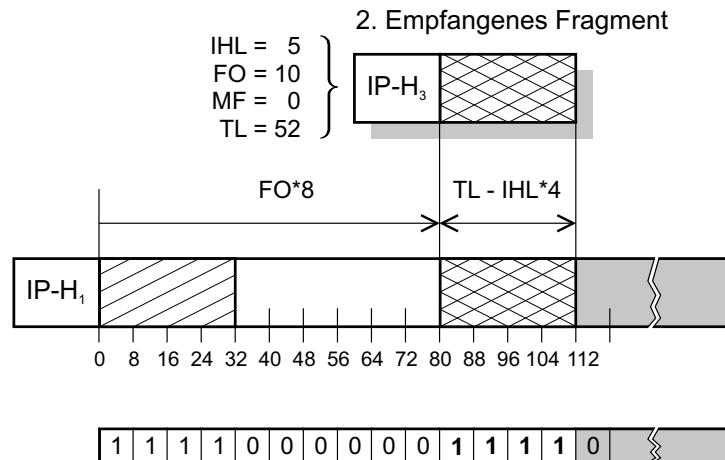


Abbildung 7.25: Beispiel: Reassembly des hintersten Fragments

- Anhand der Fragment-Block-Bit-Tabelle wird geprüft, ob mit diesem Fragment das Datagramm vollständig empfangen ist. In diesem Fall wird es an den übergeordneten Layer weitergeleitet und die allozierten Datenstrukturen werden freigegeben (siehe Beispiel in Abbildung 7.26).

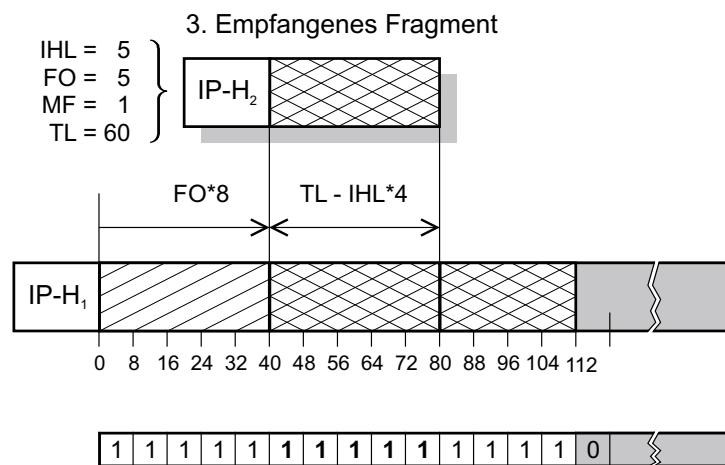


Abbildung 7.26: Beispiel: Reassembly des letzten Fragments

Das ganze Verfahren wird durch einen Timer überwacht, der typischerweise auf ca. 15 Sekunden eingestellt ist. Falls nach dieser Zeit das Datagramm nicht empfangen ist, werden die bisher empfangenen Fragmente verworfen und die allozierten Datenstrukturen freigegeben.

7.6 Kapselung und Umsetzung der IP-Adresse in die Hardware-Netzadresse

In diesem Abschnitt wird anhand von Ethernet sowie IEEE 802.3 gezeigt, wie ein IP-Datagramm schlussendlich übertragen wird. Dazu müssen zwei Aufgaben gelöst werden:

- Das IP-Datagramm muss in ein Netzwerk-(Hardware-)spezifisches Frame verpackt werden.
- Die logische IP-Adresse muss in eine Hardware-Adresse umgesetzt werden.

7.6.1 Kapselung eines IP-Datagramms

Für die Übertragung von IP-Datagrammen über IEEE 802.3 Netze sind zwei Kapselungen definiert.

- Ethernet Encapsulation nach RFC 894
- IEEE 802.2/802.3 Encapsulation nach RFC 1042

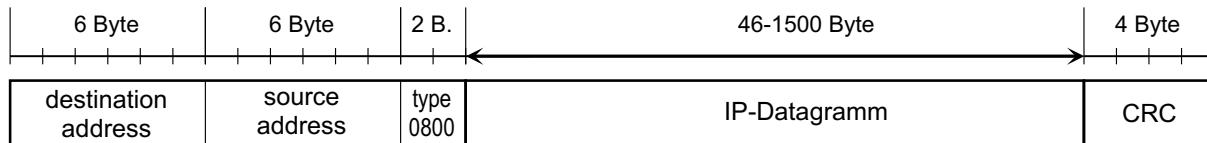


Abbildung 7.27: Kapselung eines IP-Datagramms in einem Ethernet-Frame

Die Abbildung 7.27 zeigt, wie (nach RFC 894) ein IP-Datagramm in ein Ethernet-Frame eingepackt (gekapselt) wird. Diese Art der Kapselung ist heute üblich und muss von allen Knoten unterstützt werden.

Für hundertprozentig IEEE 802.3 konforme Netze ist die Kapselung nach RFC 1042 gedacht. Wie die Abbildung 7.28 zeigt, wird zusätzlich LLC/SNAP-Header (Logical Link Control / Sub-Network Access Protocol gemäss IEEE 802.2) verwendet. Diese Art der Kapselung ist heute unüblich, sollte aber von allen Knoten empfangen werden können.

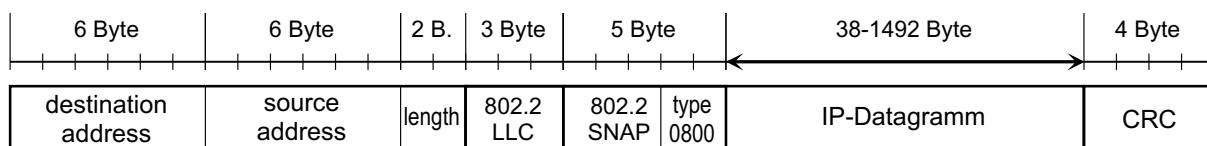


Abbildung 7.28: Kapselung eines IP-Datagramms in einem 802.2/802.3-Frame

7.6.2 Adressauflösung

Im Abschnitt 7.3 wurde das IP-Adress-Schema beschrieben. Es wurde gezeigt, dass mit Hilfe dieser abstrakten Adressen eine Hardware-unabhängige Adressierung erfolgen kann. Betrachtet man die im Abschnitt 7.6.1 gezeigten Kapselungen, so ist leicht einzusehen, dass die IP-Adresse nicht genügt, da zum Versenden die entsprechende Ethernet-Adresse (destination address) im Frame eingefügt werden muss. Beim Übergang vom Internet-Layer zum Datalink-Layer muss also die IP-Adresse in eine Hardware-Adresse umgesetzt werden.

Die Übersetzung der Protokolladresse in die entsprechende Hardware-Adresse nennt man **Adressauflösung (Address Resolution)**. Die Adressauflösung ist immer netzwerkspezifisch. Ein Knoten kann nur Adressen der Knoten, die im gleichen Netz sind, auflösen. Die Abbildung 7.29 zeigt, wie bei jedem Netzübergang das Datagramm ausgepackt, neu adressiert und wieder gekapselt wird.

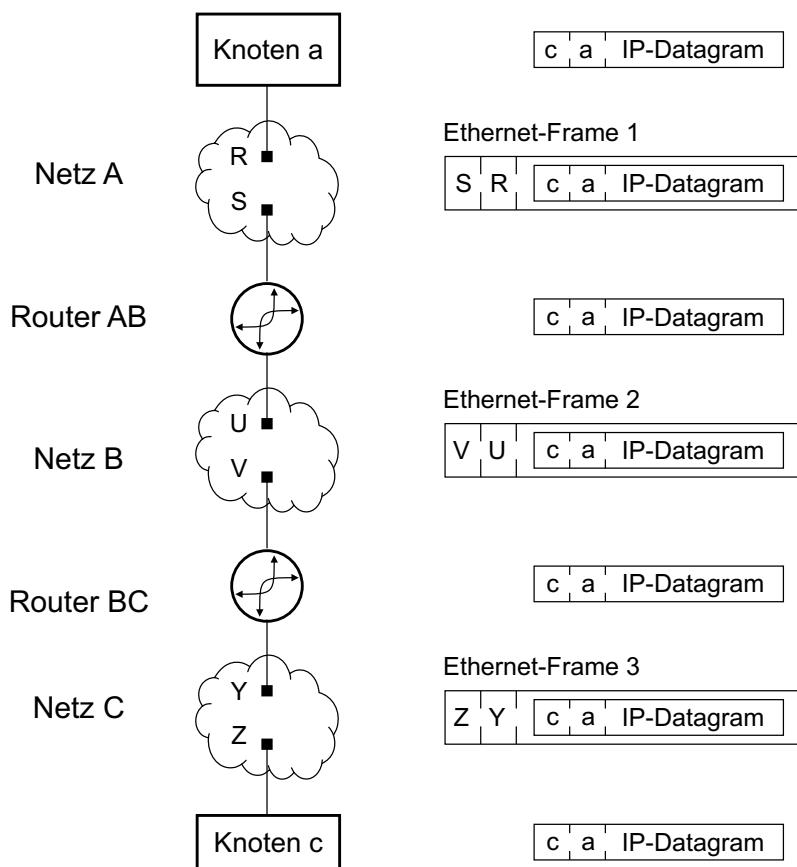


Abbildung 7.29: Kapselung und Netzübergang

Soll ein Datentransfer vom Knoten a zum Knoten c erfolgen, so wird im IP-Datagramm die Destination-Adresse vom Knoten c eingetragen.

Da der Knoten a den Knoten c nicht direkt erreichen kann, muss er das Datagramm zunächst an den Router AB senden.

Die IP-Adresse von Router AB wird in die Hardware-Adresse S übersetzt. Das Datagramm wird in ein Ethernet-Frame verpackt und an die Adresse S verschickt.

Der Router AB packt das Frame aus und stellt fest, dass er das Datagramm an den Router BC senden muss. Er übersetzt die IP-Adresse vom Router BC in die Hardware-Adresse V, verpackt das Datagramm in ein Frame mit dieser Adresse und verschickt es.

Der Router BC packt das Frame aus und stellt fest, dass er das Datagramm an den Knoten c senden muss. Er übersetzt die IP-Adresse vom Knoten c in die Hardware-Adresse Z, verpackt das Datagramm in ein Frame mit dieser Adresse und verschickt es an das gewünschte Ziel, den Knoten c.

Der Algorithmus, mit dem die Protokoll-Adresse in die Hardware-Adresse übersetzt wird, unterscheidet sich, je nachdem was für Adressformate verwendet werden. Es werden drei Verfahren eingesetzt:

Algorithmische Umsetzung: Falls zwischen den Adressformaten ein algorithmischer Zusammenhang besteht, ist die direkte Bestimmung der MAC-Adresse die eleganteste Lösung. Falls es keinen direkten Zusammenhang gibt, kann er herbeigeführt werden, indem man die MAC-Adresse software-mässig umsetzt. Solche Knoten haben dann zwei MAC-Adressen; eine global- und eine lokal-administrierte.

Beispiel: Die lokal-administrierte MAC-Adresse werde zusammengesetzt aus 2 Bytes eines Prefix AA-00- und 4 Bytes der IP-Adresse.

Ein Knoten mit der IP-Adresse 10.128.15.2 (hexadezimal 0A.80.0F.02) hätte also die MAC-Adresse AA-00-0A-80-0F-02.

Der Vorteil der algorithmischen Umsetzung ist die Effizienz und die Sicherheit. Nachteilig ist, dass für die Umstellung der Adresse auf den Knoten spezielle Software (Treiber) benötigt wird, die unter Umständen nicht verfügbar ist (z.B. bei einem Netzwerk-Drucker).

Tabellengesteuerte Übersetzung: Eine tabellengesteuerte Wandlung kann sehr gut eingesetzt werden. Das Problem bei diesem Verfahren liegt im Unterhalt der Tabellen. Falls diese manuell nachgeführt werden müssen, ist das Ändern und Verteilen der Tabellen an alle Knoten arbeitsintensiv und fehleranfällig. Ein Vorteil ist die vollständige Kontrolle, was in kontrollierten Umgebungen mit betrieblich kritischen Applikationen die Nachteile überwiegen kann.

Nachrichtenaustausch: Bei diesem Verfahren fordert ein Knoten, der eine Adresse auflösen muss, die anderen Knoten auf, ihm die erforderlichen Informationen zu senden. Der Nachteil liegt im zusätzlichen Verkehr. Im extremsten Fall würden für jedes zu sendende Paket zwei zusätzliche Frames (eine Adressanfrage und eine Antwort) übertragen.

In Abschnitt 7.6.3 wird ein Verfahren erläutert, das die beiden letzten Methoden kombiniert.

7.6.3 ARP

Bei TCP/IP können prinzipiell alle drei Adressauflösungsverfahren eingesetzt werden. Die bevorzugte Methode ist allerdings eine Kombination von Nachrichtenaustausch und tabellengesteuerter Übersetzung. Zu diesem Zweck wurde das **Address Resolution Protokoll (ARP)** definiert.

Das Verfahren ist so allgemein gehalten, dass es prinzipiell zur Wandlung beliebiger Protokoll- und Hardware-Adressen verwendet werden könnte. Im Folgenden beschränken wir uns allerdings auf die Umsetzung von 4-Byte-langen IP-Adressen auf 6-Byte-lange Ethernet-Adressen, wie dies im RFC 826 beschrieben ist.

a) Prinzip

Muss ein Knoten eine Protokoll-Adresse in eine Hardware-Adresse umsetzen, so sendet er an die Broadcast-Adresse ein ARP-Request-Paket. Dieses wird von allen Knoten im lokalen Netz empfangen. Der Knoten, der die entsprechende IP-Adresse besitzt, sendet als Antwort ein ARP-Reply-Paket an den anfragenden Knoten. Im Gegensatz zu den ARP-Anfragen werden also ARP-Antworten nicht mit Broadcast versandt.

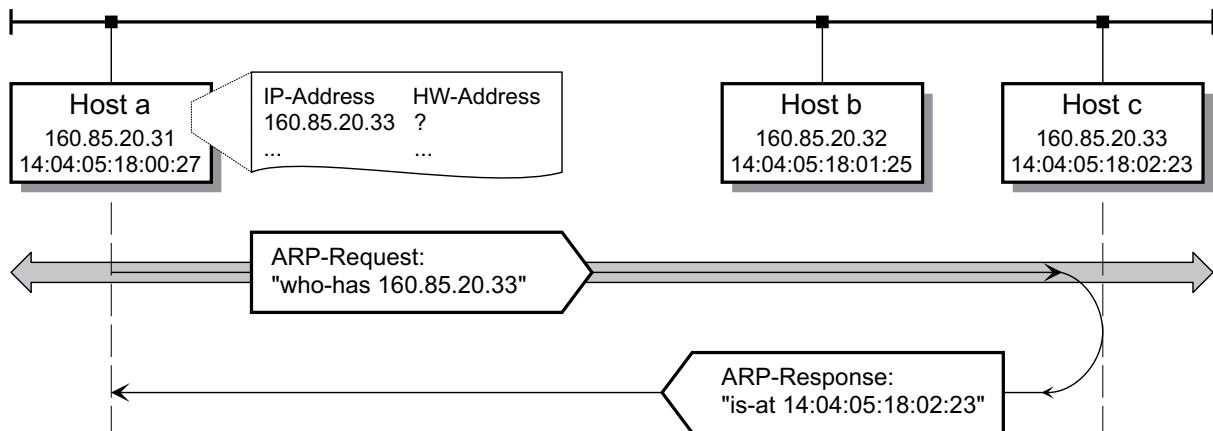


Abbildung 7.30: Ablauf einer „Address Resolution“

b) Format

Die Abbildung 7.31 zeigt den Aufbau eines ARP-Frames für die Übersetzung einer IP-Protokoll-Adresse in die Ethernet-Hardware-Adresse (ohne Präambel, SFD und CRC). Das Format ist für den ARP-Request und den ARP-Reply prinzipiell gleich. Beim Request wird als Ziel-Adresse eine Broadcast-Adresse angegeben und das Feld „Hardware Address of Target“ ist null (weil nicht bekannt).

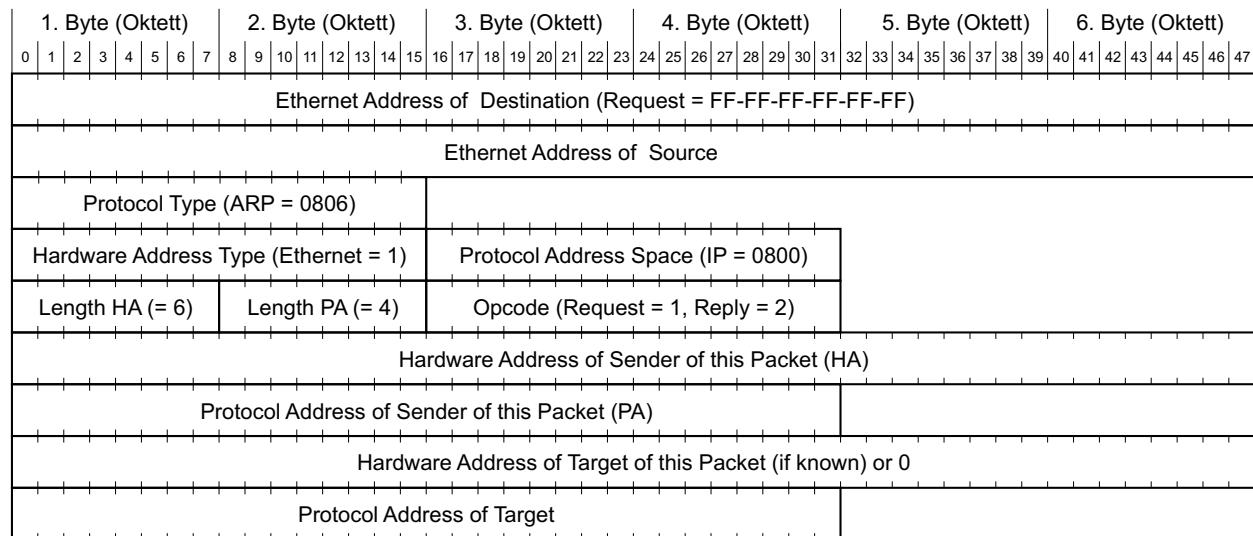


Abbildung 7.31: ARP-Frame für IP-Protokoll-Adresse und Ethernet-Hardware-Adresse

Abbildung 7.32 zeigt einen ARP-Request des Knotens 192.168.1.34. Für die IP-Adresse 192.168.1.1 wird die zugehörige MAC-Adresse (Ethernet-Hardware-Adresse) gesucht. Man beachte den Unterschied zwischen der Destination-MAC-Address (Broadcast) und der gesuchten Target-MAC-Address (null).

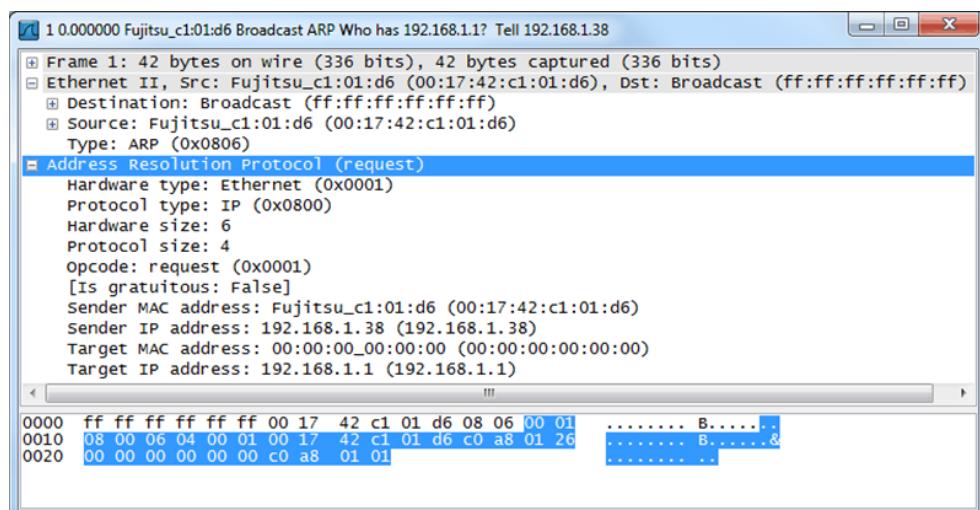


Abbildung 7.32: ARP-Request: An welcher MAC-Adresse ist die IP-Adresse 192.168.1.1?

Abbildung 7.33 zeigt den entsprechenden ARP-Reply (Unicast). Man beachte, dass die gesuchte Antwort nicht im Feld Target-MAC-Address sondern im Feld Sender-MAC-Address zu finden ist, die sich hier auch mit der Source-Address im Ethernet-Header deckt, was aber nicht zwingend der Fall sein muss.

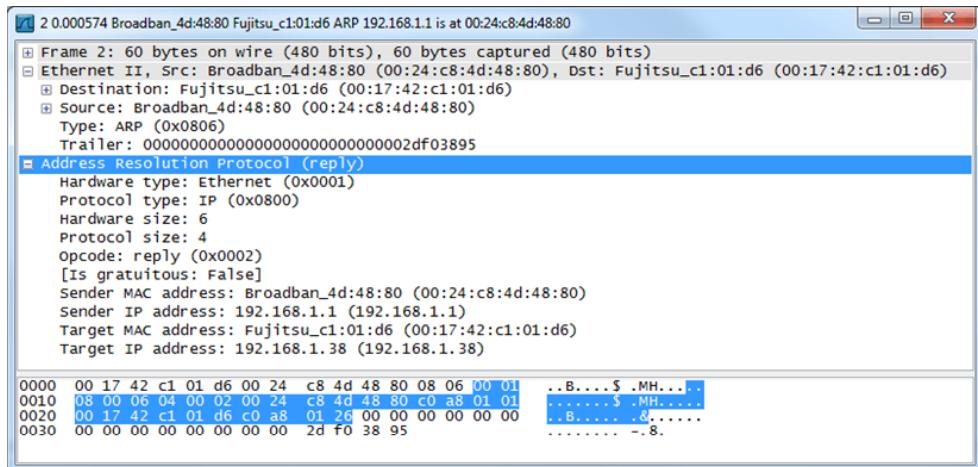


Abbildung 7.33: ARP-Reply: Die IP-Adresse 192.168.1.1 ist an der MAC-Adresse ...

c) ARP-Cache

Ohne weitere Massnahmen hat das obige Verfahren den Nachteil, dass für jedes zu sendende IP-Datagramm zusätzlich ein ARP-Request- und ARP-Reply-Paket versendet werden müsste. Dies wäre natürlich denkbar ineffizient. Jeder Knoten führt darum eine Tabelle, ein sogenanntes **ARP-Cache**, in welche er die ihm bekannten Protokoll/Hardware-Adresspaare einträgt. Nach einer gewissen Zeit werden die Einträge wieder gelöscht - sei es weil die Tabelle zu gross wurde oder weil der Eintrag lange nicht mehr verwendet wurde.

Wie alle Broadcast-Protokolle haben die ARP-Request-Pakete die unangenehme Eigenschaft, dass sie durch die Bridges nicht gefiltert werden können/dürfen. In grossen Netzen kann der durch ARP-Request-Pakete verursachte Verkehr eine massive Belastung darstellen.

d) Gratuitous ARP

Gratuitous ARP (englisch: grundlos, unbegründet, überflüssig, unnötig) bezeichnen ARP-Request und -Reply, die gemäss ARP-Spezifikation (RFC 826) eigentlich nicht notwendig wären.

Ein Gratuitous ARP-Request wird verwendet, um IP-Adresskonflikte zu erkennen. Dazu versendet der Knoten nach jeder Adresszuweisung beim Booten oder bei Änderungen der IP-Adresse einen ARP-Request für seine *eigene* IP-Adresse (Abbildung 7.34). Falls diese IP-Adresse schon vergeben ist, also ein Konflikt besteht, wird der anfragende Knoten einen ARP-Reply empfangen. Keine Antwort bedeutet also, dass (zurzeit) kein Konflikt besteht.

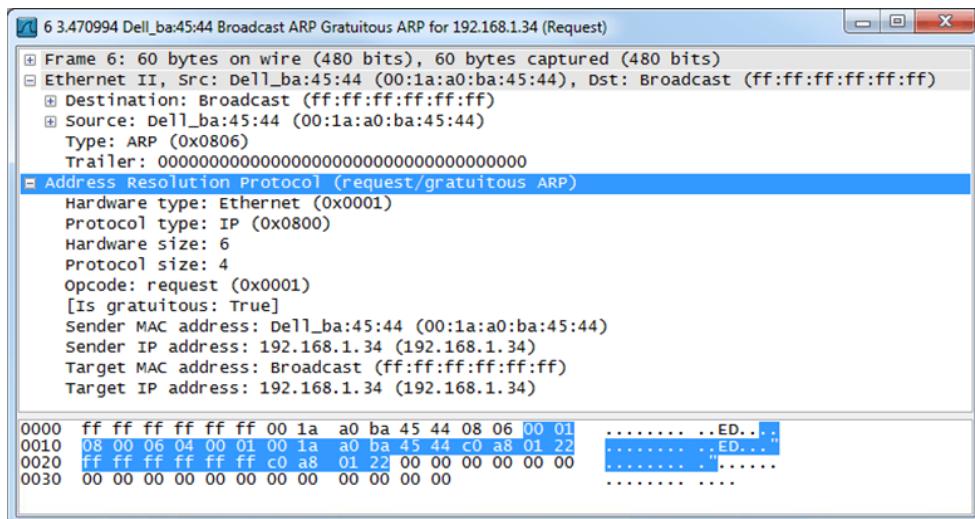


Abbildung 7.34: Gratuitous ARP-Request: Hat schon ein Knoten die IP-Adresse 192.168.1.34?

Ein Gratuitous ARP-Reply wird von einem Knoten ebenfalls nach dem Setzen oder Ändern der IP-Adresse verschickt, aber mit dem Zweck, die ARP-Cache der anderen Knoten zu berichten. Dazu wird - ohne dass ein Request vorliegt - ein ARP-Reply mit der eigenen IP/MAC-Adress-Paarung verschickt. Wie Abbildung 7.35 zeigt, wird beim Gratuitous ARP-Reply ebenfalls ein Broadcast verwendet, damit alle Knoten adressiert werden (im Gegensatz zu Unicast beim normalen ARP-Reply, Abbildung 7.33). Es gibt jedoch keine Gewähr, dass die Knoten einen Gratuitous ARP auch verwerten.

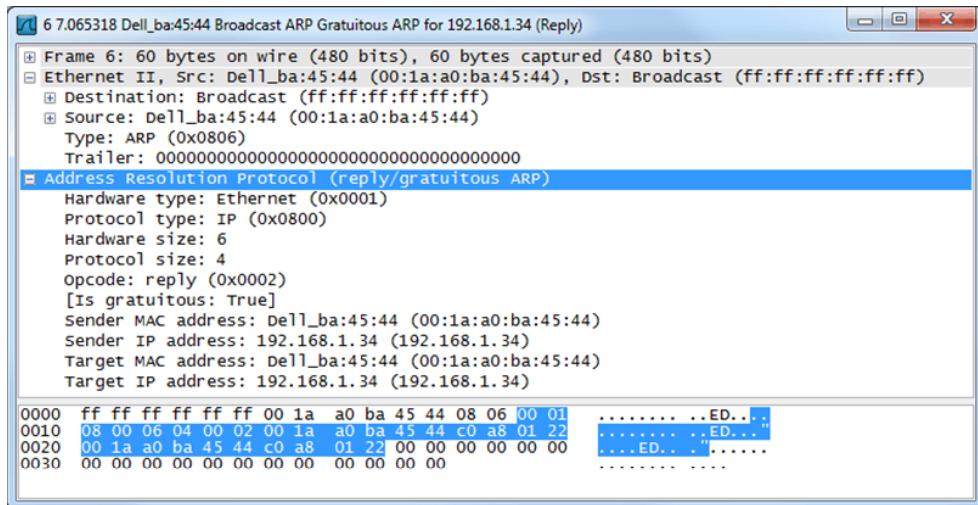


Abbildung 7.35: Gratuitous ARP-Reply: Die IP-Adresse 192.168.1.34 ist an MAC-Adresse ...!

e) ARP-Befehle

Mit Hilfe des Befehls `arp` kann der ARP-Cache auch manuell angezeigt und verändert werden. Der Befehl ist sowohl bei Unix als auch bei den MS-Betriebssystemen vorhanden und sogar fast gleich.

Operation	Befehl
Anzeigen aller Einträge	<code>arp -a</code>
Numerisches Anzeigen aller Einträge (nur Unix)	<code>arp -an</code>
Löschen eines Eintrags	<code>arp -d ip_addr</code>
Setzen eines Eintrags	<code>arp -s ip_addr hw_addr</code>

Mit Hilfe des Befehls `arping` können gezielt ARP-Request und Reply ausgelöst werden. Die folgenden Befehle wurden beispielsweise verwendet, um die obigen Gratuitous ARP zu erzeugen:

```
arping -c 1 -U 192.168.1.34
```

```
arping -c 1 -A 192.168.1.34
```

Hinweis: Da mittels ARP das Netz empfindlich gestört werden kann, soll damit nur in einer abgeschotteten Umgebung (z.B. KT Labor) experimentiert werden!

7.7 RARP – Reverse Address Resolution Protocol

ARP löst das Problem der Suche nach Ethernet-Adressen, die bestimmten IP-Adressen entsprechen. Zuweilen muss aber auch das umgekehrte Problem gelöst werden. Man muss eine IP-Adresse für eine bestimmte Ethernet-Adresse finden. Dieses Problem tritt insbesondere beim Starten einer plattenlosen Workstation auf. Diese Maschinen erhalten normalerweise das binäre Abbild ihrer Betriebssysteme (Disk-Image) von einem entfernten File-Server. Wie aber erfährt eine solche Maschine ihre IP-Adresse?

Die Lösung bietet das **Reverse Address Resolution Protocol (RARP)**. Wie im RFC 903 definiert, wird dasselbe Paketformat verwendet wie beim ARP-Protokoll.

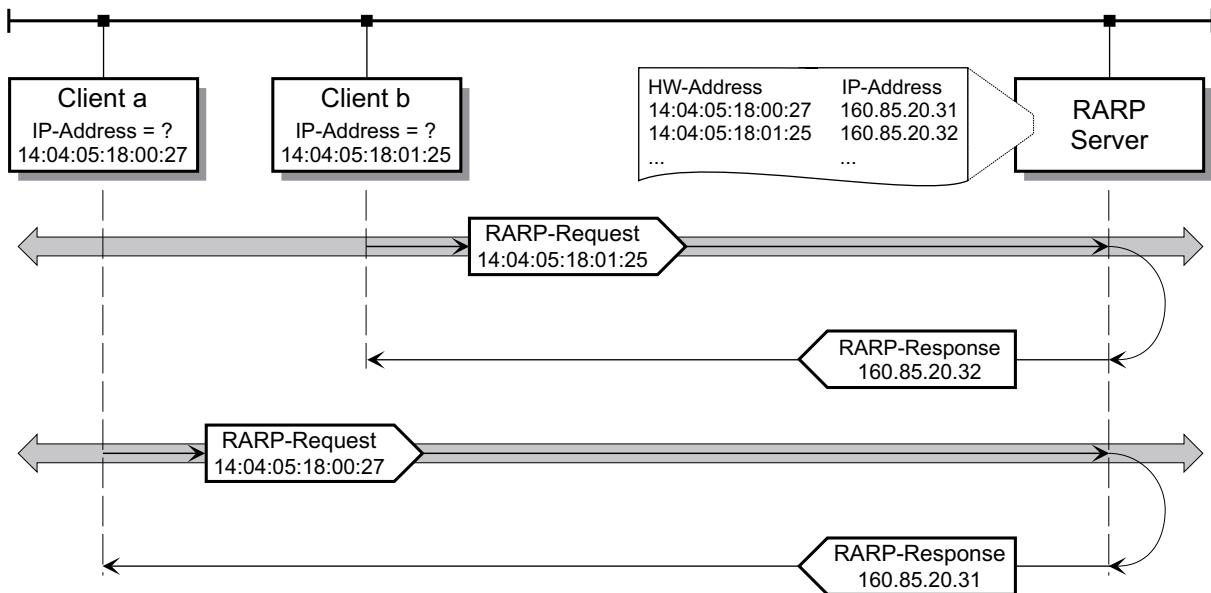


Abbildung 7.36: Ablauf einer „Reverse Address Resolution“

Mit dem RARP-Protokoll kann eine neu gebootete Workstation eine Broadcast-Anfrage mit etwa folgendem Inhalt senden: „Meine 48 Bit lange Ethernet-Adresse lautet 14:04:05:18:01:25. Kennt jemand meine IP-Adresse?“ Der RARP-Server sieht diese Anfrage, durchsucht seine Konfigurationsdateien nach der Ethernet-Adresse und sendet (falls er fündig wurde) die entsprechende IP-Adresse zurück.

Die Verwendung von RARP ist besser als das Ablegen einer IP-Adresse in einem Disk-Image, weil dadurch die gleiche Konfiguration auf allen Maschinen benutzt werden kann. Befände sich die IP-Adresse innerhalb des Disk-Images, würde jede Workstation ein eigenes benötigen.

RARP hat den einen Nachteil, dass es MAC-Layer-Broadcast benutzt, um den RARP-Server zu erreichen. Solche Broadcast-Nachrichten werden bekanntlich von Routern nicht weitergegeben, so dass in jedem IP-Subnetz ein RARP-Server erforderlich ist. Um dieses Problem zu vermeiden, wurden alternative Bootstrap-Protokolle namens BOOTP und DHCP entwickelt (RFC 951, 1048 und 1084). Im Gegensatz zu RARP benutzen diese UDP-Nachrichten, die über Router weitergeleitet werden können. Sie bieten auch zusätzliche Informationen für plattenlose Workstations, darunter die IP-Adresse des File-Servers, auf dem sich das Disk-Image befindet, die IP-Adresse des Default-Routers und die zu verwendende Subnetzmase.

7.8 Internet Control Message Protocol (ICMP)

Gelegentlich muss ein Router oder Empfänger-Host dem Sender eine Mitteilung zurücksenden, um zum Beispiel einen Fehler zu melden. Zu diesem Zweck wird das **Internet Control Message Protocol (ICMP)** benutzt (siehe RFC 792). In diesem Abschnitt werden *exemplarisch* Formate und Anwendungen einiger ICMP-Meldungen gezeigt.

Obwohl ICMP die Basisfunktionalität von IP benutzt, wie wenn es ein höheres Protokoll wäre, ist es als integraler Bestandteil von IP zu betrachten und muss in jeder Implementierung vorhanden sein.

ICMP-Meldungen werden bei verschiedenen Gelegenheiten verschickt: zum Beispiel wenn ein Datagramm das Ziel nicht erreichen kann, weil ein Router nicht genügend Buffer-Kapazität besitzt, aber auch wenn ein Router einen kürzeren Weg für ein Datagramm erkennt.

Es war nie ein Ziel des IP-Layers, die Datagramme absolut zuverlässig zu übertragen. Im gleichen Sinn ist auch nicht die Aufgabe der ICMP-Kontrollmeldungen die IP-Übertragung fehlerfrei zu gestalten, sondern nur Fehlerursachen zu melden. Nach wie vor ist es möglich, dass ein Datagramm unterwegs verloren geht, ohne dass eine ICMP-Meldung dies dem Sender meldet. Es ist und bleibt also die Aufgabe der höheren Layer, Algorithmen zur Fehlerkorrektur bereit zu stellen.

ICMP-Datagramme melden typischerweise Fehler in der Verarbeitung und Weiterleitung von Datagrammen. Um einen endlosen „Schwanzbeisser“ von Meldungen über Meldungen etc. zu vermeiden, werden keine ICMP-Meldungen über ICMP-Meldungen erzeugt. Zusätzlich wird bei fragmentierten Datagrammen nur für das erste Fragment (mit dem Fragment-Offset von Null) eine ICMP-Meldung geschickt.

7.8.1 Meldungs-Format

Wie in Abbildung 7.37 gezeigt, werden ICMP-Meldungen in einem IP-Datagramm gekapselt (siehe 7.5.1). Im IP-Header-Feld „Protocol“ wird für ICMP der Wert Eins angegeben.

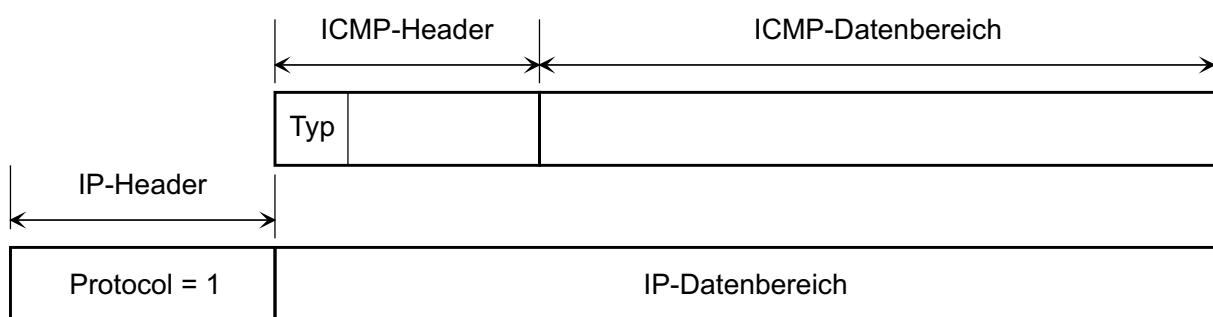


Abbildung 7.37: Kapselung einer ICMP-Meldung

Das erste Byte im Datenbereich ist das ICMP-Typenfeld. Der Wert dieses Feldes bestimmt das Format der folgenden ICMP-Meldung.

Die ICMP-Meldungen lassen sich in Fehler- und Informationsmeldungen unterteilen. Die folgende Tabelle zeigt die definierten Meldungstypen:

ICMP-Typ	Bedeutung (Fehler)	ICMP-Typ	Bedeutung (Information)
3	Destination Unreachable	0	Echo Reply
4	Source Quench	8	Echo
5	Redirect	13	Timestamp
11	Time Exceeded	14	Timestamp Reply
12	Parameter Problem	15	Information Request
		16	Information Reply

Die ICMP-**Fehlermeldungen** sind:

Destination Unreachable: Stellt ein Router fest, dass ein Datagramm nicht an das Endziel befördert werden kann, sendet er diese Meldung an den sendenden Host. Die Meldung bezeichnet den Zielhost oder das Netz, der bzw. das nicht erreichbar ist.

Beispiel: Beim Erstellen eines Datagramms kann der Host das DF-Bit (Don't Fragment) im Header setzen, um eine Fragmentierung zu verhindern. Stellt der Router fest, dass das Datagramm grösser ist als die MTU des zu benutzenden Netzes, sendet er diese ICMP-Meldung an den Host und verwirft das Datagramm.

Source Quench: Ein Router sendet diese Meldung an eine Quelle, wenn in seinem Puffer so viele Datagramme vorhanden sind, dass der Speicherplatz erschöpft ist und er weitere ankommende Datagramme verwerfen muss. In diesem Fall sendet er diese Meldung an den Host, von dem das verworfene Datagramm stammt. Der Host wird damit aufgefordert, die Rate, in der er überträgt, zu reduzieren.

Redirect: Erstellt ein Host ein Datagramm für ein entferntes Netz, sendet er das Datagramm an einen Router, der das Datagramm an das Ziel befördert. Stellt der Router fest, dass der Host das Datagramm an den falschen Router gesendet hat, sendet er diese Meldung an den Host. Der Host wird damit aufgefordert, die angegebene Route zu ändern, indem er einen bestimmten Host oder das Netz ändert. Der zweite Fall ist üblich.

