Netzwerke

Stephan Euler
FH-Giessen–Friedberg
Fachbereich MND

Version 2.95, September 2006

Dieses Skript wurde mit LaTeX 2_ε und TeX (Version 3.141592) geschrieben. Eingesetzt wurde die integrierte Benutzeroberfläche WinEdt 5.3 zusammen mit MiKTeX Version 2.2. Die Bilder wurden mit GNUPLOT für MS Windows, Version 3.7 und jPicEdt 1.3.2 erstellt.

Inhaltsverzeichnis

1 Einführung **1**
- 1.1 Einleitung . . . 1
- 1.2 Übersicht . . . 2
- 1.3 Übungen . . . 3

2 Grundlagen **5**
- 2.1 Vermittlungsverfahren . . . 5
- 2.2 Leistungsfähigkeit . . . 6
  - 2.2.1 Bedeutung von Bandbreite und Latenz . . . 8
  - 2.2.2 Verzögerung–Bandbreite–Produkt . . . 8
  - 2.2.3 Anwendungen . . . 9
  - 2.2.4 Nutzung . . . 10
- 2.3 Netzwerktypen und –topologien . . . 11
  - 2.3.1 Größe . . . 11
  - 2.3.2 Topologien . . . 12
- 2.4 Referenzmodelle . . . 14
  - 2.4.1 Protokollstapel . . . 14
  - 2.4.2 OSI Referenzmodell . . . 16
  - 2.4.3 Bedeutung des OSI Referenzmodells . . . 18
- 2.5 Übungen . . . 19

3 Rechner zu Rechner Verbindung **21**
- 3.1 Übertragung von Bits . . . 21
- 3.2 Rahmenerstellung . . . 23
  - 3.2.1 Markierungszeichen . . . 24
  - 3.2.2 Zeichenzähler . . . 24
- 3.3 Fehlererkennung . . . 25
  - 3.3.1 Parität . . . 26
  - 3.3.2 Zyklische Redundanzprüfung . . . 28
- 3.4 Sichere Übertragung von Rahmen . . . 30
  - 3.4.1 Stop–and–Wait Algorithmus . . . 31
  - 3.4.2 Sliding–Window Algorithmus . . . 32
- 3.5 Übungen . . . 33

4 Ethernet & Co **39**
4.1 Ethernet . . . 39
4.1.1 Adressen . . . 40
4.1.2 Rahmenformat . . . 40
4.1.3 Medienzugriff . . . 41
4.1.4 Physikalische Eigenschaften . . . 42
4.1.5 Bewertung . . . 43
4.2 Token Ring . . . 44
4.2.1 Medienzugriff . . . 44
4.2.2 Netz–Überwachung . . . 45
4.3 Drahtlose LAN . . . 46
4.4 Andere Netztechnologien . . . 48
4.5 Übungen . . . 50

5 Vermittlung **51**
5.1 Datagramme . . . 53
5.1.1 Lernende Bridges . . . 53
5.2 Virtuelle Verbindungen . . . 55
5.3 Design von Switches . . . 56
5.3.1 Weiterleitung . . . 57
5.3.2 Knockout-Switch . . . 58
5.3.3 Batcher- und Banyan-Netzwerke . . . 58
5.4 Übungen . . . 59

6 Internet Protokoll IP **61**
6.1 Einleitung . . . 61
6.2 Adressen . . . 61
6.3 Paketformat . . . 64
6.4 Weiterleitung . . . 66
6.5 Zuordnung IP-Adresse zu Ethernet-Adresse . . . 67
6.6 Internet Control Message Protocol . . . 68
6.7 Routing – eine kurze Einleitung . . . 69
6.8 Distanzvektor-Routing . . . 70
6.9 Interdomain Routing . . . 73
6.10 Domain Name System DNS . . . 73
6.11 Internet-Standards . . . 74
6.12 Übungen . . . 76

7 UDP und TCP **77**
7.1 Einleitung . . . 77
7.2 UDP . . . 78
7.3 TCP . . . 79
7.3.1 Segmentierung . . . 80

7.3.2 Verbindungsaufbau 81
7.3.3 Sliding Window 82
7.3.4 Überlastkontrolle 83
7.4 Übungen 85

8 ATM **87**
8.1 Einleitung 87
8.2 Grundlagen 88
8.3 Zellenformat 89
8.3.1 Größe 89
8.3.2 Header 90
8.4 ATM-Anpassungsschicht 92
8.5 Verbindung LAN mit ATM 93
8.6 Übungen 94

9 Sockets **95**
9.1 Einleitung 95
9.2 Verbindungs-orientierte Kommunikation 96
9.2.1 Funktion socket 96
9.2.2 Funktion bind 97
9.2.3 Funktion listen 99
9.2.4 Funktion accept 99
9.2.5 Funktion send und recv 100
9.2.6 Einfacher Server 100
9.2.7 Funktion connect 102
9.3 Verbindungslose Kommunikation 103
9.4 Realisierung in Perl 107
9.5 Übungen 108

10 Remote Procedure Call RPC **109**
10.1 Einleitung 109
10.2 Grundlagen 110
10.3 Beispiel 111
10.3.1 Server 112
10.3.2 Client 114
10.3.3 Compilieren und Linken 116
10.3.4 Abfrage-Prozedur 117
10.4 Übungen 119

11 Anwendungen **121**
11.1 Einleitung 121
11.2 WWW 121
11.2.1 HTTP 122

11.2.2 Universal Resource Identifier . . . 124
11.2.3 Cache . . . 125
11.3 Web-Anwendungen . . . 126
11.4 email . . . 127
11.4.1 SMTP . . . 128
11.4.2 Nachrichtenformat und MIME . . . 129
11.4.3 Adressierung . . . 131
11.5 Usenet . . . 131
11.6 Netzwerkmanagement . . . 133
11.7 Multimedia-Kommunikation . . . 133
11.7.1 Real-Time Transport Protocol . . . 134
11.7.2 Verbindungsaufbau . . . 135
11.7.3 Sprachübertragung über IP . . . 135
11.7.4 Sprachcodierung . . . 136
11.7.5 Telefon-Anwendungen . . . 138
11.8 Übungen . . . 139

12 Java **141**
12.1 Einleitung . . . 141
12.2 Socket . . . 141
12.2.1 Client-Socket . . . 141
12.2.2 Server-Sockets . . . 142
12.2.3 UDP-Socket . . . 143
12.2.4 Die Klasse URL . . . 148
12.2.5 Mailto-Links . . . 150
12.3 Übungen . . . 151

13 XML **153**
13.1 Einleitung . . . 153
13.2 Grundlagen . . . 154
13.3 Transformation . . . 155
13.3.1 Cascading Style Sheets . . . 157
13.3.2 XSLT . . . 158
13.4 Grammatik . . . 162
13.4.1 Dokumenttypdefinition . . . 163
13.4.2 XML-Schema Definition . . . 164
13.5 Programmier-Schnittstellen . . . 167
13.5.1 DOM . . . 167
13.5.2 SAX . . . 170
13.5.3 Vergleich DOM und SAX . . . 173
13.6 Übungen . . . 174

14 Web-Services **177**
14.1 Einleitung . . . 177
14.2 Nachrichtenformat . . . 178
14.3 Beispiel mit SOAPLite . . . 178
14.3.1 Grundstruktur . . . 178
14.3.2 Erweiterungen . . . 180
14.4 Java-Client . . . 183
14.5 Web Service Description Language . . . 184
14.6 Beispiel Google . . . 189

15 Datensicherheit und Verschlüsselung **191**
15.1 Einleitung . . . 191
15.2 Firewall . . . 191
15.3 Virtuelle private Netze . . . 193
15.4 Verschlüsselung . . . 194
15.4.1 Einleitung . . . 194
15.4.2 Monoalphabetisch Verschlüsselung . . . 195
15.4.3 Digitale Verschlüsselung . . . 197
15.4.4 Gemeinsame Schlüsselvereinbarung . . . 198
15.4.5 Öffentliche Schlüssel . . . 201
15.5 Anwendungen . . . 203
15.5.1 PGP . . . 203
15.5.2 Sichere Transportschicht . . . 203
15.6 Übungen . . . 204

A RSA im Detail **205**
A.1 Mathematische Grundlagen . . . 205
A.2 Implementierung . . . 208

B Endliche Automaten und Maschinen **211**
B.1 Einleitung . . . 211
B.2 Endliche Automaten . . . 211
B.2.1 ε-Übergänge . . . 215
B.2.2 Endliche Maschinen . . . 215

C Ressourcen **217**
C.1 Tools . . . 217
C.1.1 Ethereal . . . 217
C.1.2 Snort . . . 217
C.1.3 OpenH323 Gatekeeper - The GNU Gatekeeper . . . 217
C.2 Nützliche Links . . . 217

D Abkürzungen **219**

E Ausgewählte Lösungen **227**

Literaturverzeichnis **232**

Kapitel 1

Einführung

1.1 Einleitung

Netzwerke dienen der Kommunikation. Netzwerke überbrücken dabei eine mehr oder weniger grosse Distanz zwischen den Teilnehmern. Betrachten wir zunächst den einfachen Fall einer Kommunikation zwischen nur zwei Teilnehmern. Dabei sei zu einem Zeitpunkt einer der Teilnehmer der Sender (Quelle) und der andere Teilnehmer der Empfänger (Ziel) der Nachricht. Diese Rollen – Sender und Empfänger – sind nicht für alle Zeit fest sondern werden im Allgemeinen wechseln. Der Sender formuliert die Nachricht und schickt sie dann über das Netzwerk an den Empfänger. Zu verschiedenen Zeiten und für unterschiedliche Anwendungen wurden und werden dabei sehr unterschiedliche Netzwerke eingesetzt. Einige Beispiele:

- Der römische Kaiser schickt einen Befehl an einen Statthalter in einer entfernten Provinz. Als Netzwerk dienen berittene Boten.
- Sie schicken eine Ansichtskarte aus dem Urlaub nach Hause.
- Ich rufe im Sekretariat der FH an.
- Die Bankmitarbeiterin sendet eine elektronische Kursabfrage an die Börse.
- Ein Student verschickt eine Nachricht als SMS an einen Kommilitonen.
- Ein Benutzer schickt eine Datei zum Ausdrucken an einen Druckserver.

Anhand dieser einfachen Beispiele kann man bereits einige der Grundfragen von Netzwerken identifizieren. Das erste fundamentale Problem ist „Wie kommt die Nachricht vom Sender zum Empfänger?“ Daraus ergeben sich die Fragen der Adressierung und Vermittlung. Zunächst benötigt man eine eindeutige Zieladresse, um den Empfänger identifizieren zu können. Dann kann ausgehend von der Startadresse ein Weg zum Ziel festgelegt werden.

Weitere Grundfragen betreffen die Leistungsfähigkeit des Kanals: „Wie schnell können wie viele Daten übertragen werden?“ Dabei sind einerseits physikalische Grenzen zu betrachten („Wie viele Pakete passen in ein Postauto?“) aber auch die Verteilung der Last unter den verschiedenen Teilnehmern im Netzwerk.

Im Zusammenhang mit Entwurf und Betrieb von Netzwerken spielen noch viele andere Gesichtspunkte eine Rolle. Einige davon sind:

- Physikalische Realisierung
- Netzwerk-Architektur
- Gebühren
- Sicherheit gegenüber Dritten
- Zuverlässigkeit gegen Übertragungsfehler
- Ausfallsicherheit

Im Rahmen dieser Vorlesung werden wir uns mit der Verbindung zwischen Rechnern beschäftigen. Traditionell unterscheidet sich diese in Anforderungen und Techniken von Kommunikationsnetzen, wie sie für Telefonverbindungen genutzt werden. Bedingt durch die fortschreitende Digitalisierung verschwimmen aber zunehmend die Grenzen zwischen den verschiedenen Netzwerken. und insbesondere zwischen Rechnernetzen und anderen Kommunikationsnetzen.

Beispielsweise finden Systeme, um über Computer-Verbindungen zu telefonieren (Voice over IP), zunehmende Verbreitung. Ein großer Vorteil ist in diesem Fall der Wegfall einer zweiten, getrennten Verkabelung. Bei mobilen Endgeräten werden bereits Kombinationen von Telefon und Rechner angeboten. Auch im Unterhaltungsbereich ist der Übergang zu digitalen Inhalten etwa bei Audio oder Fotografie bereits weit fortgeschritten. Ganz allgemein geht der Trend zu intelligenten Endgeräten. Damit kommt der Vernetzung all dieser Geräte eine immer wichtigere Rolle zu. Man denke etwa an die Vision eines intelligenten Hauses, bei dem Sensoren und Geräten zur Klimasteuerung, Beleuchtung, Sicherheit, Kommunikation und Unterhaltung untereinander verknüpft sind.

Im Zuge dieser so genannten Konvergenz verschiedener Netze ist abzusehen, dass die Techniken der Rechnerkommunikation in immer mehr Bereiche angewendete werden wird. Allerdings führt die Kombination verschiedener Dienste mit sehr unterschiedlichen Anforderungen bezüglich Ausfallsicherheit und Antwortzeit zu neuen, anspruchsvollen Herausforderungen für den Entwurf von Netzwerken.

1.2 Übersicht

Im nächsten Kapitel werden einige grundlegende Begriffe und Konzepte eingeführt. Davon ausgehend wird anschließend zunächst die direkte Kommunikation

zwischen zwei Rechnern diskutiert. In den darauf folgenden Kapiteln wird dann schrittweise der Umfang der betrachteten Netze erweitert bis hin zu dem globalen Internet. Ein abschließendes Kapitel behandelt das Thema Datensicherheit und Verschlüsselung. Die Vorlesung folgt dabei im wesentlichen den derzeit dominierenden Standards Ethernet und TCP/IP. Auf die Vielzahl alternativer oder konkurrierender Technologien kann nur in Einzelfällen näher eingegangen werden. Die dargestellten Grundlagen sind jedoch weitgehend unabhängig von der konkreten Technologie und lassen sich daher leicht übertragen.

Vier Kapitel behandeln Themen der Programmierung von Netzwerkanwendungen. Dabei wird zunächst die Programmierung von Sockets und Remote Procedure Calls in der Sprache C vorgestellt. In einem weiteren Kapitel wird eine kurze Einführung in die Netzwerkprogrammierung mittels Java gegeben. Schließlich werden in Kapitel 14 die Grundlagen von Web-Services an einem einfachen Beispiel behandelt.

In vielen Fällen können die Ansätze in Rechnernetzen auf vertraute Konzepte aus den klassischen Bereichen von Post und Telefon zurück geführt werden. Insbesondere die beiden Grundproblem Adressierung und Zustellung bleiben trotz aller Unterschiede in der technischen Realisierung die gleichen. Im folgenden wird häufig explizit auf diese Analogie hingewiesen. Darüber hinaus ist es immer wieder sinnvoll, sich die Algorithmen durch Übertragung auf z.B. Briefe oder Postpakete zu veranschaulichen.

1.3 Übungen

Übung 1.1 *Unter Betriebssystemen wie Windows oder UNIX gibt es eine Reihe von Anwendungen zum Testen und Konfigurieren von Netzwerken:*

- `hostname`
- `ipconfig`
- `ping`
- `tracert`
- `nslookup`
- `net`
- `netsh`
- `netstat`

(Namen der Windows-Versionen). Informieren Sie sich an Hand der Hilfe-Funktionen über die Möglichkeiten dieser Anwendungen. Benutzen Sie die Anwendungen um die folgenden Fragen zu beantworten bzw. Aufgaben zu lösen:

- *Welchen Namen und welche IP Adresse hat Ihr Rechner?*
- *Testen Sie die Verbindung zu einem anderen Rechner.*
- *Welche Route wird zwischen den beiden Rechnern genommen?*
- *Eröffnen Sie auf einem zweiten Rechner eine Session mit* `telnet` *(sofern der andere Rechner als Server für* `telnet` *arbeitet).*

Übung 1.2 *Schreiben Sie ein Programm, um eine Liste von Adressen auf Erreichbarkeit zu testen. Sie können dazu aus dem Programm heraus wiederholt die Anwendung* `ping` *starten. In vielen Programmiersprachen wie z.B. C oder Perl steht für solche Zwecke eine Funktion* `system` *zur Verfügung.*

Kapitel 2

Grundlagen

2.1 Vermittlungsverfahren

Lange Zeit waren die beiden wohl wichtigsten Kommunikationsnetze Post und Telefon. Diese beiden Netze sind gleichzeitig Repräsentanten für zwei Netzwerktypen. Bei dem Postdienst werden einzelne Sendungen (Briefe, Pakete, o.ä.) vermittelt. Jedes Paket trägt die Zieladresse. Innerhalb des Netzes wird anhand dieser Zieladresse der Laufweg festgelegt: Briefkasten – örtliches Postamt – Postverteilzentrum – . . . – Empfänger. Aber selbst wenn man mehrere Briefe gleichzeitig an den gleichen Adressaten aufgibt ist nicht garantiert, dass die Briefe den gleichen Weg nehmen und gleichzeitig ankommen. Typischerweise wird an jedem Punkt nur über den nächsten Schritt entschieden. Diese Art der Vermittlung nennt man „**Paketvermittlung**“. Charakteristisch sind:

- diskrete Einheiten (Pakete)
- jedes Paket trägt die Adresse
- jedes Paket wird einzeln weitergeleitet
- im Allgemeinen keine Garantien bezüglich Laufweg und Laufzeit
- Robustheit gegenüber Unterbrechung einzelner Verbindungen

Demgegenüber wird beim klassischen Telefonnetz eine feste Verbindung zwischen den beiden Teilnehmern aufgebaut. Der Anrufer meldet seinen Verbindungswunsch an. Daraufhin wird im System eine Leitung zu dem Ziel aufgebaut. Wenn die Gesprächsverbindung zustande kommt, wird für die Dauer des Gesprächs eine feste Übertragungsrate exklusiv reserviert. Man spricht daher von „**Leitungsvermittlung**“. Wesentliche Eigenschaften sind:

- Aufbau einer Verbindung (Leitung) zwischen Quelle (Anrufer) und Ziel (Angerufenem)

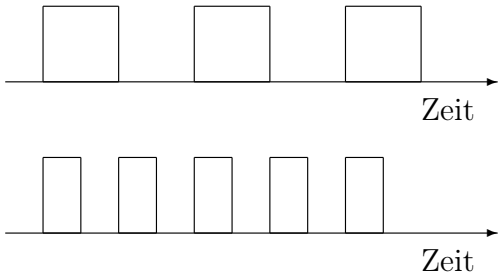


Abbildung 2.1: Einzelne Bits als zeitliche Impulse

- Erreichbarkeit des Zieles wird vor Beginn der Kommunikation sichergestellt
- Kommunikation erfolgt unter exklusiver Nutzung der Verbindung
- Zum Abschluss wird die Verbindung abgemeldet (Verbindungsabbau)
- Vorteil: konstante und garantierte Qualität
- Nachteil: schlechte Ausnutzung bei variabler Übertragungsrate
- Nachteil: empfindlich gegenüber Störung im Verbindungspfad

2.2 Leistungsfähigkeit

Für die Realisierung verschiedener Dienste sind unterschiedlich leistungsfähige Netzwerke erforderlich. Primäre Kriterien sind dabei Durchsatz und Verzögerung. Durchsatz oder Bandbreite gibt an, wie viele Daten in einer Zeiteinheit übertragen werden können. Das übliche Maß ist Bit pro Sekunde beziehungsweise die daraus abgeleiteten Einheiten wie etwa Mbit/s oder Gbit/s. Die Angabe 10Mbit/s beispielsweise bedeutet, dass das Netzwerk 10 Millionen Bits in einer Sekunde übertragen kann. Umgekehrt kann man aus dieser Angabe berechnen, wie viel Zeit die Übertragung eines einzelnen Bits benötigt. Für das Beispiel des 10Mbit/s Netzwerkes ergibt sich der Wert von 0,1μs (Mikrosekunden). Dies ist allerdings nur die Zeit, bis der Sender das nächste Bit abschicken kann. Man darf diesen Wert nicht mit der Laufzeit bis zum Empfänger verwechseln. Wenn man sich die einzelnen Datenbits als ein Folge von Impulsen auf der Zeitachse vorstellt, kann man sagen, dass ein Bit (Impuls plus Pause) 0,1μs breit ist.

Die Bandbreite wird durch die technische Realisierung der Leitung bestimmt. Eine Bandbreite von 10 Mbit/s bedeutet, dass irgendwo im System ein Takt von 1 MHz vorgegeben wird. Die Angabe von Frequenzen folgt der üblichen Konvention von Zehnerpotenzen, d.h. 1 MHz bedeutet 10^6 Hz. Daraus abgeleitet versteht man unter 1 Mbit/s auch 10^6 bit/s. Demgegenüber erfolgt die Größenangabe von Computerspeicher in Potenzen von 2. Ein kbit umfasst $2^{10} = 1024$ bit und

ein Mbit $2^{20} = 1048576$ bit. Die unterschiedliche Interpretation muss im Prinzip immer berücksichtigt werden. So dauert etwa die Übertragung einer Datenmenge von 1 kbit über eine 1 kbit/s Verbindung nicht exakt eine Sekunde. Vielmehr berechnet sich der exakte Wert zu

$$\frac{1024\text{bit}}{1000\text{bit/1s}} = 1.024\text{s}$$

Glücklicherweise ist der Unterschied recht gering (1.024 im Vergleich zu 1.000). Da in den allermeisten Fällen nur die ungefähre Größenordnung benötigt wird, kann man diese Feinheit zunächst ignorieren und den kleinen Fehler in Kauf nehmen.

Die zweite wichtige Charakteristik einer Verbindung ist die Verzögerung oder Latenz (von lat. lateō verborgen, versteckt, unbekannt sein). Latenz bezeichnet die Zeit, die eine Nachricht für den Weg vom Sender bis zum Empfänger benötigt. Die Latenz beinhaltet drei Komponenten:

1. Ausbreitungsverzögerung
2. Übertragungsverzögerung
3. Wartezeit

Die Ausbreitungsverzögerung t_A resultiert aus der endlichen Geschwindigkeit mit der sich Signale ausbreiten. Kein Signal kann schneller als mit Lichtgeschwindigkeit übertragen werden. Die Lichtgeschwindigkeit hängt vom Übertragungsmedium ab. Als Richtwert kann der Wert $c = 3.0 \times 10^8$ m/s für die Ausbreitung im Vakuum dienen. Damit kann man näherungsweise für die Ausbreitungsverzögerung bei einer Leitungslänge l den Wert $t_A = l/c$ ansetzen. Bei der Strecke Friedberg – Berlin (ca. 500km) erhält man

$$t_A = (500 \times 10^3)/(3 \times 10^8) = 166.7/10^5 = 1.667 \times 10^{-3} = 1.667ms \qquad (2.1)$$

Die Übertragungsverzögerung t_U ist durch die bereits oben diskutierte Dauer eines Bits bestimmt. Wir hatten gesehen, dass bei einer Bandbreite von B bit/s ein einzelnes Bit $1/B$ s an Übertragungszeit benötigt. Für ein Objekt der Größe K ergibt sich dann

$$t_U = K/B \qquad (2.2)$$

Die Definition bezieht sich auf größere Objekte, da in der Regel nicht ein einzelnes Bit sondern immer größere Datenblöcke als eine Einheit gesendet werden.

Die dritte Komponente – Wartezeit t_W – spielt dann eine Rolle, wenn die Übertragungsstrecke nicht nur aus einer einzelnen Leitung sondern aus mehreren Teilstücken besteht. Dann entsteht in den dazwischen liegenden Vermittlungsknoten eine Wartezeit. Die gesamte Wartezeit hängt von der Verarbeitungsgeschwindigkeit der einzelnen Knoten, der Größe der Datenpakete sowie der Anzahl der

Knoten ab. In praktischen Fällen ist die Wartezeit oft der größte Beitrag zur Latenz. Bei Paketvermittlung wartet ein Vermittlungsknoten häufig, bis er ein Paket vollständig erhalten hat, bevor er es weiter schickt. Bei hohem Verkehrsaufkommen muss das Paket zusätzlich auf eine freie Lücke warten. Typisch für beispielsweise einen transkontinentalen Kanal ist eine Latenz in der Größenordnung von 50 ms.

Die Latenz ist die Verzögerung vom Sender zum Empfänger. Häufig findet man auch die Angabe, wie lange ein Paket vom Sender zum Empfänger und wieder zurück benötigt. Man spricht dann von Roundtrip–Zeit (engl. Round–Trip Time, RTT). Zur besseren Unterscheidung benutzt man in Fällen, in denen eine Verwechslung auftreten könnte, für die einfache Verzögerung den Ausdruck Einweglatenz.

2.2.1 Bedeutung von Bandbreite und Latenz

Prinzipiell kann man sagen, je größer die Bandbreite und je kleiner die Verzögerung desto „besser“ ist die Verbindung. Allerdings hängen die konkreten Anforderungen von der jeweiligen Anwendung ab. Bei kleinen Objekten – z.B. der Übermittlung eines Tastendrucks – dominiert die Verzögerungszeit. Bei einem Kanal mit 1 Mbit/s beträgt die Übertragungsverzögerung für 1 Byte

$$t_U = 8/(1 \times 10^6) = 8\mu s = 0.008ms \tag{2.3}$$

Dieser Wert ist im Vergleich zur gesamten Latenz nahezu vernachlässigbar. Bei größeren Objekten erfolgt die Übertragung in vielen kleineren Paketen. Die gesamte Übertragungsdauer – die vom Benutzer wahrgenommene Latenz – berechnet sich dann aus der Latenz eines einzelnen Pakets und der benötigten Zeit, um entsprechend viele Pakete zu übertragen. Für die Übertragung einer Videodatei mit 20 MByte berechnete man bei dem 1 Mbit/s Kanal für die reine Übertragungsverzögerung

$$t_U = 8 \cdot 20 \times 2^{20}/(1 \times 10^6) = 168s \tag{2.4}$$

In diesem Fall spielt die Laufzeit eines einzelnen Pakets keine Rolle. Die Übertragung lässt sich nur durch Erhöhen der Bandbreite signifikant verbessern. In Tabelle 2.1 sind für verschiedene Bandbreiten und RTT-Werte die resultierenden Werte der wahrgenommenen Latenz zusammen gestellt. Die Werte zeigen, dass die höhere Bandbreite erst bei größeren Datenmengen zu spürbaren Verbesserungen führt.

2.2.2 Verzögerung–Bandbreite–Produkt

Während die ersten Daten auf dem Weg vom Sender zum Empfänger sind, kann der Empfänger bereits weitere Daten schicken. Ansonsten, wenn der Sender beispielsweise auf eine Empfangsbestätigung wartet, ist der Kanal nur schlecht ausgenutzt. Die Anzahl der Daten, die gleichzeitig auf dem Weg sind wenn der Sender

Tabelle 2.1: Wahrgenommene Latenz in ms für verschiedene Verzögerungszeiten und Objektgrößen

	1ms RTT		100ms RTT	
Größe	1 Mbit/s	10 Mbit/s	1 Mbit/s	10 Mbit/s
1Byte	1.008	1.0008	100.008	100.0008
1kByte	9.192	1.8192	108.192	100.8192
1MByte	8389.60	839.86	8488.61	938.86

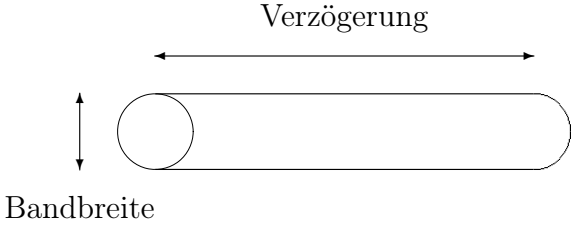


Abbildung 2.2: Verzögerung–Bandbreite–Produkt als Pipeline

permanent Pakete schickt, berechnet sich als Produkt aus Latenz und Bandbreite. In Bild 2.2 ist das so genannte Verzögerung–Bandbreite–Produkt grafisch als Röhre beziehungsweise Pipeline dargestellt. Ein Kanal mit einer Bandbreite von 10Mbit/s und einer Latenz von 50ms nimmt ein Volumen von

$$50 \times 10^{-3}s \cdot 10 \times 10^{6}Bit/s = 500 \times 10^{3}Bit \sim 62kByte \tag{2.5}$$

auf. Wenn der Empfänger das erste Bit empfängt sind bereits etwa 62 kByte Daten unterwegs. Signalisiert der Empfänger dem Sender, dass er keine weiteren Daten mehr aufnehmen kann, so benötigt der Sender wiederum die Latenzzeit von 50ms um reagieren zu können. Der Empfänger muss daher ausreichend Speicher vorhalten, um die Leitung noch leeren zu können. Andernfalls – wenn die Daten verloren gehen – müssen sie später erneut gesendet werden.

2.2.3 Anwendungen

In der Praxis sind die Anforderungen von Anwendungen an die Leistungsfähigkeit des Übertragungskanals sehr unterschiedlich. Häufig schwankt die Anforderung auch sehr stark während der Verbindungsdauer. So benötigt man während des Surfens im Internet während der meisten Zeit beim Lesen und Anschauen nur geringe Bandbreite. Demgegenüber wird beim Wechsel zu einer neuen Seite kurzfristig viel Bandbreite benötigt, um aufwändige Bilder oder Videos schnell zu laden. Am anspruchsvollsten im Hinblick auf die Netzwerksanforderungen sind

Tabelle 2.2: Datenraten für einige Anwendungen nach [4]

Anwendung	Datenrate
Sprachübertragung (Telefonqualität)	64 kbit/s
Audiosignale (hohe Qualität)	1–2 Mbit/s
Videosignale (komprimiert)	2–10 Mbit/s
Video	1–2 Gbits/s

Anwendungen, die einen kontinuierlichen Datenstrom mit engen Randbedingungen für die Verzögerung erfordern. Ein klassisches Beispiel ist das Telefonnetz. Die erforderliche Datenrate ist zwar gering (64 kbit/s bei ISDN, 13 kbit/s bei GSM), aber die Daten müssen ohne grosse Verzögerung und für die Dauer der Verbindung ohne erkennbare Pausen übertragen werden. Aus diesem Grund werden Telefonverbindungen bis heute in großen Teilen mit leitungsvermittelnden Netzwerken realisiert. In Tabelle 2.2 sind für einige Anwendungen die erforderlichen Datenraten angegeben.

Neben der absoluten Verzögerung kann auch die relative Schwankung eine Rolle spielen. Bei einer Videoanwendung ist beispielsweise die absolute Verzögerung nicht entscheidend. Ob das Video nach 50 ms oder 200 ms beginnt ist nur ein geringer Unterschied. Aber nach dem Starten des Videos müssen die einzelnen Bilder immer rechtzeitig im erforderlichen Takt (z.B. 30 Bilder pro Sekunde) geliefert werden. Bei einem paketvermittelnden Netzwerk kann es zu Schwankungen der Laufzeit – so genanntem Jitter – kommen. Im Extremfall überholt ein Paket das vor ihm gestartet Paket. Beim Empfänger kommt in solchen Fällen das Abspielen des Videos ins Stocken. Wenn die absolute Verzögerung keine große Rolle spielt, kann der Empfänger das Problem umgehen, indem er zunächst einige Bilder speichert und das Video erst mit einer entsprechenden Verzögerung abspielt. Dann können bei eventuellen Verzögerungen vermehrt Bilder aus dem Speicher wieder gegeben werden, bis der Sender neue Bilder schickt.

2.2.4 Nutzung

Um einen Eindruck von der Nutzung der verschiedenen Anwendungen zu erhalten, wurde eine Umfrage unter Studenten und Studentinnen der Wirtschaftsinformatik durchgeführt. Das Ergebnis ist in Tabelle 2.3 dargestellt. Offensichtlich gehören das Suchen im World Wide Web sowie Emails mehr oder weniger zum täglichen Leben. Demgegenüber sind die älteren Anwendungen wie news oder FTP wenig verbreitet.

Tabelle 2.3: Nutzung von Anwendungen, Umfrage unter 50 Studenten im 2. Semester in 2003 (Angaben in Prozent)

Anwendung	Nie	Manchmal	Ständig
www	0	6	94
email	0	8	92
news	68	28	4
Internet-Telefon	76	20	0
Internet-Radio	52	44	2
Instant Messager	42	18	40
Peer-to-peer Tauschbörsen	28	28	44
Spiele	-	-	6
FTP	-	-	10
Telnet	-	-	2

2.3 Netzwerktypen und –topologien

2.3.1 Größe

Aus der Größe eines Netzes ergeben sich wichtige Randbedingungen und Konsequenzen für den Betrieb. Größe umfasst dabei sowohl die geographische Ausdehnung als auch die Anzahl der angeschlossenen Teilnehmer. Zwei wichtige Kategorien sind lokale Netze (LAN, local area network) und Fernnetze (WAN, wide area network). Ein LAN ist ein geschlossenes Netz innerhalb eines Gebäudes oder Firmengeländes. Die Reichweite ist damit auf wenige Kilometer beschränkt. Die Anzahl der angeschlossenen Teilnehmer ist bekannt. Weiterhin kann der Betreiber die eingesetzte Technik festlegen. Diese Faktoren vereinfachen das Netzwerkmanagment erheblich.

Ein WAN umfasst demgegenüber einen großen Bereich – beispielsweise ein Land oder einen Kontinent. Die Signallaufzeiten können daher unter Umständen sehr groß werden. In der Regel ist das Netz offen. Ständig werden neue Teilnehmer angeschlossen oder auch vorhandene Teilnehmer wieder abgemeldet. Für die Benutzung des Netzes müssen die Teilnehmer Gebühren bezahlen. Ein öffentliches Netz ist gerade dadurch gekennzeichnet, dass jeder der die technischen Voraussetzungen für einen Netzanschluss erfüllt und die Gebühren bezahlt das Netz auch benutzen darf. Der Betrieb eines solchen Netzes erfordert daher eine klare Regelung der technischen und rechtlichen Fragen. Durch Standards wird sicher gestellt, dass die in aller Regel sehr heterogenen Endgeräte von verschiedenen Herstellern mit dem Netz zusammen funktionieren.