Time Exceeded: Diese Meldung wird in zwei Fällen gesendet. Senkt ein Router das Feld „Time To Live“ eines Datagramms auf null, verwirft er das Datagramm und sendet diese Meldung. Des Weiteren wird diese Meldung von einem Host gesendet, wenn sein Timer abläuft, bevor alle Fragmente eines bestimmten Datagramms angekommen sind.

Parameter Problem: Diese Meldung kann gesendet werden, falls ein Router oder Host im IP-Header eines Datagramms einen ungültigen Wert findet, so dass er es nicht verarbeiten kann.

Daneben gibt es die folgenden drei Paare von **Informationsmeldungen**:

Echo Request/Reply: Die Meldung Echo Request kann an die ICMP-Software jedes Rechners gesendet werden. Als Reaktion darauf muss die ICMP-Software eine Antwort – Echo Reply – senden. Die Antwort enthält die gleichen Daten wie die Anfrage.

Timestamp Request/Reply: Diese Timestamp-Meldungen verhalten sich analog zu den obigen Echo-Meldungen. Zusätzlich geben Sender und Empfänger ihre Zeit (32-Bit, Millisekunden seit Mitternacht) mit.

Information Request/Reply: Bei einem Information Request kann sowohl die Sende- als auch die Empfangsadresse Null sein. Router, die diese Anfrage erhalten, senden eine Antwort, in der die Nummer und die 32-Bit-Maske des zu benutzenden Subnetzes enthalten ist. Dies ist *eine* Möglichkeit, wie Hosts die Nummer des angeschlossenen Netzwerks bestimmen können.

7.8.2 Destination Unreachable Message (Type 3)

„Destination Unreachable“-Meldungen werden von Routern oder Ziel-Hosts an den Absender zurückgesendet, wenn ein Datagramm nicht zustellbar ist. Typische Gründe dafür sind:

- Ein Router stellt anhand seiner Routing-Tabelle fest, dass das im Datagramm angegebene Ziel nicht erreichbar ist.
- Ein Router stellt fest, dass das Port, über welches das Datagramm gesendet werden müsste, (z.B. wegen eines Leitungsunterbruchs) nicht aktiv ist.
- Der Ziel-Host kann mit dem Datagramm nichts anfangen, weil z.B. das höhere Protokoll oder das (Applikations-)Port nicht bekannt oder deaktiviert ist.
- Ein Router müsste ein Datagramm zur Weiterleitung fragmentieren, dessen „Don't Fragment“-Flag gesetzt ist. In diesem Fall muss er das Datagramm verwerfen.

a) Format und Felder

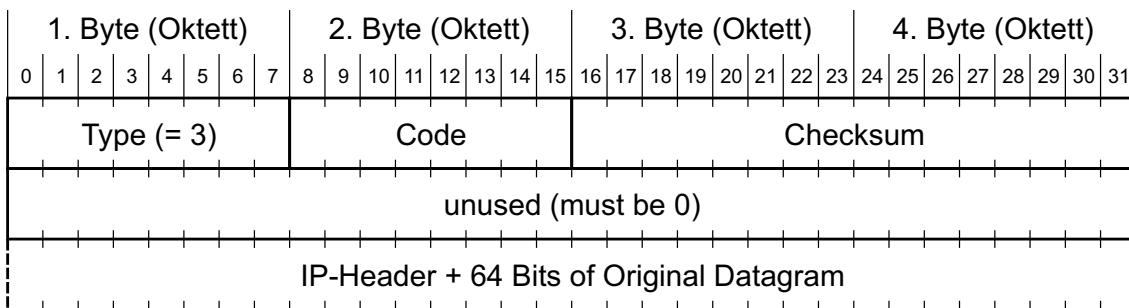


Abbildung 7.38: Format einer „Destination Unreachable“-Meldung

Das Format einer „Destination Unreachable“-Meldung ist für alle obigen Anwendungen gleich (siehe Abbildung 7.38).

Das Type-Feld ist bei einer „Destination Unreachable“-Meldung konstant drei.

Das Code-Feld gibt den Grund der Unzustellbarkeit an. Die Codes 0, 1, 4, und 5 können von Routern zurückgemeldet werden, die Codes 2 und 3 von Ziel-Hosts.

Code	Bedeutung (unvollständig)
0	net unreachable
1	host unreachable
2	protocol unreachable
3	port unreachable
4	fragmentation needed and DF set
5	source route failed
...	
13	communication administratively prohibited
...	

Das Checksum-Feld wird gebildet, indem der Header in Worte (16 Bit) zerlegt wird. Diese werden invertiert (Einerkomplement) und dann addiert, wobei das zu berechnende Feld „Checksum“ als Null angenommen wird. Die nochmals invertierte Summe (16-Bit) ergibt die Prüfsumme.

Internet Header plus 64 Bits of Original Datagram enthält den ersten Teil des Datagramms, das die ICMP-Meldung ausgelöst hat. Damit ist der ursprüngliche Absender in der Lage, den Fehler genauer zu bestimmen.

b) Beispiel:

Der Host 160.85.31.3 versucht das folgende IP-Paket an Host 160.85.29.99 zu senden. Die Mitteilung beginnt nach den 10 Word des IP-Headers mit dem Wert (8b0d...).

```
4500 0028 8b10 0000 0711 a8a4 a055 1f03  
a055 1d63 8b0d 829d 0014 a348 030a 0000  
7504 1137 407c 0800
```

Der dazwischenliegende Router kennt aber keinen Weg und schickt eine „Destination Unreachable“-Meldung zurück:

```
160.85.130.30 > 160.85.31.3: icmp: host 160.85.29.99 unreachable  
  
4500 0038 8038 0000 fd01 5bc0 a055 821e  
a055 1f03 0301 4bf7 0000 0000 4500 0028  
8b10 0000 0711 a8a4 a055 1f03 a055 1d63  
8b0d 829d 0014 a348
```

In der obigen Word-Darstellung beginnt nach 10 Blöcken der ICMP-Header (0301), nach 4 weiteren Word folgen 10 Word des IP-Headers (4500...) und auf der letzten Zeile steht der Anfang (4 Word = 64 Bit) der ursprünglichen Meldung.

7.8.3 Time Exceeded Message (Type 11)

„Time Exceeded“-Meldungen werden aus zwei Gründen geschickt:

- Ein Router erhält ein Datagramm, dessen Lebensdauer abgelaufen ist („Time to Live“-Feld ist Null): Das Datagramm wird weggeworfen und dem Absender wird dies mittels einer „Time Exceeded“-Meldung (mit Code=0) gemeldet.
 - Ein Ziel-Host kann innerhalb einer bestimmten Zeit ein Datagramm nicht vollständig zusammenstellen, weil Fragmente fehlen: Das ganze Datagramm muss weggeworfen werden und an den Absender wird eine „Time Exceeded“-Meldung (mit Code=1) gesendet.
- Falls das erste Fragment (Offset=0) fehlt, wird keine „Time Exceeded“-Meldung erzeugt.

a) Format und Felder

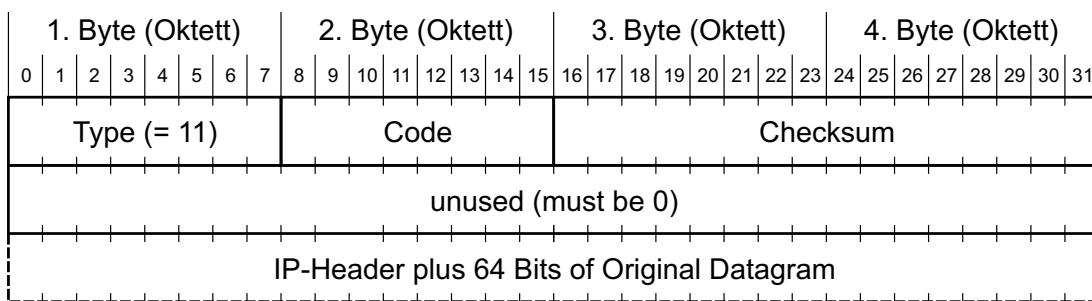


Abbildung 7.39: Format einer „Time Exceeded“-Meldung

Das Type-Feld ist bei einer „Time Exceeded“-Meldung konstant elf.

Das Code-Feld gibt den Grund der Unzustellbarkeit an.

Code	Bedeutung
0	time to live exceeded in transit
1	fragment reassembly time exceeded

Das Checksum-Feld wird gebildet, indem der Header in Worte (16 Bit) zerlegt wird. Diese werden invertiert (Einerkomplement) und dann addiert, wobei das zu berechnende Feld „Checksum“ als Null angenommen wird. Die nochmals invertierte Summe (16-Bit) ergibt die Prüfsumme.

Internet Header plus 64 Bits of Original Datagram enthält den ersten Teil des Datagramms, das die ICMP-Meldung ausgelöst hat. Damit ist der ursprüngliche Absender in der Lage, den Fehler genauer zu bestimmen.

b) Anwendung: traceroute-Programm

Das Internet ist ein grosser, komplexer Verbund von Netzen und Routern. Den Weg eines Datagramms zu bestimmen resp. einen fehlerhaft konfigurierten Router zu finden, ist darum äusserst schwierig.

Das Programm `traceroute` versucht den Weg zu einem Ziel-Host zu bestimmen, indem es UDP-Testpakete mit kleinen „Time to Live“-Werten verschickt und die „Time Exceeded“-Meldungen der Router aufzeichnet.

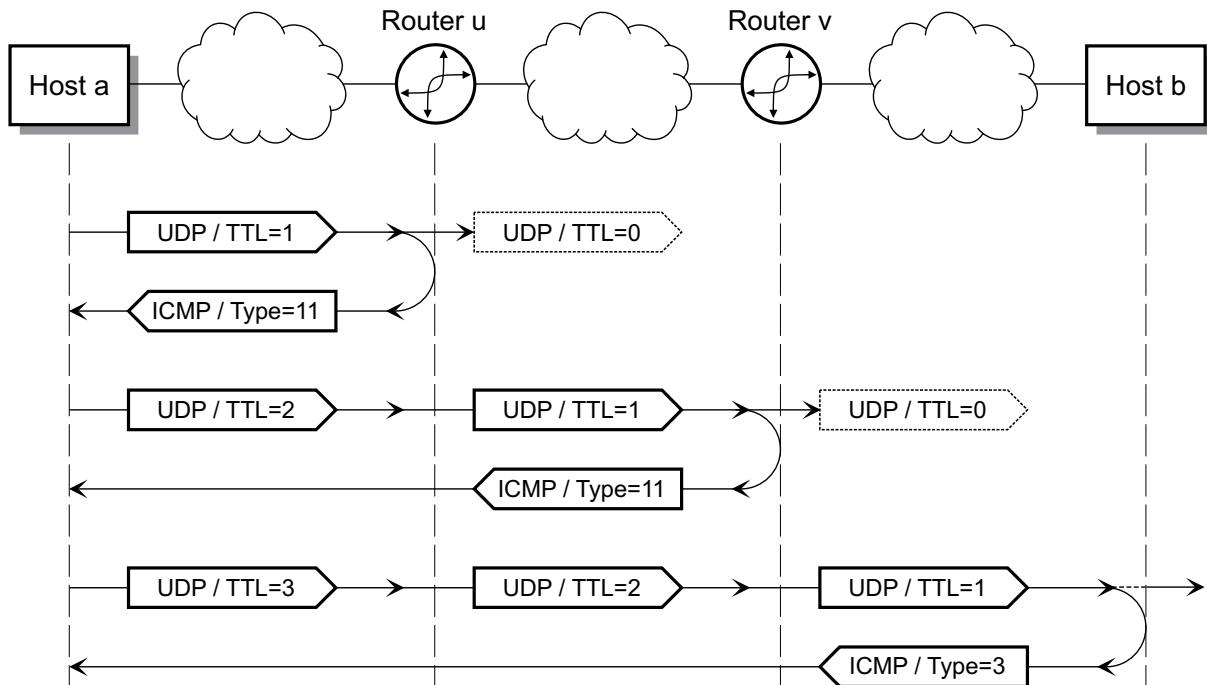


Abbildung 7.40: Prinzip des `traceroute`-Programms

Wie in Abbildung 7.40 gezeigt, wird mit einem „Time to Live“-Wert von eins begonnen. Dieser wird in Einerschritten erhöht, bis der Ziel-Host erreicht wird.

Um zu verhindern, dass der Ziel-Host das Test-Datagramm verarbeitet, wird im UDP-Paket eine hohe, zufällige Port-Nummer (Default ist 33434) gewählt. Der Ziel-Host wirft darum das empfangene Datagramm fort und sendet dafür eine ICMP-Meldung „Destination Unreachable/Port Unreachable“ zurück.

Zusätzlich wird die Port-Nummer bei jedem Schritt (hop) um eins erhöht.

Das Programm traceroute wird wie folgt aufgerufen:

```
traceroute -optionen Ziel-Host
```

Auf den Microsoft-Betriebssystemen heisst das entsprechende Programm tracert. Das Programm traceroute (resp. wo mit MS: vermerkt tracert) hat einige nützliche Parameter, wie zum Beispiel:

- | | |
|--------------------------|--|
| -f TTL | Setzt den „Time to Live“-Wert <i>TTL</i> des ersten Paketes, mit dem die Pfadsuche begonnen wird. |
| -I | Verwendet ICMP-ECHO-Meldungen (siehe nächsten Abschnitt) anstelle von UDP-Datagrammen. |
| -m max
(MS: -h max) | Setzt den maximalen „Time to Live“-Wert <i>max</i> nachdem die Pfadsuche abgebrochen wird. Der Standardwert ist 30 (gleicher Default-Wert wie bei TCP-Verbindungen). |
| -n
(MS: -d) | Gibt die IP-Adresse numerisch aus und nicht als Name. Damit kann ein Zugriff auf den Name-Server gespart werden. |
| -p port | Setzt den Anfangswert für die UDP-Port-Nummer, die in den Testpaketen verwendet wird (Default ist 33434). Traceroute hofft, dass die Router und der Ziel-Host die UDP-Ports mit den Nummern <i>port + nhops – 1</i> nicht benutzen. Falls das Port belegt ist, kann mit diesem Parameter ein anderer Bereich gewählt werden. |
| -q count | Erlaubt die Anzahl <i>count</i> der „Messungen“ anzugeben (Default ist drei). |
| -v | Detailliertere Angaben (Verbose output). |
| -w wait
(MS: -w wait) | Setzt die Zeit (in Sekunden), die auf eine Antwort gewartet wird (Default 5 Sek.). |

Das folgende Beispiel zeigt einen Aufruf. Darin ist ersichtlich, dass auch noch die Zeit angegeben wird, die benötigt wird, bis eine Antwort zurückkommt.

```
# traceroute -n -q1 160.85.128.10
traceroute to 160.85.128.10 (160.85.128.10), 30 hops max, 40 byte packets
 1  160.85.17.1      34.956 ms
 2  160.85.128.226  37.558 ms
 3  160.85.128.10   43.692 ms
```

7.8.4 Echo or Echo Reply Message (Type 8 / Type 0)

Empfängt ein Host eine an ihn adressierte „Echo“-Meldung, so schickt er eine „Echo Reply“-Meldung mit dem gleichen Dateninhalt zurück. Das heisst, die Absenderadresse der „Echo“-Meldung wird zur Zieladresse in der „Echo Reply“-Meldung.

a) Format und Felder

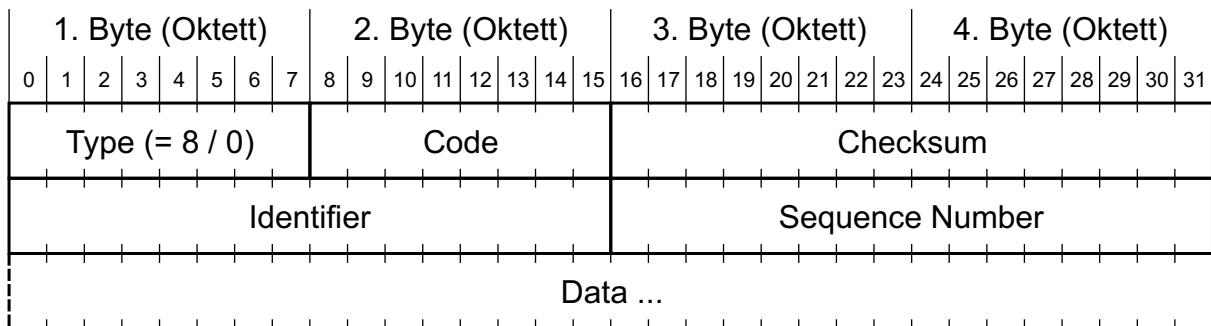


Abbildung 7.41: Format einer „Echo“- oder „Echo Reply“-Meldung

Das Type-Feld identifiziert die Art der Meldung.

Type	Bedeutung
8	Echo Message
0	Echo Reply Message

Das Identifier-Feld schafft den Bezug zum Prozess, der die Echo-Meldung abgeschickt hat. Das heisst, für jeden Echo-Prozess des Absenders wird ein unterschiedlicher Identifier-Wert verwendet. Der Echo-Host gibt das Identifier-Feld unverändert zurück.

Das Sequence-Number-Feld wird (vom Absenderprozess) bei jedem Echo-Request um eins inkrementiert. Die Sequence-Number erlaubt die Zuordnung von einer Antwort zu einer Anforderung. Der Echo-Host gibt den gleichen Wert zurück.

Das Code-Feld ist null.

Das Checksum-Feld wird gebildet, indem der Header in Worte (16 Bit) zerlegt wird. Diese werden invertiert (Einerkomplement) und dann addiert, wobei das zu berechnende Feld „Checksum“ als Null angenommen wird. Die nochmals invertierte Summe (16-Bit) ergibt die Prüfsumme.

b) Anwendung: ping-Programm

Das Programm `ping` verwendet ICMP-Echo-Meldungen um eine Verbindung zu einem Router oder Host zu prüfen. Da ICMP als Teil des IP-Layers implementiert sein muss, funktioniert diese „Verbindungskontrolle“ auch dann, wenn übergeordnete Konfigurationsfehler vorliegen, also z.B. keine TCP-Verbindung erstellt werden kann.

Der Aufruf erfolgt mit:

```
ping -optionen Zieladresse
```

Folgende Optionen sind verfügbar:

- `-c count` (MS: `-r count`) Bricht nach *count* Messungen ab. (Der Default ist unbegrenzt, bei MS ist er vier).
- `-f` Flood-Ping: Sendet ein Datagramm, sobald eine Antwort zurückkommt oder spätestens nach 10ms. Für jedes gesendete Datagramm wird ein Punkt angezeigt und für jeden Empfang wird ein Punkt gelöscht. Diese Funktion benötigt Privilegien, da ein Netz damit gefüllt werden kann.
- `-i wait` Zeit *wait* in Sekunden, die zwischen dem Senden von zwei Datagrammen gewartet wird. (Default ist eine Sekunde; kann nicht mit `-f` kombiniert werden.)
- `-n` (MS: `-a`) Zeigt die IP-Adressen numerisch an und versucht keine Übersetzung in symbolische Namen. (MS: numerische Anzeige ist Default. `-a` bewirkt Übersetzung in symbolische Namen.)
- `-s size` (MS: `-l size`) Anzahl der Datenbytes, die gesendet werden sollen. Default ist 56, was zusammen mit dem 8 Byte ICMP-Header 64-ICMP-Daten-Bytes ergibt.

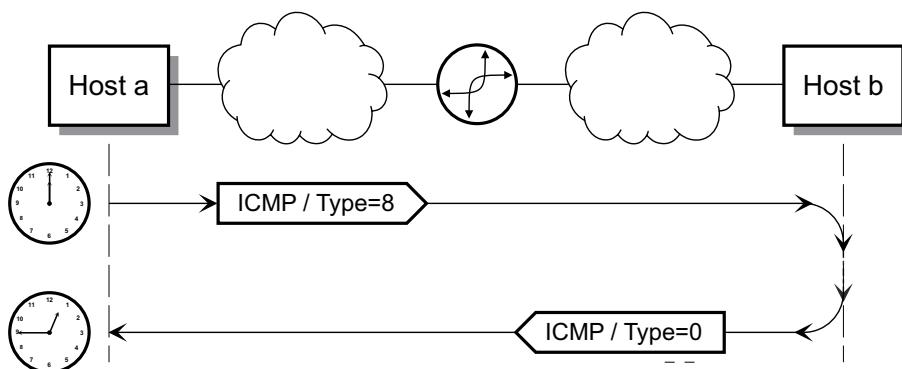


Abbildung 7.42: Prinzip des `ping`-Programms

Wie in Abbildung 7.42 gezeigt, wird mit einer „Echo Request“-Meldung beim Ziel-Host eine „Echo“-Meldung ausgelöst. Aus dem Zeitunterschied zwischen dem Senden und dem Empfangen wird die Laufzeit berechnet. Diese ist allerdings mit Vorsicht zu interpretieren, da die Antwortzeit des Zielhosts mit eingerechnet wird.

7.9 Übungen

7.9.1 IP-Protokoll

a) IP benutzt eine logische Adresse (nicht die MAC-Adresse).

- Warum eignet sich eine IP-Adresse besser für ein weltweites Netz als eine (Ethernet-) MAC-Adresse?
-
.....

- Welche Netzkomponenten „verarbeiten“ die IP-Adresse?

- | | |
|---|---|
| <input type="radio"/> Sender-Netzinterface-Karte | <input type="radio"/> Hub |
| <input type="radio"/> Empfänger-Netzinterface-Karte | <input type="radio"/> Router |
| <input type="radio"/> Switch | <input type="radio"/> Gateway (IP-Terminologie) |

- Wie nennt man das Verfahren, das eine MAC- in eine IP-Adresse „umwandelt“?
-

- Wie nennt man das Verfahren, das eine IP- in eine MAC-Adresse „umwandelt“?
-

b) Eine Firma benötigt 1325 Internet-Adressen. Es wird ein Bereich von mehreren aneinander liegenden C-Netzen zugewiesen, der bei der Adresse 192.188.128.0 beginnt.

- Wie lautet die höchste Adresse des gesamten Bereichs?
-
.....

- Wie lautet die Subnetzmaske für den Bereich?
-
.....

c) Ein IP-Datagramm soll über das 10Base-T-Netz übertragen werden.

- Welche beiden Kapselungsarten sind in den RFCs definiert (Name oder Prinzip angeben)?
-
.....

- Man skizziere die übliche, einfachere Kapselungsart und gebe an, woran der Empfänger erkennt, dass ein Frame ein IP-Datagramm enthält?
-
.....

d) Auf einem IP-Pfad hat *eine* Teilstrecke (irgendwo in der Mitte) eine kleinere Maximum Transmission Unit (z.B. MTU=500) als die übrigen (z.B. MTU=1500):

- Wer ist verantwortlich für die Anpassung (Fragmentierung) langer Datagramme (von z.B. 1500 Bytes) an die kleinere MTU der Teilstrecke?

.....

- Was geschieht dabei mit einem Datagramm und bezüglich Signalisierung, wenn im Header das Don't Fragment Flag (DF) gesetzt ist (zwei Angaben gesucht)?

.....

.....

- Wo erfolgt die Defragmentierung? (Wer setzt die Fragmente wieder zusammen?)

.....

- Warum ist dies so sinnvoll? (2 Gründe)

.....

.....

e) Für die Defragmentierung sind folgende IP-Header-Felder von Bedeutung: (IHL: Internet Header Length, FO: Fragment Offset, MF: More Fragments, TL: Total Length)

Wie wird bei der Defragmentierung

- die gesamte Länge des Datagramms bestimmt?

.....

- erkannt, dass das ganze Datagramm empfangen wurde.

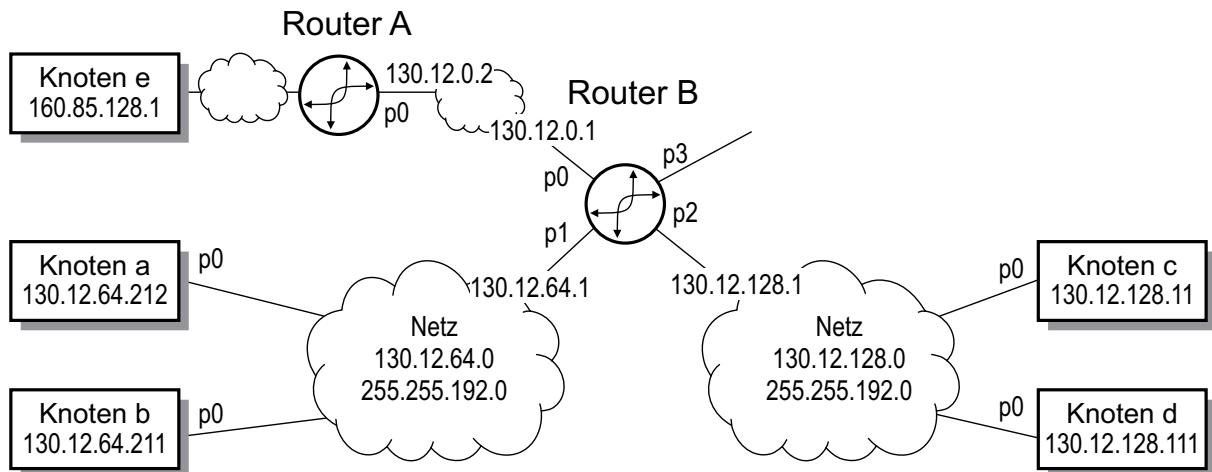
.....

- der Verlust eines Fragments erkannt und behandelt.

.....

7.9.2 Routing

In Abbildung 7.43 ist die Routing-Tabelle der Knoten a und Knoten c nicht richtig. Alle anderen Knoten sind korrekt konfiguriert.



Routing-Tabelle Router B

Netzadresse	Netzmaske	Port	Gateway
130.12.0.0	255.255.192.0	p0	(direkt)
130.12.64.0	255.255.192.0	p1	(direkt)
130.12.128.0	255.255.192.0	p2	(direkt)
130.12.192.0	255.255.192.0	p3	(direkt)
default		p0	130.12.0.2

Routing-Tabelle Knoten a

Netzadresse	Netzmaske	Port	Gateway
130.12.64.0	255.255.224.0	p0	(direkt)
default		p0	130.12.64.1

Routing-Tabelle Knoten c

Netzadresse	Netzmaske	Port	Gateway
130.12.0.0	255.255.0.0	p0	(direkt)
default		p0	130.12.128.1

Abbildung 7.43: Netzwerk mit Routing-Tabellen

- a) Geben Sie in der folgenden Tabelle an, welche Ziele vom Knoten a resp. Knoten c aus (trotz der falschen Konfiguration) erreichbar sind (z.B. mit ping).

Erreichbar	von Knoten a aus? Erklärung!	von Knoten c aus? Erklärung!
Router A		
Router B		
Knoten a		
Knoten b		
Knoten c		
Knoten e		

- b) Man gebe die korrekten Einträge der Routing-Tabellen an (nur die zu korrigierenden).

- Knoten a

.....

.....

- Knoten c

.....

.....

- c) Man gebe die Broadcast-Adresse (All Ones Broadcast) an für

- Knoten b:

.....

- Knoten d:

.....

- d) Was für ein Eintrag / Einträge werden im Router A für die obigen Subnetze benötigt? (Begründung!)

.....

.....

.....

7.9.3 ICMP

Der korrekt konfigurierte Knoten d im Netz von Abbildung 7.43 hat Verbindungsprobleme (z.B. mit dem Router B). Der Verdacht fällt auf die Netzwerkkarte vom Knoten d.

- a) Nach dem Austauschen der Netzwerkkarte kann der Router B zunächst überhaupt keine Verbindung mehr zum Knoten d erstellen. Erklären Sie warum:

.....
.....

- b) Nach einiger Zeit verschwindet das Problem plötzlich (ohne weitere Massnahmen) und alles funktioniert bestens.

.....
.....

- c) Was hätte man nach dem Austauschen der Netzwerkkarte tun sollen, um die Verbindung vom Router B zum Knoten d (sofort, ohne zu warten) testen zu können?

.....
.....

8

Transport Layer

8.1 Einleitung

Im Kapitel 7 wurden das verbindungslose Datagramm-Vermittlungsprotokoll IP und das Fehlerberichtsprotokoll ICMP beschrieben. In diesem Kapitel werden die darauf basierenden Transportprotokolle UDP und TCP der TCP/IP-Protokoll-Familie behandelt.

Der Transport-Layer ist nicht einfach eine weitere Schicht. Wie in Abbildung 8.1 gezeigt, bildet der Transport-Layer in der Regel die Schnittstelle zwischen dem Betriebssystem und den Applikationen.

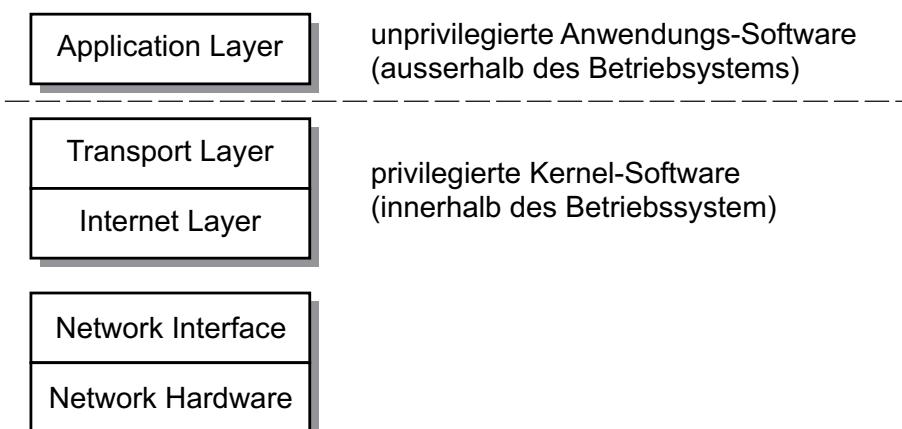


Abbildung 8.1: Der Transport-Layer stellt die Schnittstelle zur Applikation dar

8.1.1 Kapselung

Die Applikationsdaten werden, wie in Abbildung 8.2 gezeigt, mit einem (UDP- oder TCP-) Transport-Header versehen. Das resultierende **Transport Protocol Packet** wird auch **User Datagramm** (UDP) oder **Segment** (TCP) bezeichnet.

Anschliessend werden die Applikationsdaten mit einem IP-Header versehen, wobei dessen Protocol-Feld aussagt, ob es sich um ein UDP- oder ein TCP-Paket handelt.

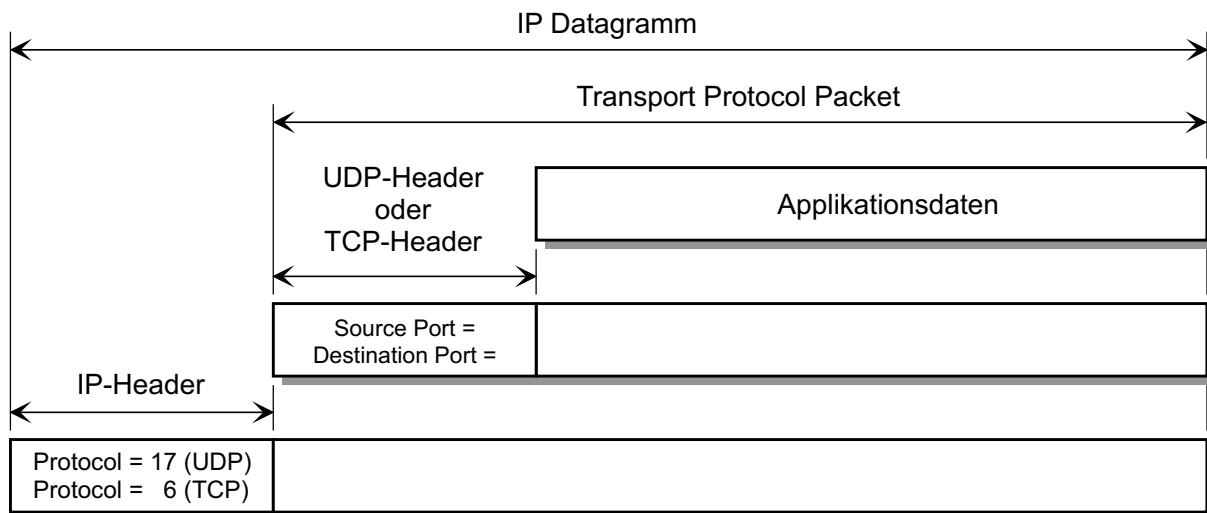


Abbildung 8.2: Kapselung der Applikationsdaten durch den Transport- und IP-Layer

8.1.2 Multiplexen und Demultiplexen

Wir haben gesehen, dass bereits IP in der Lage ist, Datagramme zwischen zwei Hosts zu übertragen. Speziell bei UDP, dem verbindungslosen, unzuverlässigen Transport-Protokoll, stellt sich natürlich die Frage, wozu der Overhead des Transport-Headers dient.

Eine wichtige Aufgabe des Transport-Layers ist das Verteilen der eintreffenden Daten auf die verschiedenen Applikationen. Dieser Vorgang, **Demultiplexen** genannt, basiert auf dem Destination Port des Transport-Headers. Dieses bezeichnet einen *logischen* I/O-Kanal und soll nicht mit Hardware-mässigen I/O-Ports verwechselt werden.

Die Abbildung 8.3 zeigt exemplarisch, wie ein ankommendes Ethernet-Frame an die richtige Applikation weitergeleitet wird.

- Die erste Unterscheidung erfolgt anhand des Type-Feldes im Ethernet-Frame. Ist der Wert z.B. 800h, so wird das Internet-Datagramm dem IP-Modul weitergereicht.
- Das IP-Modul untersucht das Protocol-Feld im Datagramm-Header und leitet das Transport-Protocol-Packet an das entsprechende, höhere Protokoll-Modul weiter.
- Anhand des Destination Ports im TCP-Header (siehe 8.3.8) oder UDP-Header (siehe 8.2.1) wird schliesslich die richtige Applikation für den Empfang der Daten bestimmt.

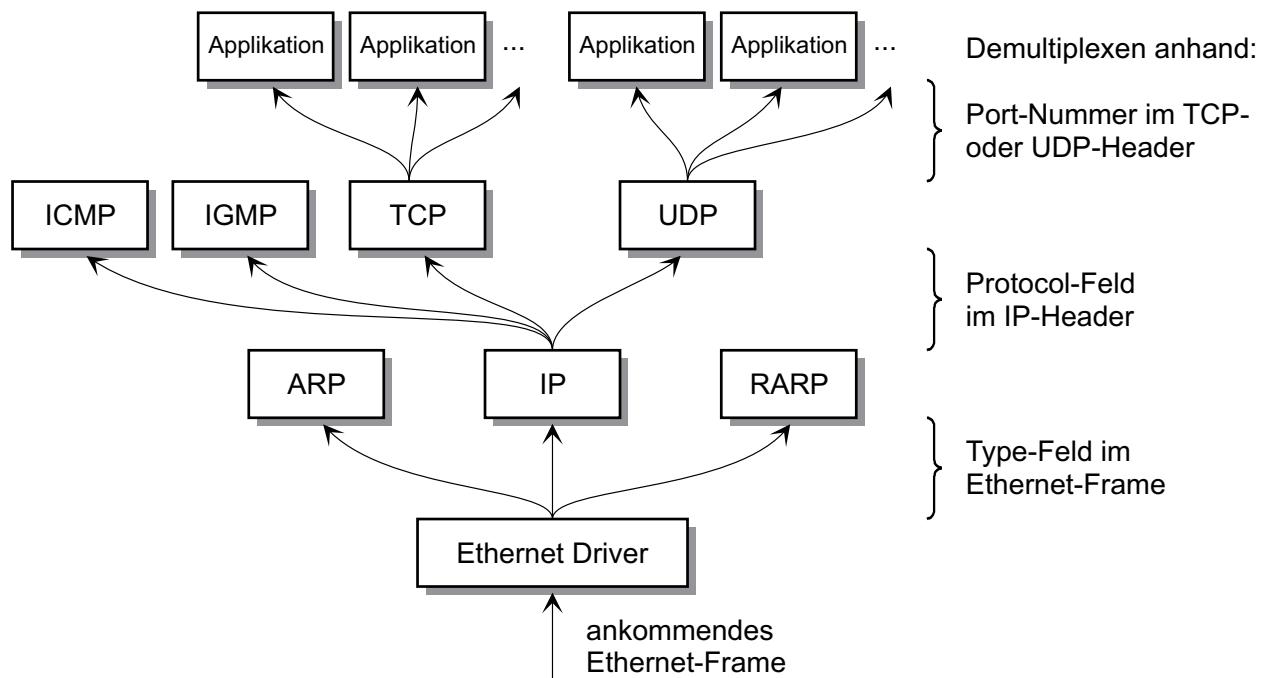


Abbildung 8.3: Demultiplexen am Beispiel einer Ethernet-Schnittstelle

8.2 User Datagram Protocol (UDP)

Das User Datagram Protocol (UDP) ist ein verbindungsloses, unzuverlässiges Transport-Protokoll. Es dient im Wesentlichen dem Multiplexen und Demultiplexen der Datagramme auf die Applikationen.

8.2.1 UDP-Header

Die Kapselung wurde im Abschnitt 8.1.1 schon vorgestellt. Die Abbildung 8.4 zeigt den Aufbau des UDP-Headers.

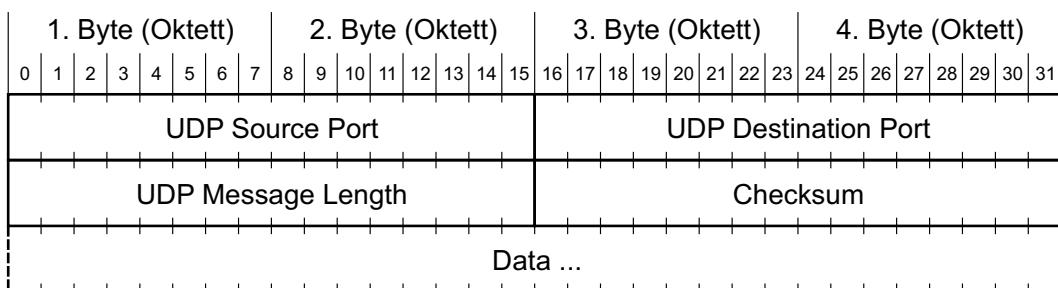


Abbildung 8.4: Felder des UDP-Headers

UDP Source Port und **Destination Port** enthalten je ein Word (16 Bit), das zum Verteilen der eintreffenden Datagramme auf die entsprechenden Prozesse verwendet wird. Das Source

Port kann optional verwendet werden, um dem Empfänger mitzuteilen, an welches Port er eine Antwort zurücksenden soll. Falls es nicht verwendet wird, soll sein Wert null sein.

UDP Message Length gibt die Länge (in Byte) des Transport-Protocol-Packets an. Dieses besteht aus dem UDP-Header und den Applikationsdaten. Der minimale Wert (UDP-Header allein, ohne Daten) ist acht.

Checksum dient zur Fehlererkennung. Für die Berechnung wird zunächst ein sogenannter Pseudo-Header gebildet, der einige Felder des IP-Headers (IP-Source- und Destination-Address, Protocol) und die Länge des UDP-Datagramms enthält. Die eigentliche Berechnung erfolgt über den Pseudo-Header, den UDP-Header *und* die Daten, indem das Einerkomplement der Summe aller Einerkomplemente aller 16-Bit-Wörter gebildet wird.

Anstelle der Prüfsumme kann auch null angegeben werden. Damit kann eine zeitkritische Applikation etwas Prozessorzeit sparen.

8.2.2 Ports

Port-Nummern werden für UDP und / oder TCP reserviert. Es werden drei Bereiche unterschieden:

Ports 0–1023 Diese Ports werden System Ports oder Well Known Ports genannt. Für bekannte Applikationen sind bei der IANA feste Port-Nummern reserviert. Neue Zuordnungen erfolgen durch die Internet Engineering Task Force (IETF). Die Liste kann unter <http://www.iana.org/assignments/service-names-port-numbers/> eingesehen werden.