Neben diesen beiden Grundtypen unterscheidet man noch zwischen verschiedenen Mischformen. Häufig benutzt man die Kategorie Stadtnetz (MAN, metropolitan area network). Das weltweite Netz kann man als Netzverbund mehrerer WANs oder als ein GAN (global area network) sehen. In [4] findet man weiterhin

die Definitionen

- PAN (personal area network) oder Piconetz für die Vernetzung eines Arbeitsplatzes
- SAN (storage / system area network) für die Vernetzung eines Raumes – typischerweise ein Rechenzentrum

Bezüglich der Offenheit kann man weiterhin unterscheiden zwischen

- Internet: das öffentliche und für alle offene Netz.
- Intranet: ein Netzwerk basierend auf der gleichen Technik wie das Internet, aber nur für einen geschlossenen Benutzerkreis (z.B. Mitarbeiter einer Firma, Angehörige einer Hochschule) zugänglich.
- Extranet: ein Intranet, das ausgewählten, externen Benutzern (z.B. Kunden) einen Zugang ermöglicht.

2.3.2 Topologien

Bei dem Entwurf eines Netzes gibt es zahlreiche Möglichkeiten, die einzelnen Rechner zu verbinden. Dies betrifft sowohl die physikalische Verkabelung als auch die logische Struktur. So kann es beispielsweise sinnvoll sein, bestimmte Teile des Netzes gegen andere abzuschirmen um etwa den Zugriff auf sensible Daten zu kontrollieren. Für die Verkabelung spielen Fragen wie die erforderlichen Bandbreiten und einzuhaltenden Verzögerungen aber auch wirtschaftliche Überlegungen eine Rolle. Innerhalb eines Netzwerkes gibt es einige Standardstrukturen oder Topologien für Netzwerke. Im folgenden werden diese Standardtopologien kurz vorgestellt. In der Praxis wird man selten die reinen Formen sondern durch Kombination verschiedener Typen Mischformen realisieren.

Punkt-zu-Punkt Topologie

Beim Maschennetz sind je zwei Endpunkten direkt verbunden. Dadurch ist eine sehr schnelle Kommunikation zwischen Hosts möglich. Darüber hinaus können einzelne Verbindungen an betreffende Geräte angepasst werden. Insofern wäre eine komplette Vermaschung aller Rechner innerhalb eines Netzes die leistungsfähigste Lösung. Allerdings ist dies bei wachsender Anzahl von Endgeräten nicht mehr praktikabel. Bei N Endgeräten braucht jedes Endgerät $N - 1$ Anschlüsse und insgesamt hätte man $N * (N - 1)$ Leitungen. Daher wird meist nur eine teilweise Vermaschung durchgeführt. Nicht direkt verbundene Endgeräte kommunizieren dann über Zwischenstationen.

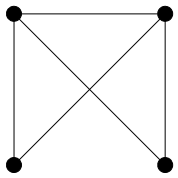


Abbildung 2.3: Teilweise vermaschtes Netz

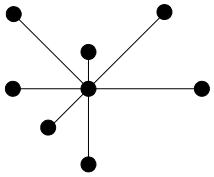


Abbildung 2.4: Sternnetz

Stern–Topologie

Eine Topologie mit sehr einfacher Struktur erhält man durch sternförmige Anordnung der Endgeräte um einen zentralen Knoten. Jedes Endgerät kommuniziert dann direkt nur noch mit dem Zentralknoten. Der Zentralknoten muss entsprechend leistungsfähig sein. Sinnvoll kann es sein an den Zentralknoten Komponenten wie Fileserver oder Backup-Geräte anzuschließen.

Baum–Topologie

Verwandt mit der Stern–Topologie ist die Baum–Topologie. Im Unterschied zur Stern–Topologie sind hierbei nicht alle Endgeräte an einen Zentralknoten angeschlossen sondern es gibt Zwischenknoten, die jeweils mehrere Endgeräte bedienen.

Ring–Topologie

Eine Topologie mit gleichmäßiger Arbeitsteilung – zumindest bezüglich der Kommunikation – ist die Ring–Topologie. Dabei ist jeder Knoten nur mit zwei anderen Knoten verbunden. Jedes Datenpaket wird dabei in jedem Knoten gelesen und –

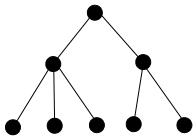


Abbildung 2.5: Baum

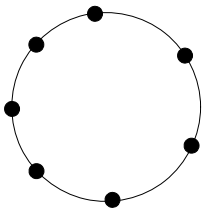


Abbildung 2.6: Ring

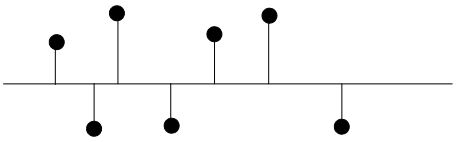


Abbildung 2.7: Bus

falls es nicht für diesen Knoten bestimmt ist – weiter gereicht. Da auf jeder Leitung nur jeweils ein Knoten sendet und empfängt, hat man gute Kontrolle über die Leitung und kann einen hohen Durchsatz erzielen. Nachteilig ist die hohe Abhängigkeit des Gesamtsystems von jeder einzelnen Leitung.

Bus–Topologie

In den bisher besprochenen Topologien verbindet eine Leitung stets zwei Knoten. Eine grundsätzliche Alternative ist die Nutzung einer gemeinsamen Leitung – dann auch als Bus bezeichnet. Jeder Knoten wird direkt mit diesem Bus verbunden. Um einen anderen Knoten zu erreichen, schickt ein Knoten eine Nachricht auf den gemeinsamen Bus. Alle Knoten müssen ständig die Daten auf dem Bus beobachten und die an sie adressierten Nachrichten aufnehmen. Eine Stärke dieser Anordnung ist die effiziente Realisierung von Broadcast Nachrichten, d.h. Nachrichten die an mehrere oder sogar alle Knoten im Netz gerichtet sind. Ein derartiges System stellt eine Reihe von besonderen Anforderungen. So muss man mit entsprechenden Protokollen sicher stellen, dass nicht zwei Knoten gleichzeitig senden oder ein Knoten den Bus dauerhaft belegt.

2.4 Referenzmodelle

2.4.1 Protokollstapel

In vielen Fällen kommunizieren Sender und Empfänger nicht direkt sondern über Zwischenstufen. Betrachten wir folgendes Beispiel: Herr A. möchte 12 Flaschen Rotwein von Chateau Rothschild in Bordeaux kaufen. Seine Nachricht kann man wie folgt darstellen:

Herr A.	12 Flaschen Rotwein	Chateau Rothschild

Herr A. spricht weder Französisch noch kennt er die Anschrift von Chateau Rothschild. Daher wendet er sich an seinen Weinhändler als Vermittler. Der Weinhändler kennt einen Großhändler in Bordeaux, spricht selbst allerdings auch kein Französisch. Er übersetzt daher die Bestellung in Englisch und schickt folgende Nachricht an den Großhändler:

Herr A.	12 bottles red wine	Chateau Rothschild	Großhändler in Bordeaux

Der französische Großhändler entfernt seine eigene Adresse, übersetzt in Französisch und leitet die Bestellung weiter.

Herr A.	12 bouteilles vin rouge	Chateau Rothschild

Der gesamte Ablauf der Bestellung ist in Bild 2.8 dargestellt. Dabei ist als zusätzliches Element noch die Übertragung der Nachricht als Brief per Post eingetragen. Das Bild zeigt, wie die eigentliche Kommunikation über mehrere Zwischenstufen abläuft. Auf diesem Weg wird die Nachricht zweimal übersetzt. Wichtig ist dabei, dass die einzelnen Schritte unabhängig voneinander erfolgen. Für die Bestellung ist es unwesentlich, ob die beiden Händler sich in Englisch oder irgend einer anderen Sprache verständigen, solange sie sich auf eine gemeinsame Sprache verständigen können. Entsprechend brauchen die beiden Endbenutzer die Übertragungssprache nicht zu verstehen.

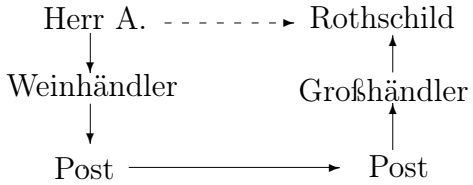


Abbildung 2.8: Ablauf Rotwein–Bestellung

Für jede Schnittstelle muss nur zwischen den beiden beteiligten Partnern ein Protokoll verabredet werden. Die gesamte Übertragung beinhaltet eine Kette von Protokollen - den so genannte Protokollstapel Protokollstapel(engl. protocol stack). Auf der Sendeseite wird die Protokollkette von oben nach unten abgearbeitet bis die Nachricht physikalisch verschickt wird. Beim Empfänger durchläuft die Nachricht die Kette dann in umgekehrter Reihenfolge. Typisch ist dabei, dass auf der Sendeseite die unteren Ebenen zusätzliche Informationen anfügen (im Beispiel die Adresse des französischen Großhändlers), die dann beim Empfänger auf der gleichen Ebene wieder entfernt werden.

Diese Situation – die beiden Kommunikationspartner verständigen sich indirekt über eine Reihe von Zwischenschichten – stellt die große Herausforderung für heterogen Netzwerke dar. Ein typisches Beispiel ist die Kommunikation zwischen einem Web-Browser, der auf einem PC unter Windows2000 läuft, und einem Web-Server auf einem Großrechner mit dem Betriebssystem Linux. Trotz

7 Anwendung		7 Anwendung
6 Darstellung		6 Darstellung
5 Sitzung		5 Sitzung
4 Transport		4 Transport
3 Vermittlung		3 Vermittlung
2 Sicherung		2 Sicherung
1 Bitübertragung	⇔ Netzwerk ⇔	1 Bitübertragung

Abbildung 2.9: ISO/OSI 7 Schichten Referenzmodell

aller Unterschiede in Hard- und Software können die beiden Anwendungen sich verständigen.

2.4.2 OSI Referenzmodell

Zum Verständnis der Abläufe in Netzwerken ist eine standardisierte Darstellung der verschiedenen Prokollebenen hilfreich. Eine klare Definition funktionaler Ebenen erlaubt eine starke Kapselung. Die Schnittstelle zwischen zwei Ebenen kann dann leicht ausgetauscht werden, ohne dass es zu Rückwirkungen auf die anderen Protokolle kommt. Eine formale Definition der verschiedenen Schichten entwickelte die ISO (International Standards Organization) ab etwa 1977. Resultat ist das ISO/OSI 7 Schichten Modell (OSI: Open Systems Interconnection). Bild 2.9 zeigt die 7 Schichten des Referenzmodells. In diesem Modell sind weiterhin die Schnittstellen zwischen den Schichten festgelegt. Man unterscheidet zwischen Protokollen und Diensten. Protokolle definieren die Schnittstelle zwischen zwei Systemen auf einer Ebene. Demgegenüber sind in der OSI–Terminologie Dienste Funktionen, die eine Schicht der nächsthöheren Schicht zur Verfügung stellt.

Bitübertragung

Die Bitübertragungschicht (physical layer) stellt einen Bitstrom zwischen Sender und Empfänger bereit. Auf dieser Ebene spielt die Bedeutung der Bits keine Rolle. Vielmehr geht es darum, wie die einzelnen Bits übertragen werden. Darunter fallen Fragen wie Übertragungsmedien, Stecker, Darstellung einzelner Bits, Aufbau einer Verbindung, etc.

Sicherung

Bei der Bitübertragung können Fehler auftreten. Aufgabe der Sicherungsschicht (data link layer) ist es, solche Fehler zu erkennen und durch entsprechende Me-

chanismen zu korrigieren. Dazu werden Blöcke von Datenbits zusammen gefasst. Die so gebildeten Datenrahmen (data frames) werden mit zusätzlichen Bits zur Kennung von Anfang und Ende versehen. Weitere Kontrollbits erlauben dem Empfänger, die korrekte Übertragung des Rahmens zu überprüfen. Erkennt der Empfänger Fehler, kann er sie bis zu einer gewissen Grenze korrigieren. Ist der Rahmen so gestört, dass eine Korrektur nicht mehr möglich ist, fordert der Empfänger den Sender auf, den Rahmen erneut zu senden.

Vermittlung

Der Weg den die Daten nehmen sollen wird in der Vermittlungsschicht (network layer) festgelegt. Diese Festlegung kann beispielsweise zu Beginn einer Verbindung erfolgen. Alternativ kann auch für jedes Paket neu die gerade optimale Route bestimmt werden. Man spricht auf der Vermittlungsschicht nicht mehr von Rahmen sondern von Paketen. Die Vermittlungsschicht enthält in der Regel auch Abrechnungsfunktionen, um die entstandenen Gebühren zu erfassen.

Wenn auf dem Weg vom Sender zum Empfänger mehrere Knoten liegen, werden in jedem der Knoten die drei untersten Schichten benötigt.

Transport

Die Transportschicht (transport layer) ist die erste Ende–zu–Ende Schicht. D.h. das Programm in der Transportschicht der Sendeseite kommuniziert mit einem passenden Gegenstück auf Empfängerseite. Aufgabe der Transportschicht ist, größere Datenmengen von den oberen Schichten zu übernehmen, falls notwendig in kleinere Einheiten zu zerlegen und dann an die Vermittlungsschicht weiter zu geben. Die Transportschicht ist auch zuständig für den Aufbau der Verbindung.

Sitzung

Die Sitzungsschicht (session layer) bietet eine übergeordnete Sicht der Verbindung. Eine Sitzung kann verschiedene Verbindungen beinhalten, z.B. für Hin- und Rückkanal oder in zeitlicher Folge wenn die Verbindung nicht permanent gehalten wird.

Darstellung

In der Darstellungsschicht (presentation layer) werden die von unten gelieferten Daten interpretiert und in die für das System richtige Darstellung gebracht. Beispielsweise werden für die Übertragung Standardrepräsentationen für Daten wie Zeichen, Festkommazahlen, etc. definiert. In der Darstellungsschicht erfolgt dann die Umsetzung aus der allgemeinen Repräsentation in das maschinenspezifische Format. Zu den weiteren Aufgaben in dieser Schicht gehören Verschlüsselung und Datenkompression.

Anwendung

Die Anwendungs– oder Verarbeitungsschicht (application layer) stellt dem Benutzer bestimmte Dienste zur Verfügung. Dazu gehören der Dateitransfer, das Anmelden auf anderen Rechnern oder die Kontrolle von Prozessen.

2.4.3 Bedeutung des OSI Referenzmodells

Das ISO/OSI 7 Schichten Modell bietet eine gute Grundlage zum Verständnis sowie zur vergleichenden Beurteilung von eingesetzten Technologien. Die gewählte Einteilung in 7 Schichten sollte allerdings nicht zu starr gesehen werden. In vielen Fällen erweist sich die Einteilung als teilweise zu fein oder zu grob. Die höheren Schichten sind oft in weniger Schichten realisiert. Beispielsweise hat die im Internet verwendete TCP/IP Architektur insgesamt nur 4 Schichten. Tanenbaum benutzt in seinem Standardwerk [5] ein vereinfachtes Modell mit 5 Schichten. Umgekehrt ist es sinnvoll, bei genauerer Untersuchung der unteren Schichten weitere Zwischenschichten einzuführen.

In der Praxis hat das Modell nie die von den Initiatoren erhoffte universelle Bedeutung erlangt. Durch den aufwändigen und zeitraubenden Standardisierungsprozess setzten sich oft am Markt mehr pragmatisch entstandene aber frühzeitig verfügbare Systeme durch. Universitäten als Vorreiter neuer Techniken haben eine andere Kultur als die ITU/ISO Gremien. Hier gilt mehr das Interesse an funktionierenden Programmen als an detailgenauen Spezifikationen (*„Grober Konsensus und laufender Code“*).

Die Vorlesung orientiert sich daher nicht streng an dem ISO/OSI 7 Schichten Modell. Vielmehr liegt der Schwerpunkt auf einzelnen Themen, die sich teilweise über mehrere Schichten ziehen. Im Zweifelsfall wird die inhaltliche Nähe stärker berücksichtigt als die Zuordnung zu den Schichten. Die folgenden vier Kapitel behandeln Rechnernetze zunehmender Komplexität. Im einzelnen lassen sich die Stufen

1. direkte Verbindung zweier Rechner
2. Kommunikation in einem lokalen Netz
3. Verbindung mehrerer lokaler Netze

unterscheiden. Anhand der Verbindung zweier Rechner werden Grundlagen wie Bildung von Rahmen oder Einfügen von Sicherungsinformationen dargestellt. Die Anforderungen, die sich aus dem Zusammenschluss mehrerer Rechner ergeben, werden zunächst für lokale Netze diskutiert. Schließlich wird die Verbindung vieler einzelner lokalen Netze unterschiedlichster Art zu einem großen Netzverbund – dem Internet – beschrieben.

2.5 Übungen

Übung 2.1

Welche Bandbreite ist für eine Echtzeitübertragung in den folgenden Fällen erforderlich:

1. *Musik von CD-ROM (75 Minuten Musik auf 650 MByte)*
2. *Video mit Auflösung 320 x 240 Pixel, 1 Byte Farbinformation pro Pixel, 10 Bilder pro Sekunde*
3. *Video mit Auflösung 640 x 480 Pixel, 3 Byte Farbinformation pro Pixel, 30 Bilder pro Sekunde*
4. *Wie stark müssen die Daten jeweils komprimiert werden, damit sie in Echtzeit über eine ISDN-Leitung (64 kbit/s) übertragen werden können.*

Übung 2.2 *In den Fernsehnachrichten wird ein Live-Interview mit dem Korrespondenten in New York gezeigt. Die Übertragung von Bild- und Sprachdaten erfolgt über getrennte Wege:*

Sprache: *eine Satellitenverbindung über einen geostationären Satelliten, Gesamtstrecke 70000 km.*

Bild: *über ein Unterseekabel, Gesamtstrecke 5000 km.*

Berechnen Sie die Ausbreitungsverzögerung für beide Strecken. Welche Laufzeitdifferenz ergibt sich?

Übung 2.3 *Eine Verbindung soll nicht mehr als 10 ms Einwegverzögerung haben.*

1. *Wie ist die maximale Länge, wenn nur die Ausbreitungsgeschwindigkeit berücksichtigt wird?*
2. *Wie ist der entsprechende Wert, wenn man zusätzlich die Übertragungsverzögerung bei einer Bandbreite von 1 Mbit/s und einer Paketgröße von 1024 Bytes einrechnet?*

Übung 2.4 *Sie sollen die Kommunikation mit einem Erkundungsfahrzeug auf dem Mond planen. Die Entfernung beträgt 385000 km und die Übertragung soll über eine Bandbreite von 1 Mbit/s erfolgen.*

1. *Wie groß ist die Ausbreitungsverzögerung? Wie groß ist damit die minimale Antwortzeit auf eine Anfrage?*

2. *Wie sind die Werte für die Verbindung mit einen Roboter auf dem Mars (Kleinster Abstand zur Erde ungefähr 54 Millionen Kilometer)?*

3. *Berechnen Sie auf der Basis der Ausbreitungsverzögerung das Verzögerungs-Bandbreiten Produkt für die Verbindung zwischen Erde und Mond.*

4. *Eine Webcam auf dem Mond nimmt Bilder in guter Qualität auf. Ein Bild benötigt 2 MByte Speicherplatz. Wie lange dauert es mindestens bis nach der Anforderung von der Erde aus ein Bild vollständig übertragen wurde.*

Übung 2.5 *Entwerfen Sie ein Netz für 10 Knoten in Form von zwei verbundenen Sternen.*

Übung 2.6 *Sie erstellen mit einem Textverarbeitungsprogramm eine Hotelreservierung, die Sie dann ausdrucken und per Fax an ein Hotel in Miami schicken.*

1. *Verfolgen Sie den Übertragungsweg. Welche Umwandlungen werden vorgenommen, wie sind die Schnittstellen?*

2. *Vergleichen Sie den Ablauf mit der Alternative, den Text als Email an das Hotel zu schicken. Wo liegen Vor- und Nachteile der beiden Möglichkeiten?*

Kapitel 3

Rechner zu Rechner Verbindung

3.1 Übertragung von Bits

Das einfachste Modell für eine Übertragung ist eine Leitung, auf der ein Signal zwei Zuständen annehmen kann. Dieses Signal bewegt sich mit der Ausbreitungsgeschwindigkeit vom Sender zum Empfänger. Die Dauer eines einzelnen Zustandes bestimmt die erreichbare Übertragungsrate. Bei einer Dauer Δt kann man pro Sekunde $1/\Delta t$ Zustände übermitteln.

Bei einer elektrischen Leitung werden die beiden Zustände beispielsweise durch zwei verschiedene Spannungen realisiert. Die beiden Zustände seien als low und high bezeichnet. Über diese Leitung sollen Daten übertragen werden. Dazu muss eine Zuordnung der Bits zu den Zuständen getroffen werden - die **Kodierung**. Die nahe liegende Vereinbarung ist, für den Wert 1 das Signal high und für 0 low zu übertragen. Bild 3.1 zeigt für eine Folge von 0–1 Werten die Zustände in zeitlicher Folge. Aus im folgenden klar werdenden Gründen nennt man diese Art der Kodierung Non-Return to Zero (NRZ).

Einen etwas allgemeineren Fall zeigt Bild 3.2. Folgen mehrere gleiche Werte aufeinander, bleibt das Signal auf einem festen Pegel und es findet kein Wechsel mehr statt. Wenn beispielsweise der Sender eine lange Folge von Nullen schickt, sieht der Empfänger für die entsprechende Zeitdauer einen konstanten Pegel.

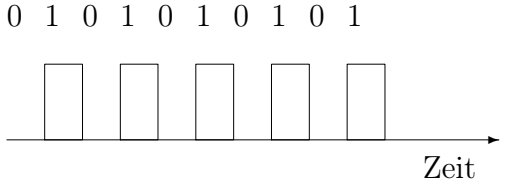


Abbildung 3.1: Folge von Bits mit Werte 0 und 1

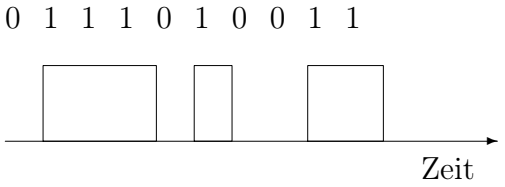


Abbildung 3.2: Allgemeine Folge von Bits

Dann kann es schwierig werden, zu zählen exakt wie viele Nullen geschickt wurden. Falls Sender und Empfänger keine gemeinsame Zeitbasis haben, können kleine Abweichungen in der Zeitmessung zwischen beiden dazu führen, dass der Empfänger zu wenige oder zu viele Nullen zählt. Im Normalfall – d.h. wenn keine zu langen Folgen mit konstanten Werten auftreten – kann der Empfänger kleine Abweichungen kompensieren, indem er die Signalwechseln nutzt, um sich immer wieder auf den Sender zu synchronisieren (Taktregenerierung).

Andererseits können Taktfehler weitreichende Folgen haben. Betrachtet man als Beispiel die Übertragung eines Stroms von Bytes. Sobald ein Bit verloren geht, verschiebt sich für alle nachfolgenden Werte die Zuordnung der Bits zu den Bytes, so dass sich komplett andere Bytewerte ergeben.

Es ist daher sinnvoll dafür zu sorgen, dass auch bei konstanten Werten Signalwechsel statt finden. Ein Ansatz dazu ist die Non-Return to Zero Inverted (NRZI) Kodierung. Dabei gilt die Regel, dass bei einer 1 der Pegel sich ändert, d.h. von low auf high oder von high auf low springt. Dadurch wird aus der Folge 0 1 1 1 1 die Signalfolge low high low high low. NRZI löst das Problem von vielen aufeinander folgenden Einsen, hilft aber nicht bei langen Folgen von Nullen.

Ein weitergehender Ansatz ist die Manchester–Kodierung. Man kann sich dabei die Übertragung mit doppelter Rate vorstellen, d.h. für jeden Wert stehen zwei Zeiteinheiten zur Verfügung. Dann überträgt man die Folge low–high für 0 und high–low für 1. So ist gewährleistet, dass für jeden übertragenen Wert mindestens ein Signalwechsel statt findet. Nachteil ist, dass jetzt für jeden Wert zwei Zeiteinheiten gebraucht werden, oder – umgekehrt gesprochen – in der gleichen Zeit nur halb so viele Werte übertragen werden können. Von den vier möglichen Signalfolgen werden nur noch zwei verwendet.

Es gibt eine ganze Reihe weiterer Schemata für die Kodierung, die auf diesen Ideen beruhen. Der verbreitete 4B5B-Code überträgt 4 bit mit 5 Signalwerten. Die Zuordnung ist dabei so gewählt, dass nie mehr als 3 Nullen hintereinander kommen. Zusammen mit einer NRZI Kodierung ermöglicht der Code bei nur geringem Mehraufwand (20%) eine deutliche Verbesserung der Taktregenerierung.

In diesem Zusammenhang wird oft das Maß Baud (Bd) benutzt. Die Einheit

ist benannt nach dem französischer Ingenieur Baudot [1], der den Baudot-Code – den Vorläufer des ASCII-Code – entwickelte. Die Anzahl der pro Sekunde erfolgten Übertragungen bezeichnet man als Baudrate. Wenn mit jeder Übertragung genau ein Bit dargestellt wird, entspricht die Baudrate dem Maß bit/s. Anderseits sind bei der Manchester–Kodierung zwei Übertragungen pro Bit notwendig. Damit ist die Baudrate doppelt so hoch wie die Anzahl der Bits pro Sekunde. Können bei der Übertragung mehr als nur zwei Zustände unterschieden werden, ist umgekehrt die Baudrate kleiner als die Zahl der Bits pro Sekunde.

3.2 Rahmenerstellung

Die Übertragung von Bits ist stets auch bis zu einem gewissen Grad fehleranfällig. Neben den bereits besprochenen Synchronisationsfehlern können auch einzelne Bits falsch dekodiert werden. Solche Fehler sind in einem kontinuierlichen Datenstrom nicht zu erkennen. Es ist daher wichtig, den Datenstrom zu strukturieren und durch Zusatzinformationen unempfindlich gegen Störungen zu machen. Weiterhin ist bei Rechnernetzen die kontinuierliche Übertragung eher die Ausnahme. Bei typischen Anwendungen (z.B. surfen im Internet) wechseln sich Phasen von Datenübertragungen mit – aus Sicht des Kommunikationskanals – Ruhephasen ab. Daher muss der Beginn einer neuen Übertragung kenntlich gemacht werden.

Aus diesen Gründen werden mehrere Bits oder Bytes zusammengefasst, um entsprechenden Informationen ergänzt und als Einheit übertragen. Man spricht dann von Rahmen oder Frames. Dabei unterscheidet man zwischen Byte-orientierten und Bit-orientierten Protokollen, je nach dem ob Bytes oder Bits als kleinste Bausteine für einen Frame benutzt werden.

Bei klassischen Kommunikationsanwendungen wie Telefon hat man eine feste Übertragungsrate, so dass man den Datenstrom gut in eine Reihe gleich grosser Rahmen aufteilen kann. Wenn eventuell am Ende der Übertragung ein Rahmen nicht mehr komplett gefüllt wird, kann problemlos mit Nullen aufgefüllt werden. Die dadurch entstehende kleine Pause wird nicht stören. Demgegenüber ist die Rechnerkommunikation durch starke Schwankungen in der Übertragungsrate gekennzeichnet. Außerdem ist das Auffüllen des letzten Rahmens beispielsweise bei dem Kopieren einer Datei nicht akzeptabel. Daher werden in der Regel Rahmenformate für eine variable Anzahl von Daten eingesetzt.

Dazu muss der Sender dem Empfänger mitteilen, wie viele Daten im aktuellen Rahmen enthalten sind. Zwei Verfahren werden dazu benutzt: Verwendung einer Endemarkierung oder explizite Angabe der Anzahl der Daten. Zusammengefasst kann man daher folgende Methoden der Rahmenbildung unterscheiden:

- Feste Anzahl von Nutzdaten
- Variable Anzahl von Nutzdaten

[1] Jean-Maurice-Émile Baudot (1845-1903)

- Markierung des Endes der Nutzdaten
- Angabe der Anzahl der Nutzdaten im aktuellen Frame

Im folgenden werden am Beispiel von Byte-orientierten Protokollen die beiden Verfahren zur Erstellung von Rahmen mit einer variablen Anzahl von Datenbytes diskutiert.

3.2.1 Markierungszeichen

Bei der Übertragung von Texten verwendet man die im ASCII–Standard definierten Steuerzeichen zum Aufbau von Rahmen. Ein einfacher Rahmen könnte dann folgende Form haben (Beispiel nach [5]):

DLE	STX	Daten	DLE	ETX

Die zeitliche Abfolge ist bei dieser Darstellung von links nach rechts, d.h. das am weitesten links stehende Element wird zuerst gesendet. Das Zeichen DLE (Data Link Escape, ASCII Code 127) markiert den Beginn einer Steuersequenz. Anfang und Ende werden durch STX (Start of TeXt, ASCII Code 2) beziehungsweise ETX (End of TeXt, ASCII Code 3) gekennzeichnet. Eingebettet zwischen diesen beiden Steuersequenzen liegen die Nutzdaten.

Dieses Verfahren funktioniert bei der Übertragung von Textdaten problemlos. Bei beliebigen Binärdaten – Gleitkommazahlen, Bilder, oder ähnliches – besteht allerdings die Möglichkeit, dass die Endsequenz DLE ETX auch in den Nutzdaten auftritt:

DLE	STX	...	DLE	ETX	...	DLE	ETX

Der Empfänger würde dann vorzeitig auf Ende des Rahmens erkennen. Ein ähnliches Problem ist bei der Programmierung in C und verwandten Sprachen die Darstellung des Zeichens " als Teil einer Zeichenkette. Eine Lösung besteht in der Einfügung eines zusätzlichen Zeichens. Man kann etwa in den Daten vor jedem DLE ein zweites DLE einfügen:

DLE	STX	...	DLE	DLE	ETX	...	DLE	ETX

Durch dieses so genannte Zeichenstopfen (engl. character stuffing) wird sicher gestellt, dass innerhalb des Datenfeldes nie ein einzelnes DLE erscheint. Vielmehr folgen stets 2 oder allgemein betrachtet eine gerade Anzahl von DLE Zeichen aufeinander. Der Empfänger muss dann nur aus jedem Paar von DLE Zeichen eines entfernen, um die gesendete Bytefolge zu rekonstruieren.

3.2.2 Zeichenzähler

Die zweite Methode zur Rahmenerzeugung beruht auf der Angabe der Anzahl der nachfolgenden Daten. Im einfachsten Fall wird ein Feld mit einem Zähler vor den Daten eingefügt:

DLE	STX	Anzahl	Daten

In dem Anzahlfeld steht, wie viele Bytes Daten in diesem Rahmen noch kommen werden. Die Größe des Nutzfeldes ist dann durch die Größe des Anzahlfeldes beschränkt. Steht nur ein Byte für die Größenangabe zur Verfügung, so kann das Nutzfeld maximal 256 Bytes lang sein. In dieser einfachsten Form ist die Übertragung recht fehleranfällig. Tritt ein Fehler im Anzahlfeld auf, berechnet der Empfänger die Rahmengröße falsch. Es kann unter Umständen lange dauern, bis der Empfänger sich wieder auf einen korrekten Rahmenanfang synchronisiert. Man verwendetet daher in der Regel den Zeichenzähler nur in Kombination mit anderen Methoden.

3.3 Fehlererkennung

Im Allgemeinen ist die Übertragung über einen Kanal nicht fehlerfrei. Ein Rahmen kann daher fehlerhafte Daten enthalten. Wie oben beschrieben, können Fehler im Extremfall sogar dazu führen, dass das Rahmenende nicht richtig erkannt wird und damit auch der Empfang der nachfolgenden Rahmen gestört wird. Aus diesen Gründen ist es wichtig, dass der Empfänger durch Analyse des Rahmens fest stellen kann, ob Übertragungsfehler auftraten.

Bei optimaler Ausnutzung der Übertragungskapazität entspricht jedem Bitmuster ein gültiges Zeichen. Durch Übertragungsfehler werden dann aus gültigen Zeichen auch wieder gültige Zeichen. Der Empfänger kann dann nicht erkennen ob Übertragungsfehler vorliegen. Um eine Fehlererkennung zu ermöglichen, muss die Übertragung so erweitert werden, dass Fehler zu ungültigen Zeichen (oder Rahmen) führen. Zu diesem Zweck fügt der Sender zusätzliche Informationen (Redundanz) in den Rahmen ein.

Eine Anwendung dieses Prinzips sind die Prüfzeichen bei wichtige oder leicht zu verwechselnden Informationen wie Kontonummer, Kreditkarten-Nummern oder die Internationale Standard Buchnummer (ISBN). Diese Nummern enthalten ein zusätzliches Zeichen, das nach einem vorgegebenen Algorithmus aus den anderen Stellen berechnet wird.

Auch in der sprachlichen Kommunikation setzten wir in schwierigen Fällen Redundanz ein. Ein Beispiel ist das Buchstabieren bei der Übermittlung von Namen. Noch mehr Redundanz kann man durch die Verwendung von Buchstabier–Alphabeten *A wie Anton, B wie Berta ...* erreichen. Die Eigennamen sind als längere Einheiten besser zu erkennen als die einzelnen Buchstaben.

Oft können nicht erkannte Fehler in der unteren Schicht auf einer höheren Schicht noch erkannt werden. Bei der Übermittlung einer Textdatei führen Übertragungsfehler mit einiger Wahrscheinlichkeit dazu, dass erkennbar falsche Wörter resultieren. Diese Fehlererkennung ist aber nur aufgrund der Redundanz der Sprache möglich. Nicht jede Buchstabenfolge ergibt ein Wort in der jeweiligen Sprache, so dass Sprache auch ein entsprechendes Maß an Redundanz enthält.