Unter Linux sind die Ports im File `/etc/services` abgelegt. Die folgende Tabelle zeigt einen Auszug davon.

Ports 1024–49151 In diesem Bereich liegen die User Ports. Neue Nummern können auf Antrag ohne Beteiligung der IETF zugeordnet werden.

Ports 49152–65535 Über den Bereich der Dynamic Ports kann frei verfügt werden.

Protokoll	Port	Beschreibung
	0	Reserved
echo	7	Echo
discard	9	Discard
daytime	13	Daytime
time	37	Time
domain	53	Domain Name Server
tftp	69	Trivial File Transfer
www-http	80	World Wide Web HTTP
pop3	110	Post Office Protocol - Version 3
ntp	123	Network Time Protocol
snmp	161	SNMP
xdmcp	177	X Display Manager Control Protocol
imap3	220	Interactive Mail Access Protocol v3
who	513	maintains data bases showing who's timeserver
timed	525	

8.3 Transmission Control Protocol (TCP)

8.3.1 Einleitung

Die Hauptaufgabe von TCP als Transport Layer Protokoll ist die Bereitstellung einer zuverlässigen Übertragung. TCP bewältigt dabei eine scheinbar unmögliche Aufgabe: Es benutzt den vom IP bereitgestellten, unzuverlässigen Datagrammdienst beim Austausch von Daten zwischen zwei Hosts, bietet den Anwendungsprogrammen aber einen zuverlässigen Datendienst. TCP muss dabei Verluste und Verzögerungen im Internet ausgleichen. Dabei soll es weder die benutzten Subnetze noch die Router überlasten.

Nach der Betrachtung des von TCP bereitgestellten Dienstes werden im Folgenden die Techniken beschrieben, mit denen TCP Zuverlässigkeit erreicht.

a) Bedarf nach zuverlässigem Transport

Im Allgemeinen gehen Anwendungsprogrammierer zu Recht davon aus, dass die Rechner zuverlässig und fehlerfrei sind. Muss z.B. eine Anwendung die Daten drucken, so schickt sie diese zum Ausgabegerät (z.B. einen Drucker) und muss nicht prüfen, ob sie dort intakt ankommen. Sie verlässt sich vielmehr darauf, dass das System zuverlässig arbeitet. Es gewährleistet, dass keine Daten verlorengehen, dupliziert werden oder in der falschen Reihenfolge ankommen.

Verteilte Anwendungen sollten möglichst gleich programmiert werden können wie lokale. Idealerweise bietet die Protokoll-Software Schnittstellen mit den gleichen Eigenschaften, wie sie für lokale Geräte zur Verfügung stellen. Das heisst, die Protokoll-Software muss eine prompte und zuverlässige Übertragung sicherstellen. Die Daten müssen in genau der Reihenfolge ankommen, in der sie gesendet wurden, und kein Teil darf dupliziert werden oder verlorengehen.

b) Das TCP-Protokoll

Zuverlässigkeit liegt im Verantwortungsbereich eines Transportprotokolls. Die Anwendungen interagieren beim Datenaustausch mit einem Transportdienst. In der TCP/IP-Protokollfamilie wird zuverlässiger Transportdienst vom TCP (Transmission Control Protocol) zugesichert. TCP ist bemerkenswert, weil es ganz nebenbei ein grosses Problem löst: Trotz der zahlreichen Protokolle, die eigens als generelle Transportprotokolle entwickelt wurden, erwies sich TCP als überlegen. Folglich wird für die meisten Internet-Anwendungen TCP benutzt.

8.3.2 TCP-Dienste für Anwendungen

Aus Sicht des Anwendungsprogramms weist der von TCP gebotene Dienst sieben wichtige Merkmale auf:

Verbindungsorientierte Übertragung: Die TCP-Schnittstelle bietet einen *verbindungsorientierten* Dienst, bei dem eine Anwendung zuerst eine Verbindung zum Ziel aufbauen muss, bevor Daten übertragen werden können. Jede TCP-Verbindung hat genau zwei Endpunkte (keine Multicast-Verbindungen).

Hohe Zuverlässigkeit: TCP garantiert die Übertragung der Daten ohne Datenverlust und in der richtigen Reihenfolge.

Vollduplexübertragung: In einer TCP-Verbindung können Daten in beide Richtungen fliessen, so dass beide Seiten jederzeit Daten senden und empfangen können.

Stream-Schnittstelle: Dies bedeutet, dass eine Anwendung eine fortlaufende, unstrukturierte Bytefolge über die Verbindung sendet oder empfängt. Die Bytes treffen in der genau gleichen Reihenfolge beim Empfänger wieder ein, in der sie gesendet wurden. So können z.B. Records (Zeilen etc.) beliebiger Länge transparent übertragen werden. Die Record-Struktur hat für TCP aber keine Bedeutung – weder für die Applikationsschnittstelle noch für die Übertragung.

Zuverlässiger Verbindungsauflaufbau: TCP fordert von zwei kommunikationswilligen Anwendungen die Vereinbarung über eine Verbindung. Eventuell aus früheren Verbindungen vorhandene Duplikatpakete erscheinen nicht als gültige Antworten und wirken sich auch nicht anderweitig auf die neue Verbindung aus.

Eleganter Verbindungsabbau: Ein Anwendungsprogramm wird üblicherweise eine Verbindung aufbauen, Daten austauschen und dann den Abbau der Verbindung fordern. TCP gewährleistet die zuverlässige Zustellung aller Daten in beide Richtungen auch beim Abbau der Verbindung.

8.3.3 Ende-zu-Ende-Dienst

TCP ist ein Ende-zu-Ende-Protokoll, weil es eine Verbindung direkt von einer Anwendung eines Hosts zu einer Anwendung eines entfernten Hosts bietet. Die Anwendungen können von TCP den Aufbau einer Verbindung anfordern, Daten austauschen und die Verbindung abbauen.

Die von TCP gebotenen Verbindungen nennt man **virtuell**, weil sie von der Software hergestellt werden. Das zugrundeliegende Internet bietet weder Hardware- noch Software-Unterstützung für Verbindungen. Statt dessen tauschen die Transport-Layer-Module auf den beiden Hosts Nachrichten aus, um die Illusion einer Verbindung zu schaffen.

TCP benutzt IP zur Beförderung von Nachrichten. Jede TCP-Nachricht wird in ein IP-Datagramm gekapselt (siehe Abschnitt 8.1.1) und im Netz übertragen. Kommt das Datagramm am Zielhost an, gibt IP den Inhalt an TCP ab. IP liest dabei aber keine Nachrichten, noch interpretiert es sie. Die Abbildung 8.5 zeigt das Beispiel eines Internets mit zwei Hosts und einem Router, aus dem die Beziehung zwischen der TCP- und der IP-Software deutlich wird.

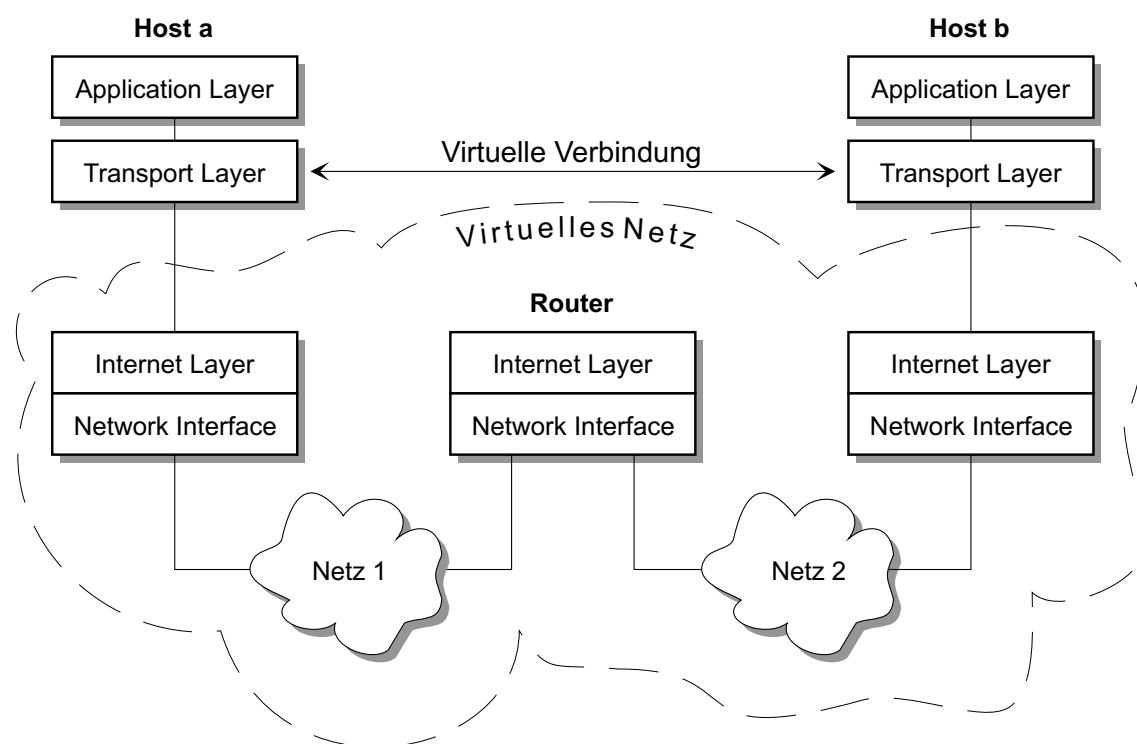


Abbildung 8.5: TCP schafft die Illusion einer fehlerfreien, virtuellen Verbindung

Wir sehen an diesem Beispiel, dass an beiden Enden einer virtuellen Verbindung die TCP-Software erforderlich ist, dass aber auf den dazwischenliegenden Routern *keine* TCP-Module benötigt werden. Aus TCP-Sicht ist das gesamte Internet ein Kommunikationssystem (virtuelles Netz), in dem Nachrichten transparent ausgetauscht werden können, also ohne dass deren Inhalt geändert oder interpretiert wird.

8.3.4 Zuverlässigkeit

Ein Transportprotokoll wie TCP muss sorgfältig ausgelegt werden, um Zuverlässigkeit zu erreichen. Die vorrangigen Probleme sind die unzuverlässige Übertragung durch das zugrundeliegende Kommunikationssystem und das An-/Abschalten von Hosts. Um diese Schwierigkeiten zu verstehen, betrachte man eine Situation, in der zwei Anwendungsprogramme eine TCP-Verbindung aufbauen, Daten übertragen, die Verbindung abbauen und dann eine neue Verbindung herstellen. Jede Nachricht kann potentiell verloren gehen, dupliziert werden, verspätet oder in einer falschen Reihenfolge ankommen. Es ist darum möglich, dass eine Nachricht der ersten Verbindung dupliziert wird und erst nach dem Aufbau der zweiten Verbindung eintrifft. Nachrichten müssen darum eindeutig einer Verbindung zugeordnet werden können. Andernfalls akzeptiert das Protokoll solche Nachrichtenduplikate von der früheren Verbindung und lässt sie in der neuen Verbindung zu.

Fehler, die durch den Neustart angeschlossener Hosts verursacht werden können, waren eine weitere grosse Herausforderung bei der Entwicklung des TCP-Protokolls. Man stelle sich eine Situation vor, in der zwei Anwendungsprogramme eine Verbindung aufbauen und einer der Hosts plötzlich neu gestartet wird. Die Protokoll-Software auf diesem Host hat zwar keine Kenntnis über die Verbindung, für die Protokoll-Software des Hosts auf der Gegenseite ist die Verbindung aber dennoch gültig. Eine besondere Schwierigkeit entsteht durch verzögerte Paketduplicaten, weil ein Protokoll in der Lage sein muss, Pakete aus einem früheren Host-Start zu verwerfen (d.h. die Pakete müssen auch dann verworfen werden, wenn die Protokoll-Software keine Aufzeichnung der früheren Verbindung hat).

a) Behandlung von Paketverlusten

Wie erreicht TCP Zuverlässigkeit? Die wichtigste Technik dafür ist die Neuübertragung der Daten (**Retransmission**). Vor jeder Datenübertragung startet der Sender einen Timer. Trifft das fehlerfreie Segment beim Empfänger ein, sendet dieser eine Bestätigung (**Acknowledgment**) an den Sender. Läuft der Timer vor Ankunft der Bestätigung ab, überträgt der Sender das Segment (unaufgefordert) nochmals. Ist das Segment fehlerhaft, erfolgt *keine* Rückmeldung vom Empfänger zum Sender.

Das Prinzip der Retransmission wird in der Abbildung 8.6 verdeutlicht. Die Elemente links entsprechen den Ereignissen im Sender, diejenigen rechts dem Ablauf im Empfänger. Die Zeit verläuft von oben nach unten.

b) Adaptive Wartezeit

Die interessante Frage bei diesem Verfahren ist, wie lange der Sender mit der Neuübertragung warten soll. Lange Wartezeiten führen bei jedem Segment-Fehler zu einer entsprechenden Verzögerung in der Übertragung. Zu kurze Wartezeiten bewirken eine unnötige Neuübertragung des Segments und belasten die Verbindung. In beiden Fällen wird der Nettodurchsatz reduziert.

Bestätigungen in einem LAN können innerhalb von wenigen Millisekunden erwartet werden; internationale Verbindungen (über Satellit) können Sekunden benötigen. In einem LAN sollte TCP

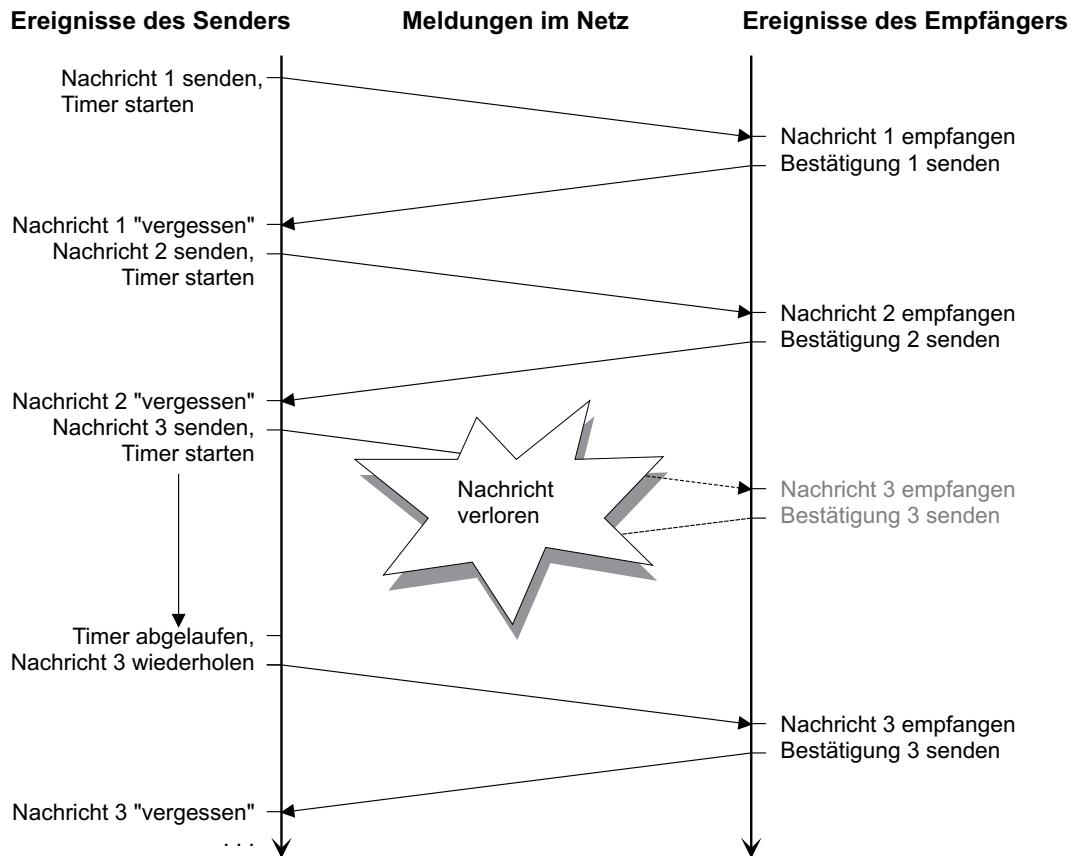


Abbildung 8.6: Beispiel einer Neuübertragung nach einem Datenverlust

deshalb mit der Neuübertragung entsprechend kurz warten. Eine erneute Übertragung nach wenigen Millisekunden wäre jedoch für Satellitenverbindungen ungünstig, weil dadurch unnötig Netzbänderbreite verbraucht wird.

Die zum Senden einer Nachricht und Empfangen einer Bestätigung benötigte Gesamtzeit wird **Umlaufverzögerung, Round-Trip Time** oder **Round-Trip Delay** genannt. Die Wartezeiten bis zur Neuübertragung wird als **Retransmission Time-Out** bezeichnet.

Vor der Entwicklung von TCP verwendeten die Transportprotokolle für den Retransmission Time-Out einen festen Wert. Die TCP-Entwickler erkannten, dass dieser Ansatz für das Internet ungeeignet ist und entschlossen sich deshalb zur Einführung einer adaptiven Neuübertragung. Das bedeutet, dass TCP auf jeder Verbindung die momentane Umlaufverzögerung überwacht und den Retransmission Time-Out entsprechend einstellt (Abbildung 8.7).

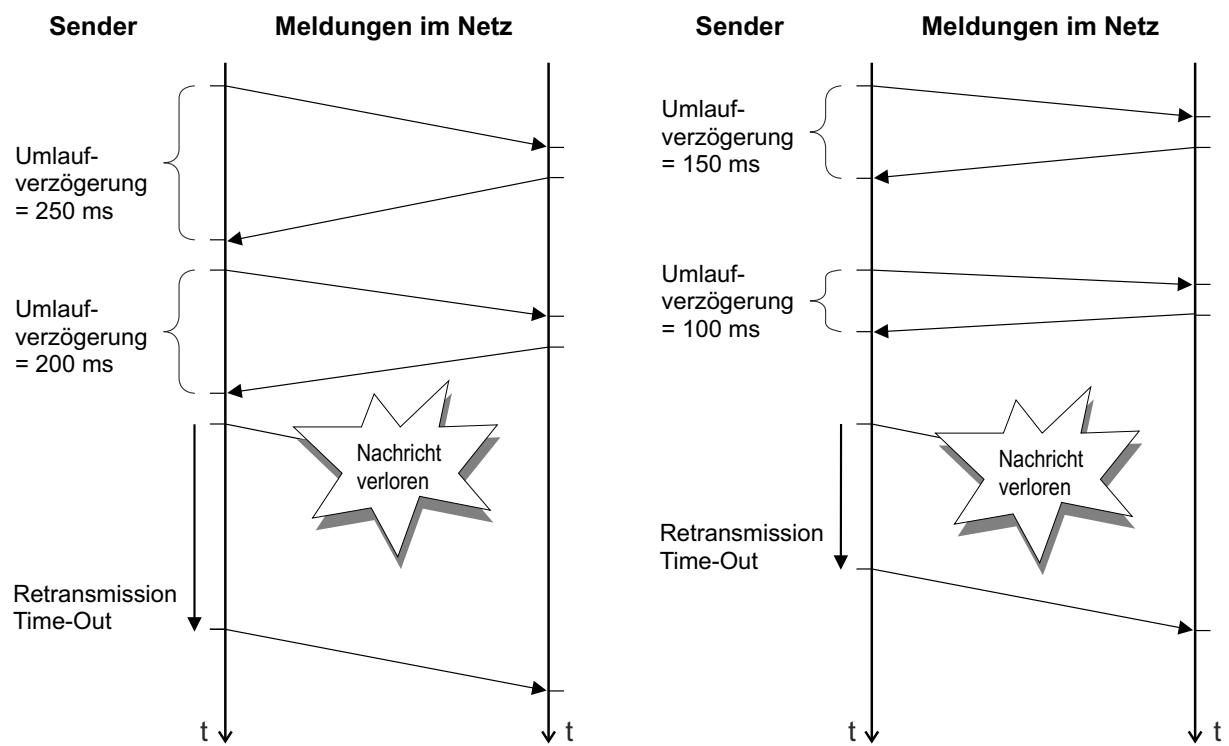


Abbildung 8.7: Retransmission Time-Out bei unterschiedlichen Umlaufverzögerungen

Das Ziel ist, ausreichend lange zu warten, dass nur ein Paketverlust zu einem Timeout führt, unnötige Wartezeiten aber zu vermeiden. Wir sehen an diesem Beispiel, dass TCP den Retransmission Time-Out länger als die mittlere Umlaufverzögerung ansetzt, damit möglichst alle auftretenden Umlaufverzögerungen zeitlich darunter liegen.

TCP muss aber ein noch schwierigeres Problem bewältigen als die Unterscheidung zwischen lokalen und entfernten Zielen: Die momentanen Verkehrsbedingungen (z.B. Lastspitzen) auf Kommunikationspfaden wirken sich auf die Verzögerung aus. TCP muss mit einer Vielzahl von sich addierenden Verzögerungen umgehen können, die sich innerhalb kurzer Zeit und über einen grossen Wertebereich ändern können.

Die Abbildung 8.8 zeigt eine Messreihe der Umlaufverzögerung (Segmente mit 1000Byte, Schweiz-China) und wie TCP den Retransmission Time-Out anpasst.

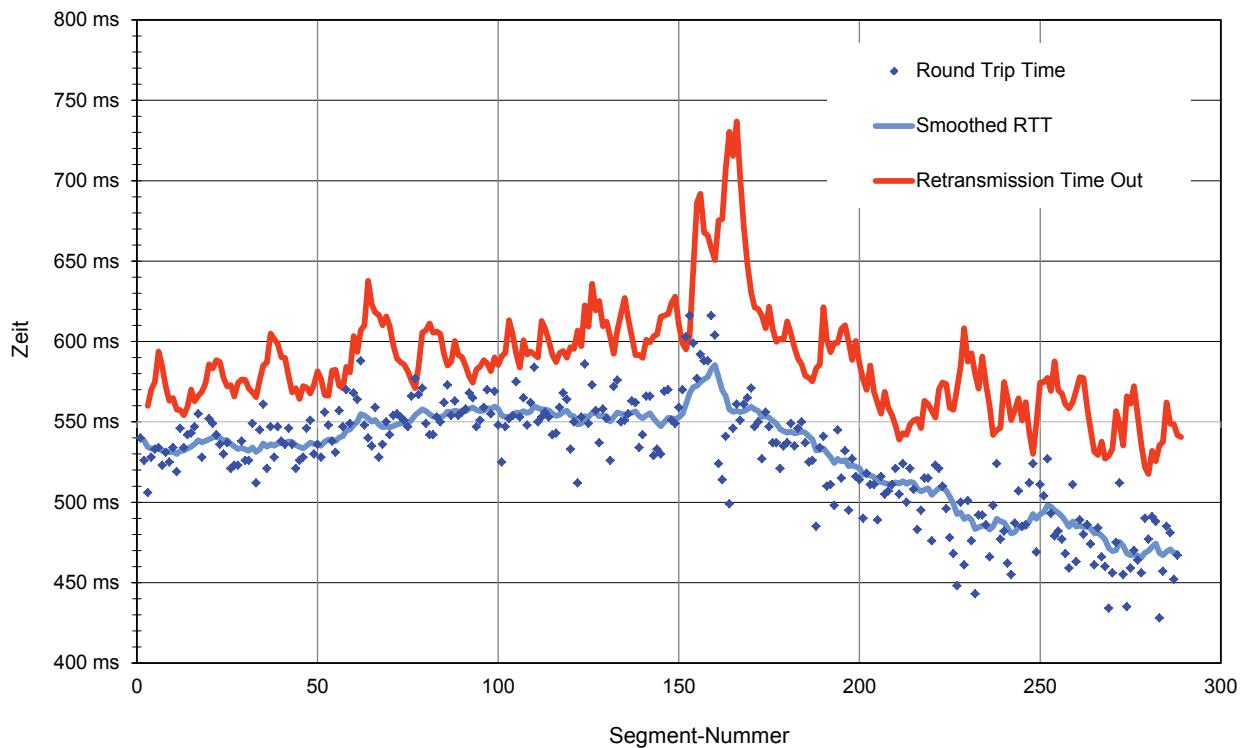


Abbildung 8.8: Beispiel von Umlaufverzögerungen und resultierendem Retransmission Time-Out

Die Berechnung des Retransmission Time-Out ist im RFC-2988 definiert. TCP misst bei jeder aktiven Verbindung die Round-Trip Time (RTT_n) und bildet daraus einen gewichteten Mittelwert $SRTT$ (Smoothed Round-Trip Time).

$$SRTT_n = (1 - \alpha) * SRTT_{n-1} + \alpha * RTT_n \quad \text{mit } \alpha = 0.125$$

Zusätzlich ermittelt TCP die Streuung $RTTVAR$ durch einen gewichteten Mittelwert der Abweichungen davon.

$$RTTVar_n = (1 - \beta) * RTTVar_{n-1} + \beta * |SRTT_n - RTT_n| \quad \text{mit } \beta = 0.25$$

Der Retransmission Time-Out RTO ist eine Kombination aus der Umlaufverzögerung und dem Mehrfachen der Streuung.

$$RTO_n = SRTT_n + 4 * RTTVar_n$$

Erfahrungen haben gezeigt, dass die adaptive Neuübertragung von TCP gut funktioniert. Bei grossen Streuungen setzt TCP den Retransmission Time-Out auf einen deutlich über dem Mittel liegenden Wert, um Spitzenaufkommen zu berücksichtigen.

8.3.5 Fluss-Steuerung

a) Flusskontrolle zur Verhinderung von Datenüberlauf

Nicht alle Computer laufen in der gleichen Geschwindigkeit. Sendet ein Computer in einem Netz Daten schneller, als sie vom Ziel aufgenommen werden können, entsteht ein Datenüberlauf (**Data Overrun**). Dadurch gehen Daten verloren. Zur Lösung dieses Problems wurden verschiedene Techniken entwickelt, die man unter dem Begriff **Fluss-Steuerung (Flow Control)** zusammenfasst.

Die einfachste Form der Fluss-Steuerung ist ein **Stop-and-Wait-Verfahren**, wie es im Abschnitt 4.11 eingeführt wurde. Dabei wartet der Sender nach jeder Paketübertragung auf eine Quittung des Empfängers. Ist der Empfänger für ein weiteres Paket bereit, sendet er eine Quittung. Stop-and-Wait-Protokolle verhindern zwar einen Datenüberlauf, können aber zu einer ineffizienten Nutzung der Netzbänderbreite führen. Um den Grund dafür zu verstehen, betrachte man ein Netz mit einer Paketgröße von 1000 Byte, einer Durchsatzkapazität von 2 Mbps und einer Verzögerung von 50 Millisekunden. Die Netz-Hardware kann 2 Mbps zwischen zwei Computern befördern. Nach der Übertragung eines Pakets muss der Sender aber 100 ms warten, bis er das nächste Paket senden kann (d.h. 50 ms, bis das Paket den Empfänger erreicht und 50 ms, bis eine Bestätigung zurückkommt). Die maximale Rate, in der Daten im Stop-and-Wait-Betrieb gesendet werden können, beträgt also 1 Paket alle 100 ms. Drückt man das als Bitrate aus, kann unter Stop-and-Wait eine Rate von höchstens 80'000 Bit/s erreicht werden. Das entspricht nur 4% der Hardware-Kapazität.

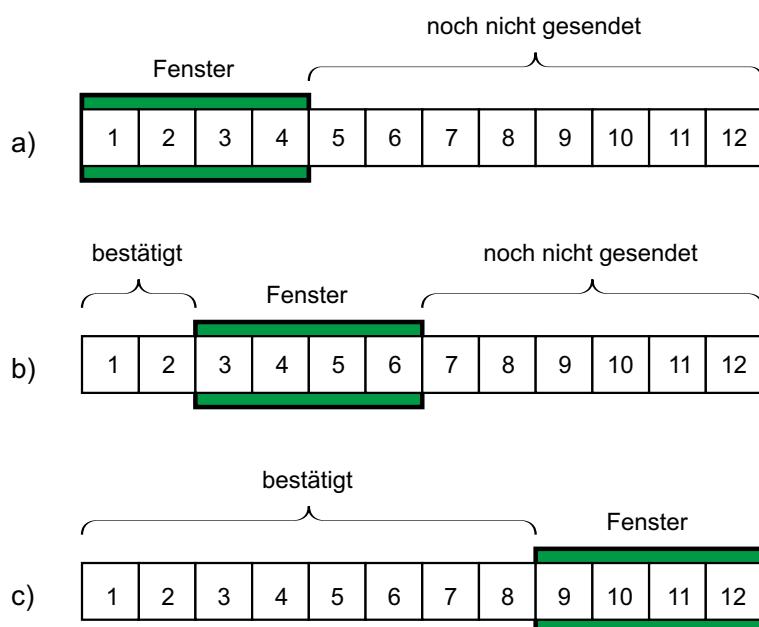


Abbildung 8.9: Prinzip eines vier Pakete umfassenden Übertragungsfensters

Höhere Durchsatzraten sind durch Protokolle mit Übertragungsfenstern möglich. Bei dieser Fluss-Steuerungstechnik, die **Sliding Window** genannt wird, werden Sender und Empfänger auf eine feste Fenstergröße programmiert. Sie entspricht der maximalen Datenmenge, die vor Ankunft

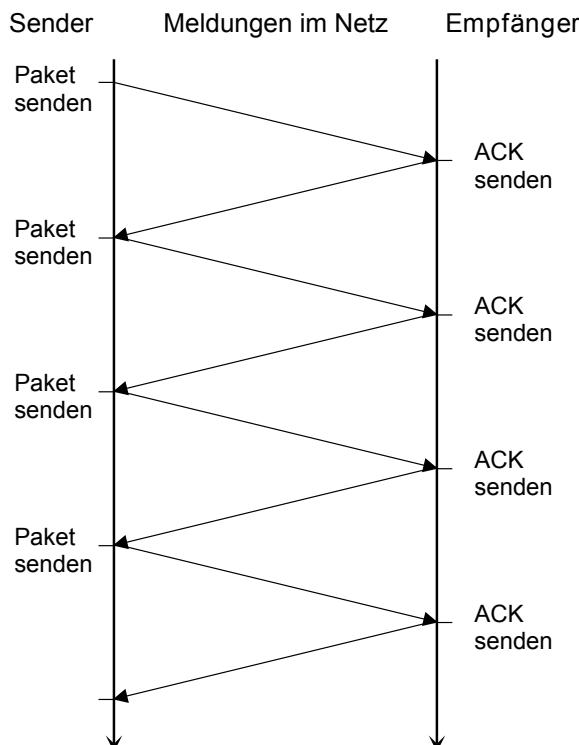
einer Bestätigung gesendet werden kann. So können Sender und Empfänger beispielsweise eine Fenstergrösse von vier Paketen vereinbaren. Der Sender beginnt mit den zu sendenden Daten, zieht die zum Füllen des ersten Fensters ausreichende Datenmenge heraus und überträgt Kopien. Ist Zuverlässigkeit angesagt, behält der Sender eine Kopie für den Fall, dass erneut übertragen werden muss. Beim Empfänger muss ein entsprechender Bufferplatz bereitstehen, um den gesamten Fensterinhalt aufzunehmen. Der Empfänger gibt das Paket an die angesprochene Anwendung weiter und übermittelt dem Sender eine Bestätigung. Nach Erhalt der Bestätigung verwirft der Sender seine Kopie des bestätigten Pakets und überträgt das nächste.

Das Beispiel (Abbildung 8.9) basiert auf einem vier Pakete umfassenden Übertragungsfenster, durch das die ausgehenden Daten fliessen. Es zeigt deutlich, warum diese Technik wörtlich übersetzt Schiebefenster heisst:

- a) Zustand bei Beginn der Übertragung
- b) Zustand nach Bestätigung von zwei Paketen
- c) Zustand nach Bestätigung von acht Paketen

Der Sender kann alle im Fenster stehenden Pakete unmittelbar übertragen. Aus der Gegenüberstellung der Stop-and-Wait- und der Sliding-Window-Verfahren in Abbildung 8.10 wird deutlich, warum sich die Durchsatzrate mit der Schiebefenstertechnik beträchtlich steigern lässt.

a) Stop and Wait Flow Control



b) Sliding Window Flow Control

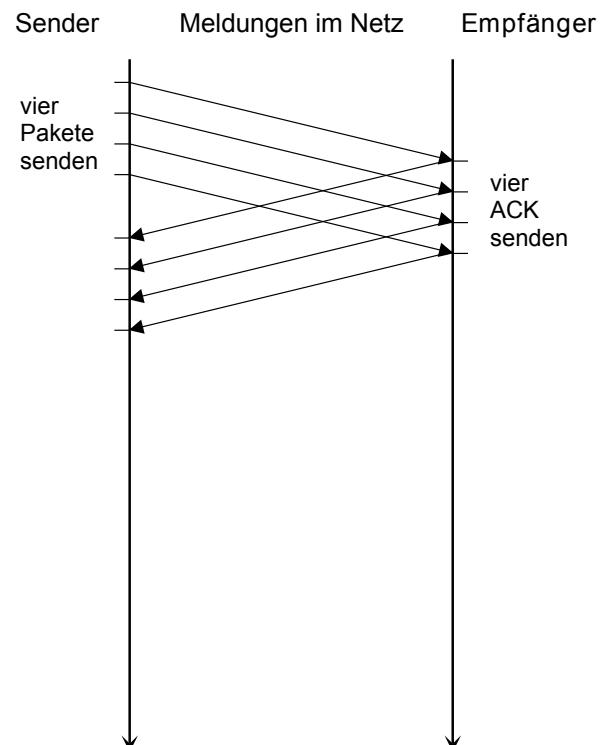


Abbildung 8.10: Übertragung von Nachrichten in einer Folge von vier Paketen

b) TCP-Fluss-Steuerung

TCP benutzt zur Steuerung des Datenflusses einen Sliding-Window-Mechanismus. Beide Seiten einer TCP-Verbindung benötigen einen Buffer für die eingehenden Daten. Die zu einem Zeitpunkt verfügbare Anzahl Bytes des Buffers ist bei TCP das Window. Beim Verbindungsaufbau wird die Grösse des angelegten Buffers der jeweils anderen Seite mitgeteilt. Nach Ankunft eines Segments gibt der Empfänger in der Bestätigung auch das restliche Window an. Dies nennt man **Window Advertisement** (Fensteranzeige). Dem Sender wird also der Empfang bestätigt und gleichzeitig der aktuelle Stand des Windows mitgeteilt.

Kann die empfangende Anwendung die Daten so schnell lesen, wie sie ankommen, zeigt der Empfänger mit jeder Bestätigung eine positive Window-Grösse an. Arbeitet die sendende Seite aber schneller als die empfangende, füllt sich der Buffer beim Empfänger, was ihn veranlasst, gelegentlich ein Window von null (**Zero Window**) anzuzeigen. Ein Sender, dem ein Zero Window angezeigt wird, muss mit dem Senden so lange innehalten, bis der Empfänger (unaufgefordert) wieder ein positives Window anzeigt.

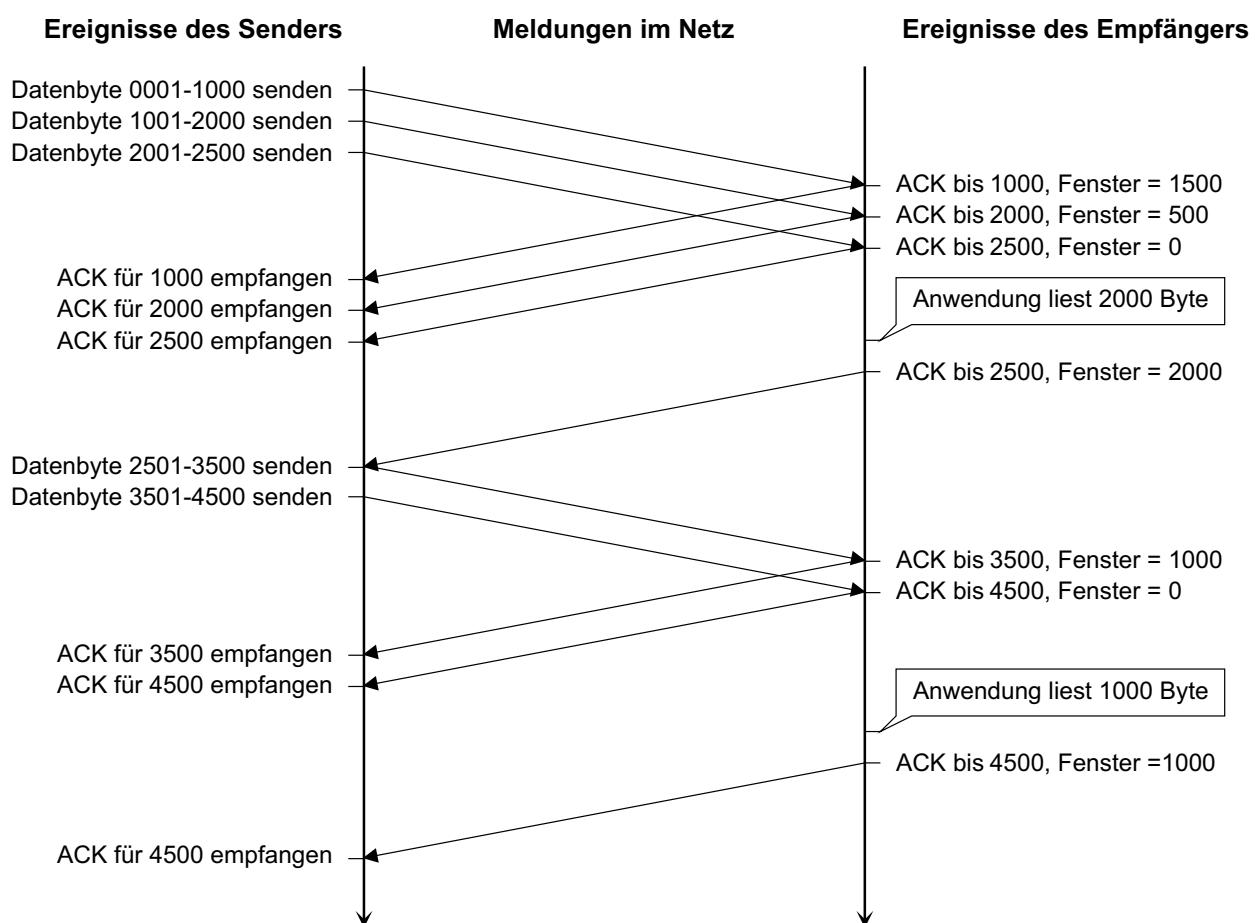


Abbildung 8.11: Übertragung von TCP-Segmenten mit einem Window von 2500 Byte

Im Beispiel von Abbildung 8.11 benutzt der Sender eine maximale Segmentgrösse von 1000 Byte. Beim Verbindungsaufbau (in der Abbildung nicht ersichtlich) wurde vom Empfänger ein anfängliches Window von 2500 Byte angezeigt. Der Sender überträgt sofort drei Segmente – zwei

mit je 1000 Byte und eines mit 500 Byte. Bei Ankunft der Segmente erzeugt der Empfänger eine Bestätigung, die zeigt, dass sich das Window um die angekommene Datenmenge reduziert hat. Die ersten drei Segmente füllen den Buffer des Empfängers, bevor die empfangende Anwendung Daten aufnehmen kann. Das Window erreicht null (Zero Window), womit keine weiteren Daten mehr gesendet werden dürfen.

Nachdem die empfangende Anwendung 2000 Byte aufgenommen hat, sendet die empfangende TCP-Software unaufgefordert eine Bestätigung, mit der ein Window von 2000 Byte angezeigt wird. Der Sender weiss nun, dass er nach den bereits gesendeten 2500 Byte weitere 2000 übertragen kann und sendet darauf zwei zusätzliche Segmente. Deren Ankunft wird vom Empfänger bestätigt, zusammen mit der Anzeige eines um jeweils 1000 Byte reduzierten Windows. Nach dem zweiten Segment hat der Empfänger wieder ein Zero Window, so dass der Sender warten muss, bis die empfangende Anwendung erneut eine gewisse Datenmenge entgegen genommen hat etc.. Dieser Ablauf wiederholt sich, bis alle Daten übertragen und erfolgreich empfangen wurden.

8.3.6 Zuverlässiger Verbindungs-Aufbau und -Abbau

Um zu gewährleisten, dass Verbindungen zuverlässig aufgebaut und beendet werden, benutzt TCP ein **Dreiweg-Handshaking**-Verfahren (**3 Way Handshake**), bei dem mindestens drei Nachrichten ausgetauscht werden. Ein dreifacher Austausch ist notwendig, um ungeachtet von Paketverlusten, Duplikaten und Verzögerungen eine eindeutige Verbindung sicherzustellen.

Eine TCP-Verbindung durchläuft beim Auf- und Abbau verschiedene Zustände:

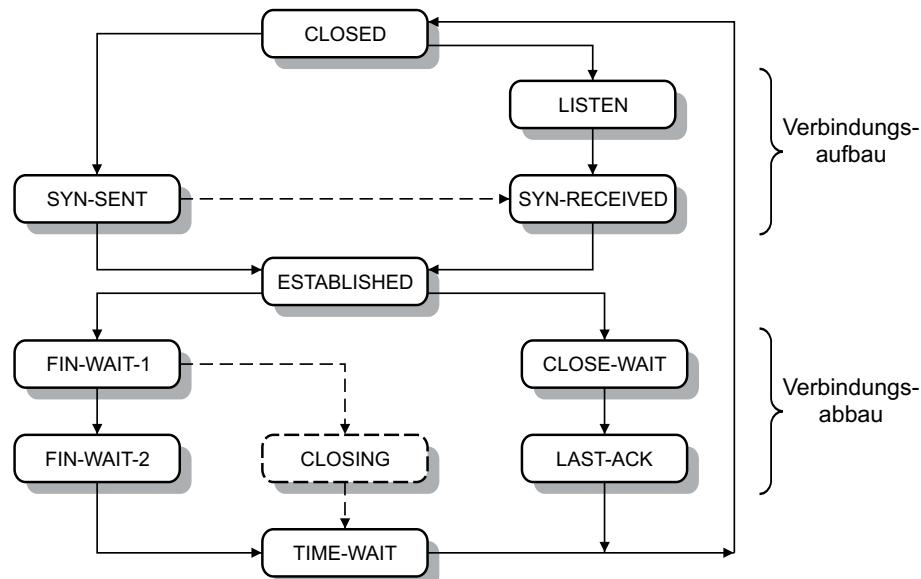


Abbildung 8.12: Übersicht der Zustände einer TCP-Verbindung

Das Diagramm in Abbildung 8.12 umfasst sowohl Aufbau, Betrieb und Abbau einer Verbindung als auch die passive Rolle von Servern, die aktive Rolle von Clients und simultane Aktivitäten von gleichberechtigen Applikationen (gestrichelt).

Zustandsübergänge werden später noch genauer erläutert. Die Zustände haben die folgenden Bedeutungen:

CLOSED	Die Verbindung existiert noch nicht: sozusagen der pränatale Ruhezustand.
LISTEN	Typischer passiver Zustand eines Servers, der auf eine Verbindungsanforderung eines Clients wartet.
SYN-SENT	Der Client hat eine Verbindungsanforderung geschickt und wartet auf eine passende Verbindungsbestätigung.
SYN-RECEIVED	Der Server wartet auf eine Bestätigung seiner Verbindungsbestätigung.
ESTABLISHED	Die Verbindung besteht. Es können Daten gesendet und empfangen werden.
FIN-WAIT-1	Die lokale Applikation will die Verbindung schliessen. Es wurde darum eine Abbauanforderung geschickt. Nun muss auf die Bestätigung des Remote-Endes gewartet werden.
FIN-WAIT-2	Die Abbauanforderung wurde bestätigt. Nun muss gewartet werden, bis das Remote-Ende bereit ist, die Verbindung auch tatsächlich abzubauen.
CLOSE-WAIT	Das Remote-Ende will die Verbindung abbauen. Es muss gewartet werden, bis die lokale Applikation auch bereit ist, die Verbindung zu schliessen.
LAST-ACK	Der Verbindungsabbau wurde bestätigt. Nun wird auf die Bestätigung der Bestätigung gewartet.
CLOSING	Das Remote-Ende hat gleichzeitig einen Verbindungsabbau verlangt. Nun muss noch auf die Bestätigung gewartet werden.
TIME-WAIT	Die letzte Bestätigung wurde gesendet. Damit diese bei einem Verlust nochmals übertragen werden kann, wird eine fixe Zeit gewartet, bevor die Verbindung endgültig abgebrochen wird.

Auf beiden Seiten einer Verbindung, also auf Server und Client, existiert eine unabhängige TCP State Machine. Wie Abbildung 8.13 zeigt, werden die State Machines durch Events beeinflusst, die einerseits von der lokalen Applikation und andererseits von der entfernten State Machine stammen können.

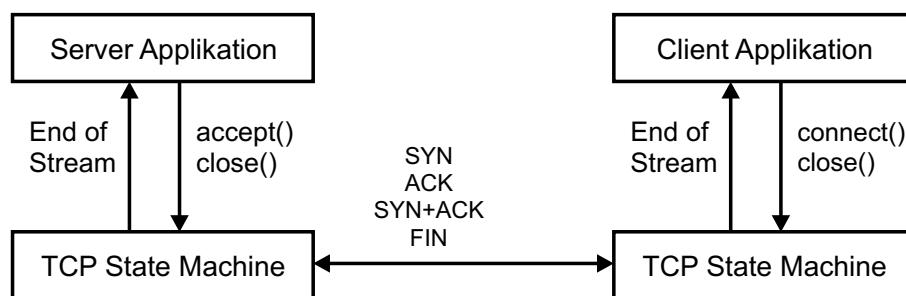


Abbildung 8.13: Systemmodell einer TCP-Client-Server-Verbindung

Applikationen beeinflussen die State Machine mit den Systemaufrufen `accept()`, `connect()` und `close()` und bekommen als Rückmeldung „End of Stream“. Die beiden TCP State Machines signalisieren sich Events mit Meldungen. Diese sind normale TCP-Segmente, die im Header (siehe Abschnitt 8.3.8) die Flags SYN, FIN und/oder ACK gesetzt haben.

a) Verbindungsauftbau

Nachrichten, die zum Aufbau einer Verbindung benutzt werden, haben das SYN-Flag im Header gesetzt und heissen SYN-Segmente. Wie andere Nachrichten werden SYN-Segmente mit einem ACK-Segment bestätigt und bei Verlust wiederholt.

Das vereinfachte State-Diagramm in Abbildung 8.14 beschreibt den TCP-Verbindungsauftbau für Server (passive open) und Client (active open). Die States eines Clients sind auf der linken Seite des Diagramms dargestellt; die States eines Servers auf der rechten. Der Fall des simultanen Verbindungsauftbaus ist gestrichelt eingezeichnet und wird im Folgenden nicht weiter betrachtet.

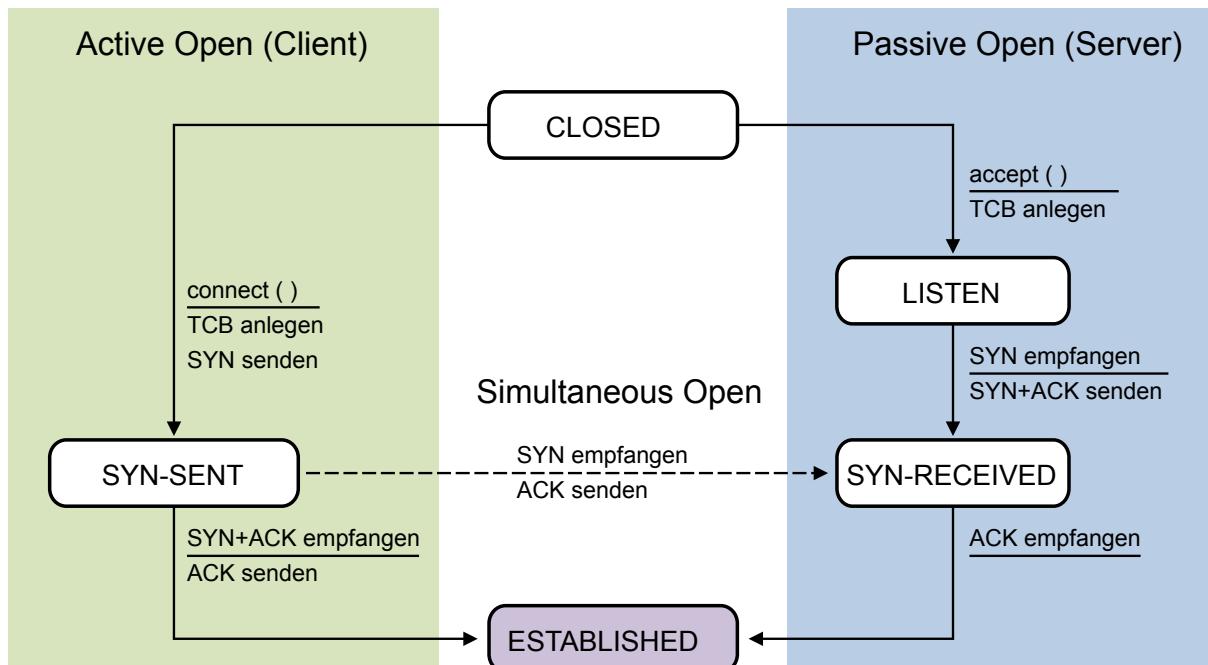


Abbildung 8.14: States des TCP Verbindungsauftbaus

Um eine Verbindung zu erzeugen, wird ein Dreiweg-Handshake benutzt. Server und Client legen vor dem eigentlichen Verbindungsauftbau eine TCP-spezifische Datenstruktur an, dem Transmission Control Block (TCB). Dieser enthält alle benötigten Informationen zur Verbindung wie z.B. die verwendeten Port-Nummern, Pointer zu Sende-/Empfangs-Buffer und die im Folgenden erläuterten Sequenz- und Acknowledge-Nummer.

Um eine eindeutige Verbindungsidentifikation zu bekommen, generieren beide Kommunikationspartner eine zufällig gewählte Sequenznummer (32 Bit). Da jede neue Verbindung so zwei neue Sequenznummern erhält, können zwei Anwendungsprogramme über TCP kommunizieren, die Verbindung beenden und später eine neue aufbauen, ohne dass Duplikate oder verzögerte Pakete nachteilig darauf einwirken.

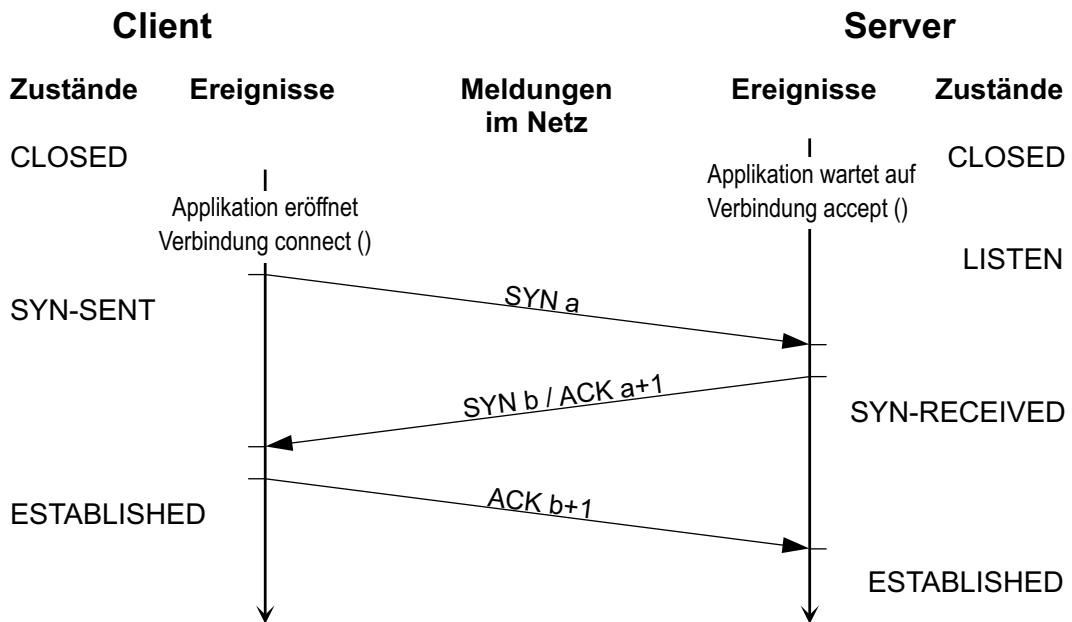


Abbildung 8.15: Dreiweg-Handshake zum Aufbau einer Verbindung

Abbildung 8.15 zeigt detailliert, wie ein TCP-Verbindungsauftbau erfolgt:

- Der Client sendet ein SYN-Segment, das die Client Initial Sequence Number (im obigen Beispiel mit a bezeichnet) enthält.
- Der Server antwortet mit einem SYN-Segment, das die Server Initial Sequence Number (b) enthält. Gleichzeitig bestätigt er die Verbindung, indem er die Initial Sequence Number des Clients (um eins inkrementiert = $a+1$) im Acknowledge-Feld zurückgibt.
- Der Client muss die Verbindung nun seinerseits bestätigen, indem er die Initial Sequence Number des Servers (um eins inkrementiert = $b+1$) im Acknowledge-Feld zurücksendet.

Alle folgenden (Daten-)Segmente verwenden als Offset der Sequenz- und Acknowledge-Nummern $a+1$ resp. $b+1$.

b) Verbindungsabbau

Um eine Verbindung abzubauen, schickt eine (beliebige) Seite ein **FIN-Segment**. Nachdem so die erste Seite ihre ausgehende Verbindung geschlossen hat, nennt man sie **half closed**, denn die andere Seite kann immer noch beliebig Daten senden. Die Ressourcen (TCB) dürfen also erst freigegeben werden, nachdem die Verbindung in *beiden* Richtungen geschlossen wurde. Der endgültige Verbindungsabbau erfordert also ein Handshake-Verfahren, das möglichst sicherstellt, dass TCP erst nach gegenseitiger Zustimmung beider Kommunikationspartner die Verbindung unterbricht und die Ressourcen freigibt.

Abbildung 8.16 zeigt den Verbindungsabbau. Die gestrichelten Verbindungen und der State CLOSING ist nur bei einem simultanen Verbindungsabbau notwendig, der im Folgenden nicht näher betrachtet wird.

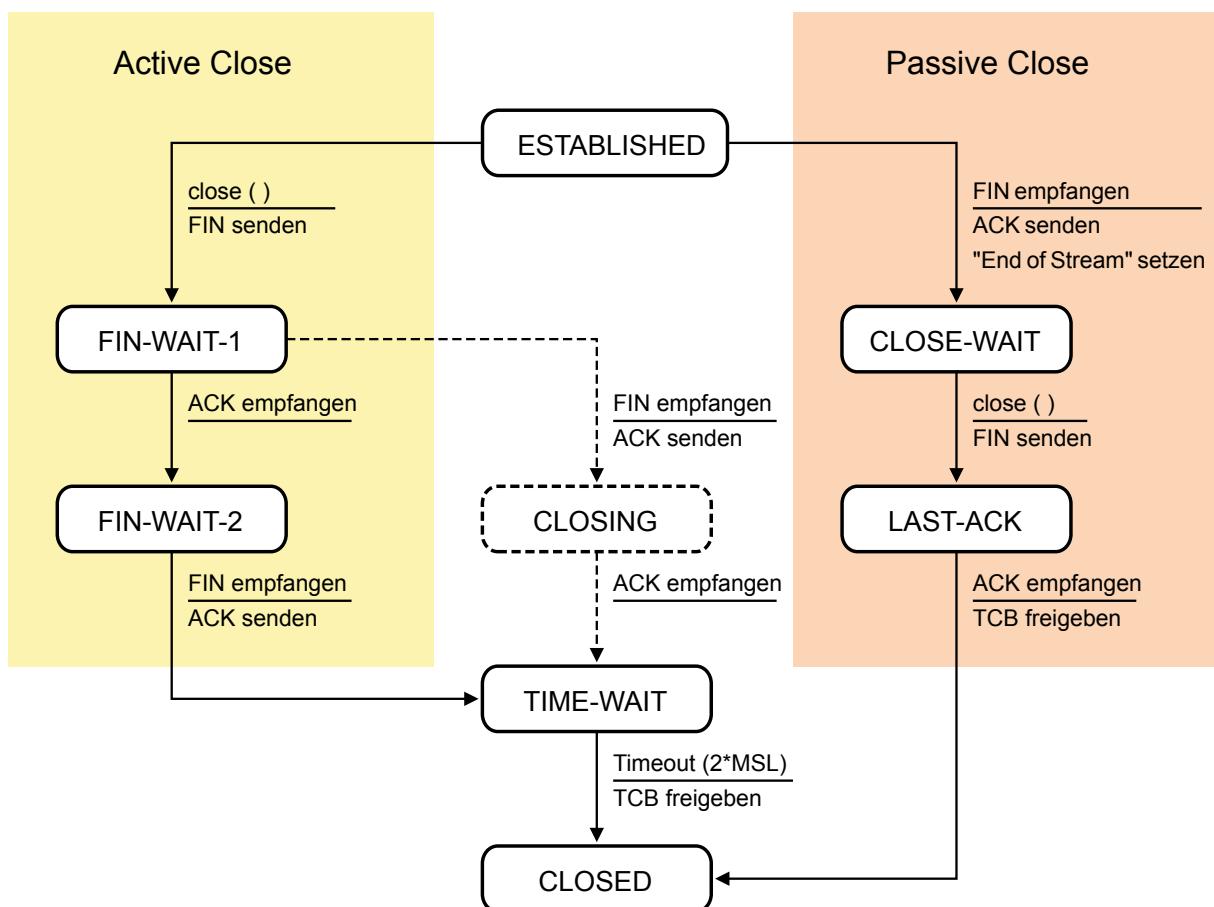


Abbildung 8.16: State Diagram des TCP Verbindungsabbaus

Abbildung 8.17 zeigt die zur Beendigung einer Verbindung benutzten Segmente. Die FIN-Segmente in beiden Richtungen werden mit ACK-Segmenten bestätigt, um sicherzustellen, dass diese angekommen sind, wobei die mittleren zwei Segmente auch zusammengefasst werden können. Da die aktive Seite nicht feststellen kann, ob das letzte ACK-Segment wirklich ankam, wird nach dessen Versenden im Zustand TIME-WAIT gewartet, bevor die Verbindung abgebaut wird. Bei Verlusten wiederholt die passive Seite solange das FIN-Segment, bis eine Bestätigung

(ACK-Segment) eintrifft. Die Wartezeit beträgt 2-mal die **Maximum Segment Lifetime (MSL)**, was typischerweise 4 Minuten entspricht.

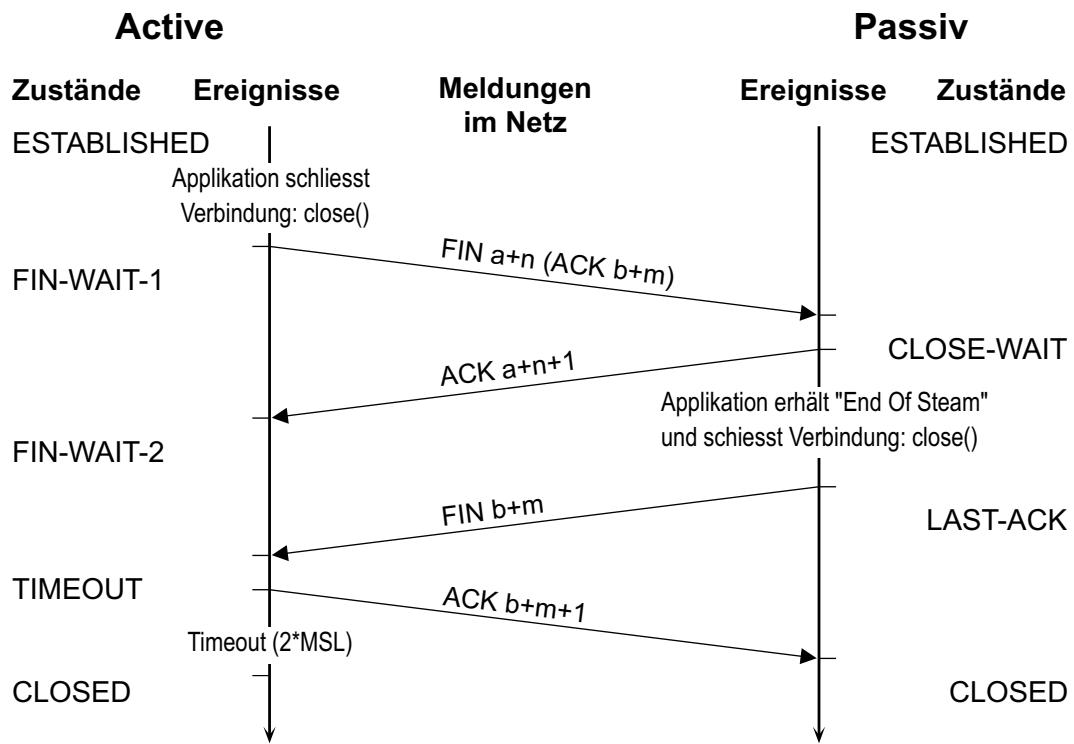


Abbildung 8.17: Meldungsverkehr zum Beenden einer Verbindung

8.3.7 Überlastüberwachung

Einer der interessantesten Aspekte von TCP ist ein Mechanismus zur Überlastüberwachung (**Congestion Control**). In den meisten modernen Netzen wird Paketverlust (oder eine extrem lange Verzögerung) viel eher durch Überlastung als durch einen Hardware-Fehler verursacht. Interessant ist in diesem Zusammenhang, dass Transportprotokolle das Überlastungsproblem noch verschärfen können, weil sie zusätzliche Kopien einer Nachricht einschleusen. Wird durch Überlastung ein übermäßig hohes Volumen an Neuübertragungen ausgelöst, kann das gesamte System einen Kollaps erleiden, der mit einem Totalstau auf der Autobahn vergleichbar ist. Um dies zu vermeiden, benutzt TCP immer den Paketverlust als Masseinheit für Überlastung und reagiert durch Absenken der Rate, in der es Daten erneut überträgt. Dieses Verfahren, das alle TCP-Implementierungen unterstützen müssen, heisst **Slow Start**.

Jeder Sender pflegt hierfür *zwei* Fenster: eines, das der Empfänger gewährt hat, und ein zweites, das sogenannte Überlastungsfenster (**Congestion Window**). Jedes spiegelt die Anzahl von Bytes wider, die der Sender übertragen kann. Das *Minimum* der beiden Fenster stellt die Anzahl von Bytes dar, die gesendet werden können. Somit ist das effektive Fenster das Minimum dessen, was der Sender für angemessen hält, und das, was der Empfänger für angemessen hält. Sieht der Empfänger diese Zahl bei 8 Kbyte, während der Sender weiss, dass bei mehr als 4 Kbyte das Netz verstopft, sendet er nur 4 Kbyte. Obwohl der Algorithmus Slow Start heisst, startet er überhaupt nicht langsam sondern verwendet eine exponentielle Charakteristik.

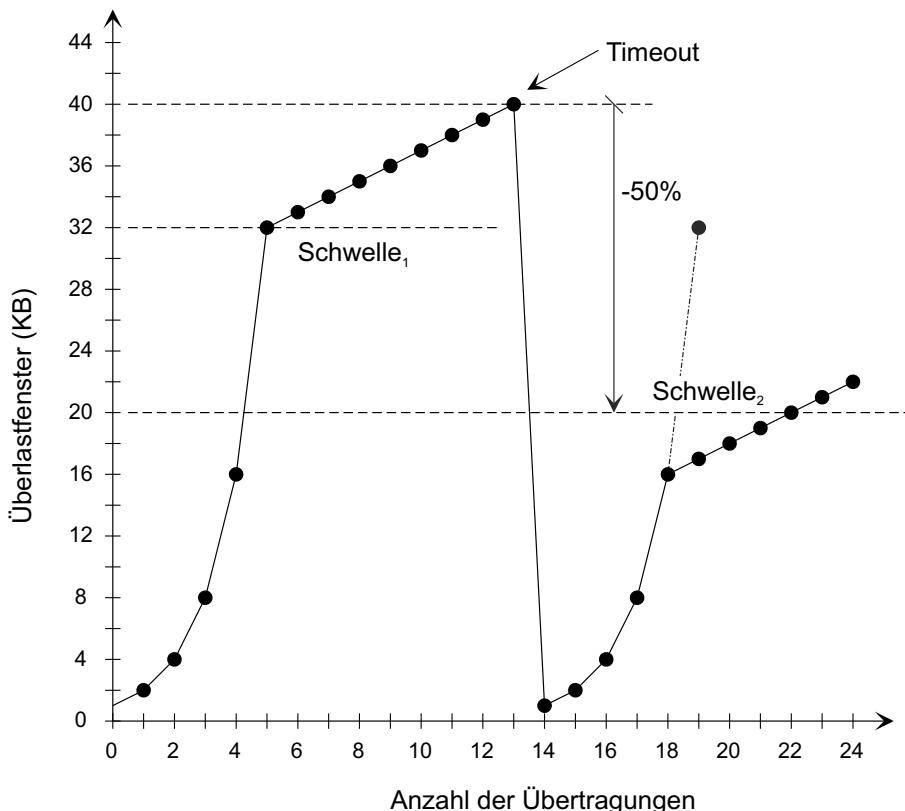


Abbildung 8.18: Beispiel zu Slow-Start-Algorithmus

Die Abbildung 8.18 stellt dar, wie der Überlastungsalgorithmus funktioniert. Die maximale Segmentgrösse ist hier 1024 Byte. Anfangs (nicht dargestellt) umfasst das Überlastungsfenster 64 Kbyte, dann findet ein Timeout statt, so dass der Schwellenwert auf 32 Kbyte gesetzt wird. Das Überlastungsfenster steht hier auf 1 Kbyte für die Übertragung 0. Dann nimmt das Überlastungsfenster exponentiell zu, bis es den Schwellenwert (32 Kbyte) erreicht. Ab dann nimmt es linear zu. Übertragung 13 hat kein Glück (sie hätte es wissen müssen), da der Timer abgelaufen ist. Der Schwellenwert wird auf die Hälfte des aktuellen Fensters gesetzt (inzwischen 40 Kbyte, d.h. die Hälfte ist 20 Kbyte), und der Slow Start beginnt von neuem. Kommen die Bestätigungen von Übertragung 14 an, wird das Überlastungsfenster bei den ersten drei jeweils verdoppelt; danach nimmt es wieder linear zu. Treten keine Timeouts mehr auf, nimmt das Überlastungsfenster bis auf die Grösse des Empfängerfensters zu. An diesem Punkt bleibt es konstant, solange keine weiteren Timeouts auftreten und sich die Grösse des Empfängerfensters nicht ändert.

Durch diese Überlastüberwachung reagiert TCP im Allgemeinen gut auf steigendes Verkehrsvolumen in einem Internet. Da es bei Netz-Verstopfungen keine Neuübertragungen durchführt, trägt es nicht wie andere Transportprotokolle zu einer weiteren Verschärfung der Lage bei.

Trotzdem wird der Algorithmus auch kritisiert: Da die Sägezahnkurven verschiedener Hosts dazu neigen sich zu synchronisieren, erhält man eine Schwingung des Lastverhaltens. An der Verbesserung von Mechanismen zur Überlastüberwachung wird darum ständig weitergearbeitet.

8.3.8 Das Format des TCP-Headers

TCP verwendet für alle Segmente, das gleiche Header-Format. Dies gilt auch für Empfangsbestätigungen und den Verbindungsauftbau und -abbau. Die Abbildung 8.19 zeigt den Header gemäss dem TCP-Standard (RFC 793).

Um das Format besser zu verstehen, erinnere man sich, dass eine TCP-Verbindung zwei Datenströme umfasst – einen in jede Richtung. Deshalb beziehen sich einige Felder des Segments auf den Datenstrom, der in Vorwärtsrichtung, und andere auf den, der in Rückwärtsrichtung fliesst. TCP kann in einem einzelnen Segment die Bestätigung für die Eingangsdaten, die Fensteranzeige und Ausgangsdaten übertragen.

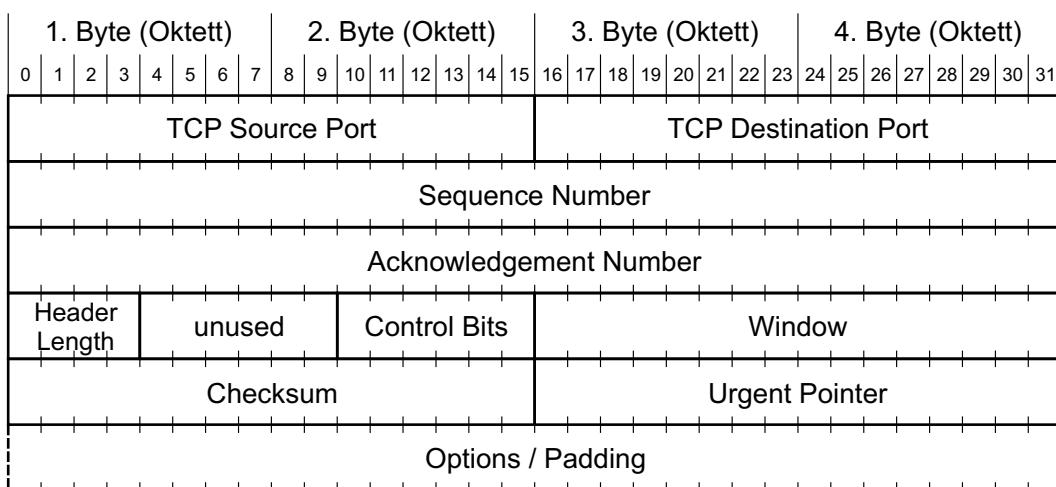


Abbildung 8.19: Felder des TCP-Headers

TCP Source Port und Destination Port definiert, welches Anwendungsprogramm auf dem empfangenden Host die Daten erhalten soll. Das Source Port bezeichnet das sendende Anwendungsprogramm. Falls das Anwendungsprogramm einem Server-Prozess gehört, bezeichnet das Port den entsprechenden Dienst.

Sequence Number bezieht sich auf die Ausgangsdaten. Es gibt die Sequenznummer der im Segment enthaltenen Daten an. Der Empfänger ordnet anhand der Sequenznummer die Segmente, die ausser der Reihe ankommen und benutzt sie zur Berechnung der Acknowledgment-Nummer.

Acknowledgment Number bezieht sich auf die Eingangsdaten. Acknowledgment Number definiert die Sequenznummer der zu empfangenden Daten.

Data Offset gibt an, wo die Daten beginnen und der TCP-Header (inklusive dem optionalen Options/Padding-Feld) zu Ende ist. Der Wert entspricht der Grösse des ganzen TCP-Headers in Doppelwörtern (32-Bit).

Control Bits (siehe Abbildung 8.20) bezeichnet eine Reihe von Flags, die unter anderem den Verbindungsauftbau und -abbau einleiten oder über die Gültigkeit von Header-Feldern Auskunft geben.

10	11	12	13	14	15
URG	ACK	PSH	RST	SYN	FIN

Abbildung 8.20: Bedeutung der Bits im Control-Feld

Die in Abbildung 8.20 verwendeten Abkürzungen haben folgende Bedeutung:

Control Bits	Bedeutung
URG	Urgent-Pointer-Feld enthält gültigen Wert.
ACK	Acknowledgment Feld enthält gültigen Wert.
PSH	Push: Empfänger soll Daten sofort an die Applikation weiterleiten.
RST	Reset: Verbindung rücksetzen.
SYN	Synchronize: Verbindung aufbauen.
FIN	Sender hat keine weiteren Daten zu übertragen.

Window zeigt die noch verfügbare Buffer-Grösse an: Wie viele weitere Daten (Bytes) ist der Empfänger (also der Sender dieses Pakets) momentan bereit zu empfangen.

Checksum Für die Berechnung des Checksum-Felds wird zunächst ein sogenannter Pseudo-Header gebildet, der einige Felder des IP-Headers (IP-Source- und Destination-Address, Protocol) und die Länge des TCP-Datagramms enthält. Die eigentliche Berechnung erfolgt über den Pseudo-Header, den TCP-Header *und* die Daten, indem das Einerkomplement der Summe aller Einerkomplemente aller 16-Bit-Wörter gebildet wird.

Urgent Pointer Falls das URG-Bits gesetzt ist, gibt das Feld an, wo in den Daten Informationen mit hoher Priorität enthalten sind (zum Beispiel Unterbrechungszeichen in Telnet-Sessions). Auf Urgent Pointer und das zugehörige Flag wird hier nicht eingegangen.

Options wird oft beim Verbindungsauftbau verwendet. Wie Abbildung 8.21 zeigt, kann ein Kommunikationspartner seiner Gegenseite beispielsweise seine maximale Segmentlänge (MSS), einen Multiplikationsfaktor für die Window-Size oder andere TCP-Optionen, hier Selective Acknowledge (SACK), mitteilen. Auf die innere Struktur des Options-Felds wird nicht eingegangen.

```
④ Ethernet II, Src: Fujitsu_c1:01:d6 (00:17:42:c1:01:d6), Dst: Broadband_4d:48:80 (00:24:c8:4d:48:80)
④ Internet Protocol, Src: 192.168.1.38 (192.168.1.38), Dst: 74.125.232.122 (74.125.232.122)
④ Transmission Control Protocol, Src Port: 58726 (58726), Dst Port: http (80), Seq: 3397135083, Len: 0
    Source port: 58726 (58726)
    Destination port: http (80)
    [stream index: 12]
    Sequence number: 3397135083
    Header length: 32 bytes
    Flags: 0x02 (SYN)
    Window size: 8192
    Checksum: 0x7ff9
    Options: (12 bytes)
        Maximum segment size: 1260 bytes
        NOP
        window scale: 2 (multiply by 4)
        NOP
        NOP
        TCP SACK Permitted Option: True
```

Abbildung 8.21: Beispiel eines SYN-Segments mit Options-Feld

8.4 Übungen

a) Man beantworte die folgenden Fragen zum TCP-Protokoll:

- Wozu dient die Port-Nummer im Header?
.....
- Nach welcher Zeit wird ein (nicht bestätigtes) Segment nochmals gesendet?
.....
- Bei welchen Übertragungsstrecken (2 Eigenschaften) ist die Sliding Window besonders wichtig?
.....
- Welches Verfahren wird (Warum?) für den Verbindungsabbau verwendet?
.....

b) Auf einer Übertragungsstrecke gehen 5% der Pakete verloren.

- Was erwarten Sie für eine Auswirkung auf den Durchsatz einer UDP-basierten Applikation?
.....
- Was erwarten Sie für eine Auswirkung auf den Durchsatz einer TCP-Verbindung (Begründung oder Namen des relevanten Verfahrens)?
.....

9

Netzwerk-Applikationen und Protokolle

9.1 Das Domain Name System

Traditionell erfolgt die Übersetzung von symbolischen Namen in numerische IP-Adressen mit Hilfe einer Hosts-Datei, in der auf jeder Zeile eine IP-Adresse und die zugehörigen Namen, sowie Abkürzungen stehen. Diese Datei heisst auf Unix-Systemen /etc/hosts, bei Windows (32 und 64 Bit) C:\windows\system32\drivers\etc\hosts oder bei Android /system/etc/hosts.

```
#  
# hosts  
#  
127.0.0.1      localhost  
160.85.17.1    gw.zhaw.ch      gateway  
160.85.17.2    angela.zhaw.ch  angela  
160.85.17.3    brenda.zhaw.ch  brenda  
160.85.17.4    carmen.zhaw.ch  carmen
```

Dieses File muss auf *allen* Knoten vorhanden sein, wo die Namen benötigt werden. Nach jeder Änderung muss es neu verteilt werden. Während dies für kleine Netze einfach und handlich ist, wird das Verfahren für grosse Netze (bezogen auf die Anzahl der Knoten) unbrauchbar.

DNS-Server (DNS) steht für **Domain Name System**) ermöglichen Socket-Anwendungen, wie z.B. FTP oder Web-Browser, Host-Namen einfach auszuwerten, indem sie eine Abfrage an einen zentralen Name-Server senden.

Die Anwendung merkt aber nicht, ob ein DNS-Server oder ein hosts-File verwendet wird. Es wird in beiden Fällen einfach die Funktion `gethostbyname` aufgerufen.

9.1.1 Überblick über das Domain Name System

Das Domain Name System, kurz **DNS** genannt, wurde bereits 1983 als Standard eingeführt (aktueller Stand siehe RFC 1034/1035). Das DNS stellt eine hierarchische Verzeichnisstruktur dar, wie sie von fast allen Betriebssystemen benutzt wird. Dazu werden Host-Namen in einer verteilten hierarchischen Datenbank indiziert, die auch dezentral verwaltet werden kann.

a) Der Domain Name Space

Die hierarchische Datenbank des DNS wird **Domain Name Space** genannt. Die Abbildung 9.1 zeigt eine einfache DNS-Hierarchie, die eine Organisation abbildet. Der Wurzelknoten eines DNS-Baums wird entweder **Wurzel (Root)** oder **Hauptdomäne (Root Domain)** genannt. Die Hauptdomäne wird auch häufig mit einem Punkt '.' bezeichnet.

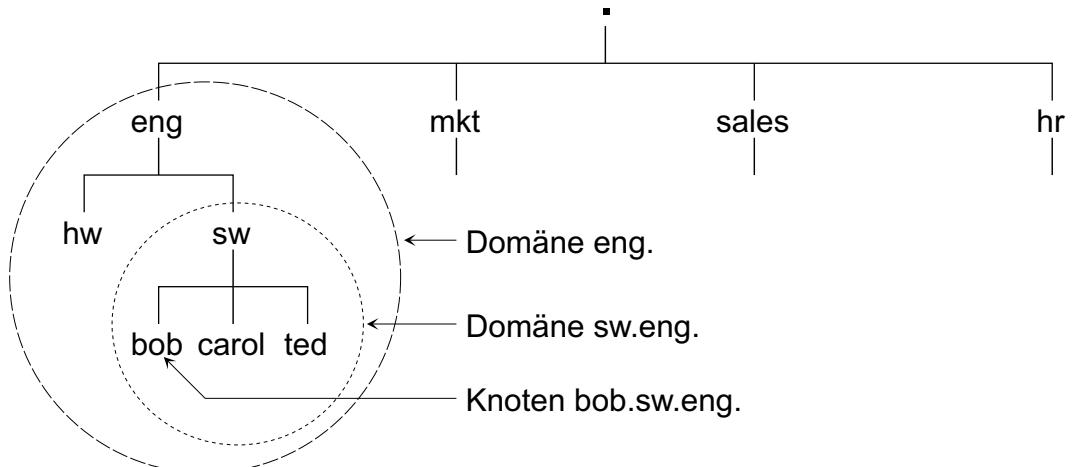


Abbildung 9.1: Beispiel einer einfachen DNS-Hierarchie

Jeder Knoten im Baum trägt einen Namen, der bis zu 63 Zeichen lang sein kann. Jeder Host im Domain Name Space hat einen eindeutigen Namen, der **Fully Qualified Domain Name** genannt wird.