Die Zusatzinformation als Sicherung gegen Fehler verringert die Netto-Übertragungsrate. Daraus ergibt sich die Herausforderung, mit möglichst wenig Zusatzinformation eine zuverlässige Fehlererkennung zu ermöglichen. Man unterscheidet dabei:

- Fehlererkennung: Der Empfänger kann erkennen, dass ein Rahmen fehlerhaft übertragen wurde.

- Fehlerkorrektur: Der Empfänger kann – bis zu einem gewissen Grad – erkannte Übertragungsfehler auch korrigieren.

Verwendet man nur eine Fehlererkennung, muss der Empfänger im Fehlerfall den Sender auffordern, den Rahmen erneut zu schicken. Will man eine Fehlerkorrektur realisieren, muss mehr Sicherheitsinformation in den Rahmen eingebaut werden. Trotzdem wird man auch damit nur einen Teil der Fehler reparieren können. Enthält ein Rahmen zuviele Fehler, kann die ursprüngliche Information nicht wieder hergestellt werden.

Welches Verfahren effizienter in Hinblick auf die Sicherheit und die insgesamt erreichbare Netto-Datenrate ist hängt von dem Übertragungskanal ab. Dabei spielen sowohl die Auftrittswahrscheinlichkeit für einen Bitfehler als auch die statistische Verteilung der Fehler eine Rolle. Bei vielen Kanälen treten Fehler nicht gleichmäßig verteilt auf sondern in Gruppen (Bursts). Wenn der Kanal in einem schlechten Zustand ist – z.B. bei einer Funkverbindung in einem Schatten hinter einem Hochhaus – kommt es zu sehr viel mehr Fehlern als im Normalzustand. Dann scheitert die Fehlerkorrektur und die entsprechenden Rahmen müssen verworfen werden.

Insgesamt ist die Fehlererkennung eine komplexe Aufgabe. Die eingesetzten Methoden erfordern ein tiefes Verständnis von Statistik und Informationstheorie. Die Informationstheorie erlaubt auch Aussagen zu den theoretischen Grenzen für die Nachrichtenübertragung über einen gegebenen Kanal. Im folgenden werden ohne Anspruch auf mathematische Tiefe anhand von zwei wichtigen Beispielen einige grundsätzlichen Prinzipien der Fehlererkennung dargestellt. Bei diesen beiden Verfahren bleiben die Daten unverändert und werden um zusätzliche Sicherungsdaten ergänzt.

3.3.1 Parität

Eine weit verbreitete Methode zur Fehlererkennung ist das Einfügen von Paritätsbits. Dazu zählt man die Bits mit Wert 1 in dem Datenwort. Dann fügt man ein weiteres Bit hinzu, so dass bei der so genannten geraden Parität (even parity) insgesamt eine gerade Anzahl von Einsen entsteht. Bei ungerader Parität (odd parity) ist die Anzahl der gesetzten Bits ungerade. Für das ASCII Zeichen C als Beispiel erhält man bei gerader Parität:

Zeichen	Bitmuster	Paritätsbit
H (0x48)	100 1000	0
A (0x41)	100 0001	0
L (0x4C)	100 1100	1
L (0x4C)	100 1100	1
O (0x4F)	100 1111	1
Längsparität	100 0110	1

Abbildung 3.3: Zweidimensionale Parität

Zeichen	Bitmuster	Paritätsbit
C (0x43)	100 0011	1

Das zusätzliche Paritätsbit eignet sich gut für ASCII–Zeichen. Wenn man sich auf den 7-Bit Zeichensatz beschränkt, kann das 8. Bit als Paritätsbit verwendet werden. Die Parität hat weiterhin den Vorteil, dass die Prüfung sich in Hardware aufwandsgünstig mit hinter einander geschalteten XOR-Elementen realisieren lässt.

Von den mit 8 Bit möglichen 256 Zeichen bleiben nach bei der Paritätsprüfung nur noch 128 gültige Zeichen übrig. Dabei gilt, dass sich zwei gültige Zeichen in mindestens zwei Bitpositionen unterscheiden. Ändert man nur ein Bit, so resultiert ein ungültiges Zeichen. Man sagt, der Hamming[2]-Abstand beträgt 2. Allgemein bedeutet ein Hamming-Abstand n, dass sich zwei Zeichen in mindestens n Positionen unterscheiden.

Mit einem Paritätsbit wird ein Fehler (oder allgemein eine ungerade Anzahl von Fehlern) innerhalb eines Bytes detektiert. Zwei Fehler kompensieren sich gegenseitig und führen wieder zu einem korrekten Paritätsbit. Betrachtet man einen Block – d.h. eine Anzahl aufeinander folgender Zeichen – gemeinsam, kann man die Sicherheit deutlich erhöhen. Dazu führt man zu der beschriebenen Parität pro Zeichen eine zweite Parität pro Position in Zeitrichtung ein. Zur Unterscheidung zur einfachen (eindimensionalen) Parität spricht man auch von zweidimensionaler Parität. Abbildung 3.3 zeigt die zweidimensionale Parität am Beispiel der Zeichenfolge `HALLO`. Die Parität in Zeilen– und Spaltenrichtung nennt man Quer– und Längsparität.

Ein einzelner Fehler macht sich sowohl in einer Zeile als auch Spalte bemerkbar und kann damit korrigiert werden. Zwei Fehler – selbst in unterschiedlichen Zeilen und Spalten – können nicht mehr korrigiert werden. Allerdings lassen sich die Fehler relativ gut lokalisieren und es gibt nur zwei Möglichkeiten, wo die Fehler sein können. Liegen die beiden Fehler zusätzlich in einer Zeile stimmen alle Querparitäten. Die Fehler werden zwar durch die Längsparität detektiert, können aber nicht mehr ausgebessert werden. Allgemein lassen sich alle 3 Bit Fehler erkennen und auch die meisten 4 Bit Fehler. Nur wenn die 4 Fehler ge-

[2]Richard Wesley Hamming, amerikanischer Mathematiker, 1915 - 1998

nau an den passenden Kreuzungspunkten liegen, stimmen wieder alle Quer– und Längsparitäten.

Die Paritätsprüfung zählt die Einsen und prüft, dass sich in Summe eine gerade Anzahl ergibt. Die Übertragung umfasst einen Block Daten und eine Anzahl von Paritätsinformationen:

Datenbits	Paritätsbits

Der Empfänger kann dann durch die Prüfoperation auf dem Gesamtblock die Integrität testen. Dieses allgemeine Schema Daten – Prüfinformation – Prüfoperation lässt sich leicht verallgemeinern, indem man andere Prüfoperationen nutzt. In Internet Protokollen wird ein Verfahren mit einer Prüfsumme angewandt. Die Prüfoperation ist dabei eine Addition im Einerkomplement. Als Prüfinformation wird das Ergebnis dieser Addition an die Daten angehängt. Der Empfänger führt dann ebenfalls die Addition aus und vergleicht das Ergebnis mit der Prüfsumme. Dieses Verfahren ist einfach zu realisieren, bietet aber nicht sehr viel Schutz. Wesentlich mächtiger ist das Verfahren der zyklische Redundanzprüfung.

3.3.2 Zyklische Redundanzprüfung

Bei dem Verfahren der zyklische Redundanzprüfung (Cyclic Redundancy Check CRC) ist im Prinzip die Prüfoperation eine Division. Betrachten wir zunächst ein Beispiel mit Dezimalzahlen. Übertragen werden soll eine große Dezimalzahl wie etwa 35628828297292. Weiterhin wählt man einen Divisor D aus. Nimmt man dafür beispielsweise den Wert 17 und führt die Division aus so erhält man bei ganzzahliger Division (Modulo–Operator % in C) einen Rest von 8. Der Sender kann damit zu der großen Zahl als Prüfwert die 8 senden. Der Empfänger rechnet dann zur Kontrolle $(35628828297292 - 8)\%17$. Bleibt bei der Division ein Rest übrig, so liegt ein Übertragungsfehler vor.

An diesem einfachen Beispiel werden einige Grundeigenenschaften deutlich. Zunächst sind nicht alle Divisoren gleich geeignet. Beispielsweise wäre 10 ein schlechter Divisor, da dann nur Fehler an der letzten Stelle sich auswirken würden. Weiterhin wird ein größerer Divisor einen besseren Schutz bieten. Allerdings muss dann entsprechend mehr Platz für den Rest vorgehalten werden. Der Rest kann maximal den Wert $D - 1$ annehmen. Der Divisor selbst hat sinnvollerweise einen festen Wert und braucht dann nicht mehr übertragen zu werden.

Die Übertragung dieses Ansatzes auf Binärdaten beruht auf der Interpretation von Bitfolgen als Polynomen. Für eine Bitfolge $abcdef$ schreibt man

$$M(x) = a \times x^5 + b \times x^4 + c \times x^3 + d \times x^2 + e \times x^1 + f \times x^0 \qquad (3.1)$$

Dabei genügt es, die Terme mit Einsen anzugeben. Für zum Beispiel die Bitfolge 100101 erhält man dann

$$M(x) = x^5 + x^2 + x^0 \qquad (3.2)$$

Die Prüfbits werden durch eine Division von solchen Polynomen bestimmt. CRC benutzt eine polynomiale Modulo-2 Arithmetik. In dieser Arithmetik reduziert sich die Subtraktion auf eine XOR–Operation. Die Division lässt sich – wie bei der üblichen Division – durch fortgesetzte Subtraktion ausführen.

Sei als Beispiel x^3+x+1 das Divisor-Polynom und damit 1011 das zugehörige Bitmuster. Übertragen werden soll die Nachricht 110101101. Zunächst wird die Nachricht entsprechende der höchsten Potenz im Divisor um 3 Bits erweitert: 110101101*000*. Genau in diese zusätzlichen Bits kommen später die Prüfbits. Der erste Schritt in der Division ist dann

1	1	0	1	0	1	1	0	1	0	0	0
1	0	1	1								
	1	1	0								

Das Ergebnis der Division wird nicht benötigt und daher nicht notiert. Dann wird die nächste Stelle von oben übernommen und der Divisor erneut subtrahiert:

1	1	0	1	0	1	1	0	1	0	0	0
1	0	1	1	↓							
	1	1	0	0							
	1	0	1	1							
		1	1	1							

Insgesamt ergibt sich

1	1	0	1	0	1	1	0	1	0	0	0
1	0	1	1	↓							
	1	1	0	0							
	1	0	1	1	↓						
		1	1	1	1						
		1	0	1	1	↓					
			1	0	0	1					
			1	0	1	1	↓	↓			
				0	1	0	0	1			
					1	0	1	1	↓	↓	
							1	0	0	0	
							1	0	1	1	↓
									1	1	0

Bei der Division bleibt ein Rest von 110. Zieht man diesen Rest von der erweiterten Zahl ab, so ist die neue Zahl ohne Rest durch den Divisor teilbar. Da Subtraktion einer XOR–Operation entspricht, erhält man 110101101*110*. Diese um die 3 Prüfbits erweiterte Bitfolge wird gesendet. Der Empfänger führt dann ebenfalls die Division durch. Falls dabei ein Rest übrig bleibt, liegt ein Übertragungsfehler vor.

Wie bei dem Beispiel mit den Dezimalzahlen gibt es auch für die Polynom-

division mehr oder weniger gut geeignete Divisor–Polynome. In der Literatur – beziehungsweise in den entsprechenden Spezifikationen – findet man geeignete Polynome. Man bezeichnet die Polynome in diesem Zusammenhang auch als Generator–Polynome. Einige davon sind in Tabelle 3.1 angegeben. Die Zahl nach dem Namen gibt an, wie viele Prüfbits jeweils genutzt werden. Details der Implementierung zusammen mit einer Realisierung in der Programmiersprache C findet man beispielsweise in [6].

CRC	Divisor
CRC-8	$x^8 + x^2 + x^1 + 1$
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + 1$
CRC-16	$x^{16} + x^{15} + x^2 + 1$

Tabelle 3.1: Beispiele für gebräuchliche CRC–Polynome

Die Prüfverfahren schützen in allen Fällen, in den die Anzahl der Bitfehler kleiner ist als die Anzahl der Prüfbits. Darüber hinaus werden die „meisten“ anderen Fehler erkannt.

3.4 Sichere Übertragung von Rahmen

Wenn der Sender einen Rahmen abgeschickt hat, dauert es eine Weile bis der Empfänger den Rahmen erhalten hat und die Korrektheit festgestellt bzw. wieder hergestellt hat. Der Empfänger kann dann den Empfang quittieren indem er dem Sender eine Bestätigung schickt. Insgesamt ergibt sich folgender Ablauf:

1. Sender schickt Rahmen
2. Empfänger wartet bis der Rahmen vollständig angekommen ist
3. Empfänger prüft Korrektheit
4. Empfänger schickt Bestätigung
5. Sender erhält Bestätigung

Dieser Zusammenhang ist in Bild 3.4 mit einem Zeitstrahl graphisch dargestellt. R1 bezeichnet den ersten Rahmen und ACK die Bestätigung (Acknowledgment). In diesem Bild ist beim Empfänger eine kleine Zeitspanne nach Erhalt des Rahmens zur Prüfung eingetragen. Diese Zeit wird zwar im Prinzip immer benötigt. Im folgenden wird aber zur Vereinfachung der Darstellung diese Zeit nicht mehr explizit eingetragen.

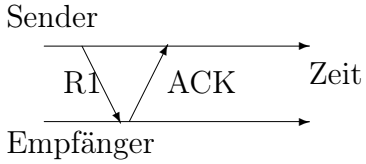


Abbildung 3.4: Zeitlicher Ablauf bei Bestätigung eines Rahmens R1

3.4.1 Stop–and–Wait Algorithmus

Eine direkte Implementierung des beschriebenen Verfahrens ist der Stop–and–Wait Algorithmus. Der Sender schickt einen Rahmen und wartet auf die Bestätigung. Erhält er innerhalb einer gewissen Wartezeit keine Bestätigung, so geht er davon aus, dass der Rahmen verloren gegangen ist und schickt ihn erneut. Die Wartezeit bezeichnet man als *Timeout*. Damit ergibt sich Bild 3.5.

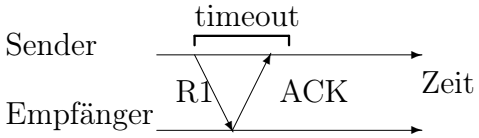


Abbildung 3.5: Zeitlicher Ablauf bei Stop–and–Wait Algorithmus

Der Timeout soll dem Empfänger genügend Zeit zur Bestätigung lassen. Andererseits soll er aber so kurz wie möglich sein, um die Wartezeit bei einem verloren Rahmen zu minimieren. Ein Problem tritt auf, wenn die Bestätigung zu spät – oder gar nicht – ankommt. Beim Sender ist mittlerweile der Timeout abgelaufen und in der Annahme, dass der Rahmen nicht korrekt empfangen wurde, schickt der Sender den Rahmen erneut. Der Empfänger hat andererseits den Rahmen bestätigt und erwartet nun bereits den nächsten Rahmen. Ohne weiter Maßnahmen würde er den zum zweiten Mal geschickten Rahmen bereits als nächsten Rahmen interpretieren.

Man benötigt daher eine Kennung für die einzelnen Rahmen. Beim Stop–and–Wait Algorithmus verwendet man in der Regel die einfachste Nummerierung mit 0 und 1. Falls ein Rahmen fälschlicherweise erneut geschickt wird, erkennt dies der Sender an der Nummer des Rahmens. Er kann daher den Rahmen ignorieren, muss aber natürlich eine Bestätigung schicken. Den vollständigen Ablauf zeigt Bild 3.6. In diesem Beispiel geht die Bestätigung für den Rahmen 0 verloren. Der Sender schickt daher nochmals den Rahmen 0. Diesmal wird die Bestätigung ordnungsgemäß zugestellt und der Sender geht zum nächsten Rahmen, der die Kennung 1 erhält.

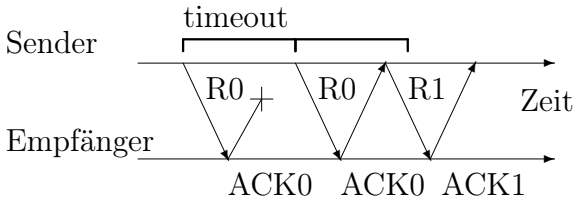


Abbildung 3.6: Stop–and–Wait Algorithmus mit Rahmennummer, Verlust einer Bestätigung.

3.4.2 Sliding–Window Algorithmus

Im Stop–and–Wait Algorithmus gibt es lange Pausen, in denen der Sender auf eine Bestätigung wartet. Daher wird der Kanal nur schlecht ausgenutzt. Eine bessere Auslastung wird erreicht, indem der Empfänger bereits während der Wartezeit weitere Rahmen schickt. Zu jedem Zeitpunkt sind dann mehrere Rahmen auf der Leitung unterwegs und der Sender wartet auf entsprechend viele ausstehende Bestätigungen.

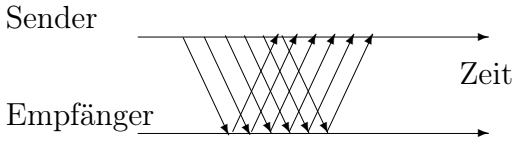


Abbildung 3.7: Sliding–Window Algorithmus

Der Sender nummeriert die Rahmen, so dass sie individuell bestätigt werden können. Die maximale Anzahl von gleichzeitig auf dem Weg befindlichen Rahmen – das Sendefenster (Send Window Size, SWS) – kann bei bekannter Leitungscharakteristik an die RTT der Leitung angepasst werden. Im Idealfall wird das Fenster so gewählt, dass der Timeout für den ersten Rahmen unmittelbar nach Abschicken des letzten Rahmens im Sendefenster abläuft. Der Sender muss alle Rahmen im Sendefenster vorrätig halten, um sie bei Bedarf nochmals senden zu können. Kommt die Bestätigung für den ersten Rahmen an, kann der Empfänger diesen Rahmen löschen und das Sendefenster weiter schieben. Anderenfalls sendet er diesen Rahmen erneut.

Im allgemeinen wird es vorkommen, dass zwar ein Rahmen verloren geht, aber die nachfolgenden Rahmen trotzdem ankommen. Eine gute Kanalauslastung erreicht man, wenn jeder angekommene Rahmen bestätigt wird (selektive Bestätigung, engl. selective acknowledgements). Allerdings wird dann der Aufwand in der Verwaltung des Sendefensters und des Empfangsfensters kompliziert. Geht beispielsweise ein bestimmter Rahmen mehrfach verloren, dann muss der Empfänger sehr viele neue Rahmen speichern, um diesen alten Rahmen an der richtigen Stelle

einfügen zu können. Wenn erforderlich, kann der Empfänger den Sender bremsen, indem er Rahmen, die zu weit vor dem letzten fehlenden Rahmen liegen, nicht bestätigt oder zumindest die Bestätigung verzögern.

Die Arbeit des Senders wird wesentlich vereinfacht, wenn der Empfänger die aus der Reihe angekommenen Rahmen nicht bestätigt. Angenommen Rahmen 4 geht verloren oder wird stark verzögert. Der Empfänger erhält statt dessen bereits die folgenden Rahmen 5 und 6. Er bestätigt diese Rahmen aber nicht sondern wartet auf den Rahmen 4 – entweder das verspätete Original oder die wieder gesendete zweite Version. Sobald der Rahmen 4 ankommt, bestätigt der Empfänger den Rahmen 6. Bei entsprechendem Protokoll erkennt der Sender daraus, dass alle Rahmen bis zur Nummer 6 gut angekommen sind (kumulative Bestätigung). Er kann dann das Sendefenster auf die Position 7 weiter schieben.

Wenn tatsächlich Rahmen verloren gehen, wird bei diesem Verfahren der Kanal nicht optimal ausgenutzt. Der Sender schickt während des Wartens nach dem wiederholten Rahmen unnötigerweise auch die nachfolgenden Rahmen zum zweiten Mal. Andererseits ist in dieser Form die Verwaltung der Rahmen in Sende– und Empfangsfenster recht einfach. Die Nummer eines Rahmen (Sequenznummer) ist eine zusätzlich zu übertragende Information. Aus Effizienzgründen sollten möglichst wenige Bits für dies Nummer verbraucht werden. Die Sequenznummer wird daher auf einen relativ kleinen Bereich beschränkt. Sobald das Maximum erreicht ist, springt der Zähler wieder auf 0 zurück. Es muss allerdings gewährleistet sein, dass die Rahmen noch eindeutig identifiziert werden können.

3.5 Übungen

Übung 3.1 *Die Zeichenkette* `Friedberg` *soll übertragen werden. Zur Sicherung gegen Übertragungsfehler werden Paritätsinformationen eingebaut. Ergänzen Sie in der Tabelle die Bits für Querparität sowie das zusätzliche Byte für die Längsparität. Verwenden Sie in beiden Fällen gerade Parität.*

Querparität	*Binär-Darstellung*	*Zeichen*
	100 0110	*F*
	111 0010	*r*
	110 1001	*i*
	110 0101	*e*
	110 0100	*d*
	110 0010	*b*
	110 0101	*e*
	111 0010	*r*
	110 0111	*g*
		Längsparität

Übung 3.2 *Wie groß ist der Hamming-Abstand bei zweidimensionaler Parität?*

Übung 3.3 *Berechnen Sie für das Divisor-Polynom $x^3 + x^2 + 1$ die Prüfzeichen zu der Nachricht 101011011.*

Übung 3.4 *Die Prüfziffer der Internationalen Standard Buchnummer ISBN wird nach folgendem Algorithmus berechnet: Zunächst wird aus den 9 Ziffern der eigentlichen Kennung eine gewichtete Summe bestimmt. Dazu wird die erste Ziffer der ISBN mit 10 multipliziert, die zweite mit 9, die dritte mit 8 usw. Die Produkte werden addiert. Für die Summe wird der Rest bei Division durch 11 bestimmt (Modulus 11). Dieser Rest wird als Prüfziffer angehängt. Der Sonderfall Rest 10 wird durch ein X als Prüfziffer markiert. Implementieren Sie diesen Prüfalgorithmus beispielsweise als C-Programm oder in Excel. Testen die Korrektheit an Hand einiger Beispiele.*

Übung 3.5 *Das nachfolgende C-Programm simuliert einen Kanal mit Bit-Fehlern. Dazu werden aus einer Textdatei* `test.txt` *nacheinander alle Zeilen gelesen. Aus jeder Zeile wird ein Rahmen erstellt. In der Grundversion wird dabei lediglich die Zeichenkette kopiert. Der Rahmen wird dann an die Kanalsimulation übergeben. Dort werden nach einer vorgegebenen Bitfehler-Wahrscheinlichkeit Übertragungsfehler in den Rahmen eingefügt. Aus dem Rahmen wird dann wieder der Text extrahiert und in eine Datei mit dem Namen* `uebertragen.txt` *geschrieben.*

1. *Untersuchen Sie mit der Grundversion den Einfluss der Bitfehler auf die Rahmenfehler, d.h. die Anzahl der fehlerhaften Rahmen (Zeilen). Erstellen Sie eine graphische Darstellung der Abhängigkeit zwischen der Häufigkeit von Bitfehlern und Rahmenfehlern.*

2. *Fügen Sie in den vorgesehenen Funktionen eine Paritätsprüfung ein. Wie wirken sich Quer- und Längsparität aus? Wie oft müssen Rahmen wiederholt werden?*

3. *Integrieren Sie eine CRC-Prüfung. (Hinweis: eine Implementierung ist auf der Seite* `www.w3.org/TR/PNG-CRCAppendix.html` *des World Wide Web Consortiums angegeben.)*

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <math.h>

#define MAX_ZEICHEN 200

/* Kanalsimulation
 * Verarbeite len Daten aus buffer
```

```
 * Mit der Wahrscheinlichkeit prob wird ein Bit verfälscht.
 * Rückgabewert: Anzahl der veränderten Bits
 */
int kanal( char *buffer, int len, double prob ){
    int i, j, limit;
    unsigned int pos;
    int count = 0;

    /* Um auch kleine Wahrscheinlichkeiten behandeln zu können,
     * werden zwei Zufallszahlen erzeugt. Nur wenn beide kleiner
     * als Wurzel(prob) sind wird das jeweilige Bit gedreht. */
    limit = (int) (sqrt(prob) * RAND_MAX);
    for( i=0; i<len; i++ ) {    /* über alle Zeichen */
        pos = 1;                /* Bit-Maske */
        for( j=0; j<8; j++ ) {  /* über alle 8 bits */
            if( rand() < limit && rand() < limit) {
                /* Bit mittels XOR umdrehen */
                buffer[i] ^= pos;
                ++count;
            }
            pos <<= 1;  /* Maske für nächstes Bit */
        }
    }
    return count;
}

void bildeRahmen( char *rahmen, char *zeile ) {
    /* Hier wird ein Rahmen gebildet */
    strcpy( rahmen, zeile );
}

void leseRahmen( char *erhalten, char *rahmen ) {
    /* Hier wird der Text aus dem Rahmen geholt */
    strcpy( erhalten, rahmen );
}

int rahmenFalsch( char *rahmen ) {
    /* Bis jetzt sind keine Rahmen falsch */
    return 0;
}


void main(int argc, char* argv[])
```

```
{
    double prob = 0.025;   /* Bitfehler-Wahrscheinlichkeit */
    int len;
    int rahmenFehler = 0;
    int c_char = 0, c_fehler = 0;
    char zeile[MAX_ZEICHEN],
        rahmen[MAX_ZEICHEN],
        erhalten[MAX_ZEICHEN];
    FILE *fp_in, *fp_out;

    /* Initialisierung des Zufallszahlengenerators */
    srand( (unsigned)time( NULL ) );

    fp_in  = fopen( "test.txt", "r" );
    fp_out = fopen( "uebertragen.txt", "w" );
    while( fgets( zeile, sizeof zeile, fp_in ) ) {
        len = strlen( zeile );
        do {
            bildeRahmen( rahmen, zeile );
            c_fehler += kanal( rahmen, len, prob );
            c_char   += len * 8;
        } while( rahmenFalsch( rahmen ) );

        leseRahmen( erhalten, rahmen);
        if( strcmp( erhalten, zeile ) ) ++rahmenFehler;
        fprintf( fp_out, "%s", rahmen );
    }
    printf( "Anzahl von Bitfehlern:    %d, relativ %f%%\n",
        c_fehler, 100.*(double) c_fehler / c_char );
    printf( "Anzahl von Rahmenfehlern: %d\n",
        rahmenFehler );