Anders als die Pfadnamen in File-Systemen, die beim Hauptverzeichnis beginnen, beginnen Pfadnamen im DNS mit dem fraglichen Knoten und gehen zum Hauptverzeichnis zurück. In der Abbildung 9.1 wäre z.B. der Pfad `bob.sw.eng.` ein vollqualifizierter Name. Die einzelnen Knoten werden durch Punkte getrennt. Das Wurzelverzeichnis kann durch einen nachgestellten Punkt gekennzeichnet werden (wie oben), der aber normalerweise weggelassen wird.

DNS-Bäume können auch in Begriffen von **Domänen** betrachtet werden, die einfache Teilbäume der gesamten Datenbank sind. Die Abbildung 9.1 veranschaulicht, wie **Subdomänen** von Domänen definiert werden können.

Die Domäne eng hat zwei Subdomänen namens sw.eng und hw.eng. Der Name einer Subdomäne ist einfach der vollqualifizierte Name des obersten Knotens in der Domäne. Subdomänen

bestehen immer aus vollständigen Teilbäumen, d.h. einem Knoten und allen seinen untergeordneten Knoten. Eine Subdomäne kann nicht so festgelegt werden, dass sie die Knoten `eng` und `mkt` enthält, die sich beide auf der gleichen Ebene eines Baums befinden.

Subdomänen sind DNS-Verwaltungsstrukturen. Es ist üblich, die Verwaltung einer Subdomäne zu delegieren, um die Verwaltungsverantwortlichkeit für den kompletten Domain Name Space zu verteilen.

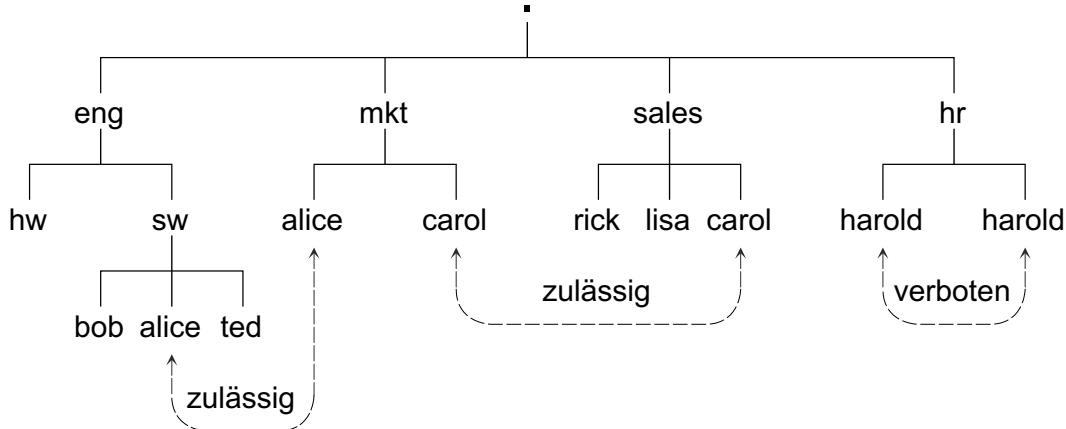


Abbildung 9.2: Benennungsregeln einer DNS-Hierarchie

Für DNS-Bäume gelten die gleichen Benennungsregeln wie für Verzeichnisbäume. Die Abbildung 9.2 zeigt, dass die Namen von Geschwistern eindeutig sein müssen. Untergeordnete Knoten verschiedener übergeordneter Knoten können die gleichen Namen tragen.

Domänennamen können **Alias** zugewiesen werden. Es handelt sich dabei um Zeiger von einer Domäne auf eine andere. Die Domäne, auf die der Alias-Name zeigt, wird auch der **kanonische Domänenname** genannt.

Domänen und Subdomänen sind relative Begriffe und werden austauschbar verwendet. Technisch gesprochen ist buchstäblich jede Domäne außer der Wurzel eine Subdomäne. Wenn ein bestimmter Knoten diskutiert wird, wird dieser im Allgemeinen jedoch als Domäne bezeichnet. Der Gebrauch der Begriffe „Domäne“ und „Subdomäne“ ist hauptsächlich eine Frage der Perspektive.

DNS-Domänen werden normalerweise mit unterschiedlichen Ebenen bezeichnet:

- Domänen auf oberster Ebene: Domänen, die direkt dem Wurzelknoten untergeordnet sind. Häufig wird auch der Begriff der **Top Level Domain (TLD)** gebraucht.
- Domänen auf der zweiten Ebene: Domänen, die den Domänen auf oberster Ebene untergeordnet sind. Sie werden eben als **Second Level Domain** bezeichnet.
- Domänen auf der dritten Ebene: Domänen, die den Domänen auf zweiter Ebene untergeordnet sind, etc.

Hinweis: Der Begriff „Domänen“ im Zusammenhang mit DNS hat nichts mit Windows-Server-Domänen zu tun. Letztere bieten eine Möglichkeit, Windows-Computer in Gruppen zu organisieren, die eine allgemeine Sicherheitsdatenbank teilen. DNS-Domänen werden nur durch den

Internet-Benennungsdienst miteinander verbunden. Ein Windows-Computer kann unter einem bestimmten Namen in einer Windows-Domäne angemeldet sein und unter einem anderen Namen in einer DNS-Domäne.

b) Die Verwaltung von Domänen

Wie erwähnt, wurde das DNS entwickelt, um das Internet zu verwalten, das zu umfangreich ist, um zentral als einzelner Name Space verwaltet zu werden. Deshalb war es unbedingt erforderlich, die Verwaltung von Subdomänen delegieren zu können.

Ein Name-Server ist ein Programm (**Daemon**, unter Unix `named`), das Daten über den Domain Name Space speichert und Informationen zu DNS-Abfragen liefert. Der gesamte Name Space kann in Zonen aufgeteilt werden, bei denen es sich um Untermengen des DNS-Baums handelt. Ein Name-Server ist (üblicherweise) für eine oder (selten) für mehrere Zonen verantwortlich. Die Abbildung 9.3 zeigt einen Beispielbaum, der in drei Zonen organisiert ist.

Im obigen Beispiel wird die Domain `eng` in einer separaten Zone auf einem eigenen Name-Server unterhalten. Zonen sind aber nicht an solche „natürliche“ Grenzlinien gebunden. Anders als Domänen, müssen sie nicht ein Teilbaum des DNS-Baums sein, sondern können verschiedene Ebenen unterschiedlicher Zweige enthalten.

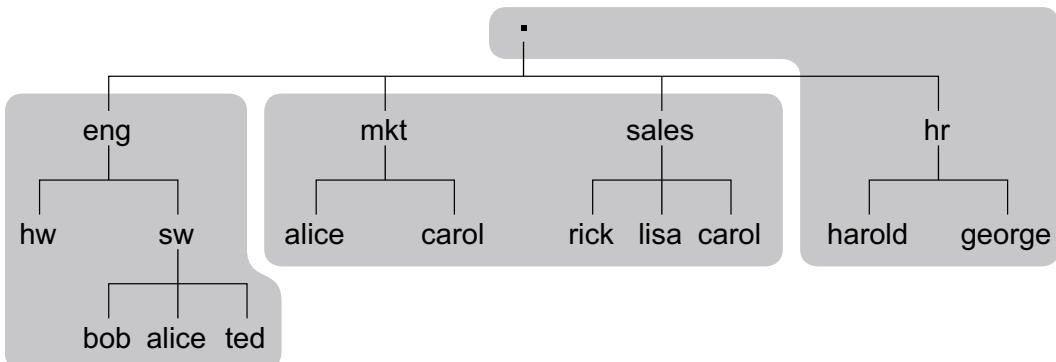


Abbildung 9.3: Zonen und die Delegation von Zuständigkeiten

Falls die Verwaltung für eine Domäne an einen Name-Server delegiert wird, wird dieser automatisch auch verantwortlich für die Subdomänen.

Jede Zone muss von einem **Master-Name-Server** bedient werden, der die Daten für die Zone aus Dateien erhält, die sich auf seinem Rechner befinden.

Slave-Name-Server erhalten Zonendaten über Transfers vom Master-Name-Server. Sie müssen ihre Datenbank in regelmäßigen Abständen aktualisieren, um die verschiedenen Name-Server der Zone synchronisiert zu halten.

DNS ist flexibel in der Art, wie Name-Server und Zonen verbunden werden können. Ein Name-Server kann für mehr als eine Zone verantwortlich sein. Darüber hinaus kann ein Name-Server aber auch in einer Zone der Master-Name-Server und in anderen Zonen der Slave-Name-Server sein.

Die Vorkehrungen für mehrere Name-Server bieten eine Ebene der **Redundanz**, durch die der DNS eines Netzwerks auch dann funktionieren kann, wenn der primäre Name-Server ausfällt.

9.1.2 DNS-Abfragen auswerten

Wenn eine Anwendung das DNS abfragt, benutzt sie dazu einen **Resolver**. Ein Resolver befindet sich auf der Client-Seite einer DNS-Client-Server-Verbindung. Er erzeugt eine DNS-Anfrage und sendet sie an einen Name-Server, verarbeitet die Antwort vom Name-Server und leitet die Information an das Programm weiter, von dem die Daten angefordert wurden.

Resolver-Anfragen werden von DNS-Servern ausgeführt (siehe Abbildung 9.4). Der Resolver in einem Host wird mit der IP-Adresse von mindestens einem DNS-Server konfiguriert. Wenn der Resolver eine IP-Adresse benötigt, nimmt er Kontakt zu einem bekannten DNS-Server auf.

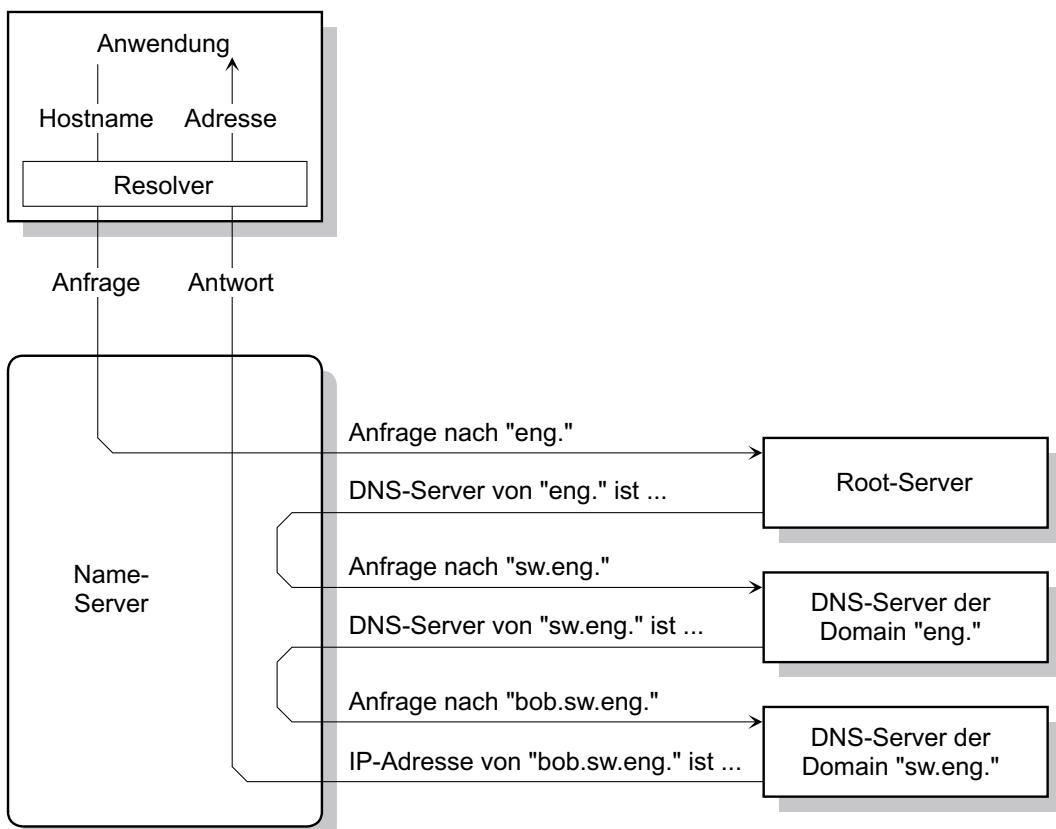


Abbildung 9.4: Auswertung einer DNS-Abfrage

Die Auswertung erfolgt, indem ausgehend vom Root-Server das Hauptverzeichnis abgefragt wird. Der Root-Server liefert die Adressen der Domain-Name-Server der ersten Ebene des abgefragten Namens.

Falls erforderlich, liefert die erste Ebene die Adresse der Domain-Name-Server der zweiten Ebene etc., bis ein Domain-Name-Server erreicht wird, der die Anfrage befriedigen kann.

Die bekannteste Implementation von DNS ist die **Berkeley Internet Name Domain** (BIND), die ursprünglich für die Version 4.3 von BSD UNIX geschrieben wurde und nun in der Version 4.8.3 verfügbar ist. BIND wurde auf die meisten UNIX- und Windows- Versionen portiert. BIND unterstützt Baumstrukturen mit bis zu 127 Ebenen. Das reicht aus, um BIND auf den Root-Name-Servern für das Internet einzusetzen.

BIND benutzt so genannte **Stub-Resolver**. Ein Stub-Resolver hat *keine* DNS-Suchkapazität. Er weiss nur, wie er eine Anfrage an einen DNS-Server sendet. Der Name-Server führt die aktuelle Auswertung der Anfrage durch.

Um den Aufwand einer DNS-Anfrage zu verringern, speichert der DNS-Server die Ergebnisse der letzten Anfragen in einem **Name-Cache**. Befinden sich die fraglichen Daten im Cache, kann der Server eine DNS-Anfrage lokal befriedigen oder das Verfahren abkürzen, indem die Suche bei einem DNS-Server begonnen wird, der für eine Domäne einer niedrigeren Ebene verantwortlich ist.

Einträgen in der DNS-Cache-Tabelle wird ein **Time-to-Live-Wert** (TTL, z.B. 1 Woche) zugewiesen, den der Domänen-Administrator konfiguriert. Einträge, deren Alter den TTL-Wert überschreitet, werden gelöscht. Das nächste Mal, wenn ein Resolver eine Anfrage für diese Domäne startet, muss der Server die Daten wieder aus dem Netzwerk abrufen. Je nach „Stabilität“ der Umgebung kann der TTL-Wert zwischen Stunden und Tagen liegen.

Resolver sind eigentlich Komponenten von Anwendungen und Prozesse, die auf Hosts ablaufen. Programme rufen Bibliotheks Routinen für den Name-Dienst auf (wie die bekannte `gethostbyname`). Auf diese Weise können Programme wie FTP und Telnet DNS-Anfragen erzeugen und die Antworten verarbeiten. Es ist jedoch der DNS-Server, der für die Suche in der Datenbank verantwortlich ist.

Sekundäre DNS-Server sind für die Gewährleistung eines zuverlässigen Name-Dienstes von grosser Bedeutung. Die Angabe der Name-Server erfolgt unter Unix im File `/etc/resolv.conf`.

```
#  
# /etc/resolv.conf  
#  
search zhaw.ch  
nameserver 160.85.128.3 # Primärer Name-Server  
nameserver 160.85.128.2 # Sekundärer Name-Server  
nameserver 160.85.128.1 # Sekundärer Name-Server
```

9.1.3 Die Organisation des Domain Name Space im Internet

Die an der University of Southern California (ISC) angesiedelte IANA übernahm schon früh die Verwaltung von Adressen und Namen. Später übertrug die amerikanische Regierung die Verwaltung der Domänen für `.com`, `.net` und `.org` der Firma Network Solutions Inc. (NSI). Dieses Unternehmen übernahm als Monopolinhaber die Registrierung der Domain-Namen für Kunden, den so genannten „Registrar“-Betrieb. Daneben hat es den zugehörigen Name-Server, sowie den Root-Server „rs.internic.net“ betrieben, der für die Delegation der Top-Level-Domains verantwortlich war.

Mit dem weltweiten Boom des Internets mussten die Bedürfnisse der Internet-Nutzer aus aller Welt stärker berücksichtigt werden. Ein Monopol im Auftrag der amerikanischen Regierung entsprach nicht den Interessen der internationalen Internet-Gemeinde. Mängel in der technischen Zuverlässigkeit sowie der Preispolitik von NSI führten schliesslich zu einer grundlegenden Neuerung und zur Gründung der Internet Corporation for Assigned Names and Numbers (ICANN).

a) Internet Corporation for Assigned Names and Numbers (ICANN)

Die Aufgabe der „Internet Corporation for Assigned Names and Numbers“ (ICANN) genannten Organisation ist die (Neu-)Regelung der Namensvergabe im Internet. Diese Non-Profit-Organisation soll die Interessen der kommerziellen und nichtkommerziellen Nutzer im Internet wahrnehmen. Privatpersonen ebenso wie Firmen sollen eine faire Chance für die Nutzung von Namen im Internet erhalten. Aber auch für die technische Administration weiterer wichtiger Internet-Protokoll-Parameter, wie IP-Adressen und Port-Nummern sowie für die Root-Server ist ICANN zuständig.

Ein besonderes Ziel von ICANN ist es, vorhandene Internet-Gremien so weit als möglich mit einzubinden (siehe Abbildung 9.5).

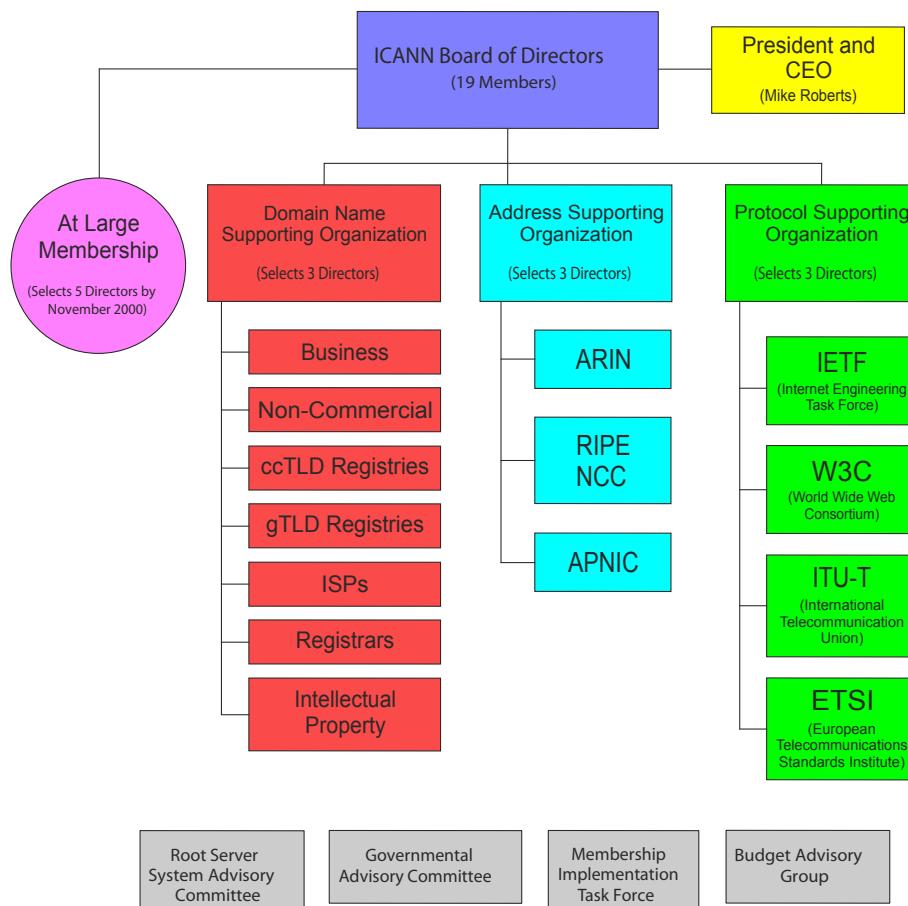


Abbildung 9.5: ICANN Organisation (www.ICANN.org)

Bei der Protocol Supporting Organization (PSO) wurden die Internet Engineering Task Force (IETF), das World Wide Web Consortium (W3C), die International Telecommunication Union (ITU-T) und das European Telecommunications Standards Institute (ETSI) einbezogen.

Die Address Support Organisation (ASO) wurde aus den drei bestehenden regionalen Internet Registries (RIRs), der Réseaux IP Européen (RIPE), der American Registry for Internet Numbers (ARIN) und des Asia Pacific Network Information Center (APNIC) gebildet.

Für den Bereich Domain-Namen wurde eine Domain Name Supporting Organisation (DNSO) mit verschiedenen Mitgliedern gegründet.

b) Organisation der Root-Name-Server

Das Internet ist natürlich der grösste Name Space. Er entwickelt sich viel zu schnell, als dass er zentral verwaltet werden könnte. Um eine effiziente und flexible Verwaltung des Internet Name Space zu ermöglichen, werden alle diskutierten DNS-Funktionen benötigt.

Die Root-Name-Server sind verantwortlich für die oberste Ebene der Domänen (Top Level Domain). Zwar verfügen die DNS-Server der tieferen Ebenen über eine gewisse Unabhängigkeit, da sie zur Entlastung der Root-Server Caches führen. Die Einträge in den Cache-Tabellen müssen aber nach einer definierten Zeit erneuert werden. Falls der Eintrag im Cache fehlt (oder veraltet ist) und kein Root-Name-Server verfügbar ist, schlägt die Namensauflösung fehl.

Die Wichtigkeit der Root-Name-Server sowie des Volumens an DNS-Abfragen im Internet erzwingen den Einsatz von mehreren Root-Name-Servers. Weltweit sind fast 400 Root-Server installiert, die von einem Dutzend verschiedener Organisationen betrieben werden (Abbildung 9.6).



Abbildung 9.6: Positionen und Betreiber der Root-Name-Server (<http://www.root-servers.org/>)

9.1.4 Adressen Namen zuordnen

Wie bereits beschrieben, ist DNS ein Meister darin, Domänennamen in IP-Adressen auszuwerten. Manchmal jedoch ist genau das Gegenteil gefordert: Es muss der Domänenname ermittelt werden, der einer vorgegebenen IP-Adresse zugeordnet ist. Für umgekehrte Zuordnungen steht eine spezielle Domäne im Internet zur Verfügung: die Domäne `in-addr.arpa`.

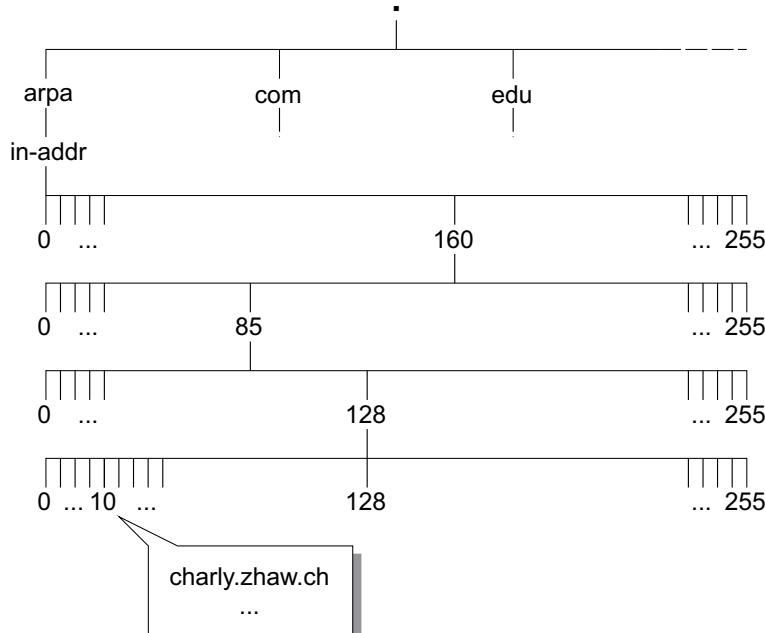


Abbildung 9.7: Beispiel zur Invers Address Domain: 10.128.85.160.in-addr.arpa

Die Abbildung 9.7 veranschaulicht die Struktur von `in-addr.arpa`. Die Knoten in der Domäne werden nach IP-Adressen benannt. Die Domäne `in-addr.arpa` kann 256 Subdomänen (0..255) haben, die dem ersten Byte der IP-Adressen entsprechen.

Jede Subdomäne von `in-addr.arpa` kann ihrerseits 256 Subdomänen haben, die mit den möglichen Werten des zweiten Bytes der IP-Adressen übereinstimmen.

In gleicher Weise kann die nächste Subdomäne in der Hierarchie 256 Subdomänen haben, die dem dritten Byte der IP-Adressen entsprechen. Schliesslich enthält die letzte Subdomäne Knoten, die dem vierten Byte von IP-Adressen entsprechen.

Die Abbildung 9.7 zeigt, wie ein Datensatz in der IN-ADDR.ARPA-Hierarchie gespeichert werden könnte. Der Rechnername `charly.zshaw.ch` ist mit der IP-Adresse 160.85.128.10 verknüpft. Um den Namen zu ermitteln, durchsucht DNS den Baum und beginnt mit dem Knoten `160.in-addr.arpa`.

Die Suche wird so lange fortgeführt, bis der Datensatz `10.128.85.160.in-addr.arpa` erreicht ist. Der Wert dieses Datensatzes enthält den Rechnernamen `charly.zshaw.ch`.

9.2 Bootstrap Protocol (BootP)

In einem IP-Netzwerk muss bekanntlich jeder Knoten über eine einmalige IP-Adresse verfügen.

Knoten ohne Laufwerke (Diskless Stations, wie Workstation, Printer, managed Hubs etc.), die nur mit einem Boot-ROM ausgerüstet sind, können Adressinformationen und die dazu gehörenden Programme nicht lokal ablegen. Mit Hilfe eines kleinen Startprogramms, das im ROM abgelegt ist, wird das Gerät darum über das Netz initialisiert und die benötigte Software geladen. Heute werden in solchen Geräten zwar häufig Flash-Speicher eingesetzt, die es erlauben, Betriebsparameter (IP-Adressen etc.) und Programme lokal zu speichern. Aus Management-Gründen ist es aber oft zweckmässiger, IP-Adressen sowie Programme über das Netz zu laden.

Zur Regelung des Ladevorgangs dienen zwei Protokolle: das **Bootstrap Protocol (BootP)** und das **Trivial File Transfer Protocol (TFTP)**, das im Abschnitt 9.3 beschrieben wird.

9.2.1 Funktionsprinzip des BootP

Ein allgemeingültiges Startup-Programm für eine Diskless-Station darf keine definierten IP-Adressen enthalten, da jeder Rechner in einem IP-Netzwerk eine IP-Adresse haben muss. Die IP-Adresse wird von einem BootP-Server dynamisch auf die Diskless-Station geladen. BootP ist im RFC 951 beschrieben. Ergänzend zum BootP-Standard enthält RFC 1084 herstellerspezifische Erweiterungen.

Ein BootP-Paket enthält sehr detaillierte Informationen (siehe Abbildung 9.8). Neben der IP-Adresse, der IP-Adresse eines Gateways und der Adresse des File Servers kann auch der Name der zu bootenden Datei mitgeteilt werden.

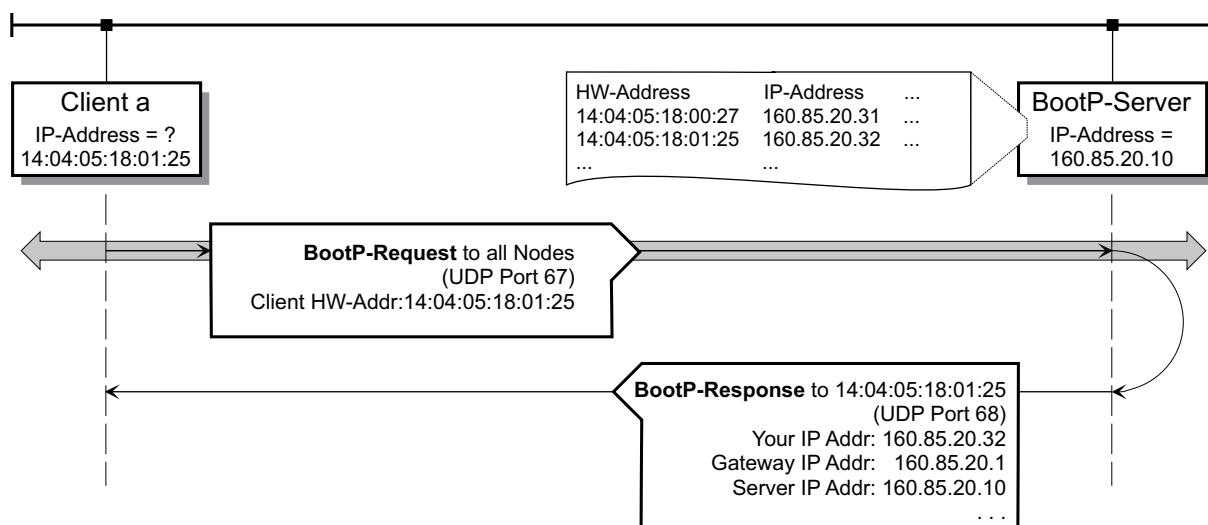


Abbildung 9.8: Ablauf BootP-Request/Reply

In einem optionalen Feld (herstellerspezifische Erweiterungen) können hersteller- oder netzspezifische Daten (Name Server, Subnetzmasken, Datum, Zeit etc.) übermittelt werden.

Wie in Abbildung 9.8 gezeigt, arbeitet BootP nach dem Client-/Server-Prinzip und tauscht nur ein einziges Paket aus. BootP setzt als Anwendung unmittelbar auf dem User Datagram Protocol (UDP) auf. Es verwendet UDP Port 67 zur Kommunikation mit dem BootP Server und UDP Port 68 zur Kommunikation mit dem BootP Client.

Dadurch, dass der Server seine Antwort an einen festen Port sendet, können diese mehrere Clients gleichzeitig empfangen – falls die Antwort als Broadcast verschickt wird. Dies macht natürlich nur dann Sinn, wenn Client-unabhängige Informationen verschickt werden.

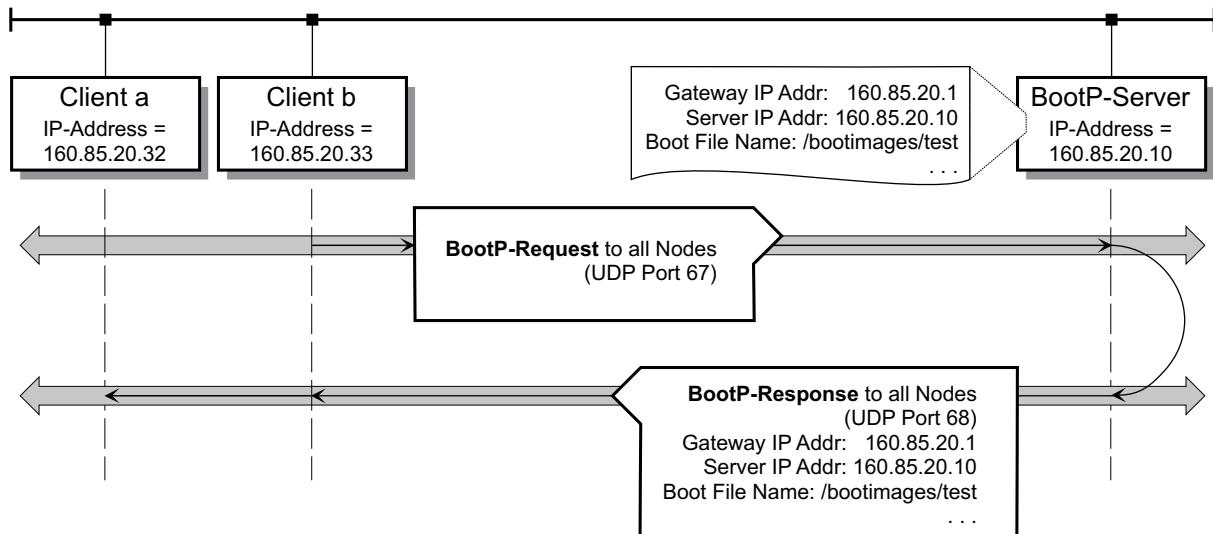


Abbildung 9.9: Ablauf BootP-Request mit Broadcast-Reply

Im Gegensatz zum RARP-Protokoll, dessen Einsatz auf ein LAN beschränkt ist, kann beim BootP-Protokoll der Server auch in einem anderen IP-Subnetz liegen.

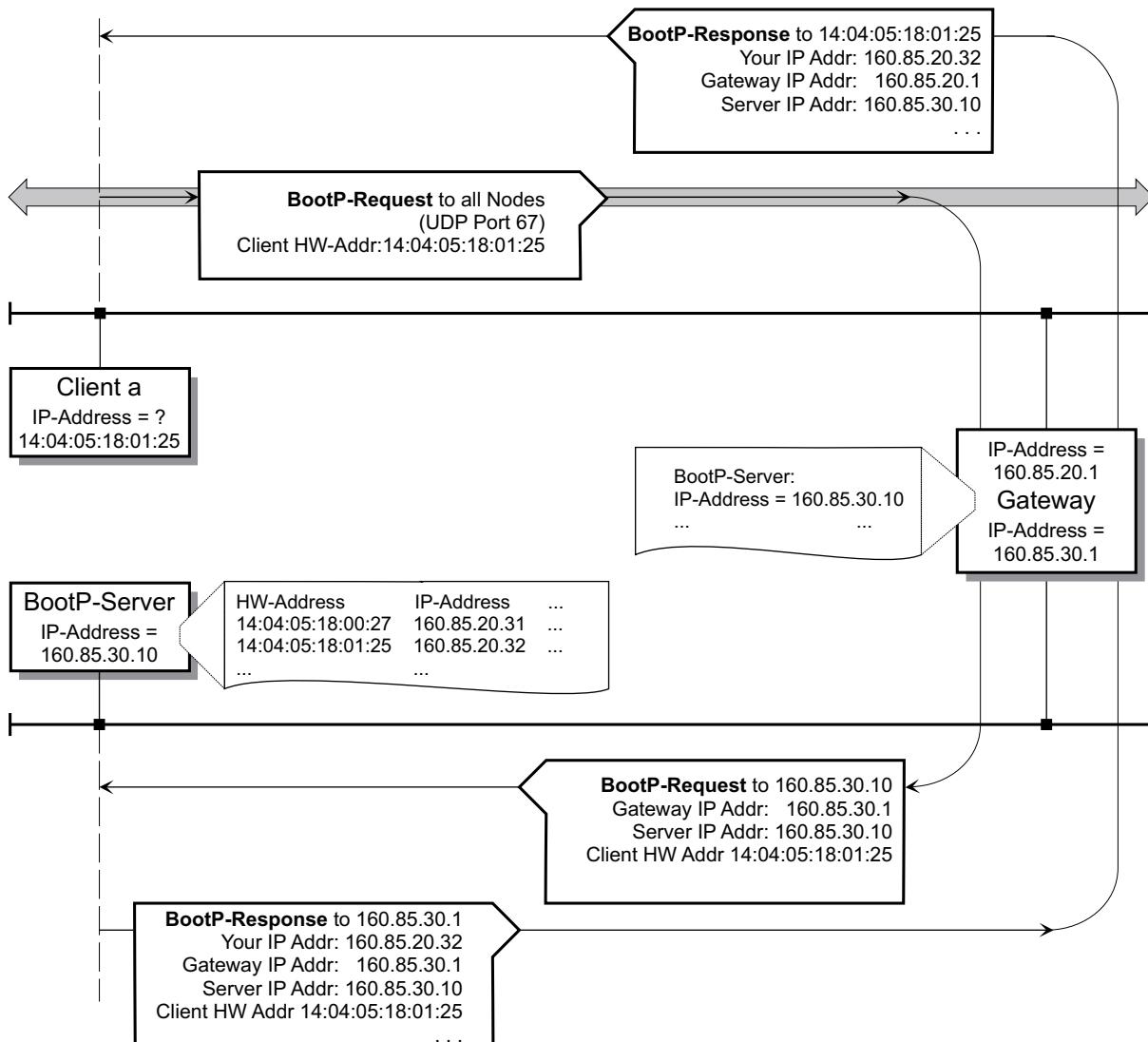


Abbildung 9.10: BootP-Protokoll über Gateway

Der Gateway muss speziell konfiguriert sein. Er muss die IP-Adresse des BootP-Servers kennen und er muss bereit sein, BootP-Requests (Broadcast) am Port 67 entgegenzunehmen.

Im Folgenden setzt er seine IP-Adresse im Gateway-Feld ein und leitet den BootP-Request – nun als Unicast – an den BootP-Server weiter.

Dieser sendet die Antwort an den Gateway zurück, der sie dann anhand der Hardware-Adresse an den BootP-Client sendet.

9.2.2 BootP-Paketformat und Bedeutung der Felder

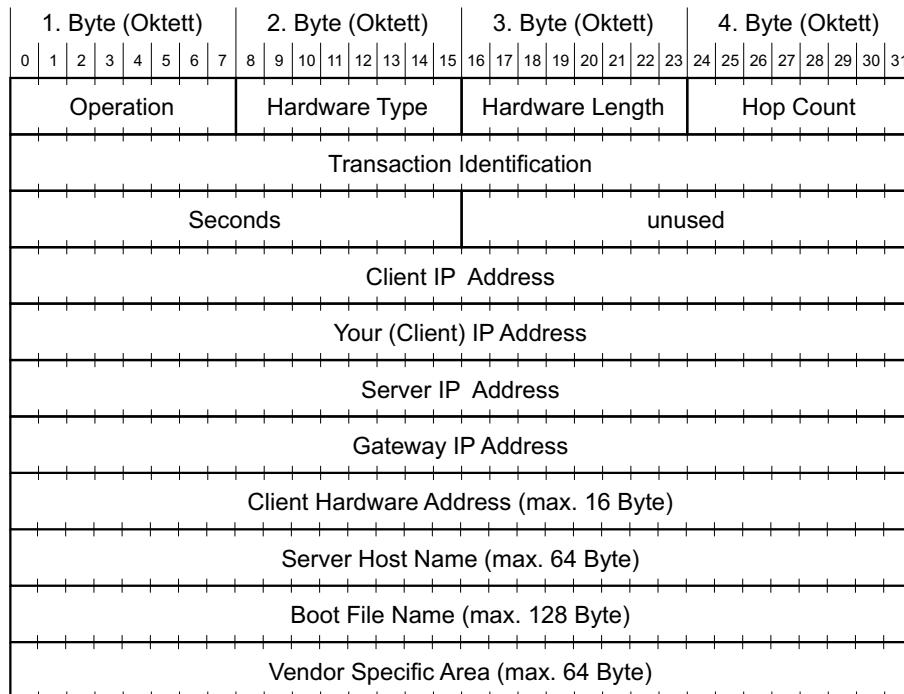


Abbildung 9.11: Aufbau eines BootP-Pakets

Operation gibt an, ob das Paket einen BootP-Request (1) oder einen BootP-Reply (2) enthält.

Hardware Type definiert die verwendete Technologie, die Geschwindigkeit und die Datenstrukturen. Durch Festlegungen in diesem Feld kann BootP auf unterschiedlichen Netzen eingesetzt werden:

Netztyp	Beschreibung
1	Ethernet (10 MB)
2	Experiment Ethernet (3 MB)
3	Amateur Radio AX.25
4	Proteon Token Ring
5	Chaos-Netz
6	IEEE 802-Netzwerke
7	ARCNET

Hardware Length gibt die Länge der Hardware-Adresse (in Byte) an. Bei Ethernet-Netzwerken beträgt der Wert immer 6.

Hop Count – Ein BootP Client muss das Feld immer auf den Wert 0 setzen. Ein Router, der einen Request weiterleitet, soll den Hop Count erhöhen oder den Request verwerfen.

Transaction Id enthält einen Integer-Wert, um der Diskless-Station die Zuordnung von Anfragen und Antworten zu ermöglichen.

Seconds verzeichnet die Zeit in Sekunden, die vergangen ist, seitdem der Client den Boot-Vorgang gestartet hat (maximal 60 Sek.).

unused ist ein Leerfeld, das für zukünftige Entwicklungen reserviert ist.

Client IP Address – Falls BootP von einem Client benutzt wird, der seine IP-Adresse bereits kennt, trägt dieser seine Adresse in dieses Feld ein. Clients, die ihre IP-Adresse noch nicht kennen, füllen dieses Feld mit 0-Werten. Empfängt der BootP Server ein Client IP Address 0, beantwortet er diese Anfrage, indem er die IP-Adresse des Clients im Feld „Your IP Address“ einträgt.

Your IP Address ist die Protokolladresse des BootP Clients, die er als Antwort auf einen BootP-Request erhält.

Server IP Address – Ein Client, der die IP-Adresse eines Servers kennt, von dem er Informationen abrufen möchte, trägt sie in dieses Feld ein und stellt dadurch sicher, dass nur der adressierte Server auf die Anfrage antwortet. Enthält dieses Feld den Wert 0, kann jeder BootP Server im Netz antworten.

Gateway IP Address enthält die IP-Adresse des ersten Gateways (Routers) auf dem Weg zum BootP Server.

Client Hardware Address enthält die Hardware-Adresse des BootP Client.

Server Host Name – Ein Client, der den Host-Namen eines Servers kennt, von dem er Informationen abrufen möchte, trägt diese Information in dieses Feld ein und stellt dadurch sicher, dass nur der adressierte Server auf den Request antwortet. Enthält dieses Feld den Wert 0, so kann jeder BootP Server im Netz antworten.

Boot File Name erlaubt einem BootP Client die Spezifikation einer Datei, die er vom BootP Server laden möchte. Der BootP Server kann über eine Datenbank die angeforderte Datei identifizieren und (zum Beispiel per TFTP) an den Client übermitteln.

Vendor Specific Area enthält optional herstellerspezifische Information, die vom Server zum Client übertragen werden kann. Die ersten vier Byte des Feldes werden „Magic Cookie“ genannt und legen das Format der folgenden Daten fest. Anschliessend folgt eine Liste von Bezeichnungen, die jeweils aus dem Typ und einem Wert bestehen.

9.2.3 Dynamic Host Configuration Protocol (DHCP)

Das BootP Protokoll hat zwei Nachteile:

- a) In Umgebungen mit einer grossen Anzahl von Knoten, von denen aber nur wenige gleichzeitig aktiv sind (z.B. Internet Provider), möchte man IP-Adressen gemeinsam nutzen können. Mit BootP ist das nicht praktikabel.
- b) Grosse Netzwerke mit häufig ändernden Knoten und Topologien erzeugen einen erheblichen Verwaltungsaufwand, da für jeden Knoten (resp. jedes Interface) per Hand ein Eintrag in der BootP-Datenbank unterhalten werden muss.

Das DHCP (Dynamic Host Configuration Protocol, RFC 2131) ermöglicht mit Hilfe eines entsprechenden Servers die Zuweisung einer *dynamisch* gewählten IP-Adresse. Der Administrator muss nur einen entsprechenden Adressbereich definieren. Bei einer entsprechenden Anfrage eines Clients sucht der DHCP-Server im zugeordneten Adressbereich eine freie IP-Adresse und vergibt diese mit einer zeitlichen Beschränkung. Läuft diese sogenannte Lease-Time ab, muss sie vom DHCP-Client erneuert werden, sonst wird die Adresse wiederverwertet.

Unter Windows kann mit dem Befehl ipconfig die aktuelle DHCP Situation angezeigt werden (Status, Server, der Zeitpunkt der Vergabe und das Ende der Gültigkeit):

```
ipconfig /all
.
.
.
DHCP Enabled . . . . . : Yes
DHCP Server . . . . . : 10.0.0.11
Lease Obtained . . . . . : Montag, 17. Januar 2099 15:11:58
Lease Expires . . . . . : Montag, 17. Januar 2099 17:11:58
```

Mit ipconfig /renew [adapter] kann die Adresse vorzeitig erneuert werden.

Mit ipconfig /release [adapter] wird eine Adresse vorzeitig freigegeben.

DHCP verwendet das BOOTP-Protokoll und ist weitgehend aufwärtskompatibel. Ein DHCP-Server kann auch BOOTP-Clients bedienen.

Durch DHCP können neben den bekannten IP-Parametern auch viele weitere Konfigurationsparameter vergeben werden. Damit ist die Einbindung eines neuen Computers in ein bestehendes Netzwerk ohne weitere Konfiguration möglich.

9.3 Trivial File Transfer Protocol (TFTP)

TFTP ist neben dem populären File Transfer Protocol (FTP) ein weiteres Protokoll zur Datenübertragung. Das TFTP ist im RFC 1350 beschrieben und wurde durch den RFC 2348 erweitert.

TFTP unterstützt nur wenige Kommandos und verwendet das einfache User Datagram Protocol (UDP), im Gegensatz zu FTP, das auf TCP aufsetzt. Durch den geringen Code-Umfang kann TFTP problemlos auf Microcontroller oder im Boot-Loader implementiert werden.

Da TFTP auf dem ungesicherten UDP aufsetzt, mussten eigene Sicherungsmechanismen implementiert werden, um die fehlerfreie Übertragung der Daten zu gewährleisten.

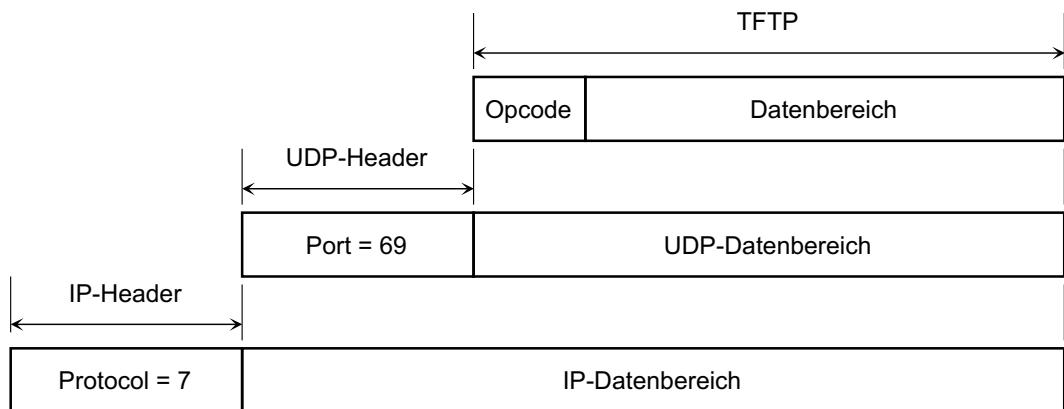


Abbildung 9.12: Aufbau eines TFTP-Pakets

9.3.1 TFTP-Funktionen

Beim TFTP sind nur fünf Funktionen (beziehungsweise Opcodes) definiert:

Read Request (RRQ) fordert eine Datei von einem TFTP-Server an. Dabei wird eine TFTP-Verbindung über das UDP-Port 69 aufgebaut. Optional kann beim Read Request eine Blockgrösse angegeben werden (RFC 1350).

Write Request (WRQ) schickt eine Datei an einen anderen Host. Dabei wird eine TFTP-Verbindung über das UDP-Port 69 aufgebaut. Optional kann beim Write Request eine Blockgrösse angegeben werden (RFC 1350).

Acknowledgement (ACK) bestätigt ein korrekt empfangenes WRQ- oder DATA-Paket. Bei der Bestätigung wird die Blocknummer angegeben.

Data (DATA) übermittelt die eigentlichen Daten. Die Datenübertragung erfolgt in Blöcken von konstanter Grösse. Sofern beim Verbindungsaufbau keine andere Blockgrösse angegeben wurde, beträgt sie 512 Byte. Kürzere Datenfelder beenden die Datenübertragung. Jeder Block wird fortlaufend nummeriert.

Error (ERROR) beschreibt den aufgetretenen Fehler und beendet die Verbindung.

9.3.2 TFTP-Funktion im Überblick

Will ein TFTP-Client ein File senden, so wird zuerst ein WRQ-Paket übertragen, das den Filenamen enthält. Es gibt keine Authentisierung. Der Server bestimmt Ziel-Directory und je nach Server-Setup muss das zu übertragende File dort bereits angelegt sein. Im positiven Fall erlaubt der Server mit einem ACK-Paket die Übertragung und etabliert so die TFTP-Verbindung (siehe Abbildung 9.13).

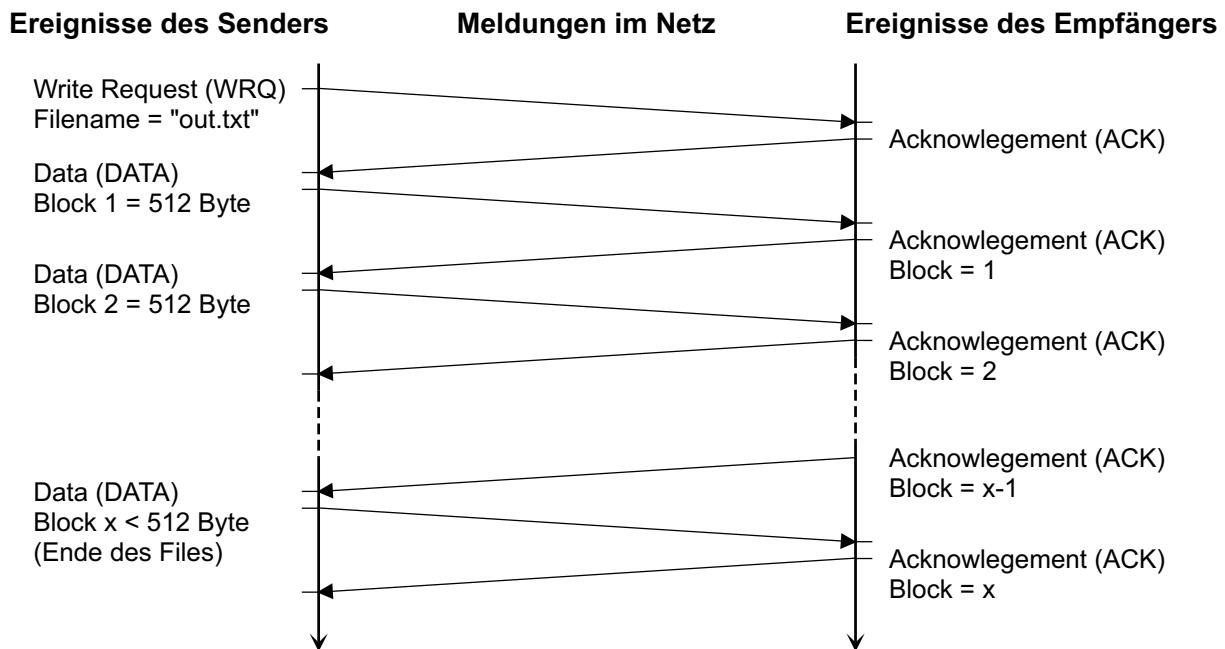


Abbildung 9.13: Senden eines Files mit TFTP

Will ein TFTP-Client ein File empfangen, so wird zuerst ein RRQ-Paket übertragen, das den Filenamen enthält. Es gibt keine Authentisierung. Der Server bestimmt Quellen-Directory, in dem sich das zu übertragende File befinden muss. Im positiven Fall beginnt der TFTP-Server direkt mit dem Senden des ersten DATA-Pakets (siehe Abbildung 9.14).

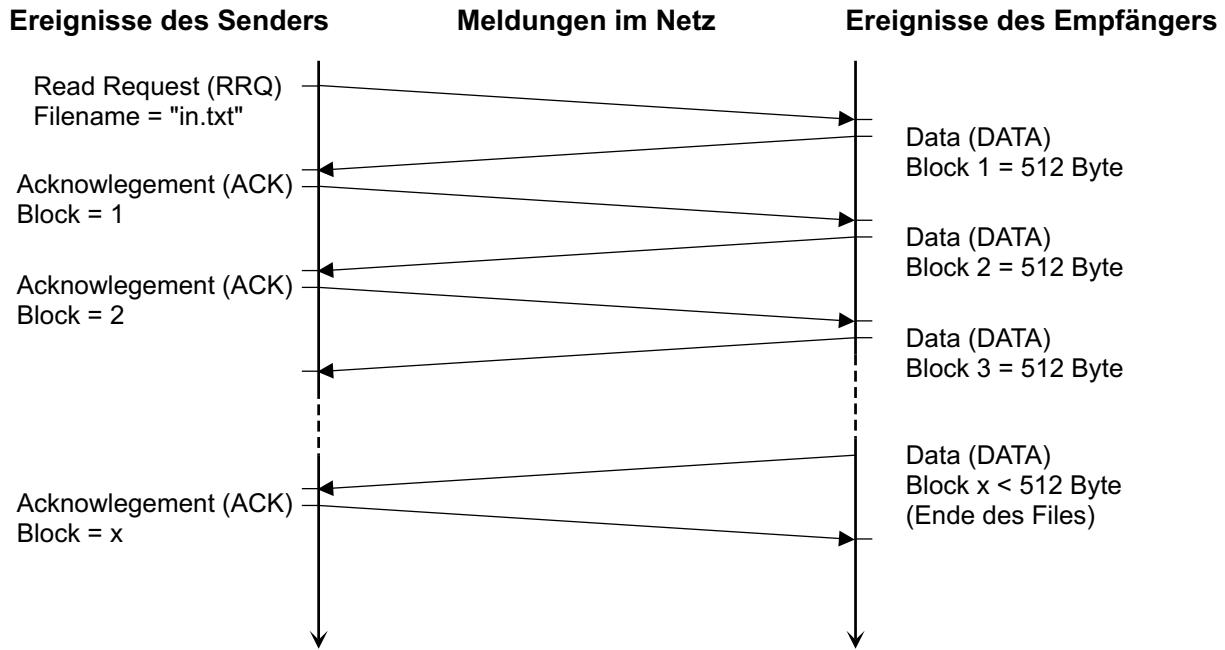


Abbildung 9.14: Empfangen eines Files mit TFTP

Die einzelnen Pakete werden durch DATA übermittelt und fortlaufend nummeriert. Jedes übermittelte Paket muss vom Empfänger mit einem ACK-Paket bestätigt werden, bevor ein weiteres Paket gesendet wird. TFTP ist also ein reines Stop-and-Wait-Protokoll (kein Übertragungs-Window).

Sender und Empfänger haben einen Timeout/Retransmission-Mechanismus implementiert. Wird nach Ablauf des Sender-Timers keine Bestätigung empfangen, so wird das Paket erneut gesendet. Trifft beim Empfänger nicht innerhalb einer bestimmten Zeit ein neues Paket ein, so wird die letzte Bestätigung erneut gesendet.

Das TFTP sendet immer Pakete mit einer festen Grösse (typischerweise 512 Byte). Ein Paket, das weniger Daten enthält, signalisiert das Ende der Datenübertragung. Fehlermeldungen (ERROR-Pakete) können entweder anstelle von Daten oder statt einer Bestätigung versendet werden. Fehlermeldungen führen immer zum Abbruch der Datenübertragung.

9.4 Simple Mail Transfer Protocol (SMTP)

Das Simple Mail Transfer Protocol (SMTP) ist ein Standardprotokoll zum Versenden und Weiterleiten von Electronic Mail (E-Mail).

SMTP ist in den RFC 5321 und RFC 7504 definiert und setzt auf TCP Port 25 auf.

SMTP legt nicht fest, wie eine E-Mail von und zu einem Benutzer bzw. vom und zum SMTP-System vermittelt wird. Wie die empfangene E-Mail zwischengespeichert und wie oft der Benutzer über ihre Existenz informiert wird, bleibt ebenso offen, wie die Präsentationsform der E-Mail auf der Benutzeroberseite. Diese Aufgaben überlässt SMTP den Anwendungsprogrammen.

Auf dem Zielrechner (Mail-Server) werden die Daten in einer Mailbox zwischengespeichert. Beispielsweise wird unter Unix pro Benutzer ein separates File im Directory `/var/spool/mail/` angelegt. Die Mails werden dort gespeichert, bis der Benutzer diese z.B. mit dem Programm `mail` direkt (liest und) löscht. Alternativ können die Protokolle **POP3 (Post Office Protocol, RFC 1939)** oder **IMAP (Internet Message Access Protocol, RFC 2060)** verwendet werden, um von einem entfernten Mail-Client über ein Netzwerk auf den Mailserver zuzugreifen. Mit dem POP3-Protokoll können Mails im Wesentlichen vom Server herunter geladen und gelöscht werden. Das IMAP-Protokoll erlaubt wesentlich weitergehende Funktionen zur Verwaltung der Mail auf dem Server. IMAP erlaubt z.B. das Einrichten von Unterverzeichnissen sowie das Kopieren und Umbenennen von Mails. Auf die Protokolle POP3 und IMAP wird im Folgenden nicht weiter eingegangen.

Die mit SMTP versendeten E-Mails richten sich meist nach dem im RFC 822 definierten Darstellungsformat. SMTP berücksichtigt weder Form noch Inhalt der zu übertragenden E-Mail: Die auf dem Briefumschlag enthaltenen Informationen reichen als Transportinformationen für SMTP aus.

9.4.1 SMTP 822-Format

Eine SMTP-Nachricht besteht aus einem Header, auf den die Nachricht folgt. Dieser eigentliche Nachrichtentext wird auch als Message Body bezeichnet. Der Header wird durch eine Leerzeile vom Nachrichtentext getrennt. Eine Header-Zeile besteht meist aus Schlüsselworten, einem Doppelpunkt und einem Wert (Text, Adresse) für das jeweilige Schlüsselwort. Gängige Schlüsselworte sind u.a.: From, To, Subjekt, Date. Sie definieren Absender, Empfänger, Überschrift und das Datum, an dem die E-Mail aufgegeben wurde. Die Header-Zeile der E-Mail darf in mehrere Zeilen unterteilt sein. Der Nachrichtentext der E-Mail besteht aus reinem Text (7-Bit-ASCII) und kann beliebig lang sein.

a) SMTP-Zeichensatz

Die ASCII-Darstellung wird von jeder SMTP-Implementierung unterstützt, da überwiegend Textdateien übertragen werden. Andere Daten wandelt der sendende Rechner vor ihrer Übergabe an das SMTP in eine ASCII-Darstellung um. Diese Reduktion gewährleistet die Übertragung der Daten in einer neutralen Form. Es berücksichtigt aber nicht die rechnerinternen Eigenschaften zur Datendarstellung.

b) SMTP-Funktionen

SMTP verwendet eine TCP-Verbindung, über die ein SMTP-Sender mit einem SMTP-Empfänger Daten austauscht. Jede Sitzung erfordert folgende Schritte:

- Aufbau einer gesicherten Verbindung zwischen Sender und Empfänger
- E-Mail-Übertragung, die aus einer oder mehreren Transaktionen besteht
- Gesicherter Abbau der zwischen Sender und Empfänger bestehenden Verbindung

c) Aufbau der Verbindung

Das SMTP baut immer eine gesicherte Verbindung (TCP Port 25) auf. Anschliessend wird überprüft, ob SMTP-Sender und SMTP-Empfänger kooperieren. Ist dies der Fall, können Nachrichten übermittelt werden.

Der SMTP-Verbindungsauflauf erfolgt in vier Schritten:

- Der Sender öffnet eine TCP-Verbindung zum Empfänger.
- Der Empfänger identifiziert sich gegenüber dem Sender.
- Der Sender identifiziert sich gegenüber dem Empfänger.
- Der Empfänger akzeptiert die Identifikation des Senders.

d) E-Mail-Übertragung

Eine SMTP-Verbindung kann zum Versand zahlreicher E-Mails genutzt werden. Jede Datenübertragung stellt eine logisch eigenständige Transaktion dar. Jede Übertragung durchläuft drei Phasen:

- Der Absender weist sich durch das MAIL-Kommando aus.
- Durch ein oder mehrere RCPT-Kommandos (Recipient; Beispiel siehe Abschnitt 9.4.4) werden die Empfänger der E-Mail definiert.
- Durch das DATA-Kommando wird der Nachrichtentext der E-Mail übermittelt.

e) Abbau der Verbindung

Der Verbindungsabbau durch den SMTP-Sender erfolgt in zwei Schritten: Nach Übertragung des QUIT-Kommandos wird auf eine Antwort (Reply) des Empfängers gewartet. Anschliessend wird ein TCP-CLOSE für die Verbindung initiiert. Unmittelbar nach Empfang des QUIT-Kommandos baut der Empfänger die TCP-Verbindung ab.

Beispiel:

```
Sender: Quit
Empfänger: Bye-Bye
Empfänger: <TCP-Close>
Sender: <TCP-Close>
```

f) Mailbox-Spezifikationen

Sender und Empfänger einer SMTP-E-Mail werden durch eine einfache ASCII-Zeichenfolge dargestellt. Die ASCII-Zeichenfolge hat immer das Format *Benutzer@Domain*.

„Benutzer“ definiert die Person, die unter diesem Account Zugang zur spezifizierten Mailbox hat.
„Domain“ definiert die Organisation, die Abteilung oder den Rechner, der die E-Mail empfangen soll.

Bei Personen, die nicht direkt erreichbar sind, muss der komplette Pfad zum Empfänger definiert werden. Über SMTP Intermediate Hops wird in diesen Fällen anhand der Domains, über die die E-Mail zu übermitteln ist, der genaue Weg zur Mailbox beschrieben.

Jeder SMTP Intermediate Hop wird durch das vorangestellte @-Zeichen kenntlich gemacht. Mehrere SMTP Intermediate Hops werden durch Kommata getrennt. Der letzte SMTP Intermediate Hop wird durch einen Doppelpunkt markiert.

```
@Billo.EDU:Kerstin@Logo.COM  
@MBBV.COM, @Gaston.COM, @Billo.EDU:Kerstin@Logo.COM
```

9.4.2 SMTP-Kommandos

Das Standardformat der SMTP-Kommandos ist die einzelne Textzeile. Jedes Kommando beginnt mit einem vierstelligen Code. Die SMTP-Kommandos sind:

- HELLO
- MAIL
- RECIPIENT
- DATA
- RESET
- SEND
- SEND OR MAIL
- SEND AND MAIL
- VERIFY
- EXPAND
- HELP
- NO OPERATION
- QUIT
- TURN

Hello (HELO domain) meldet einen E-Mail-Client beim E-Mail-Server an.

Mail (MAIL FROM: <absender>) identifiziert den Absender der E-Mail. An diese Adresse können Antwort, Fehler- und Statusmeldungen zurückgesendet werden.

Recipient (RCPT TO: <empfaenger>) definiert die Empfängeradresse, an welche die E-Mail zu senden ist. Durch mehrere RCPT TO:-Kommandos können unterschiedliche Empfängeradressen definiert werden, so dass die E-Mail gleichzeitig an mehrere Empfänger versandt wird. Dabei ist es gleichgültig, ob die Empfänger die Mail als 'To:' oder 'Cc:' erhalten sollen. Diese Angabe erfolgt erst im Header des Datenteils.

Data (DATA) bestimmt den eigentlichen Nachrichtentext einer E-Mail. Der Nachrichtentext muss im 7-Bit-ASCII-Zeichensatz darstellbar sein. Das Ende eines Nachrichtentextes wird durch eine Zeile angezeigt, die nur einen Punkt enthält. Diese Zeile wird auf der Empfängerseite automatisch aus dem Dokument entfernt.

Reset (RSET) bricht die aktuelle E-Mail-Übertragung ab.

Send (SEND FROM: <absender>) sendet eine E-Mail direkt an ein Terminal. Sollte die Gegenstelle nicht aktiv sein oder kann der so adressierte Rechner E-Mail nicht direkt verarbeiten, wird ein 450 Reply Code (Operation nicht durchgeführt; Mailbox nicht vorhanden) generiert. Das Send-Kommando ist erfolgreich durchgeführt, wenn die E-Mail an den adressierten Rechner weitergegeben wurde.

Verify (VRFY <empfaenger>) erlaubt die Überprüfung von Benutzernamen. Der Empfänger des Kommandos soll (falls vorhanden) mit dem vollständigen Benutzernamen und der Mailbox-Adresse antworten.

Expand (EXPN <empfaenger>) ermöglicht die Abfrage vollständiger Mail-Listen. Der Empfänger des Expand-Kommandos soll mit einer vollständigen Liste aller Benutzernamen und der zugehörigen Mailbox-Adressen antworten.

Help (HELP) zeigt Hilfe-Informationen für die Bedienung an.

No Operation (NOOP) bewirkt den Versand einer OK-Bestätigung durch den Server. Es hat keinen Einfluss auf vorher eingegebene Kommandos bzw. Parameter.

Quit (QUIT) beendet die SMTP-Anwendung.

Turn (TURN) kehrt die Senderichtung um: Der aktuelle Empfänger einer E-Mail wird zum Sender einer E-Mail.

9.4.3 SMTP-Bestätigungen

SMTP-Bestätigungen (Reply) bestehen aus einer dreistelligen Zahlenfolge, auf die Klartext folgt. Der Text dient der einfacheren Entschlüsselung der Zahlenfolge durch den Anwender. Im Gegensatz zu den festgelegten Zahlenfolgen (ihre Auswertung erfolgt durch SMTP-Prozesse), ist der Wortlaut der Texte nicht eindeutig definiert. Auf jedes SMTP-Kommando muss mindestens eine Bestätigung folgen.

Die einzelnen Stellen der dreistelligen Zahlenfolge haben definierte Bedeutungen. Die erste Stelle beschreibt den Zustand der Ausführung eines Kommandos (Kommando positiv/negativ/noch nicht abgeschlossen). Die zweite Stelle vertieft die Aussage der ersten Stelle (zum Beispiel: Syntaxfehler). Die dritte Stelle beinhaltet eine weitere Detaillierung der Beschreibung.

Code	Bedeutung
211	Systemstatus oder Bestätigung auf Hilfefunktion
214	Hilfeinformation
220	Rechner bereit für Benutzer
221	Verbindung im Abbau
250	Gewünschte Operation abgeschlossen
251	Benutzer nicht lokal am Rechner, E-Mail wird weitergeleitet zu (Pfadname)
354	Mail-Eingabe bereit. Beenden der Mail-Eingabe mit (CRLF).(CRLF)
421	Rechner nicht verfügbar; Verbindung in Abbau
450	Operation nicht durchgeführt; Mailbox nicht vorhanden
451	Lokaler Fehler; gewünschte Operation abgebrochen
452	Speicher belegt; gewünschte Operation nicht durchgeführt
500	Fehler in Syntax. Kommando unbekannt
501	Syntax-Fehler bei Kommando oder Parameter
502	Kommando nicht unterstützt
503	Fehler in Kommandofolge
504	Parameter für Kommando nicht unterstützt
550	Kein Zugriff auf Mailbox möglich; gewünschte Operation nicht durchgeführt
551	Benutzer nicht lokal, E-Mail weiterleiten zu (Pfadname)
552	Speicher belegt; gewünschte Operation nicht durchgeführt
553	Fehler in Mailbox-Bezeichnung; gewünschte Operation nicht durchgeführt
554	Übermittlung konnte nicht durchgeführt werden

9.4.4 Anwendungsbeispiel

Der SMTP-Daemon kann gut mit Hilfe des Telnet-Programms bedient werden, da das SMTP ein reines ASCII-Protokoll ist. Im folgenden Beispieldialog sind die Zeilen mit einem Prefix zur Identifikation versehen: 'S:' für SMTP-Server und 'C:' für Client.

```
> telnet mail.zhaw.ch 25
...
S: 220 mail.zhaw.ch ESMTP Sendmail 8.8.8/8.8.8
C: HELO zhaw.ch
S: 250 mail.zhaw.ch Hello mth@mail.zhaw.ch, pleased to meet you
C: MAIL FROM:<mth@zhaw.ch>
S: 250 OK
C: RCPT TO:<kls@zhaw.ch>
S: 250 <kls@zhaw.ch>... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Date: 06 Jan 2009 16:34:25 +0100
C: To: kls@zhaw.ch
C: Subject: Test einer Mail.
C:
C: Damit ist der SMTP-Teil beendet.
C: .
S: 250 OK
C: QUIT
S: 221 mail.zhaw.ch closing connection
```

9.4.5 Der MIME-Standard

Das E-Mail-System im Internet und SMTP waren ursprünglich nur für ASCII-Text ausgelegt. Binärwerte waren nicht zulässig. Das heisst, dass keinerlei Sonderzeichen einer Sprache, die im ASCII-Zeichensatz nicht vorkommen, verwendet werden konnten. Um Binärdaten, z.B. eine Grafikdatei oder ein ausführbares Programm, mittels E-Mail senden zu können, waren zusätzliche Mechanismen erforderlich. Mehrere einschlägige Arbeitsgruppen haben Methoden zur Kodierung von Binärdaten in ASCII-Text entwickelt. Eine dieser Methoden nutzt beispielsweise die hexadezimale Darstellung von Zeichen (wie bei einem Speicherdump). Je vier Datenbit werden in eines der 16 Zeichen von 0 bis 9 und von A bis F konvertiert. Die Zeichenfolge wird dann in einer E-Mail-Nachricht versendet. Die Software des Empfängers übersetzt sie in das Binärformat zurück. Zwei dieser Verfahren werden im Abschnitt 9.4.6 erläutert.

Zur Koordination und Vereinheitlichung der verschiedenen Methoden zum Austauschen von Binärdaten im Internet entwickelte die IETF den Standard **MIME (Multipurpose Internet Mail Extensions)**. MIME diktirt aber nicht den Standard zum Kodieren von Binärdaten, sondern ermöglicht einem Sender und Empfänger, eine angebrachte Kodierung zu wählen. Bei der Verwendung von MIME fügt der Sender zusätzliche Zeilen in den Header ein, die besagen, dass die Nachricht im MIME-Format erstellt wurde. Ausserdem werden einige Zeilen in den Rumpf eingefügt, die den Datentyp und die Kodierung definieren. Abgesehen von der Kodierung von Nachrichten ermöglicht es MIME einem Sender auch, eine Nachricht in mehrere Teile aufzuteilen und die Kodierung der einzelnen Teile unabhängig voneinander festzulegen. Ein Benutzer kann somit in einer Nachricht Text und Grafiken senden.

MIME fügt zwei Zeilen in einen E-Mail-Header ein: eine zum Deklarieren, dass die Nachricht mit MIME erstellt wurde, und eine zum Festlegen, wie die MIME-Daten in den Rumpf eingebunden wurden, z.B.:

```
MIME-Version: 1.0
Content-Type: Multipart/Mixed;
Boundary="Mime-Separator-WAB02622.926626671"
```

Diese Zeilen deklarieren, dass die Nachricht mit der MIME-Version 1.0 erstellt wurde und dass eine Zeile mit Mime-Separator-WAB02622.926626671 vor jedem Teil der Nachricht im Rumpf erscheint.

Wird zunächst z.B. eine Textnachricht gesendet, lautet die zweite Zeile wie folgt:

```
Mime-Separator-WAB02622.926626671
Content-Type: text/plain
```

Hallo! Hier folgt der Text.

Folgt anschliessend eine HTML-Page, so wird diese mit dem Separator getrennt und der neue MIME-Type angegeben.

```
Mime-Separator-WAB02622.926626671
Content-Type: text/html
<HTML>
. . .
</HTML>
```

Es sind heute sehr viele MIME-Types definiert. Hier eine kleine Auswahl:

audio/basic	au snd
audio/midi	mid midi kar
audio/mpeg	mpga mp2
audio/x-aiff	aif aifc aiff
audio/x-realaudio	ra
audio/x-wav	wav
image/gif	gif
image/ief	ief
image/jpeg	jpeg jpg jpe
image/png	png
image/tiff	tiff tif
text/html	html htm
text/plain	asc txt c cc h hh cpp.hpp
text/richtext	rtx
text/tab-separated-values	tsv
text/x-vCalendar	vcs
text/x-vCard	vcf
video/mpeg	mp2 mpe mpeg mpg
video/quicktime	qt mov

Unter Unix sind die unterstützten MIME-Types in einem Text-File `/etc/mime.types` zu finden. Es zeigt den Zusammenhang zwischen einem MIME-Type und den File-Typen an. Bei Windows werden diese Informationen in der Registry `HKEY_CLASSES_ROOT` abgelegt:

```
HKEY_CLASSES_ROOT\.gif
Content Type = "image/gif"
```

Die MIME-Types werden heute auch im HTTP-Protokoll verwendet. Der wesentliche Vorteil von MIME ist Flexibilität. Der Standard definiert keine bestimmte Kodiermethode, die alle Sender und Empfänger anwenden müssen. Vielmehr erlaubt MIME jederzeit die Entwicklung und Verwendung neuer Kodiermethoden. Ein Sender und ein Empfänger können zur Kommunikation ein konventionelles E-Mail-System benutzen. Sie müssen sich nur auf eine Kodiermethode und einen eindeutigen Namen dafür einigen. Außerdem gibt MIME keinen bestimmten Wert für die verschiedenen Teile einer Nachricht oder für die Kodiermethode vor. Der Sender kann jedes beliebige Begrenzungssymbol (Separator) wählen, das nicht im Rumpf vorkommt. Wie die Nachricht dekodiert werden muss, ermittelt der Empfänger anhand der im Header enthaltenen Informationen.

MIME ist mit älteren E-Mail-Systemen kompatibel. Ein E-Mail-System, das eine Nachricht überträgt, muss die für den Rumpf oder die MIME-Header-Zeilen angewandte Kodiermethode nicht kennen. Die Nachricht kann behandelt werden wie jede andere E-Mail-Nachricht. Das Mail-System überträgt die Header-Zeilen, ohne sie zu interpretieren, und behandelt den Rumpf als einzelnen Textblock.

9.4.6 Content-Transfer-Encoding

Wie oben erläutert, müssen für die Übertragung binäre Daten oft in 7-Bit-Symbole (ASCII-Zeichen) umcodiert werden. Um die Decodierung zu ermöglichen, kann der MIME-Type zusätzlich mit einer der folgenden Codierungsart versehen werden:

- "B"-Encoding, "Base64"
- "Q"-Encoding, "Quoted-Printable"
- Keine Codierung (binäre Übertragung): "7bit" / "8bit" / "binary"
- Für spätere Erweiterungen: "ietf-token" / "x-token"

Die Angabe der Codierung erfolgt mit einer weiteren Zeile im MIME-Header:

Content-Transfer-Encoding: *encoding*

Beispiel:

```
Content-Type: image/gif
Content-Transfer-Encoding: base64
```

a) Quoted-Printable

Eine einfache Umcodierung erfolgt mit Quoted-Printable (RFC 2045). Jeder 8-bit Wert wird durch 3 ASCII-Zeichen dargestellt. Eine solche Sequenz beginnt immer mit einem Gleichheitszeichen „=“ gefolgt von 2 Hex-Ziffern (0..9, A..F). Dabei müssen Gleichheitszeichen, Fragezeichen, Underscore, Space and Tab immer als Quoted-Printable übertragen werden.

Beispiele (ISO-8859-1 Zeichensatz):

”1“ → =31

”=“ → =3D

” “ → =20

Dieses Verfahren eignet sich gut für Texte, die gelegentlich mal einen 8-Bit Wert (z.B. Umlaut) enthalten. Mit etwas Übung kann man die Texte sogar noch in codierter Form lesen. Für grosse Datenmengen und rein binäre Daten stört die schlechte Effizienz (30%) des Verfahrens.

b) Base64 Encoding

Die Base64-Codierung (RFC 2045) ist für rein binäre Daten wie Bilder oder Programme besser geeignet als Quoted-Printable. Jeweils drei (binäre) Bytes werden in vier Bitgruppen à 6 Bit aufgeteilt. Das Prinzip ist in Abbildung 9.15 dargestellt.

Die 6-Bit-Gruppen (Werte 0 bis 63) mit druckbaren ASCII-Zeichen dargestellt:

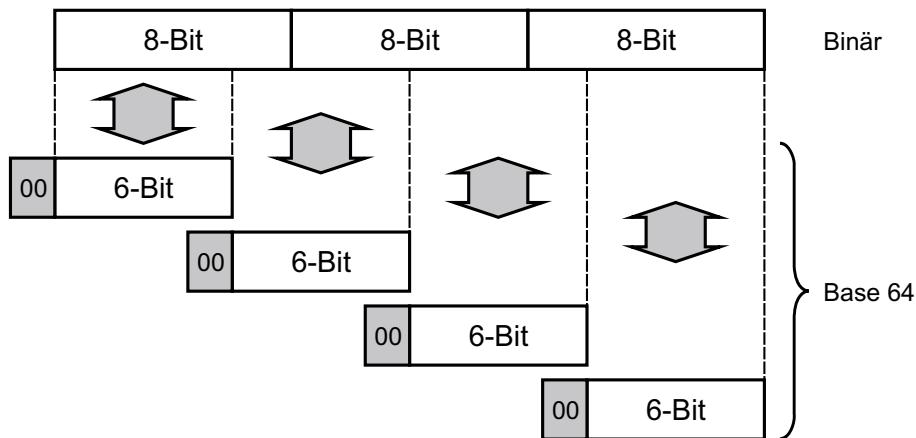


Abbildung 9.15: Codierung von 3 Bytes mit 4 ASCII Symbolen

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Die resultierenden ASCII-Zeichen werden in Zeilen mit einer Länge von maximal 76 Zeichen übertragen. Damit wird eine Effizienz von 73% erreicht.

Da eine Datei innerhalb einer solchen Dreiergruppe enden kann, muss ein Symbol für „keine weiteren Daten“ vorgesehen werden. Dazu wird das Gleichheitszeichen benutzt „=“.

9.5 Hypertext Transfer Protocol (HTTP)

Das Hypertext Transfer Protocol (HTTP) ist ein Application Layer Protocol für den Zugriff auf verteilte Hypermedia-Informationssysteme.

HTTP ist sehr populär, da das World-Wide-Web (WWW) darauf basiert. Die Aufgabe von HTTP ist dabei die Übertragung von HTML-Dokumenten, Images und anderen Daten (**Ressourcen**) zwischen Web-Browser und Web-Server. Jede Ressource wird durch einen URL (Unique Resource Locator) identifiziert.

Die erste Version 0.9 von HTTP erschien 1989. Sie war überaus einfach und elegant. Heute ist die Version HTTP/1.1 aktuell, die im RFC 2616 beschrieben ist. Zu den neuen Möglichkeiten zählen unter anderem:

- Erweiterte Unterstützung für Caching.
- Antworten können in mehreren Teilen (Chunks) gesendet werden. Dies führt insbesondere bei Skripten, die grosse Datenmengen zurückliefern oder umfangreiche Berechnungen durchführen, zu schnelleren Antworten.
- Unterstützung von mehreren „logischen“ Servern pro IP-Adresse (sogenannte **multi-homed Server**).
- Persistente Verbindungen – die Verbindung zwischen Client und Server wird über mehrere Transaktionen hinweg aufrechterhalten, um bessere Antwortzeiten zu erhalten.

9.5.1 Funktionsweise

Das Funktionsprinzip von HTTP weist viele Ähnlichkeiten mit SMTP auf: Es ist auch ein ASCII-Protokoll, es werden MIME-Typen und Status-Codes verwendet.