}
```

Übung 3.6 *Sie wollen eine Datei von 3 MByte von Frankfurt nach New York (8000 km inklusive Umwegen) übertragen. Berechnen Sie die Dauer vom Beginn der Übertragung (Senden des ersten Zeichens) bis zum Ende (Empfang des letzten Zeichens) für eine Leitung mit 1 Mbit/s. Auf der Strecke befinde sich in der Mitte (bei 4000 km) ein Knoten, der die korrekte Übertragung prüft. Dazu nimmt er Pakete von jeweils 1024 Bytes an. Dann benötigt er 1 ms zur Paritätsprüfung jedes Pakets. Anschließend verschickt er das Paket. Sobald er das letzte Zeichen verschickt hat, ist er zum Empfang des nächsten Pakets bereit. Dann schickt er eine Bestätigung (8 Bytes) an den Sender, damit dieser das nächste Paket*

absenden kann. Welche Zeit benötigt nun die Übertragung? Hinweis: Zeichnen Sie ein Zeitstrahldiagramm mit Sender, Knoten und Empfänger.

*Verwenden sie die Lichtgeschwindigkeit c=300000 km/s als Ausbreitungsgeschwindigkeit. Rechnen Sie bei der Dateigröße mit 1 MByte = 1024*1024 Byte.*

Kapitel 4

Ethernet & Co

In diesem Kapitel wird die Funktionsweise von lokalen Netzen beschrieben. Unter den LAN–Technologien ist Ethernet derzeit am weitesten verbreitet. Eine aktuelle Technologie mit wachsender Bedeutung sind LANs auf der Basis von Funktnetzen. Daneben gibt es für Spezialanwendungen eine große Anzahl von weiteren Technologien.

4.1 Ethernet

Unter der Bezeichnung Ethernet fasst man eine Familie von Netztechnologien zusammen. Den verschiedenen Technologien gemeinsam ist die Adressierung, das Rahmenformat und die Zugriffskontrolle auf das Übertragungsmedium. Ethernet wurde in den siebziger Jahren im PARC (Palo Alto Research Center), dem Forschungslabor der Firma XEROX, entwickelt. Später wurde daraus in Zusammenarbeit mit den Firmen DEC und Intel ein offener Standard entwickelt. Dieser wiederum bildete die Grundlage für den offiziellen IEEE-802.3 Standard.

IEEE (sprich *Eye-triple-E*) ist die Abkürzung für Institute of Electrical and Electronics Engineers. Von dieser Organisation (www.ieee.org) werden neben vielen anderen Aktivitäten Standards entwickelt. Die dreistellige Zahl im Namen eines Standards bezeichnet das allgemeine Thema. Die Nummer 802 umfasst diverse Standards für *Local and Metropolitan Area Networks (LAN/MAN)*. Durch die Zahl hinter dem Punkt (*Part*) wird der Standard genauer bezeichnet. Die Gruppe 802.3 trägt den Titel *CSMA/CD Access Method*. Ein anderes Beispiel ist die Gruppe 802.11 *Wireless LANs*. Einzelne Standards innerhalb einer Gruppe werden durch angehängte Buchstaben oder Ziffern sowie eine Jahreszahl unterschieden (z.B. 802.11g-2003, 802.15.3-2003).

Der IEEE-802.3 Standard umfasst wie gesagt eine ganze Familie von Techniken. Die jeweiligen Namen werden nach dem Schema

<Datenrate><Übertragungsverfahren><Segmentlänge oder Kabeltyp>

gebildet. So bezeichnet 10BASE5 eine Variante mit 10Mbit/s Basisband-Übertragung und einer maximalen Segmentlänge von 500m.

4.1.1 Adressen

Die Adressen für Ethernet bestehen aus 6 Byte. Üblich ist die Angabe der einzelnen Bytes mit je 2 Hex-Zeichen, getrennt durch Doppelpunkte. Führende Nullen werden oft weggelassen. Das Programm `ipconfig` gibt die Adresse mit allen Nullen und Bindestrichen aus.
Beispiele:
8:34:3e:0:55:a2
00-D1-59-6B-88-63

Jeder Rechner oder genauer gesagt jede Netzwerkkarte in der Welt hat eine eindeutige Ethernet-Adresse. Die Adresse enthält einen Anfangsteil, der den Hersteller kennzeichnet. Das erste Bit hat eine besondere Bedeutung. Ist es gesetzt, handelt es sich nicht um eine individuelle Adresse sondern die Adresse einer ganzen Gruppe von Rechnern (Multicast). Der Extremfall ist eine Adresse, die nur Einsen enthält. Mit dieser Adresse (Broadcast) werden alle Rechner gleichzeitig angesprochen.

4.1.2 Rahmenformat

Die Rahmen im Standard IEEE-802.3 haben folgendes Format:

8 Byte	6	6	2	46-1500	4
Präambel	Zieladresse	Quelladresse	Länge	Daten	CRC

Die Präambel enthält im wesentlichen eine Folgen von abwechselnden Nullen und Einsen und erlaubt den Empfängern die Synchronisation auf den Rahmen. Anschließend folgen die Adressen von Ziel(en) und Sender. Bei IEEE-802.3 gibt das nächste Feld die Größe der darauf folgenden Daten an. Dabei gilt eine Mindestgröße von 46 Byte und eine maximale Größe von 1500 Byte. Im ursprünglichen Ethernet–Standard enthält dieses Feld eine Kennung für den Rahmentyp. Wenn man sich für die Typkennzeichnung auf Werte größer als 1500 beschränkt, können beide Rahmentypen ohne Verwechslungsgefahr parallel verwendet werden. Nach den Daten folgen noch die Prüfbits einer 32 Bit CRC.

Diese Darstellung ist etwas vereinfacht. Die Realität ist etwas komplizierter und es gibt mehrere Ausprägungen von 802.3-Rahmen. Auf die Details soll hier nicht näher eingegangen werden. Es sei lediglich erwähnt, dass die ersten drei Bytes der Nutzlast als Information für die so genannte Logical Link Control (LLC) gemäß IEEE-802.2 dienen.

4.1.3 Medienzugriff

Ethernet wurde für die gemeinsame Nutzung eines Mediums durch viele Stationen (Rechner) entwickelt (Mehrfachzugriffsnetz). Der erste Vorläufer war ein Funknetz zwischen den Hawaii–Inseln mit dem Namen Aloha. In diesem Fall war die Atmosphäre das gemeinsam genutzte Medium. Bei Ethernet war es ursprünglich ein Kabel, über das die Rechner in einer Bus-Topologie verbunden waren.

Der gemeinsame Zugriff (Medienzugriffssteuerung, engl. Media Access Control MAC) wird über die Methode CSMA/CD (Carrier Sense Multiple Access with Collision Detection) geregelt. Das Grundprinzip ist einfach: jede Station kann feststellen, ob die Leitung momentan frei ist (Carrier Sense). Wenn eine Station einen Rahmen senden möchte und die Leitung frei ist, beginnt sie sofort mit der Übertragung. Die Übertragungsdauer ist durch die maximale Datenmenge von 1500 Bits begrenzt. Alle Stationen hören permanent die Leitung ab. Entdecken sie einen Rahmen, der für sie bestimmt ist, übernehmen sie ihn.

Jede Station entscheidet selbständig, wann sie senden will. Damit kann es passieren, dass zwei Stationen in der Annahme, dass die Leitung frei ist, gleichzeitig anfangen zu senden. Die beiden Rahmen kollidieren auf der Leitung und werden damit unbrauchbar. Die Stationen erkennen die Kollision (Collision Detection), brechen die Übertragung ab und schicken ein 32 Bit langes Stausignal.

Nach einiger Zeit versuchen die Stationen erneut, ihre Rahmen zu schicken. Um zu verhindern, dass wieder beide Stationen gleichzeitig senden, wird die Wartezeit zufällig variiert. Im ersten Wiederholungsversuch beträgt die Wartezeit 51,2 μs und jede Station entscheidet zufällig, ob sie 0 μs oder 51,2 μs wartet. Falls es erneut zu einer Kollision kommt, verdoppelt sich die maximale Wartezeit und jede Station wartet zufällig für 0, 51,2, 102,4 oder 153,6 μs. Diese Verfahren der Verdopplung der Wartezeit (exponential backoff) wird bis zu einer gegebenen Maximalzahl von Versuchen angewandt. Im Versuch i wählt jede Station zufällig eine Wartezeit aus der Menge $0, 51, 2, \ldots, (2^i - 1) * 51, 2$ aus. In der Praxis wird allerdings i nicht größer als 10 und nach 16 erfolglosen Versuchen bricht der Netzwerkadapter mit einer Fehlermeldung ab. Mit dieser Strategie werden einerseits Sendekonflikte zwischen zwei oder einigen wenigen Stationen schnell geklärt. Andererseits löst sich auch in dem Fall, dass viele Stationen gleichzeitig senden wollen, nach einigen Verdoppelungen der Gesamtwartezeit der Stau rasch auf.

Aus der CSMA/CD Strategie ergeben sich Grenzwerte für die erlaubte maximale Laufzeit und die Rahmengröße. Betrachten wir dazu den extremen Fall zweier Stationen A und B an den entgegengesetzten Enden der Leitung. Station A sendet einen Rahmen. Der Rahmen erreicht nach der Latenz τ die Station B. Für B war bis dahin die Leitung frei. Im ungünstigsten Fall hat B daher unmittelbar bevor dieser Rahmen bei ihm eintrifft selbst mit der Übermittlung begonnen. B entdeckt sofort den Konflikt und bricht die Übertragung ab. Allerdings dauert es wiederum die Zeit τ bis der von B begonnene Rahmen bei A eintrifft. Damit

Tabelle 4.1: Kabeltypen für 10Mbit/s Ethernet

Bezeichnung	Typ	Maximale Länge	Bemerkung
10Base5	Dickes Koaxialkabel (ThickWire)	500m	Kabel wird für Anschlüsse (Transceiver) angebohrt
10Base2	Dünnes Koaxialkabel (ThinWire)	200m	Anschluss über T-Stück
10Base-T	Verdrilltes Paar (Twisted pair)	100m	Punkt–zu–Punkt Verbindungen
10Base-F	Glasfaser	2000m	

A seinen Rahmen noch zurück nehmen kann, darf A zu diesem Zeitpunkt noch nicht mit der Übertragung fertig sein. In Summe muss daher die Übertragung eines Rahmens stets mindestens die längste RTT 2τ innerhalb des Netzes dauern. Die minimale Übertragungsverzögerung muss die Beziehung

$$t_U = K/B \geq 2\tau \tag{4.1}$$

erfüllen. Bei der Übertragungsrate 10 Mbit/s und einer minimalen Rahmengröße von 64 Bytes beziehungsweise 512 bits ergibt sich als maximale RTT 51,2μs. Erhöht man die Übertragungsrate beispielsweise auf 100 Mbit/s so reduziert sich die maximale RTT bei gleichen Rahmen auf 5,12μs.

4.1.4 Physikalische Eigenschaften

In der ursprünglichen Form wird Ethernet über ein Koaxialkabel (10Base5) in Bus–Topologie betrieben. Das Kabel hat eine maximale Länge von 500 m. Direkt an den Kabeln werden Transceiver angebracht (Mindestabstant 2,5 m), von denen aus Kabeln zu den Rechnern führen. Als preisgünstiger Alternative wird dünneres Koaxialkabel (10Base2) verwendet, bei dem eine Abzweigung über ein T–Stück realisiert wird. Zwei Kabelsegmente können über einen Repeater verbunden werden.

Die moderne Form 10Base-T basiert auf verdrillten Kabelpaaren (twisted pair). Damit sind nur noch Punkt-zu-Punkt-Verbindungen möglich. Mehrere Stationen werden dann auf einen mehrwegigen Repeater (Sternkoppler oder Hub (engl. Nabe, Mittelpunkt)) geführt. Die einzelnen Kabeltypen sind in Tabelle 4.1.4 zusammen gestellt. Repeater und Hubs arbeiten auf Bitebene und leiten die Signale ohne Analyse der Adressen weiter. Dann bleibt – unabhängig von der technischen Realisierung – das Netz eine logische Einheit. Jede Nachricht wird über das gesamte Netz verteilt, so dass Kollisionen an beliebiger Stelle auftreten können. Das Netz bildet eine Kollisionsdomäne (collision domain).

Einen höheren Datedurchsatz erreicht man durch Trennung eines Netzes in mehrere Teilnetze, die durch vermittelnde Geräte wie Switches oder Router verbunden sind. Diese Geräte leiten Pakete nur an die Teilnetze weiter, in denen sich die Zielknoten befinden. Dadurch werden die anderen Teilnetze von unnötigem Verkehr frei gehalten. Der Durchsatz lässt sich dann durch geeignete Topologien weiter optimieren. So ist es sinnvoll, Stationen die viel miteinander kommunizieren, an ein gemeinsames Teilnetz anzuschließen. Weiterhin können Stationen mit sehr hohem Datenaufkommen einem eigenen Port am Switch erhalten, so dass ihnen die Bandbreite exklusiv zur Verfügung steht. Ein Leistungssteigerung erreicht man durch Full Duplex Betrieb mit getrennten Kanälen für die beiden Richtungen.

Aufbauend auf dem Standard für 10 Mbit/s wurden mit 100 Mbit/s Fast Ethernet und 1000 Mbit/s Gigabit Ethernet zwei Nachfolgetechnologie mit höheren Bandbreiten entwickelt. Die beiden Verfahren benötigen Punkt zu Punkt Verbindungen, wobei sowohl Twisted Pair als auch Lichtleiterkabel eingesetzt werden können. Die weitgehende Kompatibilität erleichtert die Migration bestehender Ethernet-Netze zu den höheren Datenraten.

4.1.5 Bewertung

Ethernet ist eine überaus erfolgreiche Technologie. Die Bandbreite wird zwar nicht optimal ausgenutzt, aber solange die Belastung nicht zu hoch ist (kleiner 30%), ist der Verlust durch die Kollisionen nicht störend. Vorteilhaft ist die Einfachheit von Installation und Betrieb. Es ist kein Problem, Stationen einzufügen oder aus dem Netz zu nehmen. Auch der Ausfall einzelner Stationen hat in der Regel keine negativen Auswirkungen auf den Netzbetrieb. Durch das robuste Protokoll gibt es auch kaum Situationen, in denen eine einzelne Station durch Fehlfunktionen das ganze Netz in Mitleidenschaft zieht.

Für normale Computer-Anwendungen wie etwa Fileservice ist Ethernet gut geeignet. Schwierig sind zeitkritische Anwendungen, die enge Anforderungen an die Reaktionszeit stellen. Darunter fallen auch Applikationen wie etwa Telefonverbindungen, bei denen Schwankungen in der Laufzeit sich störend bemerkbar machen. Durch den zufälligen Charakter des Medienzugriffs gibt es keine Garantie, dass ein Rahmen innerhalb einer vorgegebenen Zeit sein Ziel erreicht. In der Praxis kann man diese Probleme weitgehend vermeiden, indem man das Netzwerk nicht zu stark belastet.

Ethernet ist die bei weitem am häufigsten eingesetzte LAN Technologie. Die Tendenz geht zu immer höheren Datenraten, wobei aber stets noch die Kompatibilität mit den älteren Techniken gewahrt bleibt.

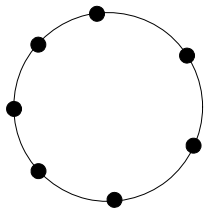


Abbildung 4.1: Netzwerk mit Ring-Topologie

4.2 Token Ring

Im Zugriffsverfahren CSMA/CD können die Rechner zu jeder Zeit versuchen, eine Nachricht zu senden sofern die Leitung frei ist. Die dadurch immer wieder auftretenden Kollisionen reduzieren den Durchsatz. Eine grundsätzliche Alternative ist ein geregelter Zugriff auf das Medium, bei dem zu jedem Zeitpunkt nur jeweils eine Station das Senderecht hat. Eine flexible Regelung basiert auf der Verwendung eines „Tokens“ (engl. für Zeichen, Erkennungsmarke). Nur die Station, die jeweils den Token besitzt, darf senden. Die Details des Medienzugriffs werden dann durch den Umlauf des Tokens sowie die Regeln zur Verwendung des Tokens bestimmt. Eine vergleichende Analyse der Leistungsfähigkeit von CSMA/CD und Token Ring findet man in [7].

Die erfolgreichste Technik auf dieser Basis sind Token-Ring-Netze. Weit verbreitet ist der IBM–Token-Ring, der weitgehend identisch ist mit dem IEEE–Standard 802.5. Ein neuerer Standard ist FDDI (Fiber Distributed Data Interface).

Bei dem Token–Ring sind alle Stationen – zumindest logisch – in Form eines Ringes angeordnet (Bild 4.1). Die Daten werden stets nur in eine Richtung weiter gegeben. Jede Station hat genau eine Vorgängerin und eine Nachfolgerin. Die Stationen werden über spezielle Anschaltvorrichtungen an den Ring angeschlossen. Fällt eine Station aus oder wird sie vom Netz genommen, schließt ein Relais wieder den Ring.

4.2.1 Medienzugriff

Der Token ist eine spezielle Nachricht, die zunächst frei im Ring kreist. Möchte eine Station senden, so nimmt sie den Token aus dem Ring und schickt statt dessen ihre Nachricht in einem oder mehreren Rahmen. Ähnlich wie bei Ethernet enthält jeder Rahmen die Zieladresse. Die Rahmen laufen dann im Netz von Station zu Station. Die Zielstation kopiert die Nachricht, nimmt die Rahmen aber nicht aus dem Netz sondern lässt sie weiter kreisen. Erst die sendenden Station selbst nimmt die Nachricht wieder aus dem Netz. Anschließend gibt sie den Token weiter, so dass die nachfolgende Station die Gelegenheit zum Senden hat.

Für eine sichere Übertragung ist ein Protokoll mit zwei Bits im Rahmen vorgesehen. Die beiden Bits mit der Bezeichnung A und C werden vom Sender auf 0 gesetzt. Erkennt eine Station einen an sie gerichteten Rahmen, setzt sie das A–Bit. Bei erfolgreicher Kopie setzt sie zusätzlich auch das C–Bit. Der Sender kann auf diese Art erkennen, ob

1. der Empfänger aktiv ist
2. der Empfänger den Rahmen übernehmen konnte

Wenn eine Station den Token übernommen hat, kann sie ihre Daten verschicken. Optimal im Sinne der Netznutzung wäre es, diese Station ihre gesamten Daten direkt hintereinander schicken zu lassen. Insbesondere bei ansonsten geringen Aktivitäten könnte man die Wartezeit, bis der Token jeweils umgelaufen ist, einsparen. Dann hätten allerdings die anderen Stationen keine Chance, in der Zwischenzeit eigene Nachrichten zu versenden. Ein umfangreicher Transfer würde dann auch viele kleine Nachrichten blockieren. Daher wird die Zeit, für die eine Station den Token behalten darf (THT, Token Hold Time), beschränkt. Vor jedem weiteren Rahmen prüft die Station, ob sie diesen Rahmen noch innerhalb der THT schicken kann. Anderenfalls hält sie die Nachricht zurück und gibt den Token weiter. Im Standard 802.5 beträgt die THT 10 ms.

Damit kann man die obere Grenze für die Wartezeit, bis eine Station wieder an die Reihe kommt, angeben. Im ungünstigsten Fall (*worst case*) übernimmt jede Station den Token und nutzt ihre THT voll aus. Für den Umlauf des Tokens (TRT, Token Rotation Time) gilt damit

$$TRT \leq AnzahlStationen \times THT + RingLatenz \tag{4.2}$$

Das Protokoll gibt allen Stationen eine gleiche Chance zum Senden. Die maximale Dauer für eine Übertragung (ohne Wiederholung von Rahmen aufgrund von Fehlern) kann aus der oberen Abschätzung 4.2 für TRT bestimmt werden.

4.2.2 Netz–Überwachung

Der Betrieb eines Token-Ring-Netzes erfordert eine Überwachung oder Wartung während des Betriebs. Einige mögliche Fehlerfälle sind:

- Der Token geht aufgrund von Bitfehlern verloren
- Die Station im Besitz des Tokens stürzt ab
- Rahmen werden verfälscht und kreisen ewig im Ring
- Ein Rahmen „verwaist“ weil die Sendestation abstürzt, bevor sie ihn wieder aus dem Ring nimmt

Tabelle 4.2: Standards für WLAN

802.11	erster Standard von 1997	2 bis 3 Mbit/s, 2,4 GHz-Band
802.11b	Nachfolger von 802.11	bis 11 Mbit/s, 2,4 GHz-Band
802.11g	Nachfolger von 802.11b	bis 54 Mbit/s, 2,4 GHz-Band
802.11a	Nachfolger von 802.11b	bis 54 Mbit/s, 5 GHz-Band

Daher übernimmt eine der Stationen im Netz die Funktion eines Monitors. Jede Station kann Monitorstation werden. Es gibt ein wohl definiertes Protokoll, nach dem die Stationen bei der Inbetriebnahme des Rings oder nach Ausfall der Monitorstation aushandeln, welche Station diese Rolle übernimmt. Entdeckt die Monitorstation, dass nach der maximalen Umlaufszeit gemäß Gleichung 4.2 der Token immer noch nicht zurück gekommen ist, erzeugt sie einen neuen. Weiterhin kann sie anhand eines speziellen Monitorbits Rahmen erkennen, die mehrfach durch den Ring gereist sind. Diese Rahmen nimmt sie aus dem Ring.

4.3 Drahtlose LAN

Drahtlose Netze (wireless LAN, WLAN) gewinnen immer mehr an Bedeutung. Nur durch drahtlose Netze ist freie Beweglichkeit möglich. Dadurch können auch bewegliche Teilnehmer – sowohl Rechner (Laptops) als auch Geräte wie Scanner, Barcode-Leser, etc. – in ein Netz eingebunden werden. Der besondere Vorteil liegt im schnellen Netzaufbau. Dieser Vorteil kommt in Fällen zu tragen, in denen die Teilnehmer häufig wechseln oder einen schnellen, unkomplizierten Zugriff auf ein Netz brauchen. Typische ist der Einsatz als Netzzugang in öffentlichen Bereichen wie Flughäfen, Bahnhöfen oder Hotels (Hotspots).

Die wichtigste Standards für drahtlose Netze sind IEEE 802.11 und die daraus abgeleiteten Varianten. Eine Übersicht über die Standards dieser Familie gibt Tabelle 4.3. Derzeit aktuell ist 802.11b auch bekannt als Wi-Fi für *wireless fidelity*. Der Standard 802.11g ist kompatibel mit 802.11b. Erste Produkte dafür werden für Mitte 2003 erwartet. Als potentieller Nachfolger erscheint der Anfang 2003 verabschiedete Standard IEEE 802.16 aussichtsreich. Bei Entfernungen bis 50 km und Übertragungsleistungen bis zu 134 Mbit/s soll er eine großflächige Bereitstellung von drahtlosen Netzen ermöglichen. In diesem Zusammenhang spricht man bereits von Wireless Metropolitan Area Networks (WMAN).

IEEE 802.11 wurde für zwei Methoden der Funkübertragung und eine Art der Infrarot-Übertragung ausgelegt. Die Kollisionsvermeidung erfolgt ähnlich wie bei Ethernet. Allerdings ergeben sich aus der endlichen Reichweite der Signale spezielle Probleme. Anders als bei Ethernet erreicht ein Rahmen nicht alle Stationen. Abbildung 4.2 zeigt zur Veranschaulichung ein Netz mit 4 Stationen A, B, C und D. Die jeweilige Reichweite ist durch einen Kreis dargestellt.

In einer solchen Konstellation kann ein Sender nicht mehr alle Kollisionen

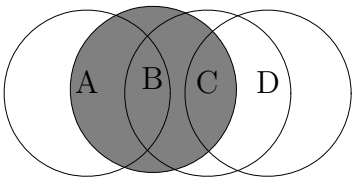


Abbildung 4.2: Drahtloses Netz mit 4 Stationen

feststellen. Angenommen A und C senden gleichzeitig an B. Da A und C weit auseinander liegen, empfangen sie nicht das Signal der jeweils anderen Station, so dass die Kollision unbemerkt bleibt. 802.11 sieht verschiedene Zugriffsverfahren vor, um solche Probleme zu umgehen.

Ein Konzept beruht auf dem Austausch von kurzen Protokollrahmen (MACA, Multiple Access with Collision Avoidance). Der Sender schickt zunächst eine Benachrichtigung (RTS, Request To Send) an den Empfänger. Ist der Empfänger bereit, antwortet er mit einer Bestätigung (CTS, Clear To Send). Der CTS-Rahmen dient als Bestätigung, dass der Sender mit der Übertragung beginnen kann. Bleibt das CTS aus, schließt der Sender – nach einem timeout – dass das RTS nicht angekommen ist, weil es z.B. zu einer Kollision kam. Beide Protokoll-Rahmen enthalten die Information über den Umfang der zu schickenden Daten. Daraus können benachbarte Stationen berechnen, wie lange der Transfer dauern wird und warten entsprechend lange mit eigenen Übertragungen.

Ähnlich wie bei Mobiltelefonsystemen mit Funkzellen organisiert man Netze nach 802.11 in Regionen. Die Rolle der Basisstationen übernehmen dann so genannte Anschlußpunkte (AP, Access Point). Die Anschlußpunkte können untereinander beispielsweise über Ethernet verbunden sein. Jeder mobile Knoten meldet sich bei einem Anschlußpunkt an und kommuniziert bis auf weiteres nur über ihn. Der Standard definiert die Protokolle, nach denen sich Rechner anmelden. Ein Knoten kann einerseits selbst nach APs suchen (aktives scanning). Andererseits schicken auch die APs Rahmen, mit denen sie ihre Dienste anbieten (passives scanning). Bei mobilen Knoten kann es allerdings durchaus vorkommen, dass ein Knoten den Bereich seines AP verlässt. In einem solchen Fall findet er wiederum mit einer der Scanning-Methoden einen besser geeigneten AP. Idealerweise erfolgt dieser Wechsel zwischen zwei APs ohne Unterbrechung der Verbindung.

Neben der Kommunikation über mindestens einen gemeinsamen AP (*infrastructure mode*) können auch mehrere Stationen direkt miteinander Verbindung aufnehmen (*ad hoc* oder *peer-to-peer mode*). Eine Sonderform sind so genannte Manets (*mobile ad hoc networks*), bei denen mehrere mobile Knoten untereinander ein gemeinsames ad hoc Netz betreiben.

Der Aufbau und Betrieb von drahtlosen Netzen erfordert deutlich höheren Aufwand. Die beschränkte Reichweite der Knoten und ihre Beweglichkeit stellen

neue Herausforderungen an die Kontrolle des Medienzugriffs. Auch die Anforderungen an die Sicherheit der Daten gegen Verfälschungen oder Abhören wachsen. Im Gegensatz zu einem leitungsgebundenen Netz ist es sehr einfach, Pakete abzuhören oder eigene Pakete einzuschleusen. Die Inhalte sollten daher durch Verschlüsselung geschützt werden. Bei Netzen mit geschlossenem Benutzerkreis ist es sinnvoll, nur Geräten mit einer registrierten Netzwerkadresse den Zugang zu erlauben. Der derzeit noch nicht verabschiedete Standard 802.11i bietet Erweiterungen bezüglich Sicherheit und Authentifizierung.

4.4 Andere Netztechnologien

Neben den beschriebenen Technologien gibt es eine ganze Reihe von Alternativen. Für viele Spezialfälle gibt es angepasste Lösungen. So bedingen manche Anwendungen hohe Anforderungen an Robustheit und Zeitverhalten. Häufig sind mehrere Alternativen verfügbar. Nicht in allen Fällen entwickelte sich ein einheitlicher Standard.

- LAN
  - Token Bus IEEE 802.4: in diesem Protokoll wird ein Token-basiertes Verfahren auf einer Bus-Architektur realisiert. Der Vorteil liegt in dem deterministischen Zeitverhalten. Diese Technik hat nur in speziellen Anwendungsbereichen Verbreitung gefunden. So basiert das von der Firma General Motors für die industrielle Fertigung entwickelte Manufacturing Automation Protocol (MAP) auf diesem Ansatz.
  - Fiber Distributed Data Interface (FDDI): eine ursprünglich für die Übertragung über Glasfaser entwickelte Technik basierend auf einem doppelten Token-Ring. Mit einer Übertragungsrate von 100 Mbit/s bei Ringlängen bis maximal 100 km wurde es häufig als *Backbone* (engl. Rückgrat) zur Vernetzung größerer Bereiche (Firmen, Uni-Campus) eingesetzt. Einzelne Rechner werden in der Regel nicht direkt, sondern über Konzentratoren an den Ring angeschlossen. Auch Ethernet-Netze können mittels FDDI/Ethernet Bridges mit dem Ring verbunden werden. Durch die Entwicklung von Fast Ethernet und Gigabit Ethernet hat FDDI kaum noch Bedeutung.
- Drahtlose Netze
  - Bluetooth[1]: ein Standard für die drahtlose Vernetzung von (kleinen) Geräten mit geringer Reichweite. Typisch ist die Vernetzung von mobilen Kleingeräte wie Mobiltelefone und PDAs. Ein solches Netzwerk wird auch als *Wireless Personal Area Network* (WPAN) bezeichnet.

[1]Benannt nach dem Wikinger-König Harald Blauzahn, um 910-986

- Netze für Peripheriegeräte
  - SCSI Small Computer System Interface: Ein paralleler Bus für Peripheriegeräten wie Festplatten oder Scanner. Über einen SCSI-Bus können bis zu 15 Geräte angeschlossen werden. Der SCSI-Bus wird vornehmlich im High-End Bereich eingesetzt.
  - USB Universal Serial Bus: Ein schneller, serieller Bus zum Anschluß von Peripheriegeräten (Drucker, Scanner, Maus, etc.) an einen Rechner. In den Versionen 1.0 und 1.1 beträgt die Übertragungsrate bis zu 12 Mbit/s, bei der aktuellen Version 2.0 bis zu 480 Mbit/s. Durch Einsatz von USB-Hubs können an einen USB-Port bis zu 127 Geräte angeschlossen werden.
  - Firewire IEEE 1394: Eine weitere schnelle serielle Schnittstelle mit Übertragungsraten bis zu 400 Mbit/s bei maximal 63 anschließbaren Geräten. Firewire wird beispielsweise als Verbindung zwischen Rechner und externer Festplatte oder Videokamera verwendet.
- Feldbusse: Feldbusse sind für den Einsatz in der Automatisierungstechnik ausgelegt. Sie verbinden Steuergeräte mit Sensoren und Aktoren Wichtig ist die Robustheit gegenüber Störungen. Die meisten garantierten eine Zeitspanne, innerhalb der Daten immer übermittelt werden (Echtzeitverhalten).
  - PROFIBUS: Profibus ist ein offenes, flexibles System für Anwendungen in den Bereichen Prozeßleittechnik, Gebäudeautomation, etc.. Es unterstützt die Kombinationen von Geräten in unterschiedlichen Konfigurationen (z.B. Mono-Master, Multi-Master, Master-Slave). Ein Token-basiertes Protokoll regelt den Buszugriff.
  - CAN Controller Area Network: CAN benutzt eine Bus-Topologie mit Vielfachzugriffsverfahren und Prioritätsregeln. CAN wird auch zur Vernetzung von Komponenten in Kraftfahrzeugen eingesetzt. Neuere Entwicklungen für zeitkritische Anwendungen im Kfz-Bereich sind TTP (Time Triggered Protocol) und, basierend auf dem von BMW entwickelten ByteFlight Konzept, FlexRay.
- Gebäudevernetzung Darunter versteht man die Vernetzung der verschiedensten Sensoren und Geräte innerhalb eines Gebäudes. Die Vision ist eine einheitlich Steuerung von Geräten zur Klimasteuerung, Beleuchtung, Sicherheit, Kommunikation, Unterhaltung und so weiter. Derzeit wird die Technik überwiegend in gewerblichen Gebäuden eingesetzt, soll aber zunehmend auch in Privathaushalten Einzug halten.
  - EIB Europäischer Installationsbus: EIB ist ein europaweit einheitlicher Standard zur Vernetzung von Komponenten der Gebäudesystemtechnik.

– Local Operating Network LON: auch LONWORKS® genannt ist ebenfalls ein universelles Automatisierungsnetzwerk, entwickelt von der Echelon Corporation [8].

4.5 Übungen

Übung 4.1 *Planen Sie das Netzwerk für Ihre Firma. Die Firma befindet sich in einem dreistöckigen Gebäude. In jedem Stockwerk sind die Räume wie folgt verteilt:*

<table>
<tr><td rowspan="3">Labor</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>Schacht</td></tr>
<tr><td colspan="5"></td><td rowspan="2"></td></tr>
<tr><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td></tr>
</table>

Jeder Raum ist 5 m breit und 10 m lang. Die Labore sind 10 m breit und 23 m lang und die Geschosshöhe beträgt 4 m. Die normalen Räume benötigen je 4 Netzanschlüsse, die Labore je 10. Die Server werden in Raum 5, 1. Stockwerk untergebracht. Von dort sollen Kabel zu allen Anschlüssen im gleichen Stockwerk gelegt werden. Über den Kabelschacht können Verteiler in den beiden anderen Stockwerken erreicht werden.

- *Skizzieren Sie die notwendige Verkabelung.*
- *Informieren Sie sich im Internet über die entsprechenden Kosten der benötigten Teile (z.B. www.transtec.de). Welche Gesamtkosten ergeben sich inklusive eines großen Servers?*
- *Wie groß sind die Preisunterschiede bei verschiedenen Alternativen (unterschiedliche Übertragungskapazitäten je nach Anforderung wie z.B Büroräume oder CAD-Arbeitsplätze)?*
- *Wie sieht eine Alternative mit WLAN aus?*

Kapitel 5

Vermittlung

Im letzten Kapitel haben wir gesehen, wie Stationen direkt miteinander kommunizieren können. Die direkte Verbindung stößt an Kapazitätsgrenzen, wenn es gilt viele Stationen zu verbinden oder eine hohe Datenrate zu gewährleisten. Allgemein gesehen handelt es sich um ein Reihe von Teilnetzen, die zusammen geschaltet werden sollen. Im Extremfall kann jedes Teilnetz aus nur einer einzelnen Station bestehen. Man kann die Teilnetze beispielsweise in Form eines Sternes über ein zentrales Koppelelement verbinden (Bild 5.1).

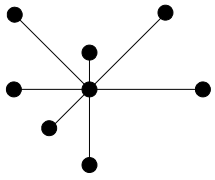


Abbildung 5.1: Sterntopologie

Bei einem einfachen Ethernet-Hub werden wie beschrieben alle Pakete über den gesamten Stern geschickt. Effizienter ist, wenn das Koppelelement nicht alle Pakete an alle Teilnetze und damit an alle Knoten weiter gibt, sondern die Adresse auswertet und die Pakete nur an die betroffenen Teilnetze schickt. Ein Gerät mit entsprechender Funktionalität ist ein Vermittler oder Switch. Der grundsätzliche Aufbau ist in Bild 5.2 dargestellt.

Ein Switch hat eine feste Anzahl von Ein- und Ausgängen. An diesen Anschlüssen – auch als Ports bezeichnet – werden Teilnetze oder Knoten angeschlossen. Dabei können verschiedene Typen von Ports vorhanden sein. Zum Beispiel kann ein Switch ein Reihe von Ports mit 10BaseT zum direkten Anschluss von Stationen und breitbandige Anschlüsse zur Vernetzung mit weiteren Switches haben. Empfängt der Switch an einem Eingangs-Port ein Paket, so liest er die Adresse und entscheidet dann, auf welchen Ausgangs-Port er es schickt.

Bevor wir näher auf Switches eingehen, sind in der folgenden Auflistung die

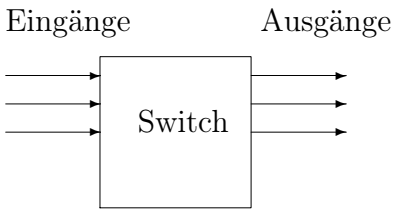


Abbildung 5.2: Prinzip eines Switchs

unterschiedlichen Geräte zur Verbindung von Netzen zusammen gestellt. Im wesentlichen unterscheiden sich die Gerät nach der Anzahl der Anschlüsse sowie in der „Intelligenz“. Einfache Geräte arbeiten nur auf der Ebene des physikalischen Signals während aufwändige eine komplette Analyse der Daten über alle Ebenen des ISO/OSI-Referenzmodells durchführen.

- **Repeater**: Ein Repeater verbindet zwei gleichartige Netze. Er arbeitet auf Ebene 1 des ISO/OSI-Referenzmodells und behandelt nur das physikalische Signal. Ein Repeater kann beispielsweise dann verwendet werden, wenn ein Ethernet-Strang die maximale Länge erreicht hat. Über einen Repeater kann dann ein weiterer Strang angeschlossen werden. Der Repeater regeneriert das Signal, bevor er es in den zweiten Strang überträgt.

- **Hub**: Ein Hub entspricht einem Repeater mit mehreren Ports (Multiport-Repeater).

- **Bridge**: Eine Bridge stellt eine Verbindung zweier Netze her, wobei Ebene 1 und 2 des ISO/OSI-Referenzmodells berücksichtigt werden. Bei der Weiterleitung werden Adressinformationen ausgewertet. Fehlerhafte Rahmen werden blockiert.

- **Switch**: Multiport-Bridge

- **Router**: Router verbinden Netzwerke unterschiedlicher Art. Falls erforderlich, können Sie Datenpakete in ein anderes Format umsetzen. Sie arbeiten auf Schicht 3.

- **Gateway**: Als Schnittstelle zwischen unterschiedlichen Netzen (z.B. TCP/IP mit einem herstellerspezifischen Protokoll wie DECnet) dienen Gateways. Sie arbeiten auf allen 7 Schichten.

Bei der Entwicklung von Switches stellen sich zwei Herausforderungen. Das erste Problem ist die Realisierung der Vermittlungsfunktionalität.

- Woher weiß der Switch, an welchen Ausgangs-Port ein Paket mit der Adresse *xyz* geschickt werden soll?

Der zweite Themenkomplex ist die effiziente Realisierung eines Switches. Intern muss zunächst ein schneller Datentransfer gewährleistet sein. Weiterhin muss der Switch auf jeder angeschlossenen Leitung ein sicheres Protokoll realisieren und beispielsweise auch Verzögerungen durch Kollisionen handhaben können. Die Kernfrage ist dann:

- Wie kann ein leistungsfähiger Switch preisgünstig in Hardware realisiert werden?

Im folgenden werden zunächst Grundprinzipien der Vermittlung beschrieben. Anschließend werden einige Beispiele für die Realisierung von Switches vorgestellt. Die Frage, wie Pakete ihren Weg durch ein weltweites Netz finden, ist Gegenstand des nächsten Kapitels. Ausgeklammert aus der Darstellung ist die Technik ATM für Zellenvermittlung, die in einem späteren Kapitel behandelt werden wird.

5.1 Datagramme

In der Grundform der Paketvermittlung trägt jedes Datenpaket die Zieladresse. Anhand dieser Zieladresse wird das Paket durch das Netzwerk geleitet. Jeder Switch führt eine Tabelle mit der Information, auf welchen Port er Pakete mit gegebener Zieladresse weiter leitet (Weiterleitungstabelle oder Routing-Tabelle).

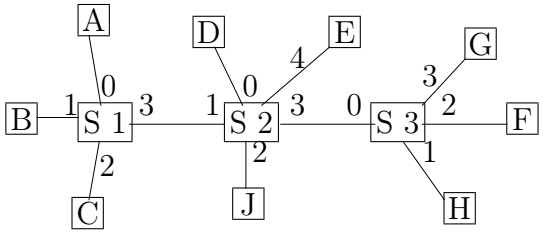


Abbildung 5.3: Netz mit 9 Knoten A, ..., J und 3 Switches

Bild 5.3 zeigt ein Beispielnetz mit 9 Knoten und Switches. Die Ports sind nummeriert. Der Einfachheit halber sind Ein- und Ausgänge zusammen gelegt. Jeder Switch führt eine Tabelle, in der für jedes Ziel (Knoten) der richtige Port eingetragen ist. Entweder hängt der Knoten direkt an diesem Port oder der Port führt zum nächsten Switch in Richtung Ziel. Die Weiterleitungstabelle für Switch 2 ist in Tabelle 5.1 angegeben.

5.1.1 Lernende Bridges

Eine einfache Strategie zum Aufbau der Weiterleitungstabellen nutzen lernende Bridges. Bridges sind Geräte, die mehrere LANs verbinden, indem sie ankommenden Rahmen an die Ausgänge weiter geben. Wenn eine Bridge die Rahmen

Tabelle 5.1: Weiterleitungstabelle für Switch 2

Zielknoten	Port
A	1
B	1
C	1
D	0
E	4
F	3
G	3
H	3
J	2

analysiert, kann sie feststellen an welchem Port Rahmen von welchem Knoten ankommen. Dieser Port wird in die Weiterleitungstabelle für den Knoten eingetragen. Von da an werden Rahmen an diesen Knoten nur noch auf diesem Port ausgegeben. Durch diese einfache Strategie wird für einen Knoten, sobald er Rahmen sendet, automatisch der richtige Eintrag in die Weiterleitungstabelle aufgenommen.

Am Anfang ist die Tabelle leer. Erst im Laufe der Zeit wird sie automatisch gefüllt. Auch wenn die Tabelle das Netzwerk nicht vollständig erfasst, wird der Netzwerkverkehr verbessert. Fehlt für einen Zielknoten der Eintrag, so wird der Rahmen nach wie vor an alle Ports (außer dem Eingangs-Port) weiter geschickt. Um Änderungen im Netz zu erfassen, wird der Eintrag eines Knotens wieder gelöscht, sobald über eine gewisse Zeit keine Rahmen mehr von diesem Knoten gekommen sind.

Ein Problem entsteht bei der Verknüpfung von LANs über Bridges, wenn das Netz Schleifen enthält. Solche Schleifen entstehen entweder aus Versehen oder bewusst als Redundanz gegen Ausfall einer Bridge. Ohne Gegenmaßnahmen kann es dazu kommen, dass Rahmen ewig in den Schleifen kreisen. Eine Lösung dieses Problems stammt von R. Perlmann. Um Schleifen zu erkennen, tauschen die Bridges regelmäßig Informationen über den ihnen bekannten Teil des Netzes aus. Aus den Informationen berechnen sie dann einen Baum (Spanning tree), der die jeweils kürzesten Verbindungen enthält. Eventuell vorhandene zusätzliche Verbindungen, die im vollständigen Netz zu Schleifen führen, werden im Baum nicht berücksichtigt. Während des Betriebs wird der Baum regelmäßig aktualisiert, um eventuelle Änderungen zu erfassen. Das Protokoll wird im Standard IEEE 802.1D „Media Access Control (MAC) Bridges“ beschrieben.

5.2 Virtuelle Verbindungen

Die Vor- und Nachteile der Nachrichtenübertragung mit Datagrammen wurden bereits im Abschnitt 2.1 diskutiert. Datagramme sind besonders geeignet, wenn nur kurze Nachrichten geschickt werden oder Wert auf eine ausfallsichere Kommunikation gelegt wird. Andererseits wird häufig nicht nur eine einzelne Nachricht übermittelt, sondern die Kommunikation erfolgt über einen längeren Zeitraum (z.B. Verbindung zwischen zwei Standorten einer Firma). In solchen Fällen ist die Datagramm-Übertragung nicht effizient, da für jedes Paket wieder der gesamte Vermittlungsprozess abläuft. Daher wird häufig die Kommunikation über virtuelle Leitungen abgewickelt.

Virtuelle Leitungen (Virtual Circuit VC) simulieren mit Methoden der Paketvermittlung klassische Leitungsvermittlung. In einer Aufbauphase wird zunächst ein Weg vom Sender zum Empfänger festgelegt. Jeder dabei beteiligte Switch reserviert dann Kapazität für die Verbindung. Sobald die Verbindung steht, können Pakete über die Verbindung geschickt werden. Die Pakete benötigen nicht mehr die Zieladresse sondern nur noch einen Bezeichner für die VC. Zum Ende der Verbindung müssen die Switches informiert werden, so dass sie die VC wieder abbauen und die Ressourcen für neue Verbindungen frei machen.

Virtuelle Leitungen können entweder vom Netzadminstrator dauerhaft angelegt werden (Permanent Virtual Circuit PVC) oder von einem Knoten bei Bedarf aufgebaut werden (Switched Virtual Circuit SVC). Man kann sich den Netzaufbau ähnlich wie die Weiterleitung eines Datagramms vorstellen. Der Sender schickt ein Paket mit dem Wunsch zum Verbindungsaufbau. Das Paket wird durch das Netz zum Empfänger geleitet. Beim Passieren des Pakets reserviert ein Switch die Ressourcen für die Verbindung. An dieser Stelle können bestimmte Vorgaben für die Qualität der Verbindung bezüglich beispielsweise Bandbreite oder Verzögerungszeit getroffen werden. Dadurch wird für die Verbindung eine definierte Dienstgüte (Quality of Service, QoS) garantiert.

Ist die VC aufgebaut, braucht jeder Switch sich nur noch die Verbindung zwischen Eingang und Ausgang zu merken. Da über einen Port mehrere VC kommen können, reicht nicht die Port-Nummer sondern eine Kennzeichnung für die VC ist zusätzlich erforderlich. Diese Kennzeichnung gilt nicht global für den VC, sondern nur innerhalb jedes Switches und dort wieder für einen Port.

In Bild 5.4 ist ein Beispiel für ein Switch mit 4 Ein- und 4 Ausgangs-Ports dargestellt. Eingetragen sind 2 virtuelle Verbindungen, eine von Port 2 nach Port 1 und die zweite ebenfalls von Port 2 nach Port 3.

Um die Verbindungen zu kennzeichnen, vergeben die Switchs eindeutige Identifikatoren, so genannte Virtual Circuit Identifier VCI. Für die Eingänge kann der Switch selbst die Nummerierung übernehmen. Sobald ein anderer Switch auf einem Port eine neue Verbindung anmeldet, vergibt der Switch eine freie VCI Nummer. Diese Nummer meldet er dem Vorgänger-Switch, der sie dann als Ausgangs-VCI übernimmt. In Summe sind die Schritte für eine Verbindungsteilstrecke von

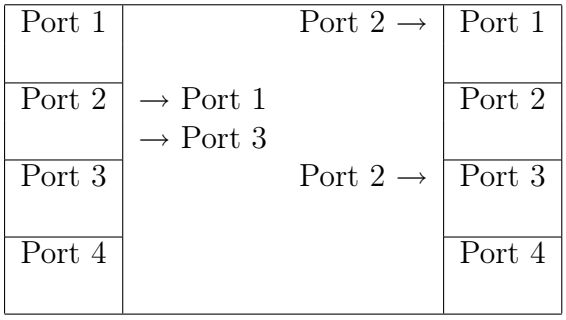


Abbildung 5.4: VC innerhalb eines Switches

Eingangs-Port	Eingangs-VCI	Ausgangs-Port	Ausgangs-VCI
2	1	1	5
2	2	3	3

Tabelle 5.2: VC-Tabelle mit 2 Einträgen.

Switch N zu Switch M:

1. Switch N sendet eine Anfrage an Switch M
2. Switch M entscheidet, ob er der Anfrage akzeptiert
3. Falls Switch M die Anfrage akzeptiert, wählt er einen freien VCI für seinen Eingangsport aus.
4. Switch M meldet seinen Eingangs-VCI an Switch N
5. Switch N trägt den erhaltenen VCI als Ausgangs-VCI in seine eigene Tabelle ein

Die Rückmeldung der VCI an den Vorgänger-Switch muss nicht unmittelbar erfolgen, sondern kann auf dem Weg einer Bestätigung vom Empfänger zurück zum Sender bei allen Switches nacheinander übergeben werden. Das Resultat trägt der Switch in eine Tabelle ein. Für das obige Beispiel könnte die Tabelle 5.2 resultieren.

Techniken mit virtuellen Leitungsvermittlung sind Frame-Relay und ATM. Frame-Relay wird für die Übertragung in Weitbereichsnetzen eingesetzt. Eine wichtige Anwendung ist die Kopplung mehrerer LANs zu einem Virtual Private Network VPN. Dazu werden vornehmlich PVC benutzt.

5.3 Design von Switches

Wie oben beschrieben besteht ein Switch aus einer Reihe von Ein- und Ausgängen – den Ports – sowie einer Einheit zur Verteilung der Pakete – dem Schaltnetzwerk

(Switching fabric). Die Herausforderung beim Design ist, Pakete von möglichst vielen Ports möglichst schnell zu verteilen. Dazu muss intern eine entsprechend hohe Verarbeitungsgeschwindigkeit bereit gestellt werden.

Eingangs-Ports nehmen Pakete in Empfang und speisen sie in das Schaltnetzwerk ein. Ausgangs-Ports erhalten Pakete vom Schaltnetzwerk und geben sie auf die Leitung. Diese zunächst trivial erscheinende Aufgaben erweisen sich bei genauerer Analyse als komplex. Das Grundproblem liegt in der ungleichen Verteilung des Verkehrs. In aller Regel wird nicht stets ein Eingang mit genau einem Ausgang kommunizieren. Beispielsweise wird ein Server Pakete an viele Knoten schicken. Damit werden an dem Port, an dem der Server angeschlossen ist, viele Pakete ankommen, die auf mehrere Ausgänge verteilt werden müssen. Ein Extremfall ist ein Broadcast-Paket, das an einem Eingang ankommt und an alle Ausgänge weitergereicht wird.

Entsprechend gilt für die Ausgänge, dass auch dort Pakete von vielen Eingängen „gleichzeitig“ ankommen können und dann in eine Leitung eingespeist werden müssen. Daher müssen Ports in der Lage sein, Pakete in einen Zwischenspeicher (Puffer) zu legen, bis sie weiter geschickt werden können. Bei dem Entwurf gilt es, einen guten Kompromiss zu finden, derart dass einerseits möglichst selten die Puffer überlaufen aber anderseits auch die Puffer nicht überdimensioniert sind.

Prinzipiell könnten die Puffer an den Eingängen oder an den Ausgängen plaziert werden. Zwischenspeicher an den Eingängen können allerdings schnell zu unnötigen Blockaden führen. Betrachten wir folgendes Beispiel von Paket-Folgen an zwei Eingängen:

Port N:	Ziel 4	Ziel 4	Ziel 5	→
Port M:	Ziel 7	Ziel 6	Ziel 5	→

Die beiden nächsten Pakete an der Reihe kollidieren, da sie beide an Port 5 gerichtet sind. Bei Eingangspufferung würde eines der Pakete ausgewählt und die Ausgabe am zweiten Port verzögert. Damit werden auch weiter hinten liegende Pakete blockiert, obwohl ihr Ziel (4 oder 6) eventuell frei ist (Head-of-Line Blocking). Daher erfolgt die Pufferung in der Regel an den Ausgängen.

5.3.1 Weiterleitung

Eine weitere Frage ist, wann ein Paket weiter geleitet werden soll. Eine Möglichkeit ist, zunächst das Paket vollständig abzuwarten, dann die Integrität zu prüfen und es – sofern alles in Ordnung ist – dann weiter zu geben (*Store-and-Forward* Prinzip). Auf diese Art und Weise wird die weitere Verbreitung unvollständiger oder fehlerhafter Pakete verhindert. Daraus resultiert aber auch, dass die Verarbeitungszeit im Switch mindestens so groß ist wie die Übertragungsverzögerung für das Paket.

Die Alternative ist, ein Paket frühzeitig weiter zu geben. In der Regel steht die Zieladresse im vorderen Teil des Paketes. Sobald dieser Teil gelesen ist, kann die

Weiterleitung aufgenommen werden (*Cut-Through* oder *On The Fly* Prinzip). Allerdings wird damit in Kauf genommen, dass fehlerhafter Pakete in andere Netze übertragen werden.

Je nach Anwendungssituation kann eine der beiden Strategien besser sein. Betrachtet man ein lokales Netz mit relativ geringer Auslastung, so kann man durch die beschleunigte Weiterleitung die Gesamtverzögerung reduzieren. Ist umgekehrt das Netz stark belastet, so dass es häufiger zu Kollisionen kommt, dann wird die Lage durch die ungehinderte Verbreitung von unvollständigen Paketen noch verschärft.

Einige Hersteller bieten daher Geräte an, die dynamisch zwischen den beiden Betriebsweisen hin und her schalten können. Abhängig von z.B. der Anzahl der festgestellten fehlerhaften Pakete wird dann zwischen der schnellerem oder der sicheren Alternative gewählt.

5.3.2 Knockout-Switch

Bei einem Knockout-Switch [9] werden alle ankommenden Pakete an alle Ausgangs-Ports verteilt. An jedem Ausgangs-Port sind Paketfilter, die nur an den Port gerichtete Pakete passieren lassen. Die Pakete werden in ein Warteschlange eingereiht und von dort aus abgeschickt. Falls die Kapazität der Warteschlange nicht ausreicht alle Pakete aufzunehmen, muss eine Auswahl getroffen werden. Diese Aufgabe übernimmt ein Konzentrator, der aus den L Paketen $N < L$ Pakete auswählt. Die günstige Realisierung beruht auf einfachen Auswahlbausteinen mit nur 2 Eingängen. Ein Baustein wählt zufällig aus den beiden ankommenden Paketen eines als „Sieger“ aus. Wie bei einem Turnier in K.O.-Modus treten alle Pakete gegeneinander an, bis ein Sieger übrig bleibt. Die Verlierer kommen in eine zweite Runde, die nach dem gleichen Modus den zweiten Sieger ermittelt. Das Verfahren wird fortgesetzt bis N Sieger feststehen. Die verbleibenden Pakete werden verworfen.

5.3.3 Batcher- und Banyan-Netzwerke

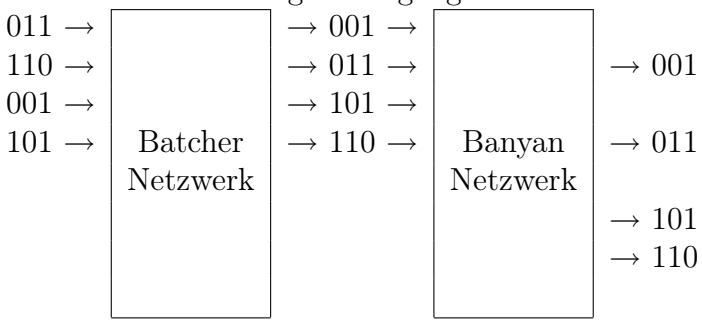
Schaltnetzwerke können günstig als Verbund gleichartiger kleiner, einfachen Elemente gebaut werden. Solche Elemente haben 2 Eingänge und 2 Ausgänge und entscheiden nur zwischen 2 Möglichkeiten:

- Eingang 1 auf Ausgang 1 und Eingang 2 auf Ausgang 2
- Eingang 1 auf Ausgang 2 und Eingang 2 auf Ausgang 1

Das 2x2 Schaltelement trifft seine Entscheidung aufgrund von Informationen im Header des Pakets. Beispielsweise kann bei einer VC der Eingangs-Port aus der aktuellen Tabelle den Ausgangs-Port entnehmen und als internen Header an das Paket anfügen. Man spricht dann von eigenvermittelnden Schaltnetzwerken.

Der Entscheidungsprozess aus Sicht eines Paketes verläuft ähnlich wie die Suche in einem binären Baum. An jedem Knoten - Schaltelement - wird ein weiter führender Ast aus zwei Möglichkeiten ausgewählt. Bei einem Header mit n Bit erreicht ein Paket dann in n Schritten einen aus 2^n Endknoten (Ausgangs-Ports). Da es aber mehrere Eingänge gibt, sind mehrere Wurzeln notwendig. Alle Wurzeln führen aber zu den gemeinsamen Blättern, wobei die Pakete auf dem Weg nicht zusammen stoßen dürfen. Eine trickreiche Lösung dieses Problems sind Banyan-Netzwerke, bei denen diese Vergleichsbäume in geschickter Weise ineinander verflochten sind. Bei n Eingängen werden $n/2$ Schaltwerke pro Stufe und $\log_2 n$ Stufen benötigt. Banyan ist der englische Name eines Baumes (*Ficus benghalensis*), bei dem Ableger von den Ästen aus neue Wurzeln bilden.

Banyan-Netzwerke funktionieren allerdings nur gut, wenn die eingehenden Pakete in aufsteigender Reihenfolge sortiert sind. Daher wird ein nach seinem Erfinder benanntes Batcher-Netzwerk vorgeschaltet, das zunächst die Pakete in die richtige Reihenfolge bringt. Das nachgeschaltete Banyan-Netzwerk bringt die Pakete dann an die richtigen Ausgangs-Ports.



Das kombinierte Netzwerk befördert alle Pakete an den richtigen Ausgangs-Port. Es kann allerdings stets nur ein Paket zu jedem Ausgang befördern. Daher werden zusätzliche Komponenten benötigt, die durch Pufferung und eventuelle parallele Strukturen Zielkollisionen so weit wie möglich vermeiden.

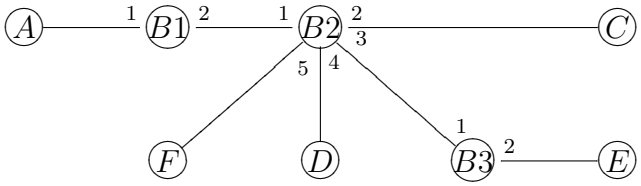
5.4 Übungen

Übung 5.1 *Entwerfen Sie ein Netz für 10 Knoten in Form von zwei verbundenen Sternen. Wie sieht die Weiterleitungstabelle in den beiden Knoten im Zentrum der Sterne aus?*

Übung 5.2 *Vergleichen Sie die minimale Verzögerungszeit für einen Ethernet-Rahmen mit 1000 Bytes Nutzdaten in einem Switch in Abhängigkeit davon, ob das Store-and-Forward oder das Cut-Through Prinzip verwendet wird.*

Übung 5.3 *Lernende Bridges*
Betrachten Sie folgendes Netzwerk mit 5 Knoten A, C, D, E und F sowie den

lernenden Bridges B1 bis B3.



Zu Beginn sind die Weiterleitungstabellen in den Bridges leer. Als erstes schickt Knoten D ein Paket an Knoten A. Zeichnen Sie in das Bild ein, welche Wege das Paket nimmt. Was haben die Bridges über die Zuordnung von Ports und Knoten aus diesem Paket gelernt? Tragen Sie diese gewonnene Information in die folgenden Tabellen ein:

B1	*Knoten*				
	Port				
B2	*Knoten*				
	Port				
B3	*Knoten*				
	Port				

Wie ist der jeweilige Stand, nachdem

- *C mit einem Paket an A geantwortet hat?*
- *D ein Paket an C gesendet hat?*

Angenommen, alle Knoten haben irgendwann Pakete verschickt. Wie sehen dann die vollständigen Tabellen aus?

B1	*Knoten*				
	Port				
B2	*Knoten*				
	Port				
B3	*Knoten*				
	Port				

Kapitel 6

Internet Protokoll IP

6.1 Einleitung

Der Erfolg des Internets zeigt, dass eine weltweite Vernetzung unterschiedlichster Rechner und Systeme möglich ist und damit funktionierende Anwendungen angeboten werden können. Das Internet ist kein einheitliches Netz sondern ein Verbund vieler Netze mit unterschiedlicher Technologie. Insgesamt entstand ein riesiger Verbund von Knoten mit weiterhin starkem Wachstum [10]. Bei dem Aufbau des bisherigen Netzes und seinem weiterer Ausbau sind Kernprobleme:

- Heterogenität: wie können die verschiedensten Netze zusammen funktionieren?
- Adressierung: wie werden eindeutige und aussagefähige Adressen vergeben?
- Routing: wie findet eine Nachricht den Weg durch das Netz?
- Skalierbarkeit: wie kann das Netz das zu erwartenden starke Wachstum aufnehmen?

Grundlage des Internets ist IP – das Internet Protokoll. IP beinhaltet die Adressvergabe und einen verbindungslosen, unzuverlässigen Datagramm-Dienst. Im OSI Modell stellt IP die Schicht 3 (Vermittlungsschicht) dar. Unzuverlässig bedeutet, dass keine Garantie bezüglich der richtigen Zustellung von Paketen gegeben wird. Pakete können verfälscht werden, verloren gehen und mehrfach oder in der falschen Reihenfolge ankommen. Das Netzwerk bemüht sich die Pakete zuzustellen, aber um eventuelle Fehler müssen sich Sender und Empfänger selbst kümmern. Man sagt, das Netz arbeitet nach dem Prinzip Best-Effort.

6.2 Adressen

Eine IP-Adresse (in Version 4, IPv4) besteht aus 32 Bit. Üblicherweise schreibt man die Adressen als 4 durch Punkte getrennte Dezimalzahlen, also z.B. 120.56.222.94,

Tabelle 6.1: Aufbau der IP-Adressen in verschiedene Klassen

8				16				24				32				Klasse
0	Netz-ID							Host-ID								A
1	0	Netz-ID								Host-ID						B
1	1	0	Netz-ID										Host-ID			C
1	1	1	0	Multicast-Adresse												D

wobei jede Zahl 8 Bit repräsentiert. Jede Zahl kann damit maximal den Wert 255 annehmen. Insgesamt sind 2^{32} verschiedene Adressen zwischen 0.0.0.0 und 255.255.255.255 möglich.

Im Gegensatz zu den Ethernet-Adressen sind die IP-Adressen hierarchisch aufgebaut (Ethernet-Adressen sind hierarchisch bezüglich des Herstellers der Netzwerkkarte, eine Information die allerdings beim Routing nicht hilft). Der erste Teil der IP-Adresse bezeichnet ein Netzwerk und der zweite Teil einen Knoten. Dadurch vereinfacht sich das Routing, da zunächst nur die Netzwerk-Adresse ausgewertet werden muss.

Netzwerke sind unterschiedlich groß. Beispielsweise haben Firmen sehr unterschiedlich große Netze. So unterscheidet sich der Bedarf eines weltweiten Großkonzerns sehr von dem einer kleinen Software-Firma. Daher wurde ein flexibles Schema mit mehreren Klassen eingeführt. Die Position der ersten Null in der Adresse legt die Klasse fest. Ist das erste Bit Null (d.h. in der Hälfte der Fälle), so handelt es sich um eine Adresse der Klasse A. In diesem Fall sind die nächsten 7 Bit die Netzwerk-Adresse und die restlichen 24 Bit bezeichnen den Knoten. Berücksichtigt man noch zwei Sonderfälle verbleiben 126 Adressen für Netze der Klasse A. Diese 126 Netze belegen bereits die Hälfte aller zur Verfügung stehenden Adressen. In Tabelle 6.1 ist die Zuordnung der Bits zu den Feldern für die verschiedene Klassen zusammen gestellt.

Wie gesagt erkennt man die Klasse einer IP-Adresse an der Position der ersten Null. Betrachten wir als Beispiel die Adresse 212.201.24.18. Das erste Byte hat den Dezimalwert 212. Umwandlung in die Darstellung als Binärzahl ergibt 1101 0100. Die erste Null ist an der dritten Stelle von links. Es handelt sich also um eine Adresse aus der Klasse C.

Die Grundidee bei der Festlegung der Klassen war, dass es nur wenige sehr grosse Netze mit vielen Knoten, aber sehr viele kleine Netze mit wenigen Knoten gibt. Die jeweilige Anzahl der Netze und darin enthaltenen Knoten ist in Tabelle 6.2 eingetragen. Dabei ist berücksichtigt, dass einige Adressen eine besondere Bedeutung haben. Adressen mit dem Wert 0 beziehen sich auf das eigene Netz oder den eigenen Knoten. Sind andererseits alle Bits im Knotennamen gesetzt, so handelt es sich um eine Broadcast-Nachricht. Die Adresse 127 in der Klasse A wird für Tests benutzt, bei denen ein Knoten sein gesendetes Paket zurück erhält (Loop-Back).

Tabelle 6.2: Anzahl und Größe der Netze in den einzelnen Klassen

Klasse	Anzahl Netze	Anzahl Knoten
A	126	16777214
B	16384	65534
C	2097152	254

Die Netznummern werden vom Network Information Center (NIC) bzw. jetzt durch Internet Assigned Numbers Authority (IANA) (`www.iana.org`) verwaltet. In der Praxis hat sich die Einteilung als nicht optimal erwiesen. Es gibt einerseits zu wenige Netze der Klasse B für große Firmen oder Institutionen. Andererseits ist die Klasse B für die meisten Firmen überdimensioniert während die Klasse C zu klein ist oder zumindest zu wenig Reserven hat. Eine Lösung, die die starre Einteilung überwindet, ist Classless InterDomain Routing CIDR. Wie der Name zeigt, ersetzt CIDR die feste Klasseneinteilung durch eine flexible Aufteilung der Bits zwischen Netzkennung und Knotennummer. Gleichzeitig wurden die Adressen den 4 Weltzonen Europe, Nordamerika, Mittel- und Südamerika sowie Asien und Pazifik zugeordnet, um das Routing zu vereinfachen.

Eine andere Verfeinerung des Adressraums erreicht man durch Subnetz-Adressierung. Angenommen eine Firma betreibt mehrere LANs, die aber unter einer einheitlichen Netzadresse angesprochen werden sollen. Um die interne Weiterleitung zu vereinfachen, werden die einzelnen LANs als Subnetze behandelt. Dazu wird ein Teil der Bits der Host-ID als Subnetz-ID verwendet. Wie viele Bits für diese weitere Hierarchie zur Verfügung stehen, wird in einer Subnetzmaske spezifiziert. Angeschlossen Knoten können dann an der Adresse erkennen, ob das Paket in ihrem eigenen LAN bleibt oder über einen Switch in ein anderes LAN geschickt wird.

Die Subnetzmaske hat die Form einer IP-Adresse. Gesetzte Bits kennzeichnen den Netzwerkteil inklusive Subnetzadresse. Beispielsweise bedeutet eine Subnetzmaske 255.255.255.0, dass nur die letzten 8 Bit die Knotennummer beinhaltet. Alle anderen Bits, bei denen die Subnetzmaske den Wert Eins hat, gehören zur Netzwerksadresse. Die genaue Interpretation – Netzadresse und Subnetzadresse – hängt von der Klasse der Adresse ab. Bei einer Adresse der Klasse B enthalten die ersten 16 Bit die Netzwerksadresse. Bei der Subnetzmaske 255.255.255.0 bilden dann die nächsten 8 Bit, die ursprünglich zur Hostadresse gehörten, die Subnetzadresse.

Der Netzwerkteil einer Adresse lässt sich leicht durch bitweise UND-Verknüpfung mit der Subnetzmaske berechnen. Als Beispiel berechnet man für die Adresse 212.201.25.18 und die Subnetzmaske 255.255.252.0 als Adresse des Netzwerkteils:

IP-Adresse	212.201.25.18	11010100.11001001.00011001.00010010
Subnetzmaske	255.255.252.0	11111111.11111111.11111100.00000000
bitweise UND		11010100.11001001.00011000.00000000
Netzwerkteil	212.201.24.0	

Um das prinzipielle Problem von zu wenigen Adressen zu lösen, wird in der Version IPv6 die Größe der Adressen auf 16 Byte erhöht. Damit stehen selbst bei schlechter Ausnutzung des Adressraums rechnerisch mehr als 1000 IP-Adressen pro Quadratmeter der Erdoberfläche zur Verfügung.

Die IP-Adressen der einzelnen Knoten können – beispielsweise vom Systemadministrator – fest vergeben werden. Eine Alternative sind dynamische Adressen. Damit können insbesondere für temporäre Verbindungen zeitlich befristete Adressen vergeben werden. Über das Protokoll DHCP (Dynamic Host Configuration Protocol) erhält ein Rechner von einem DHCP-Server eine IP-Adresse. Der DHCP-Server verwaltet die Nummern und sorgt für eine eindeutige Vergabe der Nummern. Der Client erhält die Adresse für einen bestimmten Zeitraum (Lease). Vor dem Ablauf der Gültigkeitsdauer muss der Client diese verlängern oder eine neue Lease beziehen. Ein Beispiel für meinen Laptop am Netz der FH:

```
ipconfig /all

...
Ethernetadapter "LAN-Verbindung":

  Verbindungsspezifisches DNS-Suffix: fh-friedberg.de
  Beschreibung. . . . . . . . . . . : Accton EN2242 Series MiniPCI Fast
                                                    Ethernet Adapter
  Physikalische Adresse . . . . . . : 00-D0-59-6A-88-63
  DHCP-aktiviert. . . . . . . . . . : Ja
  Autokonfiguration aktiviert . . . : Ja
  IP-Adresse. . . . . . . . . . . . : 212.201.26.252
  Subnetzmaske. . . . . . . . . . . : 255.255.255.0
  Standardgateway . . . . . . . . . : 212.201.26.1
  DHCP-Server . . . . . . . . . . . : 212.201.26.1
...
  Lease erhalten. . . . . . . . . . : Dienstag, 22. Juni 2004 14:21:28
  Lease läuft ab. . . . . . . . . . : Dienstag, 22. Juni 2004 14:31:28
```

6.3 Paketformat

Tabelle 6.3 zeigt das Format der IP-Pakete (Datagramme) in der Version IPv4. Die Darstellung ist in Vielfachen von 32 bit organisiert. Die Bedeutung der ein-

Tabelle 6.3: IPv4 Paket-Format

0	4	8	16	19 31 bit
Version	IHL	TOS	Gesamtlänge	
Ident			Flags	Offset
TTL		Protocol	Checksum	
Quellenadresse				
Zieladresse				
Optionen + Pad				
Daten				

zelnen Felder ist:

- **Version:** IP Version (hier 4)
- **IHL:** Internet Header Length. Die Länge des Headers in Vielfachen von 32 bit.
- **TOS:** Type of Service. In diesem Feld können Eigenschaften für die Übertragung spezifiziert werden.
- **Gesamtlänge:** Datagrammlänge (Header plus Daten) in Bytes. Die maximale Größe ist 65535 Byte.
- **Ident:** Identifikation des Datagramm. Notwendig falls das Datagramm für die unteren Übertragungsschichten in kleiner Pakete – so genannte Fragmente – (z.B. bei Ethernet maximal 1500 Byte) zerlegt werden muss. Jedes Fragment enthält den vollständigen IP-Header zusammen mit einem Anteil der Daten.
- **Flags:** Informationen über die Fragmentierung.
- **Offset:** Die laufende Nummer des ersten Bytes im Datenteil relativ zum ersten Byte des gesamten Datagramms.
- **TTL:** Time To Live. Zähler für die maximale Lebensdauer eines Datagramms. Der Sender setzt TTL auf einen Startwert (z.Z. 64). Jeder Router auf dem Weg dekrementiert TTL. Ist der Wert Null erreicht, wird das Datagramm vernichtet.
- **Protocol:** Spezifikation für das höhere Protokoll (z.B. 6 für TCP und 17 für UDP).
- **Checksum:** Prüfsumme für den Header (Summe in Einerkomplement)

- **Quellenadresse und Zieladresse:** Die vollständigen IP-Adressen von Sender und Empfänger.

- **Optionen + Pad:** Mögliche zusätzliche Optionen und eventuelle Füllbits. Die Anzahl der Optionen berechnet sich aus der angegebenen Header Länge.

6.4 Weiterleitung

Wenn ein Knoten – ein Rechner oder ein Router – ein Datagramm verschicken oder weiter leiten will, muss er die IP-Adresse analysieren. Zunächst prüft er die Netzwerkadresse. Stimmt die Netzwerkadresse mit seiner eigenen überein, erkennt der Knoten, dass der Zielknoten sich in seinen eigenen physikalischen Netzwerk befindet. Rechner sind normalerweise nur an einem physikalischen Netzwerk angeschlossen, während Router zwei oder mehr Netzschnittstellen haben. Router überprüfen daher, ob die Netzwerkadresse zu einem ihrer Netzanschlüsse passt.

Befindet sich das Ziel im gleichen physikalischen Netzwerk, so kann der Knoten die zugehörige interne Adresse (z.B. eine Ethernet-Adresse) ermitteln und das Datagramm direkt zustellen. Wie dies im Detail geschieht werden wir im nächsten Abschnitt sehen.

Ist das Ziel außerhalb des eigenen Netzwerkes, wird das Datagramm an einen Router weiter geleitet. Der Router wirkt als Schnittstelle in die externen Netzwerke. Im allgemeinen wird der Router selbst auch keine direkte Verbindung zu dem Ziel haben, sondern das Paket an einen weiteren Router weiter reichen bis irgendwann ein Router mit Zugang zu dem Zielnetz erreicht wird.

Die Weiterleitung basiert auf Tabellen. Ähnlich den Weiterleitungstabellen in Switches gibt es Weiterleitungstabellen für die IP-Adressen. Im einfachsten Fall enthält die Weiterleitungstabelle eine paarweise Zuordnung von Netzwerksadressen und nächsten Routern. Für die Weiterleitung des Datagramms schlägt der Knoten in seiner Weiterleitungstabelle nach, an welchen nächsten Knoten er Datagramme für die gegebene Netzwerksadresse schicken soll. Den ausgewählten nächsten Router nennt man Next-Hop-Router.

Die Weiterleitungstabelle wird nicht vollständig sein. Daher gibt es noch einen Eintrag für einen Default-Router, an den alle nicht direkt zuordnenden Datagramme geschickt werden. Ein Rechner hat häufig nur Zugang zu einem Router. In diesem Fall ist die Weiterleitungstabelle leer beziehungsweise enthält nur den Eintrag für diesen Default-Router.

Insgesamt gesehen erfolgt die Zustellung in zwei Schritten. Zunächst wird das Datagramm in das richtige Netzwerk geschickt. Anschließend wird es dort an den Zielknoten zugestellt. Dadurch wird der Verwaltungsaufwand beträchtlich reduziert. Die Router müssen nicht für jeden Rechner in der Welt Einträge in ihre Weiterleitungstabellen aufnehmen, sondern nur noch für die Netze. Bleibt die Frage, wie die Router zu ihren Weiterleitungstabellen kommen. Dies wird

Thema des übernächsten Abschnittes sein.

6.5 Zuordnung IP-Adresse zu Ethernet-Adresse

Wenn ein Knoten feststellt, dass der Zielknoten sich im gleichen Netzwerk befindet, so kann er das Datagramm direkt an dessen Adresse schicken. Sind beispielsweise die beiden Knoten über Ethernet verbunden, so benötigt er dazu die Ethernet-Adresse des Zielknotens. Die Zuordnung von IP-Adressen zu Ethernet-Adressen erfolgt über das Adreßauflösungsprotokoll (Address Resolution Protocol ARP).

Wieder wird die Information in einer Tabelle gespeichert. Jeder Knoten führt eine Zuordnungstabelle mit IP-Adressen und Ethernet-Adressen, die ARP-Tabelle oder auch ARP-Cache. Da die Adresszuordnung sich jederzeit ändern kann, wird die Tabelle dynamisch verwaltet.

Angenommen ein Knoten S möchte zum ersten Mal ein Datagramm an einen bestimmten Zielknoten Z schicken. Zu diesem Zeitpunkt gibt es noch keinen Eintrag für Z in der ARP-Tabelle. Daher schickt S eine Anfrage (Rundruf) an alle Knoten im eigenen Netz. Bei Ethernet wird diese Anfrage als Broadcast gleichzeitig an alle geschickt. Die Anfrage enthält die Zieladresse und die eigene Adresse von S. Jeder Knoten empfängt die ARP-Anfrage. Knoten Z erkennt seine eigene Adresse und reagiert, indem er eine Antwortnachricht mit seiner eigenen Ethernet-Adresse zurück schickt. Gleichzeitig fügt Z die Adresse von S in seine eigene ARP-Tabelle ein, in der Annahme, dass er bald Datagramme an S schicken wird. S erhält die Antwort, erweitert seine ARP-Tabelle und schickt das Datagramm. Bei weiteren Datagrammen nutzen S und Z die Einträge in ihren ARP-Tabellen.

Um die ARP-Tabelle aktuell zu halten, wird jeder Eintrag nach 15 Minuten automatisch wieder entfernt. Findet in der Zwischenzeit eine weitere Kommunikation mit dem Knoten statt, wird die Lebensdauer des Eintrags entsprechend verlängert. Genau so wird die Lebensdauer verlängert, wenn ein Knoten eine ARP-Abfrage von einem anderen Knoten sieht. Diese Aktualisierung nehmen alle Knoten vor, die einen Eintrag für den Sender haben. Nicht betroffene Knoten ignorieren die ARP-Abfrage und erzeugen keinen neuen Eintrag.

Mit dem Programm `arp` kann man die Übersetzungstabellen anschauen und manuell ändern. Man kann Einträge löschen oder einfügen. Manuell eingefügte Einträge sind permanent und werden auch bei Zeitüberschreitung nicht aus dem Cache gelöscht. Auf meinem Laptop ergibt sich:

```
arp -a

Schnittstelle: 212.201.26.252 on Interface 0x1000003
  Internetadresse       Physikal. Adresse     Typ
  212.201.26.1          00-20-18-58-b8-1f     dynamisch
```

Das Gegenstück zu ARP ist RARP (Reverse Address Resolution Protocol). Damit können Rechner ohne eigenen Massenspeicher von einem RARP-Server ihre IP-Adresse erfragen.

6.6 Internet Control Message Protocol

Das Internet Protokoll wird ergänzt durch das Internet Control Message Protocol (ICMP). Über ICMP tauschen Hosts und Router Nachrichten aus. Dazu gehören unter anderem Fehlermeldungen, Informationen zur Steuerung der Übertragung sowie Echo-Anfragen und Antworten. Um welche Art von Nachricht es sich handelt, wird in einem Typ-Feld im Kopf des Paketes spezifiziert. Abhängig vom konkreten Typ folgen dann die dazu gehörenden Daten. Eine ICMP-Nachricht wird im Nutzdatenfeld eines IP-Paketes übertragen.

Eine einfache Anwendung auf der Basis von ICMP ist `ping`[1]. Ping steht für Packet InterNet Groper (das Verb *grope* bedeutet greifen oder suchend nach etwas tasten). Gleichzeitig ist Ping auch die Bezeichnung für einen kurzen Impuls, wie er bei Echolot oder Radarsuchgeräten ausgesendet wird. Die gleiche Funktion hat `ping` im Netzwerk: es schickt eine ICMP-Nachricht mit einer Echo-Anforderung an einen Host und wartet auf das Echo. Ein Beispiel:

```
>Ping www.w3.org [128.30.52.25] mit 32 Bytes Daten:

Antwort von 128.30.52.25: Bytes=32 Zeit=160ms TTL=55
Antwort von 128.30.52.25: Bytes=32 Zeit=150ms TTL=55
Antwort von 128.30.52.25: Bytes=32 Zeit=150ms TTL=55
Antwort von 128.30.52.25: Bytes=32 Zeit=150ms TTL=55

Ping-Statistik für 128.30.52.25:
    Pakete: Gesendet = 4, Empfangen = 4, Verloren = 0 (0% Verlust),
Ca. Zeitangaben in Millisek.:
    Minimum = 150ms, Maximum =  160ms, Mittelwert =  152ms
```

In diesem Fall wurden 4 Pakete geschickt. Bei allen kam eine Antwort. Das Programm gibt die jeweils benötigte Zeit aus und berechnet aus den einzelnen Werten eine kleine Statistik. Eine solche erfolgreiche Ausführung zeigt, dass die Netzwerkverbindung zu dem angegebenen Rechner funktioniert. Damit ist `ping` ein nützliches Prüfwerkzeug. Falls Fehler auftreten, kann man mittels `ping` schnell prüfen, ob die Kommunikation bis zur Ebene von IP zustande kommt. Damit lassen sich elementare Fehler wie z.B. ein ausgeschalteter Zielknoten oder ein falscher Knotenname aufdecken.

[1] Der Quellcode und vielen anderen Informationen zu dem Programm sind auf folgender Seite zu finden `www.ping127001.com/pingpage/ping.html`

6.7 Routing – eine kurze Einleitung

In der bisherigen Diskussion hatten wir behandelt, wie ein Router Pakete weiter leitet. Grundlage sind Weiterleitungstabellen, die die benötigte Information beinhalten. Wir hatten gesehen, wie damit das Problem der Weiterleitung effizient gelöst wird. Weitgehend ausgeblendet hatten wir bisher die Frage, wie die Weiterleitungstabellen gefüllt werden. Die zugrunde liegende Frage ist: wie wird die Verbindung zwischen zwei Knoten S und E festgelegt?

Wenn man das Netzwerk als Graphen sieht – eine Menge durch Kanten verbundener Knoten – ist die entsprechende Frage nach der kürzesten Verbindung zwischen zwei Knoten. In Bild 6.1 ist ein Netzwerk als Graph dargestellt. Der Graph repräsentiert allerdings nur die Verbindungen zwischen den Routern. In der Realität gehört dann zu einem Router wiederum ein lokales Netz, in dem Pakete mit den beschriebenen Methoden weiter verteilt werden.

Der Einfachheit halber wird nur zwischen verbundenen und nicht verbundenen Knoten unterschieden. Im allgemeinen werden die Kosten (Latenz, Bandbreite, Gebühren) für verschiedene Verbindungen (Kanten) unterschiedlich sein. Man kann dies berücksichtigen, indem man entsprechende Maßzahlen an die Kanten anbringt. Die Suche nach der optimalen Verbindung muss dann die entstehenden Kosten berücksichtigen.

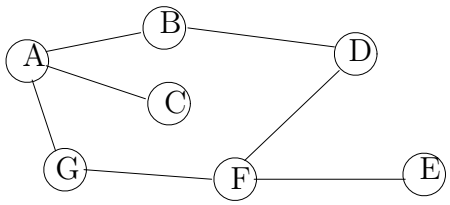


Abbildung 6.1: Netzwerk mit Knoten A,. . . ,G als Graph

Die Suche nach dem kürzesten Pfad in einem Graphen ist ein Standardproblem der Informatik. Eine klassische Anwendung ist die Bestimmung einer optimalen Route z.B. für ein Auto-Navigationssystem. Die Kosten der Kanten sind in diesem Fall die entsprechenden Fahrzeiten bestimmt durch Entfernung und Straßentyp. Zur effizienten Lösung des Problems existieren Standardlösungen. Grundvoraussetzung ist allerdings die Kenntnis des Graphs.

Die Kenntnis des Graphs – d.h. das Wissen welche Knoten aktiv sind, welche Kanten es gibt sowie die eventuellen Kosten einer Verbindungsstrecke – ist bei einem großen Netz mit ständigen Veränderungen kaum zentral verfügbar. Daher beruhen die Routingverfahren auf verteilten Algorithmen. Jeder Knoten teilt anderen Knoten sein Wissen über das Netz mit. Aus diesen Informationen erstellt jeder Knoten für sich eine Sicht des Netzes. Im Idealfall konvergieren die Algorithmen, so dass nach einer gewissen Zeit alle Knoten eine einheitliche Sicht

haben.

Die Aufgabe des Routings hängt sehr von dem betrachten Netzwerk ab. Solange es sich um ein weitgehend einheitliches Netz unter gemeinsamer Verwaltung handelt, ist die Information über den Graph relativ leicht zu erhalten. Betrachtet man demgegenüber das weltweite Internet, so ist eine einheitliche Sicht nicht mehr möglich. Verschiedene Provider haben unterschiedliche Sichtweisen der Kosten für bestimmte Verbindungen. So wird z.B. ein Provider bestrebt sein, möglichst häufig eigene Verbindungen zu benutzen, selbst wenn der resultierende Pfad nicht der kürzeste ist.

Im folgenden wird daher zwischen Intradomain- und Interdomain-Routing unterschieden. Man betrachtet ein abgeschlossenes System mit einheitlicher Administration als autonomes System (AS). Beispiele für autonome Systeme sind das interne Netzwerk eines Großunternehmens oder das Netzwerk eines Service-Providers. Dementsprechend unterscheidet man zwischen internem (intradomain) und externem (interdomain) Routing. Die Vorgehensweise bei Intradomain-Routing wird im folgenden im wesentlichen am Beispiel des Distanzvektor-Routings beschrieben. Bei der Betrachtung des Interdomain-Routings beschränkt sich die Darstellung auf einige allgemeine Eigenschaften des im Internet eingesetzten Protokolls.

6.8 Distanzvektor-Routing

Der Algorithmus des Distanzvektor-Routings basiert auf den Informationen der Knoten über ihre unmittelbaren Nachbarn. Betrachten wir nur die Anzahl der Kanten bis zu einem anderen Knoten (Hops), so erstellt ein Knoten zunächst eine Tabelle, in der alle direkten Nachbarn den Abstand 1 und alle anderen Knoten den Abstand ∞ haben. In dem obigen Beispiel hat Knoten A die Entfernungstabelle 6.4.

Tabelle 6.4: Entfernungstabelle von Knoten A

	Entfernung zu Knoten						
Knoten	A	B	C	D	E	F	G
A	-	1	1	∞	∞	∞	1

In gleicher Weise erstellen auch alle anderen Knoten ihre initiale Entfernungstabelle. Insgesamt erhält man die Tabelle 6.5.

Die Tabelle existiert allerdings nur in verteilter Form. Jeder Knoten sieht nur seine eigene Zeile. Im nächsten Schritt schickt jeder Knoten seine Information an seine Nachbarn. Knoten A erfährt beispielsweise von B, dass B mit einem Hop D erreichen kann. Damit weiß A, dass er D über B mit 2 Hops erreichen kann. Knoten A ergänzt seine Tabelle und trägt den jeweils nächsten Hop (Port) für jeden Knoten ein (Tabelle 6.6).

Tabelle 6.5: Anfangswerte der Entfernungstabellen aller Knoten

	Entfernung zu Knoten						
Knoten	A	B	C	D	E	F	G
A	–	1	1	∞	∞	∞	1
B	1	–	∞	1	∞	∞	∞
C	1	∞	–	∞	∞	∞	∞
D	∞	1	∞	–	∞	1	∞
E	∞	∞	∞	∞	–	1	∞
F	∞	∞	∞	1	1	–	1
G	1	∞	∞	∞	∞	1	–

Tabelle 6.6: Entfernungstabelle von Knoten A, nach einer Iteration

	Entfernung zu Knoten						
Knoten	A	B	C	D	E	F	G
A	-	1	1	2	∞	2	1
Port	-	B	C	B	-	G	G

Wenn alle Knoten die Informationen von ihren Nachbarn eingetragen haben, ergeben sich die Werte in Tabelle 6.7. Nach der ersten Iteration kennt Knoten G bereits die Entfernungen und die nächstgelegenen Knoten zu allen anderen Knoten im Netzwerk. Knoten E fehlen demgegenüber noch Informationen über A, B und C.

Tabelle 6.7: Entfernungstabellen aller Knoten nach einer Iteration

	Entfernung zu Knoten						
Knoten	A	B	C	D	E	F	G
A	–	1	1	2	∞	2	1
B	1	–	2	1	∞	2	2
C	1	2	–	∞	∞	∞	2
D	2	1	∞	–	2	1	2
E	∞	∞	∞	2	–	1	2
F	2	2	∞	1	1	–	1
G	1	2	2	2	2	1	–

Im nächsten Schritt verschicken die Knoten diese erweiterten Tabellen – genauer gesagt jeweils ihre eigene Zeile – an ihre Nachbarn. Die Knoten werten die Informationen aus und ergänzen ihre eigenen Tabellen. Dieses Verfahren – Versenden des aktuellen Kenntnisstandes und Aktualisieren der Tabellen – wird kontinuierlich fortgesetzt. Die Knoten prüfen dabei, ob sie zu einem Knoten eine

kürzere Verbindung als bisher in der Tabelle eingetragen finden. Für das Beispielnetz ergibt sich Tabelle 6.8.

Tabelle 6.8: Entfernungstabellen aller Knoten nach 4 Iterationen

	Entfernung zu Knoten						
Knoten	A	B	C	D	E	F	G
A	–	1	1	2	3	2	1
B	1	–	2	1	3	2	2
C	1	2	–	3	4	3	2
D	2	1	3	–	2	1	2
E	3	3	4	2	–	1	2
F	2	2	3	1	1	–	1
G	1	2	2	2	2	1	–

In dieser Art und Weise bauen die Knoten die Information für ihre Weiterleitungstabellen auf. Kein Knoten hat die vollständige Sicht über das komplette Netz, aber jeder Knoten weiß genug, um die Pakete an den jeweils richtigen Folgeknoten zu schicken.

Die Knoten senden ihre Informationen periodisch oder wenn sie eine Änderung festgestellt haben. Zum Beispiel kann ein Knoten feststellen, dass einer seiner Nachbarn keine periodischen Aktualisierungen mehr schickt. Er wird dann davon ausgehen, dass die Verbindung zu diesem Knoten gestört ist und die Einträge in seiner Tabelle, die über diesen Knoten laufen, mit der Distanz unendlich belegen. Diese Information schickt er an seine anderen Nachbarn, die daraufhin ebenfalls ihre Tabellen aktualisieren.

Im Normalfall konvergiert der Algorithmus relativ schnell wieder zu einem stabilen Zustand. Allerdings kann es zu Problemen kommen, wenn im Netz Schleifen vorhanden sind. Es existieren Verfeinerungen des Verfahrens, die das Verhalten in solchen Fällen verbessern.

Ein verbreitetes Protokoll, mit dem Router ihre Weiterleitungsinformationen austauschen ist das Routing Information Protocol RIP. RIP implementiert den beschriebenen Distanzvektor-Algorithmus. Die Informationen werden durch Pakete im RIP-Format verschickt. Die Aktualisierung erfolgt nach jeweils 30 Sekunden.

Ein alternatives Verfahren ist das Link-State-Routing. Im grundsätzliche Unterschied zum Distanzvektor-Routing verschickt dabei ein Knoten nur die Informationen über seine Nachbarn, nicht sein gelerntes Wissen über weiter entfernte Knoten. Diese eingeschränkte Information schickt jeder Router allerdings an alle anderen Router. Er setzt dazu ein als *zuverlässiges Fluten* bezeichnetes Verfahren ein.

6.9 Interdomain Routing

Wie in der Einleitung beschrieben besteht das Internet aus einer Anzahl von autonomen Systemen. Dabei gibt es verschiedene Klassen von autonomen Systemen:

- AS mit nur einer Verbindung zu einem anderen AS: z.B. eine kleine Firma mit einem internen Netz und nur einem Router (Border-Gateway) als Schnittstelle nach extern.
- AS mit mehreren externen Verbindungen: z.B. eine große Firma mit mehreren Standorten. Innerhalb des AS findet aber nur lokaler Verkehr statt.
- AS mit mehreren externen Verbindungen und der Bereitschaft, Pakete für andere AS zu übermitteln: z.B. ein Backbone-Provider.

Aufgabe des Interdomain-Routings ist es, die Erreichbarkeit zwischen den AS zu gewährleisten. Die Ansprüche sind dabei wesentlich geringer als bei dem Intradomain-Routing. Wesentliches Ziel ist es, irgendeinen Weg ohne Schleifen zu finden. Die Festlegung eines optimalen Pfads ist kaum möglich, da dann beispielsweise alle AS die gleichen Kostenmetrik verwenden müssten. Außerdem sind Kriterien wie etwa „möglichst lange im eigenen Netz“ oder „nur durch vertrauenswürdige Netze“ nur schwer zu formalisieren.

Im Internet wird das Border Gateway Protocol BGP eingesetzt. BGP ist ausgelegt, um das Routing in einen Verbund von AS zu ermöglichen. Jedes AS erhält eine eindeutige Nummer (16 bit). Jedes AS benennt mindestens einen BGP-Sprecher. Die BGP-Sprecher tauschen untereinander die Routing-Information in Form von vollständigen Weglisten aus. Ein BGP-Sprecher könnte beispielsweise mitteilen, dass er das Netzwerk 178.53.66 auf dem Weg <AS15, AS356, AS27> erreichen kann. Mit entsprechenden Aktualisierungspaketen verschickt ein BGP-Sprecher die ihm bekannten Wege oder melde Wege auch wieder ab.

Durch die Kombination von Intradomain- und Interdomain-Routing wird der große, heterogene Verbund des Internets überschaubar. Ein Router innerhalb eines AS einer Firma braucht beispielsweise nur das Border-Gateway zu kennen. Er schickt dann alle nicht intern zustellbaren Pakete an diesen Router. Umgekehrt benötigen auch der Router eines Backbone-Providers nicht alle Detailkenntnisse. Es ist ausreichend, wenn er die Netze den einzelnen AS zuordnen kann.

6.10 Domain Name System DNS

Zur besseren Übersichtlichkeit verwendet man statt der numerischen IP-Adressen Namen für Knoten. Die Auflösung der Namen erfolgt durch das Domain Name System DNS. Die Namen sind hierarchisch organisiert, wobei die einzelnen Namensteile durch Punkte getrennt sind. Die Reihenfolge ist nach zunehmender

Größe geordnet. Ein Beispiel ist `monet.fh-friedberg.de`. Der Knoten `monet` gehört zur Domäne `fh-friedberg`, die wiederum Teil von `de` ist. Die Top Level Domains sind entweder Länderkürzel oder Zuordnungen wie `edu` (Education) oder `mil` (Military).

Die gesamte Hierarchie ist in so genannte Zonen eingeteilt. Für jede Zone gibt es eine verantwortliche Instanz. Jede Zone stellt mindestens zwei Name-Server bereit, die auf Anfrage Informationen über die Adressen erteilen.

Wenn eine Anwendung die IP-Adresse eines Knoten benötigt, schickt sie eine Anfrage mit dem Namen an einen lokalen Name-Server. Wenn der lokale Name-Server den Namen nicht kennt, leitet er eine Abfrage an den Name-Server in seiner Zone weiter. Von dort erhält er entweder die gewünschte Zieladresse oder die Adresse eines weiteren Name-Servers. Im zweiten Fall fragt der lokale Name-Server bei dem weiteren Name-Server. Dieser Prozess wird fortgesetzt, bis ein Server gefunden wird, der die gewünschte Adresse angeben kann.

Zur Verringerung des Abfrageverkehrs merken sich die Name-Server in einem Cache für eine gewisse Zeit die erfragten Adressen. So kann eine neue Frage nach der Adresse direkt beantwortet werden.

Basis von DNS ist eine einfache Datenbank mit Ressourcendatensätzen. Ein Datensatz enthält ein Typfeld, mit dem verschiedene Arten von Informationen spezifiziert werden. DNS bietet dadurch Möglichkeiten wie die Definition von Alias-Namen oder Festlegung des Mail-Servers.

6.11 Internet-Standards

Das Internet ist ein loser Verbund vieler einzelner Rechner. Für den Endbenutzer ist in der Regel nur sein Internet-Provider relevant. Für die Gebühren an den Provider erhält er den Zugang und die Möglichkeit zur Datenübertragung. Die Infrastruktur des Internets kann er kostenfrei benutzen. Ob eine angefragte Adresse in der Nachbarschaft oder am anderen Ende der Welt liegt, spielt keine Rolle.

Gleichzeitig ist das Internet ein wesentlicher Teil der globalen Infrastruktur der Informationsgesellschaft. Die Ausgestaltung und die Weiterentwicklung des Internets wirft damit eine Vielzahl von technischen, wirtschaftlichen, politischen und kulturellen Fragen auf. Das Zusammenspiel verschiedenster Systeme setzt klare Standards voraus. Diese Standards bedürfen einer ständigen Anpassung an die technische Weiterentwicklung. Gleichberechtigter Zugang zu neuen Standards ist eine wesentliche Voraussetzung für einen fairen Wettbewerb. Fragen nach erlaubten Inhalten und die Garantie eines freien und sicheren Fluss von Informationen berühren unmittelbar unsere Lebenssituation.

Im Vergleich zu anderen Bereichen ist die Entwicklung des Internets geprägt von einem Geist der Offenheit und Freiwilligkeit. Weder staatliche Einrichtungen noch einzelne Firmen mit marktbeherrschender Stellung bestimmen das Inter-

net. Vielmehr entwickelte sich eine Reihe von Mechanismen zur selbstbestimmten Regulation und Weiterentwicklung. Die Dachorganisation für die verschiedenen Aktivitäten zur Koordination des Internets ist die Internet Society (ISOC, `www.isoc.org`). Ihr Ziel ist zusammen gefasst in dem Mission Statement „*To assure the open development, evolution and use of the Internet for the benefit of all people throughout the world.*“ Die Mitgliedschaft in der ISOC ist kostenfrei und steht jedem Interessenten offen. Neben der internationalen ISOC gibt es nationale Vereine. So wurde 1995 aus der 1992 gegründeten Deutsche Interessengemeinschaft Internet (DIGI e.V.) unter dem Namen ISOC.DE das „German Chapter“ der Internet Society.

Technische Themen werden in speziellen Gruppen behandelt. Allgemeine grundlegende Fragen zur Architektur des Internets sind Thema des Internet Architecture Board (IAB). Spezielle Fragen wie z.B. Protokolle werden in den verschiedenen Working Groups der Internet Engineering Task Force (IETF) behandelt. Es gibt keine formale Mitgliedschaft in der IEFT. Wer in einer der zahlreichen Arbeitsgruppen mitwirken möchte, lässt sich einfach in den entsprechenden Email-Verteiler aufnehmen. Erste Versionen von technischen Dokumente haben den Status von „Internet Drafts“. Sie werden in den Arbeitsgruppen zur Diskussion gestellt. Hat ein Internet Draft einen stabilen Stand erreicht, so kann er als Request for Comments (RFC) publiziert werden. Die einzelnen RFC werden fortlaufend nummeriert. Der derzeit (Juni 2004) aktuellste ist RFC 3846 „Mobile IPv4 Extension for Carrying Network Access Identifiers“. Über die Seite `www.ietf.org/rfc.html` hat man Zugang zu allen RFCs. Wichtig ist auch dieser Stelle die Idee des freien Zugriffs auf die Dokumente sowie die Möglichkeit, die beschriebenen Protokolle lizenzfrei nutzen zu können.

Neben technischen Dokumenten wie Spezifikationen oder ähnlichem sind auch mehr beschreibende Dokumente unter den RFC. Ein Beispiel ist RFC 1118 „Hitchhikers guide to the Internet“ mit einer allgemeinen Einführung in die Internet-Technologie. RFC 2828 „Internet Security Glossary“ ist ein umfangreiches Glossar von mehr als 200 Seiten.

Aus einigen RFC werden Internet Standards. Der Ablauf der Standardisierung ist im Detail in RFC 2026 beschrieben. Durchgeführt wird dieser Prozess von der Internet Engineering Steering Group (IESG). IESG wirkt außerdem als Schnittstelle zu anderen Standardisierungsgremien wie zum Beispiel ITU.

Einige Namen und Nummern im Internet müssen fest und eindeutig vergeben werden. Dazu zählen etwa die IP-Netzwerkadressen oder die Endungen für die Top Level Domains. Derartige Werte wurden ursprünglich von der Internet Assigned Numbers Authority (IANA) zentral vergeben und verwaltet. Mittlerweile hat Internet Corporation for Assigned Names and Numbers (ICANN) diese Aufgabe übernommen. Dabei wurde die Zuständigkeit auf vier Regional Internet Registries (RIRs) verteilt [11]. Das RIR für den europäischen Bereich ist das RIPE Network Coordination Centre (RIPE NCC) (www.ripe.net).

6.12 Übungen

1. Füllen Sie für Ihren Rechner die Felder im folgenden Formblatt aus. Benutzen Sie die Programme

   (a) `ipconfig`

   (b) `nslookup`

   (c) `ping`

   (d) `arp`

   (e) `tracert`

   mit entsprechenden Optionen, um die Informationen zu erhalten.

Hostname	
IP-Adresse	
Subnet-Mask	
Adresse ihres Netzwerks	
Netzwerk-Klasse (A, B oder C?)	
Ethernet-Adresse	
DNS-Server	
Anzahl der Hops bis zu `www.ibm.com`	

Kapitel 7

UDP und TCP

7.1 Einleitung

Bisher haben wir betrachtet, wie Pakete von einem Rechner zu einem anderen Rechner geschickt werden. Die eigentliche Kommunikation erfolgt zwischen Anwendungen, d.h. Prozessen auf den jeweiligen Betriebssystemen. Beispiele für Anwendungen sind:

- Terminalprogramme wie `telnet`
- Datentransfer `ftp`
- email Programme
- Browser

Diese Anwendungen kommunizieren mit einem entsprechenden Partner auf einem entfernten Rechner. Man spricht daher von Ende-zu-Ende-Protokollen und der Transportschicht. Diese Protokolle sind Mittler zwischen der Vermittlungsschicht und den Anwendungen. Sie müssen mit den Möglichkeiten der Vermittlungsschicht ein den Anforderungen der Anwendungen genügendes Protokoll realisieren. So muss beispielsweise für den Dateitransfer eine zuverlässige Verbindung über das unzuverlässige, best-effort IP bewerkstelligt werden.

Allerdings stellen verschiedene Anwendungen unterschiedliche Anforderungen. So benötigt `ftp` eine zuverlässigen Verbindung zur effizienten Übertragung großer Datenmengen. Ein Terminalprogramm liefert demgegenüber nur kleine Datenmengen (ein Zeichen für einen Tastendruck), die aber möglichst sofort geschickt werden sollen. Einige Anwendungen wie etwa eine Videokonferenz können gelegentliche Datenverluste akzeptieren, benötigen aber eine große Bandbreite mit einer über die Zeit relativ konstanten Qualität.

Erst auf der Ebene der Ende-zu-Ende-Protokollen können derartige Anforderung konkretisiert werden. Durch diese Arbeitsteilung wird das Protokoll der Vermittlungsebene IP entlastet. IP stellt nur die notwendige Basisfunktionalität zur

Tabelle 7.1: UDP Paket-Format

0	16 31
Quell-Port	Ziel-Port
Prüfsumme	Länge
Daten	

Verfügung und kann damit als Basis für unterschiedlichste Anwendungen dienen. Würde umgekehrt IP beispielsweise einen Schutz gegen Paketverluste gewährleisten, müssten verloren gegangene Pakete erneut geschickt werden. Durch die daraus resultierende Verzögerung wäre das Protokoll für Echtzeit-Anwendungen weniger geeignet.

Im folgenden werden die beiden Transport-Protokolle UDP und TCP vorgestellt, die die beiden Prototypen Datagrammdienst und stromorientierten Dienst über IP darstellen. Im späteren Kapitel 10 wird ein Beispiel für ein Anfrage/Antwort-Dienst vorgestellt.

7.2 UDP

Das UDP – User Datagram Protocol – ist ein einfaches Datagramm-Protokoll. Im Prinzip erweitert es lediglich den Host-zu-Host Dienst um die Adressierung einer Anwendung. Auf einen Rechner können gleichzeitig mehrere Anwendungen oder mehrere Instanzen einer Anwendung aktiv sein. Daher wird eine eindeutig Kennung für jede Anwendung benötigt.

Das Betriebssystem stellt zu diesem Zweck Kommunikationspunkte – die Ports – zur Verfügung. Eine Anwendung meldet sich an einem Port an und erhält von da an vom Betriebssystem alle an diesen Port geschickten Nachrichten. Der Port ist eine Abstraktion eines Kommunikationspunktes. Wie er konkret realisiert wird, bleibt dem Betriebssystem überlassen. Wesentlich ist, dass die UDP-Pakete die entsprechende Port-Adresse enthalten. Für die Port-Adressen stehen im UDP-Header 16 Bit zur Verfügung, wobei der Header sowohl Quell- als auch Ziel-Port enthält. Zur Vollständigkeit ist der Aufbau der UDP-Pakete in Tabelle 7.1 dargestellt. Neben den Port-Adressen enthält der Header nur noch Felder für ein optionale Prüfsumme sowie die Längeninformation.

Ein UDP-Paket wird dann für die Übermittlung in ein IP-Paket gepackt. Das IP-Paket trägt die IP-Adresse des Zielrechners. Beim Zielrechner angekommen, wird das IP-Paket entpackt und die Port-Adressen werden dem UDP-Header entnommen. Insgesamt bildet das Paar <IP-Adresse, Port-Adresse> die vollständige Adresse der Anwendung. Das IP-Paket enthält sowohl für Sender als auch Empfänger diese beiden Informationen.

Tabelle 7.2: Reservierte Portnummern

FTP-Daten	20
FTP-Steuerung	21
Telnet	23
DNS	53
WWW	80

Jeder der beiden Prozesse muss in irgendeiner Form die Port-Adressen des jeweiligen Partners kennen. Einige Dienste verwenden daher feste, öffentlich bekannte Portnummern. In Tabelle 7.2 sind für einige Anwendungen die Portnummern aufgelistet. Eine umfangreiche Liste findet man auf der Seite `www.iana.org/assignments/port` In manchen Fällen benutzen Anwendungen derartige wohlbekannte Portnummern nur, um den Kontakt aufzunehmen und für die weitere Kommunikation einen eigenen Port auszuhandeln.

7.3 TCP

UDP ist letztlich nur ein Multiplexer zwischen verschiedenen Anwendungen und dem IP-Protokoll. Es bietet keine funktionale Erweiterungen gegenüber dem unzuverlässigen Paketdienst. Viele Anwendungen benötigen aber eine zuverlässige Verbindung. Das wichtigste Protokoll für diese Fälle ist TCP Transmission Control Protocol. TCP realisiert einen zuverlässigen, verbindungsorientierten Byte-Strom. Anwendungen, die auf TCP aufsetzen, werden dadurch von vielen Implementierungsdetails entlastet. Wesentliche Eigenschaften von TCP sind:

- Verbindungsaufbau
- zuverlässige Übertragung
- Vollduplex Byte-Strom
- Flußkontrolle (Schutz des **Empfängers** vor zu vielen Daten)
- Überlastkontrolle (Schutz des **Netzes** vor zu vielen Daten)
- Multiplex für mehrere Anwendungen (analog zu UDP)

Die große Herausforderung für TCP ist, über einen unzuverlässigen Paketdienst und ein komplexes, sich ständig veränderndes Netz eine zuverlässige Verbindung mit möglichst hohem Datendurchsatz und geringer Verzögerung zu realisieren. TCP benutzt den uns aus Kapitel 3 bekannten Sliding-Window-Algorithmus. Der Basisalgorithmus wird allerdings an mehreren Punkten erweitert, um der wesentlich schwierigeren Situation Rechnung zu tragen.

7.3.1 Segmentierung

Aus Sicht der Anwendung stellt TCP einen Byte-Strom dar. D.h. die Anwendung kann einzelne Bytes schreiben oder lesen. Andererseits überträgt IP Pakete. Es ist wenig effizient, jedes Byte stets in einem eigenen IP-Paket zu verschicken. Das Verhältnis Nutzdaten zu Headerdaten ist dann sehr ungünstig. Daher setzt TCP den Byte-Strom in eine Folge von so genannten Segmenten um. Den gesamten Ablauf für eine Richtung zeigt Bild 7.1

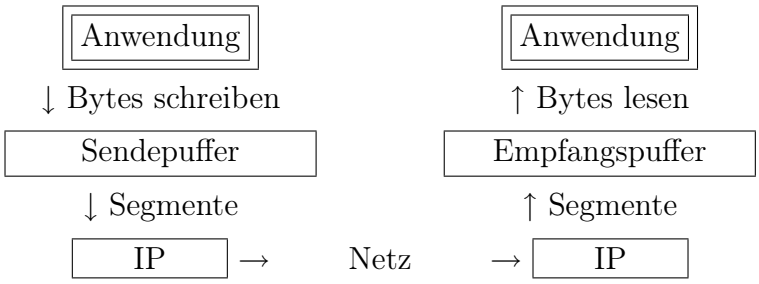


Abbildung 7.1: Segmentierung in TCP

Auf der Sendeseite sammelt TCP zunächst die geschriebenen Bytes in einem Puffer. Aus dem Puffer werden dann Bytes zu Segmenten zusammen gefasst. Im Sinne der Netzwerknutzung sind möglichst große Segmente wünschenswert, um ein gutes Verhältnis Nutzdaten zu Header-Daten zu erzielen. Anderseits führen große Segmente zu längeren Wartezeiten. Außerdem gibt es im zugrunde liegenden Netz eine Beschränkung der Paketgröße. Insgesamt verwendet TCP folgende Kriterien für das Versenden des nächsten Segments:

- Die maximale Segmentgröße (Maximum Segment Size, MSS) ist erreicht. Zweckmäßigerweise wird MSS so gewählt, dass IP das Segment nicht fragmentieren muss.
- Die Anwendung wünscht ausdrücklich den sofortigen Versand (Push-Operation). Terminalanwendungen wie `telnet` erfordern die – wenig effiziente – sofortige Weitergabe jedes Tastendrucks.
- Der Timer für die Zeit zum Aufsammeln ist abgelaufen.

Das Format der TCP-Segmente zeigt Tabelle 7.3. Die Information über die Segmentierung steckt im Feld Sequence Number. Diese Zahl bezeichnet den Index des ersten Bytes im Datenfeld. Der Header enthält keine Information zu der Länge des Datenfelds. Die Anzahl der Daten berechnet der Empfänger aus der Gesamtlänge des IP-Paketes abzüglich der beiden Header.

Wie bei UDP enthält der Header die beiden Port-Adressen, die Länge des Headers sowie eine Prüfsumme. In dem Feld Flags stehen Steuerinfomationen. Hier kann beispielsweise das Flag URG gesetzt werden, um den ersten Teil der

Tabelle 7.3: TCP Segment-Format

0	4	10	16 31
Source Port			Destination Port
Sequence Number			
Acknowledgement Number			
HdrLen	0	Flags	Advertised Window
Checksum			Urgent Pointer
Optionen + Pad			
Daten			

Daten als dringend zu kennzeichnen. Der Zeiger Urgent Pointer deutet dann auf das erste Byte, das nicht mehr zu dem dringenden Teil gehört. Die Bedeutung der anderen Felder wird im weiteren behandelt.

7.3.2 Verbindungsaufbau

TCP ist ein verbindungsorientierter Dienst. Die Kommunikation beginnt mit einer Phase zum Aufbau einer virtuellen Verbindung. Die Verbindung existiert nur auf der Ebene von TCP und nicht im physikalischen Netz. Sie geht von einer Anwendung auf einem Rechner zu einer anderen Anwendung auf einem zweiten Rechner. Die beiden Endpunkte sind jeweils durch Portadresse und IP-Adresse gegeben. Insgesamt ist die Verbindung damit durch die 4 Werte <IP-Adresse1, Port-Adresse1, IP-Adresse2, Port-Adresse2> definiert.

Eine wesentliche Aufgabe der beiden Partner ist es, sich über die Segmentnummern zu einigen. Die Designer von TCP hatten entschieden, dass für eine neue Verbindung zufällige Startwerte gewählt werden. Damit soll verhindert werden, dass eventuelle Irrläufer aus einer vorherigen Verbindung der beiden Anwendungen die neue Verbindung stören. Durch die zufälligen Startwerte können diese alten Pakete mit großer Wahrscheinlichkeit an ihren nicht mehr passenden Segmentnummern erkannt werden.

Zum Verbindungsaufbau wird ein Drei-Wege-Handshake benutzt. Dabei werden die Flags im Header zum Informationsaustausch genutzt. Der Ablauf ist wie folgt:

1. Anwendung A schickt ein Segment mit dem Flag SYN und seiner Segmentnummer N_A.

2. Anwendung B bestätigt den Eingang durch ein Segment mit den Flags SYN und ACK. Gleichzeitig setzt B die Acknowledge Number auf $N_A + 1$ um

anzugeben, welches Byte als nächstes erwartet wird. Schließlich wird die eigene Segmentnummer N_B zufällig ausgewählt und eingetragen.

3. Anwendung A erhält die Bestätigung und schickt daraufhin ebenfalls eine Bestätigung mit ACK und $N_B + 1$ als Acknowledge Number.

7.3.3 Sliding Window

Um eine zuverlässige Verbindung darzustellen, verlangt der Sender für jedes geschickte Segment eine Bestätigung. Die Bestätigung kann in einem eigenen Segment stehen oder Teil eines Segments mit Daten sein. Wir hatten bereits in der Diskussion zum einfachen Stop–and–Wait Algorithmus gesehen, dass das Netzwerk schlecht ausgelastet wird, wenn erst nach Erhalt der Bestätigung das nächste Paket geschickt wird. Diese Situation wird bei einer Verbindung im Internet noch verschärft. Die Antwortzeiten sind hier recht groß und insbesondere kann ein Paket sich z.B. durch Überlastung einer Teilstrecke deutlich verzögern. Daher muss der Sender beim Ausbleiben einer Bestätigung lange warten, bis er sicher ist, dass das Segment verloren gegangen ist.

TCP nutzt daher den Sliding Window Algorithmus und ein Sender schickt mehrere Segmente direkt nacheinander. An Hand der eingehenden Bestätigungen kann er verfolgen, welche Segmente angekommen sind. Bleibt eine Bestätigung aus, wiederholt er die alten Segmente ab dem letzten bestätigten Segment.

Schätzung der RTT

Im Internet mit den unterschiedlichsten Arten von Verbindungen, die noch dazu zeitlichen Schwankungen unterworfen sind, ist die Wahl der Wartezeit auf Bestätigungen schwierig. Ein fester Wert für den Timeout müsste an Verbindungen mit großer RTT ausgerichtet sein und wäre damit in vielen Fällen zu pessimistisch.

Daher wird bei TCP die Laufzeit adaptiv geschätzt. Der Sender misst dazu die Zeit vom Versand bis zum Erhalt der Bestätigung und benutzt diesen Wert als aktuelle Schätzung RTT_a. Diese aktuelle Schätzung kann sehr stark schwanken. Um diese Schwankungen auszugleichen, wird der Schätzwert für RTT als gleitender Mittelwert zwischen dem alten Wert und dem aktuellen Wert in der Form

$$\mathrm{RTT} = \alpha \times \mathrm{RTT} + (1 - \alpha) \times \mathrm{RTT}_a$$

gemittelt. Der Koeffizient α bestimmt, wie schnell sich die Schätzung an Änderungen adaptiert. Bei einem kleinen Wert von α geht der alte Wert nur zu einem geringen Anteil in die Berechnung ein, so dass sich Änderungen sehr schnell auswirken. Umgekehrt bewirkt ein großer Wert, d.h. α nahe an 1, dass die Änderungen langsam erfolgen.

Typische Werte liegen im Bereich von 0,8 bis 0,9. TCP benutzt dann als Wert für den Timeout das Doppelte der geschätzten RTT. Der beschriebene Algorithmus stammt aus der ursprünglichen TCP-Spezifikation. Mittlerweile werden Verfeinerungen des Algorithmus eingesetzt, die u.a. auch die Schwankung der Messwerte als Maß für die Zuverlässigkeit der Schätzung benutzen.

Flußkontrolle

Auf Seiten des Empfängers besteht die Gefahr, dass die ankommenden Daten nicht schnell genug von der Anwendung aus dem Empfangspuffer gelesen werden. Dann füllt sich der Puffer immer mehr und irgendwann ist kein Platz mehr für weitere Daten. Der Empfänger kann den Empfang neuer Segmenten nicht mehr bestätigen und der Sender muss sie später nochmals schicken. Ohne besondere Vorkehrungen würde in einem solchen Fall viel unnötiger Netzverkehr erzeugt werden. Daher verfügt TCP über einen Mechanismus, mit dem der Empfänger die Übertragungsgeschwindigkeit des Senders steuern kann.

Der Empfänger benutzt dazu das Feld Advertised Window im TCP-Header. In seinen Bestätigungsmeldungen (oder eigenen Datensegmenten) trägt er in diesem Feld ein, wie viele Bytes er zur Zeit noch aufnehmen kann. Der Sender schickt dann bis zur nächsten Bestätigung nur noch maximal so viele Bytes. Wenn beim Empfänger der Platz abnimmt weil die lokale Anwendung die Daten momentan nur sehr langsam oder gar nicht mehr liest, so reduziert er die Größe im Feld Advertised Window und bremst damit den Sender.

Im Extremfall – wenn der Puffer vollständig gefüllt ist – meldet er den Wert 0 und signalisiert damit, dass er zur Zeit keine Daten mehr aufnehmen kann. In diesem Fall schickt der Sender trotzdem weiterhin in regelmäßigen Abständen Segmente mit nur einem Datenbyte. Solange der Empfänger keine Daten aufnehmen kann, wird er die Segmente ignorieren. Hat er allerdings wieder Platz , kann er das Byte übernehmen und in der Bestätigung wieder ein größeres Advertised Window angeben. Auf diese Art und Weise wird der Transfer wieder aufgenommen, ohne dass beim Empfänger eine spezielle Vorgehensweise implementiert sein muss. Dieses Konzept, bei dem die Intelligenz beim Sender liegt, wird mit dem allgemeinen Ausdruck *smart sender/ dumb receiver rule* bezeichnet.

7.3.4 Überlastkontrolle

Ohne weitere Maßnahmen kann TCP leicht zu einer Überlastung des Netzes führen. Besonders kritisch ist, dass TCP dann wenn das Netz überlastet ist und Segmente verloren gehen mit erhöhter Aktivität reagiert indem es Segment erneut sendet. Damit verschlimmert sich eine Überlast im Netz weiter und es kann zum Kollaps kommen. Um die Situation zu verbessern, wurde Ende der achtziger Jahre eine Überlastkontrolle in TCP eingeführt. Die Überlastkontrolle basiert auf der Schätzung der Netzbelastung durch einen Sender. Auf der Basis der beob-

achteten Paketübertragungen und -verluste steuert ein Sender selbständig seine Übertragungsgeschwindigkeit.

Ein prinzipielles Problem dabei ist, dass ein Sender nur indirekt Informationen über das Netz erhält. Um das Optimum zu finden, muss er die Grenzen herausfinden. Die Grenzen findet er nur, indem er sie überschreitet. Er muss zunächst die Übertragungsrate möglichst weit erhöhen, bis er irgendwann an ein Limit stößt. Dann nimmt er die Rate zurück und nähert sich sozusagen vorsichtig von unten wieder dem Optimum. Durch diese gezielte Überschreitung des Optimums wird zusätzlich Verkehr erzeugt. Daher ist es wichtig, nach einer Überschreitung schnell wieder auf einen sicheren Wert zurück zu gehen. Umgekehrt kann das Herantasten an das Optimum langsam erfolgen.

Auf diesen Gedanken beruht das Konzept Additive Increase / Multiplicative Decrease (AIMD). Der Sender führt ein Überlastfenster, das festlegt wie viele Daten maximal bis zur nächsten Bestätigung gesendet werden dürfen. Diese Größe wird nur ausgeschöpft, wenn der Empfänger entsprechende Aufnahmekapazität signalisiert hat. Geht ein Segment verloren, vermutet der Sender eine Überlastung im Netz. In diesem Fall reduziert er sein Überlastfenster auf die Hälfte.

Umgekehrt zeigt ihm ein ACK an, dass ein Segment gut angekommen ist. Wenn alle Segmente in einem Sliding-Window Interval bestätigt wurden, erhöht er das Überlastfenster um einen kleinen Betrag. Das Fenster wird dann langsam anwachsen, bis irgendwann wieder das Limit erreicht ist. Dann wird es wieder auf die halbe Größe reduziert. Dieser Zyklus des langsamen Anwachsens und schnellen Schrumpfens wiederholt sich immer wieder. Die Priorität liegt auf dem Schutz des Netzes vor Überlast.

Das typische Verhalten des Überlastfensters zeigt Abbildung 7.2. Die Werte wurden mittels eines einfachen Simulationsprogramms erzeugt (Details dazu finden sich in der Übungsaufgabe 7.4). In der Simulation wird eine Kapazität vorgegeben. Das Überlastfenster wächst bis zum Erreichen der Kapazitätsgrenze linear an. Nach jedem Überschreiten wird das Fenster wieder auf die Hälfte des letzten Wertes reduziert. Der Wert für die Kapazitätsgrenze – in der Abbildung jeweils als Punkt eingetragen – wird in der Simulation nach jedem Zeitschritt um einen zufälligen Wert verändert.

Eine besondere Situation besteht am Anfang der Verbindung. Hier hat der Sender noch keinerlei Informationen über das Netz. Daher befindet er sich in einem Dilemma: Beginnt er mit der vollen Fenstergröße riskiert er ein Überlastung. Andererseits ist ein langsames, lineares Steigern u.U. zu pessimistisch. Daher verwendet der Sender einen Slow-Start Ansatz (im Gegensatz zu Full-Start mit der vollen Fenstergröße). In diesem Fall erhöht er das Fenster bereits nach jedem ACK. Wenn er vor dem Timeout n Segmente geschickt hat, wird im Erfolgsfall das Fenster auch n mal erhöht und nicht nur einmal wie im oben beschriebenen Additive Increase Ansatz. In Summe verdoppelt sich dadurch die Anzahl der gesendeten Segmente pro RTT. Das Verfahren ist sozusagen das Gegenstück zum Multiplicative Decrease mit umgekehrter Tendenz.

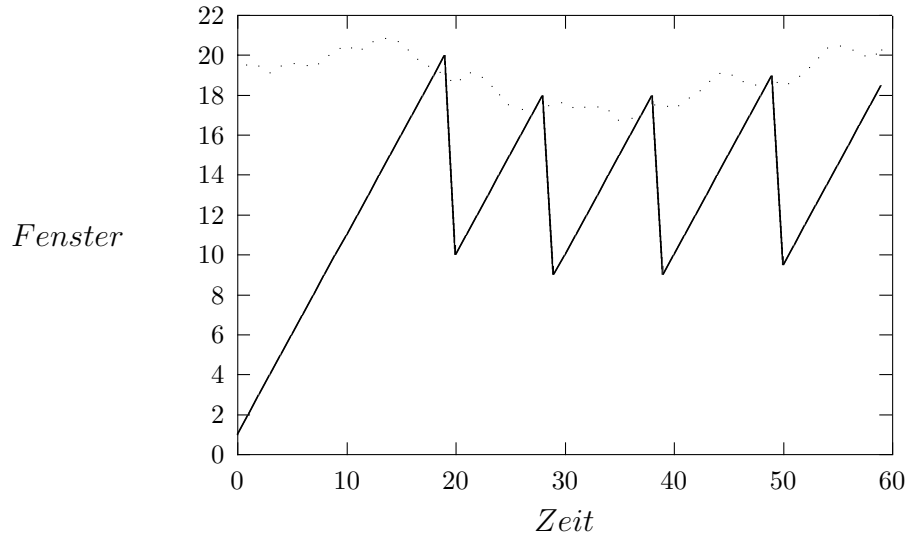


Abbildung 7.2: Regelung der Fenstergröße durch Lastkontrolle bei simulierter Kapazitätsgrenze ($\cdots$).

7.4 Übungen

Übung 7.1 *Betrachten Sie eine TCP-Verbindung mit 100Mbit/s. Wie lange dauert es bei maximaler Ausnutzung der Bandbreite, bis sich die Segmentnummer wiederholt?*

Übung 7.2 *Die Werte für die Abbildung zur Überlastkontrolle in 7.3.4 wurden mit folgendem C-Programm erzeugt.*