Eine HTTP-Transaktion besteht immer aus einer Anfrage (HTTP Client Request) und einer Antwort (HTTP Server Reply). HTTP arbeitet dabei

verbindungsorientiert: Für eine Transaktion wird eine gesicherte Verbindung (TCP) aufgebaut.

zustandslos: Zwischen zwei Transaktionen muss keine Information über die Verbindung gespeichert werden. Der Server „vergisst“ nach der Transaktion alles und ist wieder im ursprünglichen Zustand.

Der wartende Server (typischerweise an Port 80) nimmt den Request entgegen und sendet bei erfolgreicher Bearbeitung eine Antwort zurück. Andernfalls übermittelt er eine Fehlermeldung an den Client. Danach wird die Verbindung wieder abgebaut.

Wie Abbildung 9.16 zeigt, werden für eine HTML-Seite im Allgemeinen mehrere Transaktionen benötigt. Der Client muss diese mit separaten Transaktionen vom Server (oder von mehreren Servern) anfordern.

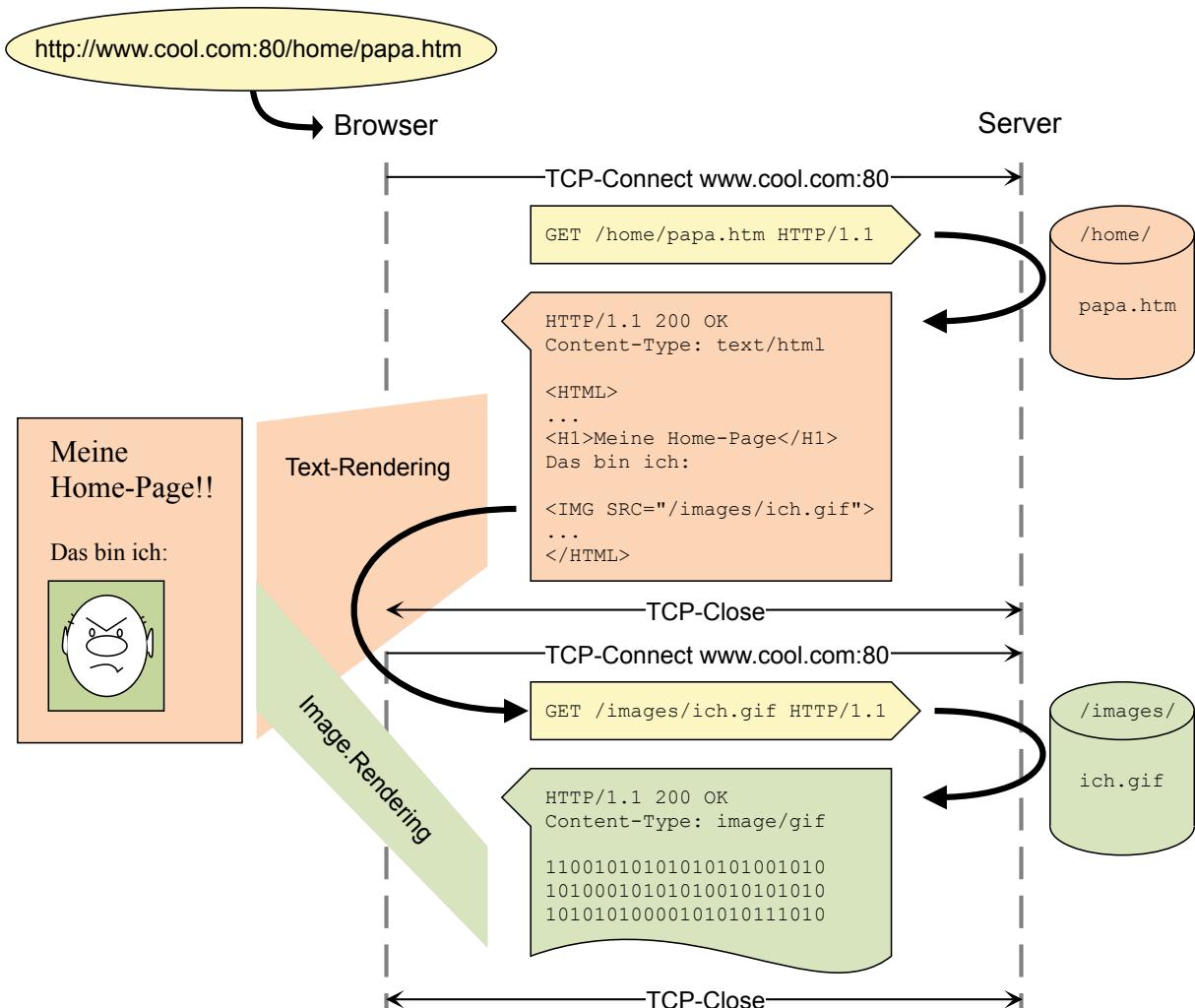


Abbildung 9.16: Abarbeitung einer HTML-Seite

Die folgende Tabelle zeigt einige HTTP Client Request Methoden:

GET	Fordert vom HTTP-Server Daten einer bestimmten URL an.
POST	Schickt Daten an den HTTP-Server zum „Abspeichern“.
HEAD	Wie GET; der HTTP-Server schickt aber nur die Header-Information.
DELETE	Löscht die Daten einer bestimmten URL.
OPTIONS	Der HTTP-Server liefert Informationen über die verfügbaren Optionen.

9.5.2 HTTP-Request

Die weitaus häufigsten Requests sind GET und PUT. Während bei HTTP/0.9 ein Request noch aus einer Zeile bestand (z.B. GET /pfad/zu/meinem/dokument.html), sieht ein typischer GET-Request heute wie folgt aus:

```
GET /pfad/zu/meinem/dokument.html HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, */
Accept-Language: de, en
User-Agent: Mozilla/4.0
Host: www.ktlabor.ch:80
```

Der obige Request besteht aus

- einer Anforderungszeile (**Initial Request Line**), welche – durch Leerzeichen getrennt – die Art des Requests (GET), die angeforderte URL und die HTTP-Version angibt.
- einer Folge von Zeilen, die Informationen über die Fähigkeiten des Clients (Browser) enthalten. Diese Zeilen besitzen ein einheitliches, in RFC 822 definiertes Format: Feldname : Wert. HTTP 1.1 definiert 17 verschiedene Felder für Request-Header, von denen bei Anfragen keines erforderlich ist. Die am häufigsten in Anfrageköpfen verwendeten Felder sind folgend:

Accept	Gibt an, welche MIME-Types der Browser darstellen kann.
Accept-Language	Gibt an, welche Sprachen der Benutzer bevorzugt.
If-Modified-Since	Es sollen nur Dokumente übertragen werden, die seit dem angegebenen Zeitpunkt modifiziert wurden. Andernfalls wird die Meldung 304 Not Modified zurückgegeben.
User-Agent	Identifiziert den verwendeten Web-Browser.
Host	Knotenname, wie er in der URL ursprünglich angegeben wurde.

- Einer leeren Zeile, die den Request-Header (analog zu SMTP) abschliesst.

9.5.3 HTTP-Reply

Nachdem der Server einen Client-Request erhalten hat, ist seine Aufgabe recht einfach: Er muss die gewünschte Ressource finden und den Wünschen des Clients folgen.

In diesem Fall, wo ein einfacher GET-Request vorliegt, heisst dies: Es wird die Position der Datei dokument.html auf dem Disk bestimmt und deren Inhalt wird über die TCP-Verbindung zum Client geschickt. Vorausgesetzt, dass das File existiert und der Client die Berechtigung hat es zu lesen, erhält er eine Antwort von etwa folgendem Format:

```
HTTP/1.1 200 OK
Date: 02 Feb 2099 15:52:12 GMT
```

Server: Apache/9.9.9 (Unix)
Content-Type: text/html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
  <HEAD>
    <TITLE> ... </TITLE>
  </HEAD>
  <BODY>
  ...
  </BODY>
</HTML>
```

Der obige Header besteht aus:

Status Line: Diese besteht ihrerseits aus mehreren Teilen:

Die **HTTP-Version** gibt an, dass der Server HTTP/1.1 konform antwortet.

Die Zahl „200“ ist der **Response Status Code**, der analog zu den SMTP-Stati aufgebaut ist.
Die zugehörige **Reason Phrase** entspricht dem Statuscode im Klartext.

Die wichtigsten Statuscodes sind:

200	OK	Allgemeine Erfolgsmeldung. Die angefragte Nachricht (Ressource) folgt.
400	Bad Request	Anfrage enthielt eine falsche Syntax.
404	Not Found	Die angefragte Ressource existiert nicht.
500	Internal Server Error	Ein unerwarteter Server-Fehler ist aufgetreten, z.B. durch ein fehlerhaftes CGI-Skript.

Response Header Fields: Mehrere optionale Zeilen, die Angaben zum Server liefern.

Unter anderen kommen folgende Felder zur Anwendung:

- Last-Modified Datum der letzten Änderung der Ressource.
- Server Identifiziert den verwendeten Web-Server.
- Content-Length Gibt die Länge der Nachricht im Body in Bytes an.
- Content-Type Legt den MIME-Type der Nachricht im Body fest.

10

Die Schnittstelle zum Transport-Layer

Dieses Kapitel enthält Details das Transport-Layer-Interface, also die Schnittstelle zwischen einer Anwendung und der Protokoll-Software. Anhand eines Beispiels werden Prozeduren aufgeführt, die eine Anwendung benutzen kann, um als Client oder Server mit einem entfernten Ziel Verbindung aufzunehmen bzw. Daten zu übertragen.

10.1 Einleitung

Die Schnittstelle zwischen einer Anwendung und der Transport-Protokoll-Software heisst **Application Program Interface (API)**, siehe Abbildung 10.1).

Ein API definiert eine Menge von Prozeduren und Datentypen, welche die Anwendung verwenden kann. Die Operationen, die eine Anwendung bei der Interaktion mit Protokoll-Software ausführen kann, bestimmen im wesentlichen die Funktionalität, die einer Anwendung zur Verfügung steht, sowie den Komfort um mit Hilfe dieser Funktionalität eine Anwendung zu erstellen.

Bei der Interaktion mit Protokoll-Software muss eine Anwendung Details angeben, beispielsweise, ob sie ein Server oder ein Client ist (d.h. ob sie passiv wartet oder aktiv die Kommunikation einleitet). Ausserdem muss der Sender die zu übertragenden Daten spezifizieren, und der Empfänger muss bestimmen, wo die ankommenden Daten gespeichert werden sollen.

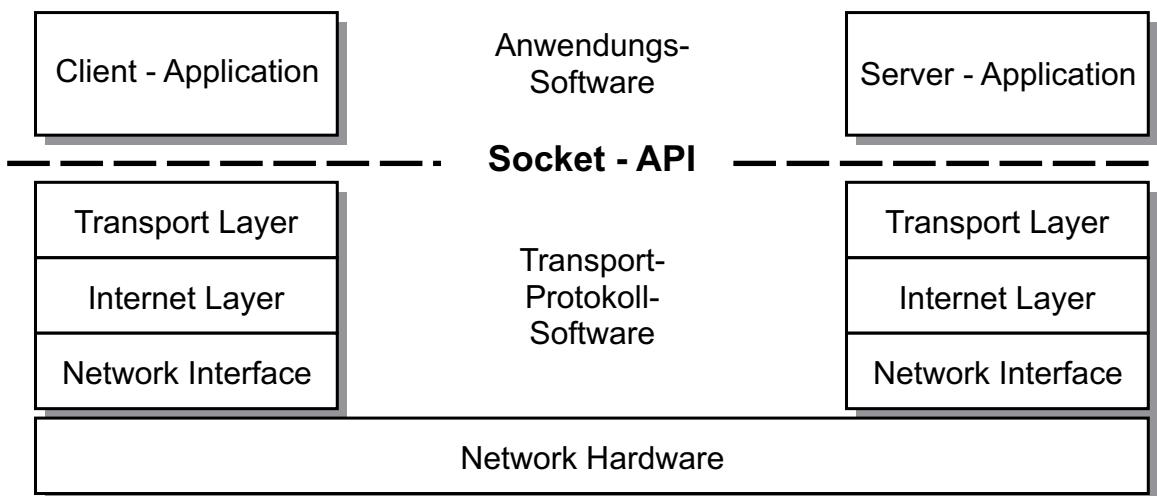


Abbildung 10.1: Das Socket-API als Schnittstelle zwischen Anwendung und Protokoll-Software

10.1.1 Das Socket-Application Program Interface

Standards für Kommunikationsprotokolle beinhalten normalerweise keine Beschreibungen, wie die Interaktion von Anwendungen mit den Protokollen erfolgen soll. Damit kann im Prinzip jedes Betriebssystem oder jede Sprache (C und Java), ein spezifisches API (Application Program Interface) definieren, das dann von den Anwendungen benutzt werden kann. Ein Protokollstandard wird also beispielsweise festschreiben, dass eine Operation zum Senden von Daten nötig ist, während das API die genaue Bezeichnung der Funktion und die einzelnen Argumente vorgibt.

Das Socket-API ist heute für praktisch alle Betriebssysteme verfügbar. Damit ist es der *De-facto-Standard* in der Branche. Mit etwas Sorgfalt können damit portable Applikationen geschrieben werden (siehe Beispiel im Abschnitt 10.6).

Die Sockets und zugehörige C-API wurden erstmals im Jahr 1982 von der **University of California at Berkeley** mit dem Betriebssystem BSD-UNIX 4.1c für VAX-Rechner (ehemals Digital Equipment Corporation) zur Verfügung gestellt. Die Entwicklungsarbeit dazu war durch US-Regierungszuschüsse gefördert worden.

Viele Computeranbieter profitierten anschliessend davon. Sie portierten das BSD-System auf ihre Hardware und benutzen es als Grundlage für kommerzielle Betriebssystemprodukte.

10.1.2 Socket und UNIX-Ein-/Ausgaben

Da Sockets ursprünglich als Teil des UNIX-Betriebssystems entwickelt wurden, weisen sie viele Konzepte aus der UNIX-Welt auf. Insbesondere sind Sockets im I/O-System integriert. Das heisst, eine Anwendung kommuniziert durch ein Socket auf ähnliche Weise, wie sie Daten von oder in eine Datei überträgt. Um die Funktionsweise von Sockets zu verstehen, ist daher eine gewisse Grundkenntnis der UNIX-Ein-/Ausgabefunktionen nötig.

In UNIX basieren alle Ein- und Ausgaben auf dem Paradigma `open-read-write-close`. Diese Bezeichnungen stammen aus den Ein-/Ausgabe-Operationen für Geräte und Dateien. So muss eine Anwendung beispielsweise zuerst `open` aufrufen, um eine Datei für den Zugriff vorzubereiten. Die Anwendung ruft dann `read` oder `write` auf, um Daten aus der Datei zu lesen oder in die Datei zu schreiben. Schliesslich ruft die Anwendung `close` auf, um zu verstehen zu geben, dass sie mit der Verwendung der Datei fertig ist.

Wenn eine Anwendung eine Datei oder ein Gerät öffnet, gibt der Aufruf `open` einen **Descriptor** – eine kleine Ganzzahl, welche die Datei bezeichnet – zurück. Die Anwendung muss diesen Descriptor bei der Anforderung zur Datenübertragung angeben (d.h. der Descriptor ist ein Argument der `read`- oder `write`-Prozedur). Ruft eine Anwendung beispielsweise `open` auf, um auf eine Datei namens „haba.kuk“ zuzugreifen, gibt die `open`-Prozedur möglicherweise den Descriptor 3 zurück. Bei dem anschliessenden Aufruf von `write` wird der Descriptor 3 angegeben, so dass Daten in die Datei „haba.kuk“ geschrieben werden. Der Dateiname erscheint nicht im `write`-Aufruf.

10.1.3 Socket-Ein-/Ausgaben und Descriptoren

Bei der Socket-Kommunikation wird ebenfalls mit Descriptoren gearbeitet. Bevor eine Anwendung kommunizieren kann, muss sie das Betriebssystem auffordern, einen Socket zu erstellen. Das System gibt für den neuen Socket einen Descriptor (16-Bit-Integer) zurück. Wenn die Anwendung Prozeduren zur Datenübertragung im Netzwerk aufruft, gibt sie den Descriptor als Argument an. So muss die Anwendung nicht bei jeder Datenübertragung Einzelheiten über das Ziel angeben.

In einer UNIX-Implementierung sind Sockets vollständig in die übrigen Ein-/Ausgaben integriert. Das Betriebssystem verwaltet eine Menge von gemeinsamen Descriptoren für Dateien, Geräte sowie die Interprozess- und die Netz-Kommunikation. Das bedeutet, dass Prozeduren wie `read` und `write` allgemein anwendbar sind. Eine Anwendung kann die gleiche Prozedur benutzen, um Daten an ein anderes Programm, eine Datei oder im Netzwerk zu übertragen. Der Descriptor stellt ein Objekt und die `write`-Prozedur eine Methode dar, die auf dieses Objekt angewandt wird. Das zugrundeliegende Objekt bestimmt, wie die Methode angewandt wird.

Der wesentliche Vorteil eines integrierten Systems ist seine Flexibilität: Mit einer einzelnen Anwendung können Daten an eine beliebige Stelle übertragen werden. Erhält die Anwendung einen Descriptor, der einem Gerät entspricht, sendet die Anwendung die Daten an dieses Gerät. Entspricht der Descriptor einer Datei, schreibt die Anwendung die Daten in diese Datei. Entspricht der Descriptor einem Socket, sendet die Anwendung die Daten in einem Internet an eine entfernte Maschine.

10.1.4 Socket-API und Parameter

Die Socket-Programmierung unterscheidet sich von konventionellen Ein-/Ausgaben, weil eine Anwendung viel mehr Parameter definieren muss. Sie muss beispielsweise ein bestimmtes Transportprotokoll wählen, die Protokolladresse eines entfernten Rechners angeben und festlegen, ob sie ein Client oder ein Server ist.

Um zu viele Parameter bei den Socket-Funktionen zu vermeiden, definiert das Socket-API eine Anzahl von Funktionen, welche die Betriebsart genau festlegen. Im wesentlichen erzeugt eine Anwendung also einen Socket und ruft dann Funktionen auf, um die Einzelheiten seiner Verwendung zu spezifizieren. Dieser Ansatz hat den Vorteil, dass die meisten Funktionen nicht mehr als drei Parameter haben. Der Nachteil dabei ist, dass sich ein Programmierer daran erinnern muss, welche Funktionen er bei der Verwendung von Sockets aufrufen muss.

Die folgende Liste enthält nur eine Auswahl der wichtigsten Funktionen des Socket-API. Sie werden im Abschnitt 10.3 einzeln beschrieben. Im Abschnitt 10.6 wird das Zusammenspiel anhand einer Applikation aufgezeigt.

socket	Erzeugt einen Socket-Descriptor, der im folgenden für die Netzwerk-Kommunikation verwendet wird.
connect	Verbindet einen Client mit einem Server-Dienst.
write	Sendet Daten über eine Netzwerk-Verbindung.
read	Liest Daten von einer Netzwerk-Verbindung.
close	Schliesst eine Netzwerk-Verbindung.
bind	Verknüpft ein Socket mit einer lokale IP-Adresse und einer Port-Nummer.
listen	Setzt ein Socket eines Servers in den passiven (Warte-)Zustand und bestimmt die Grösse der Warteschlange.
accept	(Server) wartet auf die nächste Verbindung eines Clients.
recv	Liest das nächste Datagramm von einer Netzwerk-Verbindung.
send	Sendet ein Datagramm über eine Netzwerk-Verbindung.
shutdown	Schiesst eine Netzwerk-Verbindung.
getsockopt	Liefert die aktuellen Einstellungen eines Sockets.
setsockopt	Verändert die Einstellungen eines Sockets.

10.2 Funktionen zur Integer-Konversion

TCP/IP definiert wie Integer-Werte des Headers (wie zum Beispiel das IP-Header-Feld „Protocol“) im Netz übertragen werden.

Der Internet-Standard legt fest, dass von Integer-Werten das Most Significant Byte immer zuerst übertragen wird (Network Byte Order).

Im Gegensatz zur genormten **Network Byte Order** spricht man bei der Darstellung auf den Rechnern von der **Host Byte Order**. Diese kann je nach Rechner-Typ verschieden sein (Big- oder Little Endian). Die Host Byte Order von Big-Endian-Rechnern (höchstwertiges Byte an der tiefsten Adresse) entspricht in der Reihenfolge der Network Byte Order. Intel-Rechner mit 80x86-Architektur verwenden die Little-Endian-Darstellung (tiefstwertiges Byte an der tiefsten Adresse) und benötigen daher eine Konversion.

Es gibt eine ganze Reihe von Funktionen, um diese Konversion vorzunehmen. (Unter Unix befinden sich die Deklarationen im Header-File `netinet/in.h`).

```
unsigned long int htonl(unsigned long int hostlong);  
unsigned short int htons(unsigned short int hostshort);  
unsigned long int ntohl(unsigned long int netlong);  
unsigned short int ntohs(unsigned short int netshort);
```

Die Funktionen haben folgende Aufgaben:

- htonl() Konvertiert den Long-Integer-Parameter `hostlong` von der Host-Byte-Order in die Network-Byte-Order.
- htons() Konvertiert den Short-Integer-Parameter `hostshort` von der Host-Byte-Order in die Network-Byte-Order.
- ntohl() Konvertiert den Long-Integer-Parameter `netlong` von der Network-Byte-Order in die Host-Byte-Order.
- ntohs() Konvertiert den Short-Integer-Parameter `netshort` von der Network-Byte-Order in die Host-Byte-Order.

Es wird empfohlen diese Routinen auf jeden Fall aufzurufen, auch wenn keine Konversion nötig ist, da damit der Programmcode auch auf andere Rechner übertragen werden kann.

10.3 Prozeduren zur Implementierung des Socket-API

In diesem Abschnitt werden zunächst die Datenstrukturen und Funktionen des Socket-API einzeln erläutert. Anhand des Beispiel im Abschnitt 10.6 wird deren Zusammenspiel aufgezeigt. Unter Unix sind diese in den Header-Files `sys/socket.h` und `sys/types.h` definiert.

10.3.1 Die Prozedur `socket`

Diese Prozedur erzeugt einen Socket und gibt einen ganzzahligen Descriptor zurück:

```
mysocket = socket (protofamily, type, protocol)
```

protofamily definiert die Protokollfamilie, die mit dem Socket zu verwenden ist. Der Wert `PF_INET` steht beispielsweise für TCP/IP-, `PF_IPX` für die Novell- und `PF_APPLETALK` für die AppleTalk-Protokolle.

type definiert den gewünschten Kommunikationsdienst. Die beiden üblichen Typen sind ein verbindungsorientierter Stream-Transfer (mit dem Wert `SOCK_STREAM`) und ein verbindungsloser Nachrichtentransfer (mit dem Wert `SOCK_DGRAM`).

protocol falls in einer Protokollfamilie zwei oder mehr Protokolle den gleichen Dienst zu Verfügung stellen, definiert das Argument `protocol` (zusätzlich zu `type`) genau *ein* bestimmtes Transportprotokoll. Das Argument ist optional (Null heisst keine Vorgabe).

Die Werte, die mit dem `protocol`-Argument benutzt werden können, hängen von der Protokollfamilie ab. So ist z.B. TCP zwar im TCP/IP-Protokoll-Stack, nicht aber im AppleTalk-Stack enthalten.

10.3.2 Die Prozedur `close`

Diese Prozedur weist das System an, die Benutzung eines Socket zu beenden. Sie hat die Form

```
close (mysocket);
```

wobei `mysocket` der Descriptor des zu schliessenden Socket ist. Benutzt der Socket ein verbindungsorientiertes Transportprotokoll, beendet `close` die Verbindung vor dem Schliessen des Socket. Das Schliessen eines Socket beendet die Verwendung sofort. Der Descriptor wird freigegeben, wodurch die Anwendung am Senden weiterer Daten gehindert wird, und das Transportprotokoll nimmt keine weiteren Nachrichten für den Socket an, so dass die Anwendung keine weiteren Daten mehr empfangen kann.

10.3.3 Die Prozedur bind

Ein erzeugter Socket hat zunächst weder eine Absender- noch eine Ziel-Adresse. Ein Server benutzt die `bind`-Prozedur, um eine Protokoll-Portnummer bereitzustellen, die den Port bezeichnet, an dem der Server auf Kontakt wartet. `bind` hat drei Argumente:

```
bind(mysocket, localaddr, addrlen)
```

mysocket ist der Descriptor von einem Socket, der erzeugt, aber noch nicht gebunden wurde.

Der Aufruf ist eine Anforderung, dem Socket eine bestimmte Protokoll-Portnummer zuzuweisen.

localaddr ist eine Struktur, welche die lokale Adresse (Absender-Adresse) definiert, die dem Socket zuzuweisen ist.

addrlen ist eine Ganzzahl die der Länge der Adress-Struktur `localaddr` entspricht (unter Linux z.B. 16).

Da Sockets mit beliebigen Protokollen benutzt werden können, hängt das Format einer Adresse vom benutzten Protokoll ab. Das Socket-API definiert eine allgemeine Form, die zur Darstellung von Adressen benutzt wird, und verlangt dann von jeder Protokollfamilie den Hinweis, wie die allgemeine Form von den jeweiligen Protokolladressen benutzt wird.

Das allgemeine Format zur Darstellung einer Adresse ist eine `sockaddr`-Struktur, wovon es mehrere Versionen gibt. Unter Linux enthält die `sockaddr`-Struktur zwei Felder:

```
struct sockaddr {
    unsigned short sa_family;      /* Familie der Adresse */
    char sa_data[14];              /* Adresse */
};
```

Das Feld `sa_family` definiert die Familie, zu der eine Adresse gehört (für TCP/IP-Adressen wird die symbolische Konstante `AF_INET` benutzt). Das Feld `sa_data` enthält Bytes für die Adresse.

Jede Protokollfamilie definiert das genaue Format der mit dem Feld `sa_data` einer `sockaddr`-Struktur benutzten Adressen.

Für TCP/IP-Protokolle wird beispielsweise die Struktur `sockaddr_in` zum Definieren einer Adresse benutzt:

```
/* Internet address. */
struct in_addr {
    __u32 s_addr;
};

/* Structure describing an Internet (IP) socket address. */
struct sockaddr_in {
    short int sin_family; /* Address family */           */
    unsigned short int sin_port; /* Port number */        */
};
```

```
    struct in_addr      sin_addr;    /* Internet address          */
    unsigned char       __pad[8];    /* Pad to size of sockaddr */
```

Das erste Feld der Struktur `sockaddr_in` entspricht genau dem ersten Feld der allgemeinen `sockaddr`-Struktur.

Die letzten drei Felder definieren die genaue Form der Adresse für TCP/IP-Protokolle. Sie entsprechen zusammen dem zweiten Feld der `sockaddr`-Struktur. Eine solche Adresse identifiziert einen Rechner *und* eine bestimmte Anwendung auf diesem.

Das Feld `sin_addr` enthält die IP-Adresse des Rechners. Der Typ `__u32` ist als 32-Bit Unsigned-Integer definiert.

Im Feld `sin_port` steht die Protokoll-Portnummer einer Anwendung.

Die allgemeine `sockaddr`-Struktur reserviert für die Adresse 14 Byte, obwohl TCP/IP nur 4 Byte-Adressen benutzt. Das letzte Feld in der Struktur `sockaddr_in` definiert darum ein 8-Byte-Feld (mit Nullen), mit denen die Struktur auf die Grösse von `sockaddr` aufgefüllt wird.

Ein Server definiert mit `bind` die Protokoll-Portnummer für die Kommunikationsverbindung. Zusätzlich zu einer Protokoll-Portnummer enthält die Struktur `sockaddr_in` auch ein Feld für eine IP-Adresse.

Ein Server kann nach eigenem Ermessen bei der Angabe einer Adresse die IP-Adresse ausfüllen. Dies kann aber gerade bei Hosts mit mehreren Interfaces (Multihomed-Host) zu Problemen führen, weil es bedeutet, dass der Server nur Anfragen akzeptiert, die an eine bestimmte Adresse gerichtet sind.

Damit ein Server auf einem Multihomed-Host betrieben werden kann, wird die symbolische Konstante `INADDR_ANY` als IP-Adresse angegeben. Mit dieser speziellen Konstante kann ein Server-Programm auf einem Rechners einen bestimmten Port von beliebigen IP-Adressen (Interfaces) bedienen.

10.3.4 Die Prozedur `listen`

Nachdem ein Server das Protokoll-Port definiert hat, muss er den Socket in den sogenannten *passiven* Betriebsmodus versetzen. Dazu ruft der Server die Prozedur `listen` auf, die zwei Argumente hat:

```
listen (mysocket, queuesize)
```

mysocket ist der Descriptor von einem erzeugten und mit einer lokalen Adresse gebundenen Socket.

queuesize definiert eine Länge für die Warteschlange des Socket.

Danach kann der Socket „Anrufe“ von mehreren Clients entgegennehmen. Das Runtime-System stellt für jeden Socket eine eigene Warteschlange bereit. Anfänglich ist die Warteschlange leer. Die von Clients ankommenden Anrufe werden in die Warteschlange gestellt.

Ist die Warteschlange voll, wird der Anruf vom System verworfen (besetzt). Die Warteschlange ermöglicht die Annahme neuer Anrufe, während der Server mit der Abarbeitung vorheriger beschäftigt ist. Der Parameter `queuesize` ermöglicht es jedem Server, eine maximale Grösse der Warteschlange entsprechend dem erwarteten Dienst zu wählen.

10.3.5 Die Prozedur `accept`

Alle Server beginnen damit, mit `mysocket` ein Socket zu erzeugen und mit `bind` eine Protokoll-Portnummer zu definieren. Nach der Ausführung der beiden Aufrufe ist ein Server, der ein *verbindungsloses* Transportprotokoll benutzt, für die Entgegennahme von Nachrichten bereit. Bei einem Server, der mit einem *verbindungsorientierten* Transportprotokoll arbeitet, fallen zusätzliche Schritte an, bis Nachrichten angenommen werden können:

Ein Server muss zunächst `listen` aufrufen, um den Socket in den passiven Modus zu versetzen. Erst dann kann er Verbindungsanfragen von Clients (siehe `connect`) akzeptieren.

Bei der Verbindung wird ein neuer Socket erzeugt, der für die eigentliche Kommunikation zwischen Server und Client genutzt wird. Nach be endeter Kommunikation muss dieser mit `close` wieder geschlossen werden.

Server, die ein **verbindungsorientiertes Transportprotokoll** benutzen, aktivieren die Prozedur `accept`, um die nächste Verbindungsanfrage anzunehmen. Steht eine Anfrage in der Warteschlange, gibt `accept` diese sofort zurück. Andernfalls blockiert das System den Server so lange, bis von einem Client ein Verbindungswunsch eingeht. `accept` hat folgende Form:

```
newsock = accept (mysocket, client_addr, caddr_len)
```

mysocket ist der Descriptor von einem Socket, den der Server erzeugt und mit einem bestimmten Protokollport gebunden hat.

client_addr ist die Adresse einer Struktur vom Typ `sockaddr`.

caddr_len ist ein Zeiger auf eine Ganzzahl.

`accept` füllt die Felder des Arguments `client_addr` mit der Adresse des Clients, der die Verbindung aufgebaut hat, und setzt `caddr_len` auf die Länge der Adresse.

Schliesslich erzeugt `accept` einen neuen Socket `newsock` für die Verbindung und gibt der ru fenden Seite den Descriptor des neuen Socket zurück. Der Server benutzt den Socket zur Kom munikation mit dem Client und schliesst dann den Socket. Der Original-Socket des Servers bleibt dabei unverändert. Der Server benutzt diesen wieder zur Annahme des nächsten Verbindungswunsches eines Clients.

10.3.6 Die Prozedur `connect`

Zum Aufbau einer Verbindung zu einem bestimmten Server benutzen Clients die Prozedur `connect`, die folgende Form hat:

```
connect (mysocket, server_addr, saddr_len)
```

mysocket ist der Descriptor vom einem Socket auf dem Rechner des Clients, das für die Verbindung benutzt wird.

server_addr ist eine `sockaddr`-Struktur, welche die Adresse und die Protokoll-Portnummer des Servers bezeichnet.

saddr_len definiert die Länge der Serveradresse in Bytes.

Bei Verwendung mit einem verbindungsorientierten Transportprotokoll wie TCP leitet `connect` eine Verbindung zum angegebenen Server auf der Transportschicht ein.

Im wesentlichen ist `connect` die Prozedur, die ein Client benutzt, um mit einem Server Verbindung aufzunehmen, der `accept` aktiviert hat.

10.3.7 Die Prozedur `send`

Sowohl Clients als auch Server müssen Daten versenden können. Beispielsweise sendet ein Client eine Anfrage und ein Server eine Antwort. Ist der Socket verbunden, kann die Prozedur `send` für den Datentransfer benutzt werden. `send` hat vier Argumente:

```
send (mysocket, data, length, flags)
```

mysocket ist der Descriptor des zu benutzenden Socket.

data ist die Speicheradresse der zu sendenden Daten.

length definiert die Anzahl der Datenbytes.

flags enthält Bits für bestimmte Optionen.

10.3.8 Die Prozedur `recv`

Für den Empfang von Daten empfangen kann, bietet das Socket-API mehrere Prozeduren an. So kann eine Anwendung beispielsweise `recv` verwenden, um Daten von einem verbundenen Socket zu empfangen. Die Prozedur hat folgende Form:

```
recv (mysocket, buffer, length, flags)
```

mysocket ist der Descriptor eines Socket, von dem Daten zu empfangen sind.

buffer definiert die Adresse eines Puffers, in dem die ankommende Nachricht abzulegen ist.

length definiert die Grösse des Puffers.

flags ermöglicht der rufenden Seite eine gewisse Bestimmung von Details (z.B. kann einer Anwendung gestattet werden, eine Kopie einer ankommenden Nachricht zu nehmen, ohne die Nachricht selbst aus dem Socket zu entfernen).

10.4 Socket-Datenaustausch mit `read` und `write`

Das Socket-API wurde ursprünglich als Teil von UNIX entwickelt; deshalb ermöglicht es Anwendungen, `read` und `write` für den Datenaustausch zu benutzen. Wie `send` und `recv` haben `read` und `write` keine Argumente für die Angabe eines Ziels durch die rufende Seite.

`read` und `write` haben drei Argumente: einen Socket-Descriptor, die Stelle eines Speicherpuffers zum Ablegen von Daten und die Länge des Speicherpuffers. Deshalb müssen `read` und `write` mit verbundenen Sockets benutzt werden.

`read` und `write` haben den Vorteil, dass man ein Anwendungsprogramm für den allgemeinen Datenaustausch schreiben kann, ohne dass bekannt sein muss, ob es sich beim Descriptor um eine Datei oder ein Socket handelt. Ein Programmierer kann beispielsweise eine auf der lokalen Platte gespeicherte Datei benutzen, um einen Client oder Server zu testen, ohne dass eine Verbindung im Netz aufgebaut wird. Der Nachteil der beiden Prozeduren ist, dass (je nach Implementierung) zusätzlicher Overhead entstehen kann.

10.5 Weitere Socket-Prozeduren

Das Socket-API umfasst weitere nützliche Prozeduren.

Beispielsweise sind Prozeduren vorhanden, um Socket-Optionen zu setzen und die aktuellen Werte abzurufen. Diese Optionen werden hauptsächlich für Spezialfälle benutzt, z.B. um den von der Protokoll-Software benutzten internen Buffer zu vergrößern.

Die Prozedur `setsockopt` setzt Werte von Socket-Optionen.

Mit der Prozedur `getsockopt` können die aktuellen Optionswerte abgerufen werden.

Weitere Prozeduren dienen der Übersetzung von symbolischen Namen oder numerische Adressen (Strings) in binären IP-Adressen und umgekehrt.

Die Prozedur `gethostbyname` bestimmt die IP-Adresse eines symbolisch benannten Hosts. Clients benutzen häufig `gethostbyname`, um einen vom Benutzer eingegebenen Server-Namen in die entsprechende binäre IP-Adresse zu konvertieren, (die von der Protokoll-Software benötigt wird).

Die Prozedur `gethostbyaddr` konvertiert eine binäre IP-Adresse in einen String, welcher dem symbolischen Host-Namen entspricht. Clients und Server können `gethostbyaddr` benutzen, um IP-Adressen benutzerfreundlich anzuzeigen.

Die Funktion `inet_addr` wandelt einen String, der eine numerische IP-Adresse enthält, in eine binäre IP-Adresse. Diese Funktion ist ähnlich zu bekannten `atoi`, die einen String in einen Integer wandelt.

Die Funktion `inet_ntoa` konvertiert eine binäre IP-Adresse in einen String, der die numerische Darstellung der Adresse enthält.

10.6 Client-/Server-Beispiel

Im vorherigen Abschnitt wurde das Socket-API einschliesslich der verwendbaren Prozeduren und Argumente beschrieben. Im folgenden wird das Socket-API anhand von Beispielen weiter ausgeführt. Im einzelnen werden ein Client und ein Server beschrieben, die über Sockets kommunizieren. Dieses Beispiel illustriert zwar nicht alle möglichen Client- und Serverkonfigurationen, erklärt aber den Zweck der wichtigen Socket-Prozeduren. Das Beispiel zeigt eine Abfolge von Prozederaufrufen und hebt den Unterschied zwischen Aufrufen eines Clients und eines Servers hervor.

10.6.1 Verbindungsorientierte Kommunikation

Ein Client und ein Server müssen ein Transportprotokoll wählen, das einen verbindungslosen oder verbindungsorientierten Dienst unterstützt. Mit einem verbindungslosen Dienst kann eine Anwendung jederzeit Nachrichten an ein beliebiges Ziel senden. Das Ziel muss sich nicht vor der Übertragung ausdrücklich bereit erklären, die Nachrichten entgegenzunehmen. Demgegenüber müssen bei einem verbindungsorientierten Dienst zwei Anwendungen vor dem Datenaustausch eine Verbindung aufbauen. Hierfür interagieren die Anwendungen jeweils mit ihrer Protokoll-Software, und die beiden Protokollmodule tauschen Nachrichten im Netz aus. Haben sich beide Seiten über den Aufbau einer Verbindung geeinigt, können die Anwendungen Daten senden.

10.6.2 Beispieldienst

Anhand eines Beispiels mit einem Client und einem Server können viele Details der verbindungsorientierten Interaktion geklärt werden. Ausserdem lässt sich aufzeigen, wie Software für einen verbindungsorientierten Dienst Sockets nutzt. Um den Umfang der Programme in Grenzen zu halten und uns auf die Socket-Aufrufe konzentrieren zu können, wurde als Beispiel ein sehr einfacher Dienst gewählt: Der Server führt einen Zähler über die Anzahl der Clients, die auf den Dienst zugegriffen haben, und meldet den Zählerstand, sobald ein Client den Server kontaktiert.

Um auch die Implementierung und Fehlerbehandlung zu vereinfachen, basiert der Dienst auf ASCII. Ein Client baut eine Verbindung zu einem Server auf und wartet auf Ausgaben. Kommt eine Verbindungsanfrage an, erstellt ein Server eine Nachricht im druckbaren ASCII-Format, sendet die Nachricht und baut die Verbindung wieder ab. Der Client zeigt die empfangenen Daten an und wird dann beendet.

Baut ein Client beispielsweise zum zehnten Mal eine Verbindung zum Server auf, erhält er folgende Nachricht:

Dies ist die 10. Verbindung.

10.6.3 Befehlszeilenargumente für die Programmbeispiele

Unser Beispiel-Server hat ein Befehlszeilenargument – eine Protokoll-Portnummer, d.h. die Nummer des Ports, an dem er Anfragen entgegennimmt. Das Argument ist optional. Falls das Port nicht angegeben wird, wird automatisch Port 4711 benutzt. Die beiden folgenden Aufrufe sind also gleichwertig.

```
> server  
> server 4711
```

Der Beispiel-Client hat zwei Befehlszeilenargumente: den Namen eines Hosts, auf dem sich der Server befindet, und die zu benutzende Protokoll-Portnummer. Beide Argumente sind optional. Wird keines angegeben, werden automatisch „4711“ als Portnummer und „localhost“ als Hostname verwendet. Normalerweise geben Benutzer einen Hostnamen an, weil die Anbindung an einen Server auf dem lokalen Rechner hier uninteressant ist. Die Möglichkeit, den lokalen Server auf diese Weise anzusprechen, ist in bezug auf die Fehlerbehandlung allerdings nützlich.

```
> client  
> client localhost  
> client localhost 4711
```

10.6.4 Ablauf der Socket-Prozeduraufälle

Abbildung 10.2 zeigt die Abfolge der Socket-Prozeduren für unser Beispiel. Man sieht, dass der Server sieben und der Client sechs Socket-Prozeduren aktivieren muss. Bevor ein Client `connect` ausführen kann, muss der Server `listen` aktiviert haben.

Der Server aktiviert `getprotobyname`, um den Namen eines Protokolls (hier „tcp“) in die von der Socket-Prozedur benutzte interne Binärform umzuwandeln. Dann wird mit `mysocket` ein Socket erzeugt. Anschliessend bestimmt er mit `bind` einen lokalen Protokollport und versetzt den Socket mit `listen` in den passiven Zustand. Danach tritt der Server in eine Endlosschleife ein, in der er `accept` ausführt, um weitere Verbindungsanfragen zu empfangen, und `send` aufruft, um Nachrichten an den Client zu senden. Zum Beenden der Verbindung aktiviert er `close`. Nach dem Abbau der Verbindung ruft der Server `accept` auf, um die nächste Verbindungsanfrage anzunehmen.

Der Client konvertiert zunächst mit der Prozedur `gethostbyname` den Namen der Server-Hosts in eine binäre IP-Adresse. Dann wird – wie beim Server – mit `getprotobyname` der Name eines Protokolls in die von der Socket-Prozedur benutzten internen Binärform umzuwandeln. Dann ruft der Client `mysocket` auf, um ein Socket zu erzeugen, und `connect`, um den Socket mit einem Server zu verbinden. Steht die Verbindung, aktiviert der Client mehrmals `recv`, um vom Server gesendete Daten zu empfangen. Nach dem Empfang aller Daten schliesst der Client das Socket mit `close`.

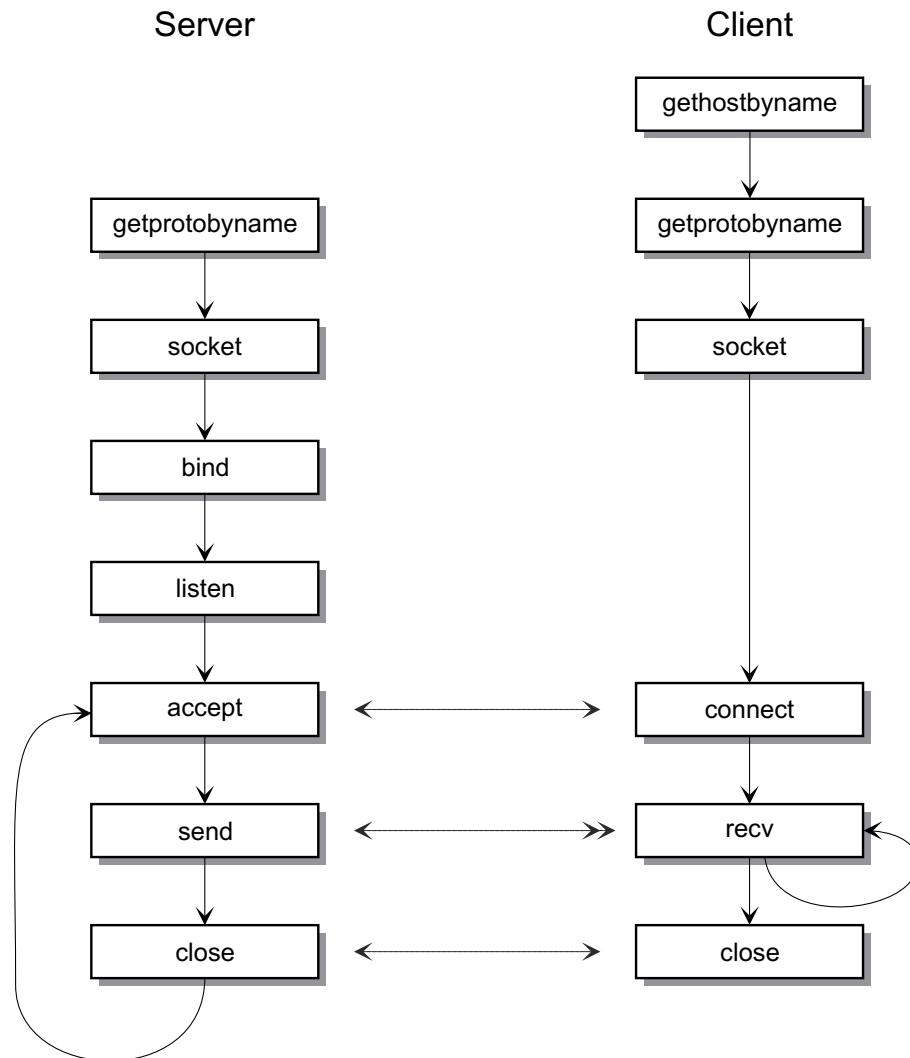


Abbildung 10.2: Abfolge der Socket-Prozeduren für unser Beispiel

10.6.5 Stream-Dienst und mehrfache `recv`-Aufrufe

Während der Server nur *einmal* `send` aufruft, um Daten zu senden, wird im Client-Code für den Empfang der Daten eine Iteration benötigt. Bei jeder Wiederholung aktiviert der Client `recv`, um Daten entgegenzunehmen. Die Wiederholung stoppt, wenn der Client das File-Ende (keine Zeichen mehr empfangen) erkennt.

In den meisten Fällen verpackt das TCP-Protokoll des Senders die gesamte Nachricht in ein einziges TCP-Segment und überträgt es in einem IP-Datagramm. Wegen der Daten-Pufferung und möglicher Fragmentierung gewährleistet TCP aber *nicht*, dass die Daten in einem Segment übertragen werden und dass jeder Aufruf von `recv` genau die gleiche Datenmenge liefert, die der Server mit einem `send` übertragen hat. TCP stellt lediglich sicher, dass die Daten verlustlos und in der richtigen *Reihenfolge* zugestellt werden. Folglich muss ein Empfangs-Programm möglicherweise mehrmals `recv` aufrufen, bis alle Daten gelesen sind.

10.6.6 Code für den Beispiel-Server

```
1  /*
2  -----
3
4  Program:    server
5
6  Zweck:      Erzeugt einen Socket und führt wiederholt aus:
7          1) Warte auf eine Verbindung von einem Client
8          2) Sende eine Meldung an den Client
9          3) Beende die Verbindung
10
11 Syntax:     server [ port ]
12             port - Protokoll-Port
13
14 Anmerkung: Die Angabe von "port" ist fakultativ. Falls nicht angegeben,
15             wird der Default-Wert "DefaultPortNumber" verwendet.
16
17 -----
18 */
19
20 /* Betriebssystem-spezifisch Header-Files einbinden */  

21 #ifndef unix  

22 # include <winsock.h>  

23 # define close closesocket  

24 #else  

25 # include <sys/types.h>  

26 # include <sys/socket.h>  

27 # include <netinet/in.h>  

28 # include <arpa/inet.h>  

29 # include <netdb.h>  

30 # include <unistd.h>  

31 #endif  

32
33 #include <stdlib.h>
34 #include <stdio.h>
35 #include <string.h>
36
37 /* Konstanten definieren */  

38 const int DefaultPortNumber = 4711;      /* Default-Protokoll-Port */  

39 const int QueueLength = 10;              /* Laenge der Request Queue */  

40
41 /* Macro um eine beliebige Datenstruktur (mittels Nullen) zu löschen */  

42 #define ClearMemory(s) memset((char*)&(s),0,sizeof(s))  

43
44 /* Prozedur zur Fehlerabfrage und Behandlung */  

45 void ExitOnError(int Status, char* Text) {  

46     if (Status < 0) {  

47         fprintf(stderr, Text);  

48         exit(1);  

49     }  

50 }
51
52 int main(int ArgumentCount, char* ArgumentValue[]) {  

53
54     typedef struct sockaddr* SockAddrPtr; /* Pointer auf sockaddr */  


```

```

55
56  /* Struktur für Server-Adresse und Pointer für Parameter-Übergabe */
57  struct sockaddr_in ServerAddr;
58  const SockAddrPtr ServerAddrPtr = (SockAddrPtr)&ServerAddr;
59
60  /* Struktur für Client-Adresse und Pointer für Parameter-Übergabe */
61  struct sockaddr_in ClientAddr;
62  const SockAddrPtr ClientAddrPtr = (SockAddrPtr)&ClientAddr;
63
64  int           ListeningSocket;    /* Socket für Verbindungsauftbau */
65  int           ConnectedSocket;   /* Socket für Datenübertragung */
66  int           UntestedPort;      /* Port Nummer (wie eingegeben) */
67  unsigned short Port;            /* Protokoll-Port-Nummer */
68  unsigned       AddrLen;          /* Laenge der Adresse */
69  char          Buffer[1000];     /* Daten-Buffer */
70  int           Status;           /* Status-Zwischenspeicher */
71  int           Visits;           /* bisherige Anzahl Verbindungen */
72
73  /* Windows-spezifische Initialisierung */
74  #ifndef unix
75      WSADATA wsaData;
76      Status = WSAStartup(0x0101, &wsaData);
77      ExitOnError(Status, "Winsock-Startup fehlgeschlagen\n");
78  #endif
79
80  /*
81  Kommandozeile verarbeiten:
82
83  Falls eine Port-Nummer angegeben wurde, soll diese für das Protokoll
84  verwendet werden, sonst der Default-Wert (Konstante DefaultPortNumber).
85  Falls der Wert ungültig ist, wird das Programm mit einer Fehlermeldung
86  beendet, sonst wird der Wert in der Variablen Port abgelegt.
87  */
88
89  if (ArgumentCount > 1) {           /* Falls Port-Parameter angegeben */
90      UntestedPort = atoi(ArgumentValue[1]);    /* String binär wandeln */
91  } else {
92      UntestedPort = DefaultPortNumber; /* sonst Default-Port verwenden */
93  }
94
95  if (UntestedPort>0 && UntestedPort<65536) { /* gueltige Port-Nummer? */
96      Port = (unsigned short) UntestedPort; /* Typ anpassen, übernehmen */
97  }
98  else {                           /* Fehlermeldung ausgeben und beenden */
99      fprintf(stderr, "Ungültige Port-Nummer %d\n", UntestedPort);
100     exit(1);
101 }
102
103 /* Socket für Verbindungsauftbau erzeugen */
104 ListeningSocket = socket(PF_INET, SOCK_STREAM, 0);
105 ExitOnError(ListeningSocket, "socket fehlgeschlagen\n");
106
107 /* Server Adresse und Port für den Dienst (Protokoll) definieren */
108 ClearMemory(ServerAddr);           /* Alles mit Nullen loeschen */
109 ServerAddr.sin_family = AF_INET;    /* Address Family InterNET */
110 ServerAddr.sin_addr.s_addr = INADDR_ANY; /* Beliebiges Interface */
111 ServerAddr.sin_port = htons(Port);   /* Port, ggf. Bytes tauschen */
112

```

```
113     /* Dem Socket die lokale Adresse und Port-Nummer zuordnen */  
114     Status = bind(ListeningSocket, ServerAddrPtr, sizeof(ServerAddr));  
115     ExitOnError(Status, "bind fehlgeschlagen\n");  
116  
117     /* Socket in passiv Modus versetzen und Warteschlagengrösse festlegen */  
118     Status = listen(ListeningSocket, QueueLength);  
119     ExitOnError(Status, "listen fehlgeschlagen\n");  
120  
121     Visits = 0; /* Noch keine Verbindungen */  
122     printf("Server wartet an Port %d auf die erste Verbindung\n", Port);  
123  
124     while (1) { /* Server Loop */  
125  
126         AddrLen = sizeof(ClientAddr); /* ... wird von accept verändert */  
127  
128         /* Auf Client-Verbindung (connect) warten */  
129         ConnectedSocket = accept(ListeningSocket, ClientAddrPtr, &AddrLen);  
130         ExitOnError(ConnectedSocket, "accept fehlgeschlagen\n");  
131  
132         Visits++;  
133         printf("%d. Verbindung von %s, Port %d\n",  
134             Visits, inet_ntoa(ClientAddr.sin_addr), ClientAddr.sin_port);  
135  
136         /* Daten-Buffer aufbereiten */  
137         sprintf(Buffer, "Dies ist die %d. Verbindung.\n", Visits);  
138  
139         /* Daten-Buffer senden */  
140         Status = send(ConnectedSocket, Buffer, strlen(Buffer), 0);  
141         ExitOnError(Status, "send fehlgeschlagen\n");  
142  
143         /* Client-Verbindung beenden */  
144         Status = close(ConnectedSocket);  
145         ExitOnError(Status, "close fehlgeschlagen\n");  
146  
147     } /* Server Loop */  
148 }
```

..... **Programm 10.1**

10.6.7 Code für den Beispiel-Client

```

1  /*
2  -----
3  Program:   client
4
5  Zweck:      Erstellt eine Verbindung zu einem Server und zeigt die
6              vom Server gesendeten Daten an.
7
8  Syntax:     client [ server [port] ]
9
10         server - IP-Adresse des Servers
11         port   - Protokoll-Port-Nummer des Servers
12
13 Anmerkung: Beide Parameter sind fakultativ.
14         - Falls "server" fehlt, wird "LocalHost" angenommen
15         - Falls "port" fehlt, wird der Default-Wert
16             "DefaultPortNumber" verwendet.
17
18 -----
19 */
20
21 /* Betriebssystem-spezifisch Header-Files einbinden */
22 #ifndef unix
23 # include <windows.h>
24 # include <winsock.h>
25 #define close closesocket
26 #else
27 # include <sys/types.h>
28 # include <sys/socket.h>
29 # include <netinet/in.h>
30 # include <arpa/inet.h>
31 # include <netdb.h>
32 # include <unistd.h>
33 #endif
34 #include <stdlib.h>
35 #include <stdio.h>
36 #include <string.h>
37
38 /* Konstanten definieren */ / /
39 const int DefaultPortNumber = 4711;    /* Default-Protokoll-Port */ /
40 const char LocalHost[] = "localhost"; /* Default-Server-Name */ /
41
42 /* Macro um eine beliebige Datenstruktur (mittels Nullen) zu löschen */ /
43 #define ClearMemory(s) memset((char*)&(s), 0, sizeof(s))
44
45 /* Prozedur zur Fehlerabfrage und Behandlung */ /
46 void ExitOnError(int Status, char* Text) {
47     if (Status < 0) {
48         fprintf(stderr, Text);
49         exit(1);
50     }
51 }
52
53 int main(int ArgumentCount, char* ArgumentValue[]) {
54

```

```

55     typedef struct sockaddr* SockAddrPtr; /* Pointer auf sockaddr
56
57     /* Struktur für Server-Adresse und Pointer für Parameter-Übergabe */
58     struct sockaddr_in ServerAddr;
59     const SockAddrPtr ServerAddrPtr=(SockAddrPtr)&ServerAddr;
60
61     const char*      ServerName;           /* Temp.Pointer auf Server-Namen */
62     struct hostent*  HostEntryPtr;        /* Rückgabewert von gethostbyname:
63                                         Zeiger auf statische Struktur */
64     unsigned long    BinaryAddr;          /* Binäre Server-Adresse */
65     int              CommunicationSocket; /* Socket (-Descriptor) */
66     int              UntestedPort;        /* Port-Nummer (wie eingegeben) */
67     unsigned short   Port;               /* Protokoll-Port-Nummer */
68     char             Buffer[1000];        /* Daten-Buffer */
69     int              Status;             /* Status-Zwischenspeicher */
70     int              CharsReceived;      /* Anzahl empfangener Zeichen */
71
72     /* Windows-spezifische Initialisierung */
73 #ifndef unix
74     WSADATA wsaData;
75     Status = WSAStartup(0x0101, &wsaData);
76     ExitOnError(Status, "Winsock-Startup fehlgeschlagen\n");
77 #endif
78
79     /*
80      1. Parameter der Kommandozeile verarbeiten:
81
82      Falls eine Server-Adresse angegeben wurde, soll diese verwendet werden,
83      sonst der Default-Wert (Konstante "LocalHost").
84     */
85
86     if (ArgumentCount > 1) {           /* Falls Server-Argument angegeben */
87         ServerName = ArgumentValue[1];
88     } else {
89         ServerName = LocalHost;
90     }
91
92     /*
93      Es wird versucht die numerische oder symbolische angegebene Server-
94      Adresse in eine binäre Adresse umzuwandeln. Falls dies nicht gelingt,
95      wird das Programm mit einer Fehlermeldung beendet, sonst wird der
96      Wert in der Variablen "BinaryAddr" abgelegt.
97      Anmerkung: Unter Unix genügt gethostbyname (inet_addr ist unnötig)
98
99     /* 1.Versuch: Numerische IP-Adresse?
100      Anmerkung: Unter Unix genügt gethostbyname (inet_addr ist unnötig) */
101     BinaryAddr = inet_addr(ServerName);
102     if (BinaryAddr == INADDR_NONE) {
103
104     /* 2.Versuch: Symbolische IP-Adresse?
105     HostEntryPtr = gethostbyname(ServerName);
106     if (HostEntryPtr != NULL) {
107         memcpy(&BinaryAddr, HostEntryPtr->h_addr, sizeof(BinaryAddr));
108     }
109     else {
110         fprintf(stderr, "Ungültiger Server-Adresse: %s\n", ServerName);
111         exit(1);
112     }

```

```

113     }
114
115     /*
116 2. Parameter der Kommandozeile verarbeiten:
117
118 Falls eine Port-Nummer angegeben wurde, soll diese für das Protokoll
119 verwendet werden, sonst der Default-Wert (Konstante DefaultPortNumber).
120 Falls der Wert ungültig ist, wird das Programm mit einer Fehlermeldung
121 beendet, sonst wird der Wert in der Variablen Port abgelegt.
122 */
123
124 if (ArgumentCount > 2) { /* Falls Port-Parameter angegeben */
125     UntestedPort = atoi(ArgumentValue[2]); /* String binär wandeln */
126 } else {
127     UntestedPort = DefaultPortNumber; /* Default verwenden */
128 }
129
130 if (UntestedPort>0 && UntestedPort<65536) { /* Port-Nummer gültig ? */
131     Port = (unsigned short)UntestedPort; /* Typ anpassen, übernehmen */
132 }
133 else { /* Fehlermeldung und Ende */
134     fprintf(stderr, "Ungültige Port-Nummer %d\n", UntestedPort);
135     exit(1);
136 }
137
138 /* Socket für Verbindungsaufbau und Datentransfer erzeugen */
139 CommunicationSocket = socket(PF_INET, SOCK_STREAM, 0);
140 ExitOnError(CommunicationSocket, "socket fehlgeschlagen\n");
141
142 /* Server-Adresse und Port (Protokoll) definieren */
143 ClearMemory(&ServerAddr); /* Alles mit Nullen loeschen */
144 ServerAddr.sin_family = AF_INET; /* Address Family InterNET */
145 ServerAddr.sin_addr.s_addr = BinaryAddr; /* Binary Server-Adresse */
146 ServerAddr.sin_port = htons(Port); /* Port, ggf. Bytes tauschen */
147
148 /* Verbindung zum Server und Dienst erstellen */
149 Status=connect(CommunicationSocket, &ServerAddr, sizeof(ServerAddr));
150 ExitOnError(Status, "connect fehlgeschlagen\n");
151
152 /* Wiederholt Daten vom Server lesen und am Bildschirm anzeigen */
153 CharsReceived = recv(CommunicationSocket, Buffer, sizeof(Buffer), 0);
154 while (CharsReceived > 0) {
155     Buffer[CharsReceived] = 0;
156     fprintf(stdout,"%s", Buffer);
157     CharsReceived = recv(CommunicationSocket, Buffer, sizeof(Buffer), 0);
158 }
159
160 /* Close the socket */
161 close(CommunicationSocket);
162 ExitOnError(Status, "close fehlgeschlagen\n");
163
164 /* Programm mit positivem Status beenden */
165 exit(0);
166 }
167
..... Programm 10.2 .....
```

10.6.8 Socket-Prozeduren und Blockierung

Wie die meisten Ein-/Ausgabeaufrufe sind auch die meisten Prozeduren im Socket-API synchron und damit blockierend. Das heisst, wenn ein Programm eine Socket-Prozedur aktiviert, wird das Programm so lange angehalten, bis die Prozedur abgeschlossen ist. Für diese Unterbrechung gibt es keine zeitliche Grenze – sie kann beliebig lange dauern.

Aus welchem Grund benutzen Clients und Server blockierende Prozeduren? Wir betrachten dies zuerst aus der Sicht des Servers. Nachdem der Server ein Socket erzeugt, es mit einem Protokollport gebunden und in den passiven Zustand versetzt hat, ruft er `accept` auf. Hat ein Client bereits mit `connect` eine Verbindung angefordert, bevor der Server `accept` aufruft, wird der Aufruf sofort zurückgegeben. Gelangt der Server vor der Verbindungsanfrage eines Clients an den `accept`-Aufruf, wird der Server so lange angehalten, bis eine Anfrage ankommt.

Im allgemeinen verweilen Server-Programme die meiste Zeit am `accept`-Aufruf. Da das Betriebssystem den Server-Prozess blockiert, kann der Prozessor in dieser Zeit für andere Aufgaben (andere Prozesse) verwendet werden.

Im Client-Code können Aufrufe von Socket-Prozeduren ebenfalls blockieren. Bei manchen Implementierungen der Bibliotheksprozedur `gethostbyname` wird beispielsweise eine Nachricht an einen Server gesendet, dann wird auf eine Antwort gewartet. Der Client wird in diesem Fall so lange angehalten, bis eine Antwort eingeht. Ähnlich erfolgt beim Aufruf von `connect` eine Blockierung, bis TCP das Dreiweg-Handshake ausführen kann, um eine Verbindung aufzubauen.

Die wahrscheinlich grösste Unterbrechung tritt während der Datenübertragung auf. Nach dem Aufbau der Verbindung ruft der Client `recv` auf. Gehen über die Verbindung keine Daten ein, blockiert der Aufruf. Stehen beim Server Verbindungsanfragen in der Warteschlange, wird der Client so lange blockiert, bis der Server Daten an ihn sendet.

10.6.9 Codeumfang und Fehlerkontrolle

Die Programmbeispiele sind zwar recht umfangreich, der Grossteil des Codes besteht aber aus Kommentaren. Entfernt man die Leerzeilen und Kommentare, reduziert sich der Code um über 40%.

In vielen Codezeilen wird auf Fehler geprüft. Zusätzlich zur Überprüfung der von Befehlszeilenargumenten vorgegebenen Werte werden im Code auch die Rückgabewerte von den einzelnen Prozeduraufrufen geprüft, um den erfolgreichen Ablauf sicherzustellen. Fehler werden nicht erwartet, tritt aber einer auf, hat der Programmierer das Programm so vorbereitet, dass eine kurze Fehlermeldung ausgegeben und das Programm beendet wird. Ein Anteil von etwa 15% am Code entfällt auf die Fehlerkontrolle.

10.6.10 Test der Kommunikation

Sowohl der Client als auch der Server des Programmbeispiels kann mit anderen Diensten benutzt werden. Benutzt man beispielsweise den Client mit einem anderen Dienst, erhält man eine einfache Möglichkeit, einen Test des Client-Programms durchzuführen, bevor das Server-Programm geschrieben wird.

a) Benutzung des Beispiel-Clients mit einem anderen Dienst

TCP/IP definiert den Dienst DAYTIME, der das aktuelle Datum und die Uhrzeit ausgibt. Dieser Dienst arbeitet gleich wie unser Beispiel: Ein Client stellt eine Verbindung zu einem DAYTIME-Server her, empfängt von diesem Daten und gibt diese aus.

Um den Beispiel-Client mit dem DAYTIME-Dienst zu benutzen, muss das Client-Programm mit zwei Argumenten aktiviert werden – eines für einen Host, auf dem der DAYTIME-Server läuft, und eines für die Protokoll-Portnummer des DAYTIME-Dienstes, d.h. 13. Wird der Client-Code beispielsweise kompiliert und das Ergebnis in einer ausführbaren Datei namens `client` abgelegt, kann er zum Aufschalten auf den DAYTIME-Dienst von Computern rund um die Welt benutzt werden:

```
> client localhost 13
Fri May 7 09:52:00 1999
> client charly.zhwin.ch 13
Fri May 7 09:55:27 1999
```

Bei diesem Beispiel wird die Zeit von zwei Computern ausgegeben. Die Ausgabe wurde durch mehrmaliges Ausführen des Clients erzeugt. Die erste Ausgabe stammt von dem Computer, auf dem der Client läuft. Die zweite Ausgabe stammt von SE-Server.

b) Server mit einem anderen Client testen

Der Beispiel-Server kann auch getrennt vom Client getestet werden. Hierfür kann man den Server über das Client-Programm `telnet` kontaktieren. Das Telnet-Programm erfordert zwei Argumente: den Namen des Computers, auf dem der Server läuft, und die Protokoll-Portnummer des Servers. Folgende Ausgabe wurde über Telnet zum Beispielserver gewonnen:

```
> telnet localhost 4711
Trying 127.0.0.1 ...
Connected to localhost.
Escape character is '^]'.
Dies ist die 2. Verbindung.
Connection closed by foreign host.
```

Nur die fünfte der sechs Zeilen dieser Ausgabe wurde vom Server ausgegeben; die übrigen stammen vom Telnet-Programm.

10.7 Sockets und Prozesse

Da viele Server im Parallelbetrieb arbeiten, wurde das Socket-API für diese Betriebsweise ausgelegt. Bei Implementierungen des Socket-API wird im allgemeinen folgendes Prinzip eingehalten, obwohl die Details vom zugrundeliegenden Betriebssystem abhängen:

Grundsatz: Jeder neue Subprozess (oder Thread) erbt eine Kopie aller offenen Sockets von dem Prozess, der ihn erzeugt hat.

Um diese Socket-Vererbung zu verstehen, muss man wissen, dass Sockets einen Referenzähler (Reference Count) benutzen. Dieser Mechanismus funktioniert wie folgt: Beim erstmaligen Erstellen eines Socket setzt das System den Referenzähler des Socket auf 1. Solange der Referenzähler positiv ist, existiert der Socket. Erstellt ein Programm einen Subprozess (Kindprozess), stellt das System eine Liste aller Sockets zusammen, die das Programm besitzt, und erhöht den Referenzähler der Sockets jeweils um 1. Führt ein Prozess `close` für einen Socket aus, senkt das System den Referenzähler des Socket um 1 und entfernt den Socket von der Liste des Prozess.

Der Hauptprozess (Vater-Prozess) eines parallel laufenden Servers erzeugt einen Socket, den der Server zur Annahme ankommender Verbindungen benutzt. Kommt ein Verbindungswunsch an, erzeugt das System einen neuen Socket für diese Verbindung. Unmittelbar danach haben der Vaterprozess und der für die neue Verbindung erzeugte Subprozess Zugriff auf die neuen und alten Sockets. Die Referenzähler der Sockets haben dann den Wert 2. Der Vaterprozess benutzt aber nicht den neuen Socket und der neue Subprozess nicht den Original-Socket. Deshalb aktiviert der Vaterprozess `close` für den neuen Socket, während der neue Subprozess `close` für den Original-Socket ausführt, so dass sich der Referenzähler der beiden um 1 reduziert.

Ist der Subprozess fertig, führt er `close` für den neuen Socket aus, wodurch der Referenzähler auf Null fällt und der Socket gelöscht wird. Die Lebensdauer eines Socket beim parallelen Serverbetrieb ist also wie folgt:

Grundsatz: Ein Socket, den ein parallel betriebener Server für die Annahme von Verbindungen benutzt, existiert für die Dauer der Ausführung des Vaterprozess. Ein für eine bestimmte Verbindung benutzter Socket existiert nur für die Lebensdauer des Subprozess, der die Verbindung handhabt.

10.7.1 Unix-Beispiel für einen parallelbetriebenen Server

Das folgende Beispiel basiert auf dem Server-Code vom Abschnitt 10.6.6. Neu wird im Server-Loop zur Behandlung jeder Verbindung ein Subprozess erzeugt. Die Unix-Funktion `fork()` erzeugt eine identische Kopie des laufenden Prozesses. Die Prozesse können nur anhand des Rückgabewerts von `fork()` unterschieden werden. Während der Rückgabewerts im Kindprozess Null ist, entspricht sie beim Vaterprozess der Identifikationsnummer des Kindprozesses.

Da `fork()` Unix-spezifisch ist, kann der folgende Code auf Windows nicht übersetzt werden. Auch ist das Programm zwar lauffähig, aber nicht ganz vollständig. Interessierte seien auf das Praktikum verwiesen.

```

121 Visits = 0; /* Noch keine Verbindungen */
122 printf("Server wartet an Port %d auf die erste Verbindung\n", Port);
123
124 while (1) { /* Server Loop */
125
126     AddrLen = sizeof(ClientAddr);      /* ... wird von accept verändert */
127
128     /* Auf Client-Verbindung (connect) warten */  

129     ConnectedSocket = accept(ListeningSocket, ClientAddrPtr, &AddrLen);
130     ExitOnError(ConnectedSocket, "accept fehlgeschlagen\n");
131
132     Visits++;
133     printf("%d. Verbindung von %s, Port %d\n",
134            Visits, inet_ntoa(ClientAddr.sin_addr), ClientAddr.sin_port);
135
136     /* Behandlung der neuen Verbindung in einem Subprozess */  

137     NewProcessID = fork();
138     if (NewProcessID < 0) ExitOnError(-1, "fork fehlgeschlagen\n");
139
140     /* Nun muss anhand der NewProcessID unterschieden werden, ob das
141        Programm im Context des Vater- oder des Subprozesses läuft. */
142
143     if (NewProcessID > 0){           /* Dies ist der Vater-Prozess */
144
145         printf("Neuen Subprocess erzeugt: pid = %d\n", NewProcessID);
146
147     /* Referenzzaehler des Connected-Sockets dekrementieren */  

148     Status = close(ConnectedSocket);
149     ExitOnError(Status, "close fehlgeschlagen\n");
150     }
151     else {                         /* Dies ist der Kind-Prozess */
152
153     /* Referenzzaehler des Listening-Sockets dekrementieren */  

154     Status = close(ListeningSocket);
155     ExitOnError(Status, "close fehlgeschlagen\n");
156
157     /* Daten-Buffer aufbereiten */  

158     sprintf(Buffer,"Dies ist die %d. Verbindung.\n",Visits);
159
160     usleep(10000000);             /* Aktivität simulieren: 10s warten */
161
162     /* Daten-Buffer senden */  

163     Status = send(ConnectedSocket, Buffer, strlen(Buffer), 0);
164     ExitOnError(Status, "send fehlgeschlagen\n");
165
166     /* Client-Verbindung beenden; socket wird nun freigegeben */  

167     Status = close(ConnectedSocket);
168     ExitOnError(Status, "close fehlgeschlagen\n");
169
170     /* Subprozess mit positivem Status beenden */  

171     exit(0);
172     }
173
174 } /* Server Loop */

```

Index

- Übersprechen, 2-6
- Übertragung
 - parallele, 3-2
 - serielle asynchrone, 3-2
 - serielle synchrone, 3-2–3-4
- Übertragungsverfahren, 3-2
- 3 Way Handshake, 8-16
- Acknowledgment, 8-9
- Address Resolution, 7-27
- Address Resolution Protokoll, 7-29
- Adressauflösung, 7-27
- Adresstabelle, 6-11
- Alias, 9-3
- Alternate Mark Inversion, 3-6
- AMI, 3-6
- API, 10-1
 - Application Program Interface, 10-1
- ARP, 7-29
 - ARP-Cache, 7-31
 - asynchrone Übertragung, 3-2
 - Ausbreitungsgeschwindigkeit, 2-2
 - Autonegotiation, 6-17
- Bandbreite, 3-8
- Berkeley Internet Name Domain, 9-6
- BIND, 9-6
- Bit Rate, 6-11
- Bitübertragungsschicht, 3-1
- Bitstopfen, 4-4
- BootP, 9-10
- Bootstrap Protocol, 9-10
- Bridge, 6-1
 - Multiport-, 6-6
 - Remote-, 6-5
- Bridging Router, 7-6
- Broadcast Domain, 6-8
- Broadcast-Übertragung, 5-4
- Broadcast-Adresse, 7-11
- Brouter, 7-6
- Brutto bitrate, 4-5
- Buffergrösse, 6-11
- Checksum, 4-11
- Classful-Routing, 7-10, 7-12
- Classless-Routing, 7-10
- Codierverfahren, 3-5
- Collision Domain, 5-18, 6-1
- Congestion Control, 8-21
- Congestion Window, 8-21
- CRC, 4-11
- Crosstalk, 2-6
- CSMA-Verfahren, 4-20
- Cut-Through-Bridges, 6-12
- Cut-Through-Switching, 6-12
- Cyclic Redundancy Check, 4-11
- Daemon, 9-4
- Data Overrun, 8-13
- Default-Routing, 7-18
- Demultiplexen, 8-2
- Descriptor, 10-3
- Direkte Berechnung, 7-28
- DNS, 9-1, 9-2
 - DNS-Server, 9-1
 - Domänen, 9-2
 - Domain Name Space, 9-2
 - Domain Name System, 9-1
- Dreiweg-Handshaking, 8-16
- Extra Carrier Extension, 6-25
- Fast-Link Pulse, 6-17
- Fehlermeldungen, 7-36
- Fehlerwahrscheinlichkeit, 4-6
- FIN-Segment, 8-20
- Flaches Routing, 7-17
- Flags
 - Ende-, 4-3
 - Start-, 4-3
- Flow Control, 4-16, 8-13
- FLP, 6-17
- Fluss-Steuerung, 8-13
- Flusssteuerung, 4-16
- Forwarding, 7-15
- Forwarding Database, 6-3
- Forwarding Rate, 6-12
- Fragmentation, 7-22
- Fragmente, 7-22
- Fragmentierung, 7-22

- Frame Bursting, 6-25
Frame-Länge
 optimale, 4-6
Framing, 4-2
Fully Qualified Domain Name, 9-2
- Gateway, 7-1
Gleichspannungsfreiheit, 3-6
- Halbduplex, 3-2
half closed, 8-20
Hamming-Distanz
 Definition, 4-8
 Fehlererkennung, 4-8
Hauptdomäne, 9-2
Hierarchisches Routing, 7-18
Host, 7-8
Host Byte Order, 10-5
HTTP-Version, 9-31
Hub, 5-3, 5-17
- ICMP, 7-35
IMAP, 9-19
Informationsmeldungen, 7-37
Initial Request Line, 9-30
Internet, 7-1
Internet Control Message Protocol, 7-35
Internet Message Access Protocol, RFC 2060, 9-19
Interpacket Gap Shrinkage, 5-18
isochrone Daten, 4-5
- Jam-Signal, 5-12
Jitter, 4-5
- kanonische Domänenname, 9-3
Koaxialkabel, 2-4
- LAN, 5-1
Late Collision, 5-19
Latency, 6-12
Layer-3-Switches, 7-6
Leitungscode
 Alternate Mark Inversion, 3-6
 AMI, 3-6
Leitungscode, 3-5
Length, 5-16
Local Area Network, 5-1
- Manchester-Code, 5-10
- Master-Name-Server, 9-4
Master/Slave-Verfahren, 4-18
maximale Übertragungseinheit, 7-21
Maximum Segment Lifetime, 8-21
Maximum Transfer Unit, 7-21
MDI, 6-16
MDI-X, 6-14
Media Independent Interface, 6-17
Medium Dependent Interface, 6-16
MII, 6-17
mikrosegmentierten LAN, 6-5
MIME, 9-24
MSL, 8-21
MTU, 7-21
Multi-homed Host, 7-8
multi-homed Server, 9-28
Multicast-Übertragung, 5-4
Multiport-Remote-Bridge, 6-6
Multipurpose Internet Mail Extensions, 9-24
- Name-Cache, 9-6
named, 9-4
Nebensprechen, 2-6
Nettobitrate, 4-5
Network Byte Order, 10-5
Netzklassen, 7-10, 7-12
Netzwerk-Adresse, 7-11
NLP, 6-18
Normal-Link Pulses, 6-18
- On-the-Fly-Switching, 6-12
Organizationally Unique Identifier, 5-15
OUI, 5-15
- Physical Layer, 3-1
POP3, 9-19
Post Office Protocol, RFC 1939, 9-19
Prüfsumme, 4-11
Priorisierung, 6-8
- Quality of Service, 7-19
- Rapid Spanning Tree Protocol, 6-10
RARP, 7-34
Reason Phrase, 9-31
Reassembly, 7-23
Reconciliation Layer, 6-17
Reconciliation Sublayer, 6-17
Redundanz, 9-5
Remote-Bridge, 6-5

- Remote-Switch, 6-6
Resolver, 9-5
Response Status Code, 9-31
Ressourcen, 9-28
Retransmission, 8-9
Retransmission Time-Out, 8-10
Reverse Address Resolution Protocol, 7-34
Root, 9-2
Root Domain, 9-2
Root-Port, 6-10
Round Trip Delay, 5-17
Round-Trip Delay, 8-10
Round-Trip Time, 8-10
Routed Protocol, 7-5
Router, 7-1
Routing Protocol, 7-5
Routing-Tabelle, 7-15
RSTP, 6-10

Second Level Domain, 9-3
Segment, 8-2
Sekundäre DNS-Server, 9-6
serielle Übertragung, 3-2
Shielded Twisted Pair, 2-5
Simplex, 3-2
Slave-Name-Server, 9-4
Sliding Window, 8-13
Slow Start, 8-21
Spanning Tree Protocol, 6-9
Startbit, 3-3
Stop-and-Wait-Verfahren, 8-13
Stopbit, 3-3
Store-and-Forward-Bridges, 6-12
STP, 2-5
Stub-Resolver, 9-6
Subdomänen, 9-2
Subnetzmaske, 7-10
Switch, 5-3, 6-1, 6-6
Switches
 Layer-3, 7-6
synchrone Übertragung, 3-4

Tabellengesteuerte Übersetzung, 7-28
Taktrückgewinnung, 3-7
TCP, 8-2
ternäres Signal, 3-6
TFTP, 9-10
Time-to-Live-Wert, 9-6
TLD, 9-3

Token Passing, 4-18
Top Level Domain, 9-3
Topologie
 Baum-, 5-1, 5-4
 Bus-, 5-1
 Linien-, 5-1
 Ring-, 5-1, 5-2
 Stern-, 5-1, 5-3
Transceiver, 5-9
Transparent Bridging, 6-2
Transport Protocol Packet, 8-2
Trivial File Transfer Protocol, 9-10
Type, 5-16

UDP, 8-2
Umlaufverzögerung, 8-10
Unicast-Übertragung, 5-4
Universally Administered Address, 5-15
University of California at Berkeley, 10-2
Unshielded Twisted Pair, 2-5
User Datagramm, 8-2
UTP, 2-5

verbindungsorientiertes Transportprotokoll, 10-9
virtuell, 8-8
virtuelle LAN, 6-8
virtuelles Netz, 7-1
VLAN-Tag, 6-8
Vollduplex, 3-2, 6-5

Weiterleitungsverzögerung, 6-12
Window Advertisement, 8-15
Wurzel, 9-2

Zero Window, 8-15