```
#include "stdio.h"
#include "stdlib.h"
int main(int argc, char* argv[])
{
    int t;
    double fenster    = 1;
    double kapazitaet = 20;

    for( t=0; t<60; t++ ) {
       printf("%d %.2f %.2f\n", t, fenster, kapazitaet);
       if( fenster < kapazitaet ) ++fenster;
       else fenster /= 2.;
       /* Kapazitaet um einen Wert aus [-0.6,0.6] ändern */
       kapazitaet += 1.2 * (rand()/RAND_MAX - 0.5) ;
    }
```

```
    return 0;
}
```

Ausgehend von diesem Programm:

1. *Testen Sie den Einfluss der verschiedenen Parameter auf den Verlauf.*
2. *Integrieren Sie einen Zähler für die Häufigkeit der Kapazitätsüberschreitung.*
3. *Wie lässt sich das Modell für die aktuell zur Verfügung stehende Kapazität verfeinern? Wie könnte beispielsweise ein Verhalten mit zwei Zuständen – frei und stark belastet – modelliert werden?*

Kapitel 8

ATM

8.1 Einleitung

In den späten 80er Jahren begannen die Telefongesellschaften mit der Digitalisierung ihrer Netze. Ziele waren einerseits die Verbesserung des Telefondienstes durch bessere Qualität und höheren Komfort mit neue Leistungsmerkmale. Andererseits sollte in Hinblick auf das absehbare Zusammenwachsen von klassischer Telefondienst und neuen Datenanwendungen eine einheitliche Netzstruktur bereit gestellt werden. Gleichzeitig sollte die neue Technik kompatibel zu bestehenden Installationen sein und ohne neue Verkabelung auskommen. Diese Zielsetzung führte zur Entwicklung des ISDN-Netzes (Integrated Services Digital Network).

Das ISDN-Netz besteht aus Daten- und Signalisierungskanälen. Jeder Datenkanal (B-Kanal) hat eine Bandbreite von 64 kbit/s. Diese Bandbreite ist für die Übertragung von Sprache in Telefonqualität ausgelegt. Sie ergibt sich aus einer Abtastrate von 8 kHz und Kodierung eines Abtastwertes mit 8 bit. Die Signalisierung wie z.B. Verbindungsauf- und -abbau erfolgen über einen getrennten Kanal (D-Kanal). Dieser Kanal kann auch zur Übertragung von Daten benutzt werden. Kunden können über einen Basisanschluss (2 B-Kanäle und 1 D-Kanal mit 16 kBit/s) oder über einen Primärmultiplexanschluß (30 B-Kanälen und 1 D-Kanal mit 64 kBit/s) an das ISDN-Netz angeschlossen werden.

Als Erweiterung oder Ergänzung für Anwendungen mit höherem Bedarf an Bandbreite wurde B-ISDN (Broadband ISDN) konzipiert. Es sollte sowohl den zunehmenden Datenverkehr aufnehmen als auch Grundlage für neuartige Dienste wie hochwertiges Bildtelefon oder „Video on Demand“ sein. Für einen solchen allgemeinen Einsatz ist eine Einteilung in feste Kanäle nicht mehr effizient. Daher wurde als Netztechnologie ATM – Asynchronous Transfer Mode – ausgewählt. B-ISDN wurde allerdings nicht wie geplant eingeführt. Aber aufgrund seiner guten Eigenschaften bei hohen Übertragungsrate ist ATM auch für andere Einsatzgebiete attraktiv und wird heute gerne im WAN-Bereich oder zur Verbindung mehrerer LANs eingesetzt.

Im folgenden wird eine Einführung in die Prinzipien und Anwendung von ATM gegeben. Der Schwerpunkt liegt auf den Aspekten der Netzwerk-Kommunikation. Eine ausführliche Darstellung unter stärker Betonung der Telekommunikationsanwendungen findet sich beispielsweise in [12] und [13]. Aktuelle Informationen bietet die Internet-Seite des ATM Forums (`www.atmforum.com`). Das ATM Forum wurde 1991 zur Unterstützung der ATM-Entwicklung und zur Förderung der Verbreitung gegründet. Mittlerweile sind etwa 150 Firmen Mitglieder dieser Organisation. Ziel ist die Weiterentwicklung der ATM Spezifikationen, Abstimmung mit anderen Standardisierungsgremien sowie der Informationsaustausch zwischen Anbietern und Nutzern von ATM-Geräten.

8.2 Grundlagen

Die Anforderungen an die Technologie für B-ISDN waren vielfältig. Sie sollte eine einheitliche Plattform sowohl für Anwendungen aus dem Telefonbereich als auch für die Kommunikation zwischen Rechner bieten. Gleichzeitig sollte die Technologie kostengünstig in der Realisierung und skalierbar für wachsende Bandbreiten sein. Wie in Abschnitt 2.1 diskutiert, werden entsprechenden den unterschiedlichen Anforderungen der Anwendungen zwei unterschiedliche Vermittlungsverfahren – Leitungsvermittlung und Paketvermittlung – eingesetzt. ATM kann man als einen Mittelweg zwischen beiden Techniken sehen. Die Verarbeitung beruht auf Datenpaketen wobei aber feste Verbindungen zwischen den Teilnehmern bestehen. Eine Besonderheit ist, dass alle Pakete die gleiche konstante Länge haben. In der Terminologie von ATM werden diese Pakete als Zellen bezeichnet. Die Verbindung zwischen zwei Knoten besteht aus einem konstanten Strom von derartigen Zellen. Die Zellen folgen lückenlos aufeinander. Stehen zu einem Zeitpunkt keine Daten zum Versand an, so wird eine spezielle Leerzelle in den Strom eingefügt.

Die feste Länge der Zellen verkürzt nicht unbedingt die Verarbeitungszeit pro Zelle in einem Vermittlungsknoten. Der große Gewinn gegenüber der allgemeinen Paketvermittlung liegt in der Übersichtlichkeit und Berechenbarkeit. Man kann die Situation an den Eingängen eines Switchs gut mit der Bedienung an z.B. Supermarktkassen oder Flughafenschaltern vergleichen. Wenn die Zeit zur Abfertigung eines Kundens nicht genau vorher gesagt werden kann, ist die Auswahl der richtigen, d.h. schnellsten, Schlange schwierig. Es kann passieren, dass die eigene Schlange sehr langsam abgearbeitet wird und Kunden, die sich später in einer anderen Schlange angestellt haben, schneller voran kommen. Würde demgegenüber die Abfertigung für jeden Kunden exakt gleich lange dauern, wäre die Auswahl der Warteschlange einfach und die Wartezeit wäre vorab berechenbar.

Durch die Berechenbarkeit lassen sich auch leichter Verabredungen zur Qualität einer Verbindung treffen (Dienstgüte, Quality of Service QoS). Anforderungen wie eine garantierte Mindestbandbreite oder eine Obergrenze für die Verzögerung können von einem Switch garantiert werden, indem er eine entsprechende Anzahl

von Zellen pro Zeiteinheit für die Verbindung reserviert. Wie der Name ATM besagt, sind diese Reservierungen asynchron, d.h. die Lage der Zellen einer Verbindung in dem Strom ist nicht fest. Demgegenüber wären bei einer synchronen Verbindung die Zellen für die verschiedenen Verbindungen in einem festen Raster angeordnet.

Die Vereinfachung der Vermittlung ermöglicht eine kostengünstige Realisierung von Vermittlungsgeräten. Insbesondere kann die feste Zellengröße zur Parallelisierung von Aufgaben genutzt werden. Die Weiterleitung einer Zelle ist ein klar definierte Aufgabe, für die leicht eine eigene Verarbeitungseinheit entworfen werden kann. Durch diese Parallelisierung eigenen sich ATM-Switches besonders für großen Datendurchsatz.

ATM ist verbindungsorientiert. Die Kommunikation zwischen zwei Knoten erfolgt als virtuelle Verbindung wie in Abschnitt 5.2 beschrieben. Zu Beginn der Verbindung wird ein Weg als virtuelle Verbindung aufgebaut. In dieser Signalisierungsphase können auch Verkehrsparameter für die Dienstgüte ausgehandelt werden. Die virtuellen Verbindungen sind in zwei Hierarchien organisiert. Eine unidirektionale Verbindung wird durch einen virtuellen Kanal (virtual channel, VC) bereit gestellt. Mehrere virtuelle Kanäle zwischen zwei Knoten werden in einem virtuellen Pfad (virtual path, VP) gebündelt. So können beispielsweise für eine Multimedia-Verbindung zwischen zwei Teilnehmern getrennte Kanäle für Audio- und Videosignale in einem gemeinsamen Pfad liegen. Diese zweistufige Organisation erleichtert die Aufgabe der Switches, da diese nur die Kennung für den Pfad auswerten müssen.

8.3 Zellenformat

8.3.1 Größe

Nachdem ausführlich die Vorteile einer festen Zellengröße vorgestellt wurden, stellt sich die Frage nach der optimalen Zellengröße. Kleine Zellen haben den Vorteil geringer Verzögerungzeit. Dafür ist bei vorgegebener Größe des Headers das Verhältnis zwischen Nutzdaten und Zellengröße schlechter als bei großen Zellen. Große Zellen werden auf der anderen Seite bei kurzen Nachrichten (z.B. Bestätigungsnachrichten) nur zu einem geringen Teil mit Nutzdaten gefüllt.

Diese Überlegungen zeigen, dass es keine optimale Zellengröße für alle Fälle gibt. Vielmehr erweisen sich bei unterschiedliche Anwendungen auch unterschiedliche Zellengrößen als am besten geeignet. Nicht umsonst wird im Bereich der Rechnernetze mit in der Regel vielfältigen Anwendungen eine variable Länge der Pakete bevorzugt.

Bei der Festlegung der Größe der ATM-Zellen hat man sich an der wohl derzeit noch wichtigsten Anwendung, nämlich der Übertragung von Telefongesprächen, orientiert. Die Kunden erwarten beim Telefonieren die gewöhnte Qualität und

dazu gehört eine relativ geringe Verzögerungszeit. Um die konkrete Festlegung für die Länge zu verstehen, muss man etwas weiter ausholen. Die Sprachsignale eines Teilnehmers werden bei dem Gesprächspartner zu einem Teil reflektiert und werden dann von dem ursprünglichen Sprecher als Echo gehört. Beispielsweise nimmt das Mikrophon in einem Telefonhörer auch zu einem gewissen Grad die Signale des Lautsprechers auf. Diese Echos stören normalerweise nicht sondern tragen im Gegenteil zu einem natürlichen Gesprächseindruck bei. Mit zunehmender Laufzeit werden diese Echos allerdings zunehmend gesondert wahrgenommen und beeinträchtigen damit das Gespräch. Daher werden in Verbindungen mit langer Laufzeit wie z.B. bei der Übertragung über Satelliten in der Regel spezielle Vorrichtungen zur Unterdrückung solcher Echos eingebaut (Echokompensatoren).

In den Standardisierungsgremien von ATM hatten die Vertreter aus Amerika und Europa unterschiedliche Vorschläge zur Länge der Zellen. In den USA ist aufgrund der geographischen Ausdehnung der Einsatz von Echokompensatoren ohnehin erforderlich. Daher plädierten die amerikanischen Vertreter für eine Länge von 64 Datenbytes. In den meisten europäischen Ländern sind demgegenüber Echokompensatoren nicht erforderlich. Bei dem Vorschlag von 32 Datenbytes bliebe die Verzögerung innerhalb der akzeptablen Zeit und damit könnten die Echokompensatoren zumindest bei der Kommunikation im eigenen Netz eingespart werden. Die Gremien konnten sich nicht auf einen dieser beiden Vorschläge einigen und wählten als Kompromiss den mittleren Wert von 48 Datenbytes.

8.3.2 Header

Zu dem Feld von 48 Byte für Nutzdaten kommt ein Header mit 5 Byte, so dass eine ATM-Zelle insgesamt 53 Bytes lang ist (Bild 8.1). In ATM gibt es zwei

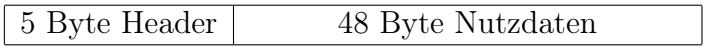


Abbildung 8.1: ATM-Zelle

Arten von Schnittstellen:

- Network node interface (NNI): die Schnittstelle zwischen zwischen zwei ATM-Switches.
- User node interface (UNI): die Schnittstelle zwischen einem ATM-Switch und einem Endpunkt.

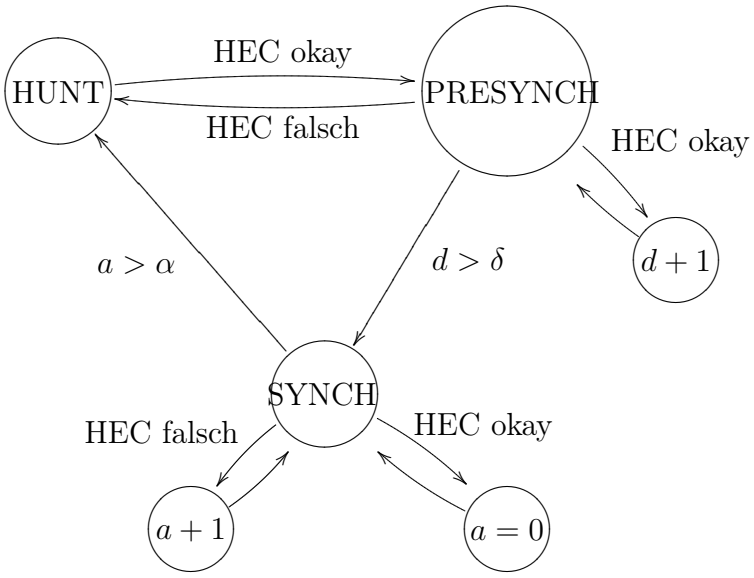
Je nach Schnittstelle unterscheiden sich die Felder im Header geringfügig. Bild 8.2 zeigt den Header für NNI. Die beiden ersten Felder enthalten die Kennung für Pfad und Kanal (Virtual Path Identifier, Virtual Channel Identifier). Das PTI-Feld (Payload Type Identifier) spezifiziert die Art der Zelle. So kann zwischen Zellen mit Benutzerdaten und Zellen zur Steuerung unterschieden werden. Mit

12 bit	16 bit	3 bit	1 bit	8 bit
VPI	VCI	PTI	CLP	HEC

Abbildung 8.2: ATM-Header bei Netzwerkinterface

dem CLP-Bit (Cell Loss Priority) können Zellen mit niedriger Priorität markiert werden. Kommt es in einem Switch zu einem Überlauf, so werden zuerst Zellen mit gesetztem CLP-Bit, d.h. Zellen mit niedriger Priorität, verworfen.

In dem Feld HEC (Header Error Control) schließlich steht die CRC-Prüfsumme über die ersten vier Bytes. Damit können Zellen mit fehlerhaften Header erkannt werden. Die Prüfsumme kann auch noch zu einem ganz anderen Zweck, nämlich der Synchronisation auf die Zellengrenzen, genutzt werden. Da die ATM-Zellen keine speziellen Anfangs- oder Endmarkierungen enthalten, sieht ein Empfänger zunächst nur einen stetigen Bitstrom. Um die Grenzen der einkommenden Zellen zu lokalisieren, entnimmt er Gruppen von jeweils 5 aufeinander folgenden Bytes. Mit jeder solchen Gruppe führt er einen Test auf korrekte Prüfsumme durch. Falls die Prüfung zu einem korrekten Ergebnis führt, handelt es sich mit großer Wahrscheinlichkeit um einen Header. Diese Hypothese kann durch Untersuchung des nächsten potentiellen Headers nach 53 Byte geprüft werden. Wenn eine vorgegebene Anzahl δ von aufeinander folgenden Headern korrekt erkannt wurde, geht der Empfänger davon aus, dass er die Synchronisation der Zellen ermittelt hat. Umgekehrt erkennt ein Empfänger auf Verlust der Synchronisation, sobald er mehr als α ungültige Header in Folge ermittelt hat. In diesem Fall sucht er wieder mit dem beschriebenen Verfahren nach gültigen Headern. Diesem Vorgehen entspricht folgendes Zustandsdiagramm:



Die Zustände haben dabei folgende Bedeutung:

- HUNT: Suche byteweise nach der ersten Kombination, die eine gültige HEC ergibt
- PRESYNCH: Prüfe weiter im Zellenabstand
  - HEC okay: Zähler d erhöhen, falls $d > \delta$ Übergang zu SYNCH
  - HEC falsch: Zurück zu HUNT
- SYNCH: Synchronisation gefunden
  - HEC okay: Zähler a auf 0
  - HEC falsch: Zähler a erhöhen, falls $a > \alpha$ Übergang zu HUNT

8.4 ATM-Anpassungsschicht

ATM bietet einen Transport von Zellen fester Größe. Es gibt keine Garantie auf Korrektheit der übermittelten Nutzdaten. Es kann auch durchaus vorkommen, dass komplette Zellen unterwegs verloren gehen. Allerdings ist gewährleistet, dass die Zellen in der korrekten Reihenfolge ankommen. Anders als bei einer reinen Paketvermittlung kann auf der virtuellen Verbindung keine Zelle eine andere überholen.

Zwischen dem Transportdienst mit Zellen und den höheren Schichten ist eine Anpassung notwendig. Eine Anpassungsschicht (ATM Adaptation Layer, AAL) übernimmt die Umsetzung von Daten der höheren Schichten auf die ATM-Schicht. Über ATM sollen verschiedenartigste Dienste mit unterschiedlichen Anforderungen transportiert werden. Die Umsetzung kann daher nicht von einer einzigen Schnittstelle gewährleistet werden. Statt dessen existieren mehrere AAL-Typen, die jeweils für eine bestimmte Art von Dienst ausgelegt sind. Im Detail unterscheidet man bei AAL zwischen zwei Subschichten:

- Convergence Sublayer (CS): Anpassung an die höheren Schichten (Konvergenzschicht).
- Segmentation and Reassembly (SAR): Aufteilung größerer Dateneinheiten auf mehrere Zellen beim Sender (Segmentation) und Rekonstruktion (Reassembly) beim Empfänger.

Zu den Aufgaben der Konvergenzschicht gehören die Erkennung von Zellenverlusten und die Synchronisation zwischen Sender und Empfänger. Dazu benötigte zusätzliche Informationen werden in den Datenteil der ATM-Zellen eingetragen. Man kann sich dies als Minizelle eingebettet in die ATM-Zelle vorstellen. Es sei betont, dass dieser Prozess nur an den jeweiligen Endpunkten einer Verbindung notwendig ist. Das ATM-Netz selbst transportiert die Zellen unabhängig von der Bedeutung der Werte im Datenteil.

Ursprünglich waren 4 verschiedene AAL-Typen vorgesehen. Es waren dies:

- AAL 1: Verbindungsorientierte Echtzeit-Dienste mit fester Bandbreite (z.B. Telefongespräche).

- AAL 2: Verbindungsorientierte Echtzeit-Dienste mit variabler Bandbreite. Hierunter fallen beispielsweise komprimierte Videosignale, bei denen der Kompressionsgrad und damit die Datenrate nicht konstant ist sondern u.a. von der Ähnlichkeit aufeinander folgender Einzelbilder abhängt.

- AAL 3: Verbindungsorientierte Dienste für Datenübertragung ohne Echtzeitanforderungen.

- AAL 4: Verbindungslose Dienste für Datenübertragung ohne Echtzeitanforderungen.

Die Typen 3 und 4 waren zur Datenübertragung zwischen Rechnern (z.B. Dateitransfer) vorgesehen. Im Laufe der Zeit wurde klar, dass beide Typen sehr ähnlich sind. Daher wurden sie zu einem gemeinsamen Typ AAL 3/4 verschmolzen. Das Protokoll beinhaltet auch die Möglichkeit zum Multiplexen mehrerer Anwendungen in einer Verbindung.

Aus Sicht der Computerindustrie war der Standard AAL 3/4 unnötig komplex. Daher entwickelte sie einen eigenen Vorschlag für Datenübertragung mit dem bezeichnenden Namen Simple Efficient Adaptation Layer (SEAL). Im Prinzip werden dabei lediglich die Daten eines Paketes der höheren Schicht auf entsprechend viele ATM-Zellen aufgeteilt und um Fülldaten, die Längeninformation sowie einen Fehlerschutz mittels CRC-32 ergänzt. Die Informationen stehen in der letzten Zelle des Pakets. Gekennzeichnet wird diese spezielle Abschlußzelle (End Of Message, EOM) durch Setzen des dritten Bits im PTI-Feld des Headers. Dieser Vorschlag wurde vom ATM-Forum als AAL 5 übernommen.

8.5 Verbindung LAN mit ATM

In großen Netzen (WAN) oder als schnelle Verbindung zwischen Routern hat ATM weite Verbreitung gefunden. Mit den beschriebenen Vorzügen und Datenraten von 25 Mbit/s bis 2,4 Gbit/s ist es eine geeignete Transporttechnologie für diese Einsatzgebiete. Demgegenüber ist der ursprünglich erwartet Durchbruch von ATM-fähigen Endgeräten weitgehend ausgeblieben. Durch Fortschritte in der Technologie und den vermehrten Einsatz von Punkt-zu-Punkt Verkabelungen bietet Ethernet bei günstigeren Preise einen ähnlich guten Datendurchsatz.

Damit ist weiterhin mit einer Koexistenz von LAN und ATM-Systemen zu rechnen. Eine Möglichkeit zur Integration ist die Nachbildung eines LANs durch ein ATM-System. In diesem Fall können alle vorhandenen LAN-Applikationen unverändert weiter verwendet werden. Ein Problem bei der Emulation ist das Fehlen

einer einfachen Möglichkeit für Broadcasts und Multicasts in ATM [14]. Insbesondere bei der Initialisierung sind Broadcasts eine wichtige Möglichkeit, um Informationen über andere Knoten zu erhalten. Ein Beispiel dafür ist das Adreßauflösungsprotokoll ARP (Abschnitt 6.5). In dem Protokoll LANE (LAN-Emulation) wird dieses Problems im wesentlichen durch einen speziellen Server gelöst. Dieser Server (Broadcast and Unknown Server, BUS) hält Multicast-Verbindungen zu allen Clients aufrecht. Ein Client schickt dann seine Multicast-Anfrage an den BUS, der die Verteilung übernimmt und Antworten an den Client zurück gibt. Der gesamte Ablauf ist recht komplex. Er beinhaltet unter anderem auch die Umsetzung der MAC-Adressen in ATM-Adressen. Ein detaillierte Beschreibung findet sich beispielsweise in der angegebenen Literatur.

8.6 Übungen

Übung 8.1

Wie viele Zellen pro Sekunde benötigt man zur Übertragung von Sprachsignalen in ISDN-Qualität über ATM? Wie ist dabei das Verhältnis zwischen Nutzdaten und Headerdaten (Gehen Sie davon aus, dass die Datenfelder komplett mit Sprachdaten gefüllt sind)?

Kapitel 9

Sockets

9.1 Einleitung

Eine elementare Schnittstelle zur Programmierung von Netzwerk-Anwendungen bieten die so genannten **Sockets** (engl. für Steckdose). Als API (Application Program Interface) stellen Sockets Funktionen zum Aufbau von Verbindungen sowie zur Kommunikation zwischen Applikationen bereit. Sockets kapseln die Details der Netzwerk-Kommunikation. Der Programmierer oder die Programmiererin kann sich auf die Realisierung der Anwendung konzentrieren. Im Idealfall funktionieren die Anwendungen dann unabhängig von Programmiersprache, Betriebssystem und Netzwerk-Interface. Das Socket API wird von allen gängigen Betriebssystemen unterstützt. Sockets setzen auf die Protokolle der Transportschicht wie TCP oder UDP auf. Aus Sicht der Anwendung ist ein Socket ein Zugang zum Netzwerk. Im folgenden wird die Programmierung am Beispiel der Programmiersprache C diskutiert. Die Verwendung in anderen Programmiersprachen ist ähnlich. Am Ende dieses Kapitels wird eine Implementierung in der Sprache Perl vorgestellt. Einige Beispiele für die Programmiersprache Java finden sich in Kapitel 12.

Das Socket API stellt zwei Verfahren zur Verfügung: verbindungs-orientierte und verbindungslose Kommunikation. Im ersten Fall müssen die beiden beteiligten Rechner zunächst eine Verbindung aufbauen, bevor sie Daten austauschen können. In beiden Fällen übernimmt einer der beiden Rechner – der Server – die Aufgabe, zunächst die Netzressource aufzubauen und dann auf Verbindungswünsche des anderen Rechner – des Clients – zu warten. Ein Server kann im allgemeinen mehrere Clients bedienen. Die Anwendung setzt nicht unbedingt eine Kommunikation zwischen Rechnern voraus, sondern kann auch für mehrere Applikationen auf dem gleichen Rechner eingesetzt werden.

9.2 Verbindungs-orientierte Kommunikation

Den prinzipiellen Ablauf zum Aufbau einer Verbindung und anschließendem Datenaustausch zeigt Tabelle 9.1. Für jeden Schritt ist der Name der entsprechenden Funktion in der Programmiersprache C angegeben.

SERVER		CLIENT	
Endpunkt anlegen	`socket()`		
Adresse festlegen	`bind()`		
Warteschlange starten	`listen()`		
Auf Anmeldung warten	`accept()`		
		Endpunkt anlegen	`socket()`
		Verbinden	`connect()`
Daten schreiben	`send()`	Daten lesen	`recv()`
Daten lesen	`recv()`	Daten schreiben	`send()`

Tabelle 9.1: Socket Funktionsaufrufe für verbindungs-orientierte Kommunikation

Eine Besonderheit ist bei der Verwendung von Sockets unter WindowsXX (Windows Sockets) zu beachten: um den Service benutzen zu können, muss zunächst eine Funktion `WSAStartup` aufgerufen werden. Dabei kann die gewünschte Version eingestellt werden. Ein Beispiel ist:

```
long WinsockStartup()
{
    long rc;

    WORD wVersionRequested;
    WSADATA wsaData;
    wVersionRequested = MAKEWORD(2, 1);

    rc = WSAStartup( wVersionRequested, &wsaData );
    return rc;
}
```

In diesem Beispiel wird die Version 2.1 spezifiziert. Nach dem Aufruf stehen in der Struktur `wsaData` Informationen über die gefundene Implementierung der Windows Sockets.

9.2.1 Funktion socket

Zu Beginn muss man das zu verwendende Protokoll spezifizieren. Dies erfolgt mit der Funktion

```
SOCKET socket( int family, int type, int protocol );
```

Das erste Argument spezifiziert die Protokollfamilie (Adressfamilie). Mögliche Werte sind unter anderem

```
AF_UNIX       Unix internes Protokoll
AF_INET       Internet Protokoll IPv4
AF_INET6      Internet Protokoll IPv6
AF_APPLETALK  AppleTalk
```

Diese Werte sind als Konstanten in entsprechenden Header-Dateien definiert. Der Präfix `AF_` steht für Adressfamilie. Gleichwertig dazu gibt es die Konstanten mit den gleichen Endungen und dem Präfix `PF_` für Protokollfamilie. Das zweite Argument `type` gibt die Art der Verbindung an. Die wichtigsten Möglichkeiten sind

```
SOCK_STREAM Strom-orientierte Verbindung
SOCK_DGRAM  Datagramm Verbindung
```

Eine weitere Option ist `SOCK_RAW`. Damit werden beispielsweise im Programm `Ping` ICMP-Pakete über einen Socket versand. Das Argument `protocol` wird in den meisten Fällen auf 0 (NULL) gesetzt. Dann wird automatisch aus den beiden anderen Argumenten das passende Protokoll ausgewählt. So resultiert aus der Kombination `AF_INET` mit `SOCK_STREAM` eine TCP Verbindung und aus `AF_INET` mit `SOCK_DGRAM` eine UDP Verbindung. Ansonsten kann man auch explizit ein Protokoll auswählen (z.B. `IPPROTO_TCP`).

Bei Erfolg liefert die Funktion einen Bezeichner für einen Socket zurück. Eigentlich handelt es sich lediglich um einen Integer Wert – sozusagen der Index des Sockets. Zur besseren Lesbarkeit des Programms kann man den Datentyp `SOCKET` verwenden. Das folgende Codefragment zeigt beispielhaft einen Aufruf, um einen Socket für TCP zu erzeugen.

```
SOCKET sockListen;

sockListen=socket(AF_INET,SOCK_STREAM,NULL);
if (sockListen == INVALID_SOCKET) {
    printf("Error: Cannot create Socket\n");
}
```

9.2.2 Funktion bind

Im nächsten Schritt muss der Server dem Socket einen Namen geben, unter dem er später erreichbar sein wird. Dazu dient die Funktion `bind` in der Form

```
int bind( SOCKET s, sockaddr *p_addr, int addrlen );
```

Das erste Argument ist der soeben angelegte Socket. Zur Übergabe der spezifischen Details wird eine Struktur `SOCKADDR` verwendet. Das zweite Argument ist ein Zeiger auf eine solche Struktur und das dritte Argument gibt die Größe der Struktur an. Für TCP verwendet man die Struktur `SOCKADDR_IN` mit folgenden Elementen:

```
struct sockaddr_in{
    short              sin_family;
    unsigned short     sin_port;
    struct in_addr     sin_addr;
    char               sin_zero[8];
};
```

Das erste Element ist die Adressfamilie. Das dritte Element ist wiederum eine Struktur, die die IP-Adresse enthält. Für unsere Zwecke reicht es aus, in dieser Unterstruktur ein Element auf `INADDR_ANY` zu setzen, um ein beliebiges Netzwerkinterface an dem eigenen Rechner zu erlauben. Entscheidend ist das Element `sin_port`. Hier wird die so genannte Portnummer eingetragen – der Name oder die Adresse für unseren Socket. Damit wird das System informiert, dass alle an diesem Port eingehenden Nachrichten an den Server weiter gereicht werden sollen. Bestimmte Dienste sind mit festen Portnummern verknüpft (z.B. Web-Server Port 80). Unter UNIX können normale Benutzer nur Portnummern oberhalb von 1024 benutzen.

Bei der Angabe der Portnummer gilt es, die für das TCP Protokoll passende Darstellung einer Integerzahl zu beachten. Verschiedene Systeme unterscheiden sich in der Reihenfolge von low-Byte und high-Byte. Das Makro `htons` (Host to network, short) wandelt Integerwerte vom Typ `short` entsprechend um. Genauer gesagt tauscht es falls erforderlich low-Byte mit high-Byte. Mit diesen Bestandteilen können wir in folgender Weise die Portnummer 5432 an den Socket binden:

```
#define  SERVER_PORT             5432

    long   rc;
    SOCKADDR_IN addr;
    int addrlen = sizeof(addr);

    addr.sin_family = AF_INET;
    addr.sin_port = htons(SERVER_PORT);
    addr.sin_addr.s_addr = INADDR_ANY;


    rc = bind(sockListen, (SOCKADDR*)&addr, sizeof(addr));
    if (rc == SOCKET_ERROR) {
        printf("Error: Cannot bind Socket\n");
```

```
}
```

9.2.3 Funktion listen

Der Socket ist jetzt angelegt und mit einer Adresse versehen. Nun kann dem System mitgeteilt werden, dass der Server bereit ist, Anfragen zu bearbeiten. Die Funktion dazu ist

```
int listen( SOCKET sock, int backlog);
```

Das erste Argument ist wieder der Socket. Das zweite Argument gibt an, wie viele Anfragen maximal in einer Warteschlange gehalten werden. Dadurch werden weitere Anfragen solange der Server noch nicht frei ist gespeichert und können dann nach und nach abgearbeitet werden. Die Größe der Warteschlange ist durch die Systemressourcen beschränkt. Eine sinnvolle Spezifikation ist `SOMAXCONN`. Dann wird automatisch die Maximalzahl eingestellt. Der entsprechende Abschnitt im Programm ist:

```
rc = listen(sockListen, SOMAXCONN);
if (rc == SOCKET_ERROR) {
    printf("Error: Cannot listen\n" );
}
```

9.2.4 Funktion accept

Wenn Anfragen in der Warteschlange sind, können sie mit der Funktion `accept` angenommen werden. Die Definition ist

```
SOCKET accept( SOCKET s, sockaddr *p_addr, int * p_len );
```

Die Funktion `accept` wartet auf eine Anfrage auf dem angegebenen Socket. Erfolgt eine Anfrage gibt die Funktion einen neuen Socket (Verbindungs-Socket) zurück. Dieser neue Socket wird dann für den Datenaustausch genutzt. Der alte Socket steht dadurch weiterhin für die Annahme weiterer Verbindungen bereit. Eine verbreitete Technik unter UNIX ist, mit dem neuen Socket einen Kindprozess zu starten. Der Kindprozess führt die gewünschte Aufgabe aus während der Elternprozess bereits wieder neue Anfragen annehmen kann [15].

Das zweite Argument enthält nach der Rückkehr aus der Funktion Informationen über den Client. Dazu wird die Struktur gefüllt, auf die der angegebene Zeiger deutet. Die Länge der Struktur wird an die im dritten Argument über Zeiger angegebene Stelle geschrieben. Beide Argumente können auch `NULL` sein. In diesem Fall wird keine Information über den Client übermittelt.

9.2.5 Funktion send und recv

Über den Verbindungs-Socket können Client und Server Daten austauschen. Aus Sicht der Applikation ähnelt der Verbindungs-Socket sehr stark einer Dateischnittstelle. In der Tat kann man auch mit den gleichen Systemfunktionen `read` und `write` aus Dateien und aus Sockets lesen beziehungsweise hinein schreiben. Eine andere Möglichkeit besteht in der Verwendung der Funktionen `recv` und `send`:

```
int recv(  SOCKET s, char * buf, int len, int flags );
int send(  SOCKET s, char * buf, int len, int flags );
```

Beide Funktionen benutzen eine Puffer, um Daten zu übertragen. Bei `recv` spezifiziert `len` die Länge des Puffers. Demgegenüber wird bei `send` die Anzahl der zu übertragenden Zeichen mitgeteilt. Das vierte Argument übernimmt optionale Spezifikationen für die Ausführung. In den meisten Fälle ist der Standardwert 0 ausreichend. Beide Funktionen geben zurück, wie viele Werte tatsächlich empfangen bzw. geschickt wurden.

9.2.6 Einfacher Server

Mit den beschriebenen Funktionen kann bereits ein Server realisiert werden. Als Client kann für Testzwecke die Anwendung `telnet` eingesetzt werden. Mit dem Befehl `telnet IP-Adresse Portnummer` wird versucht, eine Verbindung herzustellen. Ohne physikalisches Netzwerk kann man auf dem gleichen Rechner die Loopback Adresse `localhost` (127.0.0.1) verwenden. In diesem Fall werden die Nachrichten von der Netzwerkkarte direkt zurück gegeben.

Der folgende Programmcode implementiert einen einfachen Server. Der Server wartet auf Clients, schickt bei Anmeldung eine kurze Begrüßung und schließt danach unmittelbar wieder die Verbindung. Die Funktion `gethostname` wird dabei verwendet, um den Namen des eigenen Rechners zu erfragen. Dieser Name wird dann in den Begrüßungstext eingebaut. Zugunsten einer besseren Übersicht wurde auf Fehlerbehandlung verzichtet. Die Version enthält spezielle Aufrufe für Windows-Sockets. Die Socket-Funktionen stehen in der Bibliothek `ws2_32.lib`. Diese muss beim Aufruf des Linkers zusätzlich angegeben werden.

```
#include <stdio.h>
#include <winsock2.h>

#define  SERVER_PORT    1234
long WinsockStartup();

void main()
{
```

```
    long   rc;  /* Variable fuer Rueckgabewerte */

    SOCKET sockListen;
    SOCKET sockConnected;
    SOCKADDR_IN addr;
    int addrlen = sizeof(addr);

    char welcome[200];
    char hostname[200];

    addr.sin_addr.s_addr = 0;
    addr.sin_family      = AF_INET;
    addr.sin_port        = htons(SERVER_PORT);

    /* Nur unter Windows erforderlich */
    rc = WinsockStartup();

    /* Willkommenstext erstellen */
    gethostname( hostname, sizeof hostname );
    sprintf( welcome, "Willkommen bei host %s \r\n", hostname);

    /* Socket anlegen und aktivieren */
    sockListen=socket(AF_INET,SOCK_STREAM,NULL);
    bind(sockListen, (SOCKADDR*)&addr, sizeof(addr));
    listen(sockListen, SOMAXCONN);

    /* In Endlosschleife auf Clients warten
     * Begruessung schicken und
     * Verbindung wieder abbauen */
    for( ;; ) {
        printf( "Warten auf naechste Verbindung... \n" );
        sockConnected = accept( sockListen,
                                (SOCKADDR*)&addr, &addrlen);
        rc = send( sockConnected,
              welcome, strlen(welcome) + 1, NULL);
        printf("%d Bytes geschickt\n", rc );

        /* Windows Socket beenden */
        shutdown( sockConnected, SD_SEND );
        closesocket( sockConnected );
    }
}
```

```
/* ***********************************************
* Starte Socket Service unter Windows
*/
long WinsockStartup()
{
    long rc;

    /* gewünschte Version */
    WORD wVersionRequested;
    /* Datenstruktur fuer Info ueber Version */
    WSADATA wsaData;

    wVersionRequested = MAKEWORD(2, 1);  /* Short Wert aus 2 Bytes */

    rc = WSAStartup( wVersionRequested, &wsaData );
    return rc;
}
```

9.2.7 Funktion connect

Zur Realisierung eines Clients benötigt man noch die Funktion

```
int connect( SOCKET s,  sockaddr *p_addr, int addrlen );
```

Der Client kann unmittelbar einen Socket mittels `connect` verbinden. In diesem Fall erhält er automatisch vom System eine freie Portadresse. Bei Erfolg (Rückgabewert 0) steht dann bereits ein Verbindungs-Socket zur Verwendung mit `recv` und `send` bereit. Die Informationen über den Server – IP-Adresse und Portnummer – stehen in der Struktur, auf die der Zeiger `p_addr` deutet.

Der folgende Code zeigt einen einfachen Client. Dieser Client sucht nach einem Server auf Port 1234 auf dem lokalen Rechner. Nach jedem Versuch wartet er 500 ms bis zum nächsten Versuch. Gelingt die Verbindung zu einem Server, so wird eine Nachricht gelesen und der Text ausgegeben.

```
#include <stdio.h>
#include <winsock2.h>

#define  SERVER_PORT            1234
#define  RECV_BUF_MAXLEN        256

long WinsockStartup();

void main()
{
```

```
    SOCKET sock;
    SOCKADDR_IN addr;
    long   rc;
    char recvBuf[RECV_BUF_MAXLEN+1];
    char hostAdress[] = "127.0.0.1";

    rc = WinsockStartup();

    // socket anlegen
    sock=socket(AF_INET,SOCK_STREAM,NULL);

    /* Informationen fuer Verbindung */
    addr.sin_family      = AF_INET;
    addr.sin_port        = htons(SERVER_PORT);
    addr.sin_addr.s_addr = inet_addr(hostAdress);

    // socket mit Server verbinden, endlos immer wieder probieren
    while(
        connect(sock, (SOCKADDR*)&addr, sizeof(SOCKADDR))
              == SOCKET_ERROR)
    {
        printf(".");    // Lebenszeichen ausgeben
        Sleep( 500 );   // 500ms warten
    }
    // Server gefunden
    printf("Verbunden mit %s ...\n", hostAdress);
    // Nachricht abholen und ausgeben
    rc = recv(sock, recvBuf, RECV_BUF_MAXLEN, NULL);
    recvBuf[rc] = '\0';
    printf("%d Byte empfangen: %s\n", rc, recvBuf );

}
```

9.3 Verbindungslose Kommunikation

Für eine verbindungslosen Kommunikation sind weniger Schritte notwendig, da der Aufbau der Verbindung entfällt. Der Ablauf ist in Tabelle 9.2 dargestellt. Auf Seiten des Servers wird ein Socket angelegt und mit einem Port verbunden. Der Client legt ebenfalls einen Socket an. Er kann ebenfalls diesen Socket mit einem Port zu verknüpfen. In den meisten Fällen ist dies aber nicht notwendig, sondern die Portnummer wird vom System automatisch bei dem ersten Verschicken eines Paketes zugewiesen.

SERVER		CLIENT	
Endpunkt anlegen	`socket()`		
Adresse festlegen	`bind()`		
		Endpunkt anlegen	`socket()`
		Adresse festlegen	`bind()`
Daten lesen	`recvfrom()`	Daten schreiben	`sendto()`
Daten schreiben	`sendto()`	Daten lesen	`recvfrom()`

Tabelle 9.2: Socket Funktionsaufrufe für verbindungslose Kommunikation

Da mit einem Socket jetzt kein festes Ziel verbunden ist, muss die Zieladresse bei jedem Paket explizit angegeben werden. Daher wird anstelle der Funktion `send` die Funktion `sendto` verwendet. Die entsprechende Funktion zum Lesen ist `recvfrom`. Mit dieser Funktion können – ähnlich wie bei der Funktion `accept` beschrieben – Informationen über den Absender abgefragt werden. Allerdings spielt beim Lesen eines Paketes der Unterschied zwischen verbundenen und unverbundenen Sockets eine weniger große Rolle. Die Informationen über den Absender sind optional. Wenn dafür kein Bedarf besteht, kann auch die schon bekannte Funktion `recv` verwendet werden.

Der Ablauf soll wieder am Beispiel eines Programmpaares für Server und Client erläutert werden. Als minimale Funktionalität empfängt der Server Pakete und schickt jeweils eine Bestätigung an den Absender zurück. Damit eine Kommunikation überhaupt zustande kommen kann, verwendet der Server eine feste Portadresse. Der Client kann dann über den Rechnernamen und diese Portadresse ein Paket an den Server schicken. Mit diesem Paket werden die entsprechenden Adressen des Clients übermittelt. Daher kann der Client seine Portadresse frei wählen (oder vom System auswählen lassen). Der Server entnimmt die Informationen dem Paket und kann dann seine Bestätigung schicken. Für diesen Zweck stehen die beiden letzten Argumente der Funktion

```
int recvfrom(  SOCKET s, char * buf, int len, int flags,
        sockaddr *p_addr, int *p_len  );
```

zur Verfügung. Das vorletzte Argument enthält einen Zeiger auf eine Struktur, die die Informationen aufnehmen soll. Die tatsächliche Länge wird in die durch den Zeiger `p_len` angegebene Stelle geschrieben. Das Beispielprogramm wertet die Struktur aus, um die Angaben über den Absender auszugeben. Anschließend wird mit der Funktion

```
int sendto(  SOCKET s, char * buf, int len, int flags,
        sockaddr *p_addr, int len  );
```

eine Bestätigung (der Text OKAY) geschickt. Bei dem Aufruf der Funktion `sendto` wird die Struktur mit den Informationen über den Absender wiederum

zur Spezifikation der Zieladresse verwendet. Insgesamt hat der Server folgende Form:

```
/* Server für UDP */
#include <stdio.h>
#include <winsock2.h>

#define  SERVER_PORT    5432
#define  BUF_MAXLEN      256
long WinsockStartup();

void main()
{
    int   rc;  /* Variable fuer Rueckgabewerte */
    int   al;

    SOCKET sock;
    SOCKADDR_IN addr;
    SOCKADDR_IN sender;

    char buffer[BUF_MAXLEN];

    addr.sin_addr.s_addr = 0;
    addr.sin_family      = AF_INET;
    addr.sin_port        = htons(SERVER_PORT);

    /* Nur unter Windows erforderlich */
    rc = WinsockStartup();

    /* Socket anlegen und binden */
    sock = socket(AF_INET,SOCK_DGRAM,NULL);
    bind(sock, (SOCKADDR*)&addr, sizeof addr);

    /* In Endlosschleife auf Pakete warten
     *  */
    for( ;; ) {
        printf( "Warten auf naechstes Paket... \n" );
        al = sizeof sender;
        rc = recvfrom( sock, buffer, sizeof buffer, 0,
            (SOCKADDR*)&sender, &al);
        printf("%d Bytes von ", rc );
        printf("Host: %s ", inet_ntoa ( sender.sin_addr ) );
        printf("Port: %d ", ntohs( sender.sin_port ) );
```

```
        printf("<%s>\n", buffer );
        // Bestaetigung schicken
        sendto( sock, "OKAY", 5, 0, (SOCKADDR*)&sender, al);
    }
}
```

In dem Programm werden die beiden Funktionen `inet_ntoa` und `ntohs` verwendet. Die Funktion `inet_ntoa` wandelt die interne Darstellung der IP-Adresse in der Struktur in eine ASCII–Zeichenkette um. `ntohs` (*Network to host short*) ist das Gegenstück zu der uns bereits bekannten Funktion `htons`. Wie der Name sagt, konvertiert sie einen short-Wert aus dem standardisierten Netzwerkformat in die auf dem lokalen Rechner verwendetet Zahlendarstellung.

Nach dem Starten wartet der Server auf Pakete. Wenn er ein Paket erhalten hat, zeigt er die Informationen über das Paket an und schickt ein Paket mit einer Bestätigung. Der folgende Code zeigt einen dazu passenden Client.

```
/* Client für UDP */
#include <stdio.h>
#include <winsock2.h>

#define  SERVER_PORT       5432
#define  BUF_MAXLEN         256

long WinsockStartup();

int main()
{
    SOCKET sock;
    SOCKADDR_IN addr;
    long   rc;

    char buffer[BUF_MAXLEN];
    char hostAdress[] = "127.0.0.1";

    rc = WinsockStartup();

    // socket anlegen
    sock=socket(AF_INET,SOCK_DGRAM,NULL);

    // zuerst alles auf 0 setzten
    memset(&addr,0,sizeof(SOCKADDR_IN));
    addr.sin_family      = AF_INET;             // Adress Familie
    addr.sin_port        = htons(SERVER_PORT);  // Port-Nummer
```

```
    addr.sin_addr.s_addr = inet_addr(hostAdress); // host

    for( ;; ) {
        sendto( sock, "test", 5, 0, (SOCKADDR*)&addr, sizeof SOCKADDR);
        rc = recvfrom( sock, buffer, sizeof buffer, 0, 0, 0);
        if( rc > 0 ) printf("%s\n", buffer);
        else         printf(".");
        Sleep( 500 );
    }
    return 0;
}
```

Der Client schickt ein Paket an den Server und wartet anschließend auf eine Bestätigung. Eine besondere Situation entsteht, wenn der vorhergehende `sendto` Aufruf fehl schlug weil der Empfänger nicht erreichbar war. Der Socket erhielt dann eine ICMP-Meldung *Port Unreachable*. In diesem Zustand wird die Funktion `recvfrom` unmittelbar beendet und gibt einen entsprechenden Fehlerwert zurück. Im Programm werden durch den Vergleich des Rückgabewertes auf kleiner 0 solche Fehler erkannt und durch Ausgabe eines Punktes gemeldet.

9.4 Realisierung in Perl

Wie in der Einleitung erwähnt, ist die Socket-Schnittstelle nicht auf die Sprache C beschränkt, sondern steht auch in anderen Programmiersprachen zur Verfügung. Das Socket-API kapselt die Netzwerkkommunikation soweit, dass auch Anwendungen in verschiedenen Programmiersprachen über Sockets Informationen austauschen können. Als ein Beispiel sei ein UDP-Client für den oben beschriebenen Server in der Sprache Perl (Practical Extraction and Report Language) [16] angegeben. Der folgende Code zeigt einen Client, der Eingaben von der Tastatur annimmt, jede Zeile an den Server schickt, die jeweilige Antwort empfängt und am Bildschirm ausgibt.

```
# UDP-Client
use Socket;

$port = 5432;
$dest = 'localhost';

# UDP-Socket anlegen
socket( S, PF_INET, SOCK_DGRAM, 0) || die "socket: $!";

# Adresse des Servers
$hisiaddr = inet_aton($dest)       || die "unknown host";
```

```
$hispaddr = sockaddr_in($port, $hisiaddr);

# Eingaben einlesen, an Server schicken und Antwort ausgeben
while( <STDIN> ) {
    send(S, $_, 0, $hispaddr )      || die "send: $!";
    $ok = '';
    recv(S, $ok, 10, 0)             || die "recv: $!";
    print "$ok\n";
}
```

Im wesentlichen entsprechen der Ablauf und die Funktionsaufrufe dem C-Programm. Eine Vereinfachung ergibt sich aus der Tatsache, dass Zeichenketten als eigener Typ in Perl behandelt werden. Daher wird im Aufruf von `send` keine Längenangabe benötigt. Es genügt, die Variable `$_` mit der aktuellen Zeile zu übergeben.

9.5 Übungen

Übung 9.1 *Entwickeln Sie eine Anwendung* Telefonbuch. *Die Anwendung besteht aus einem Server und einem Client. Der Server verwaltet das Telefonbuch und kann z.B. die Einträge aus einer Datei lesen. Über eine Socket-Verbindung kann der Client Anfragen an den Server schicken. Eine solche Anfrage enthält einen Namen. Findet der Server zu diesem Namen einen Eintrag, so sendet er als Antwort die Telefonnummer. Andernfalls sendet er eine Fehlermeldung.*

- *Implementieren Sie den Server. (Der Schwerpunkt liegt auf der Netzwerkkommunikation, die Verwaltung des Telefonbuchs können Sie sehr einfach gestalten.)*
- *Zum Testen des Servers können Sie die Anwendung* `telnet` *verwenden.*
- *Implementieren Sie den Client.*
- *Erweitern Sie die Programme um einige Funktionalitäten, so dass z.B. über den Client auch neue Einträge eingegeben werden können.*

Übung 9.2 *In dem Protokoll UDP sind keine Maßnahmen implementiert, um größere Datenmengen in Einheiten mit jeweils nur der maximale Paketlänge der darunter liegenden Protokolle aufzuteilen. Wie kann man die maximale Paketgröße im Programm ermitteln? Was passiert, wenn diese Größe überschritten wird?*

Kapitel 10

Remote Procedure Call RPC

10.1 Einleitung

Häufig basieren Netzwerk-Anwendungen auf einem Anfrage und Antwort Schema. Ein Prozess (der Client) auf einem Rechner beauftragt einen zweiten Prozess (den Server) auf einem anderen Rechner eine bestimmte Aktion auszuführen und ein dazu gehörendes Ergebnis zu melden. In diesem Szenario steht weniger die Datenübertragung als die Aufteilung der Arbeit im Mittelpunkt.

Betrachten wir als Beispiel einen Datenbank-Server. Ein Client fragt nach allen Einträgen zu einem bestimmten Nachnamen. Der Ablauf ist:

1. Der Client schickt eine Abfrage mit dem Nachnamen und wartet anschließend.
2. Der Server sucht nach entsprechenden Einträgen.
3. Der Server sendet die Liste mit gefundenen Einträgen zurück.
4. Der Client empfängt die Antwort und arbeitet damit weiter.

Bild 10.1 zeigt den Ablauf in der Darstellung als Zeitstrahl. Bei dem Ansatz

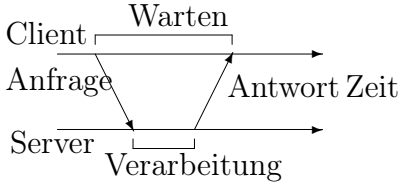


Abbildung 10.1: Ablauf in Anfrage und Antwort Schema

von Remote Procedure Call RPC betrachtet man diesen Ablauf wie den Aufruf eines Unterprogramms. Der Client entspricht dem übergeordneten Modul,

das eine Funktion mit einer Argumentliste und Rückgabewerten aufruft. Der RPC-Formalismus unterstützt diese Sichtweise durch eine Reihe von Werkzeuge. Mit deren Hilfe erfolgt die Umsetzung in den auf unterer Schicht ablaufenden Austausch von Datenpaketen weitgehend automatisch. Im allgemeinen können die beiden Prozesse in verschiedenen Programmiersprachen geschrieben worden sein und auf verschiedenen Betriebssystemen laufen. Daher ist es notwendig, eine plattformunabhängige Repräsentation der Daten einzusetzen.

Im folgenden wird zunächst eine kurze Beschreibung der grundlegenden Mechanismen von RPC gegeben. RPC ist allerdings eher ein allgemeines Konzept als ein gemeinsames Protokoll. Es gibt eine ganze Reihe von Implementierungen (sun RPC, Microsoft RPC, ...), die sich in den Details unterscheiden. Daneben stehen in anderen Programmiersprache ähnliche Möglichkeiten zur Verfügung. In Java ist dies das Konzept von Remote Methode Invocation (RMI). Im folgenden werden Details der Realisierung an Hand eines einfachen Beispiels für ein Telefonbuch besprochen. Die Entwicklung erfolgt mit Visual C unter Windows 2000. Die Darstellung versteht sich als Einführung in die Programmierung mit RPC anhand der Basisfunktionalitäten. Für weitere Möglichkeiten sei auf die umfangreiche Dokumentation verwiesen [17].

10.2 Grundlagen

RPC soll möglichst weitgehend alle Netzwerk relevanten Details übernehmen. Im Idealfall ist für den Anwender nicht sichtbar, dass die Anwendung über ein Netzwerk läuft. Die Kapselung erfolgt durch so genannte Stubs (engl. (Baum)Stumpf, Stummel). Die Stubs wirken als Mittler zwischen der Anwendung und dem RPC-Protokoll. Mit einem entsprechenden Generator oder Compiler werden die Stubs für die jeweilige Programmiersprache automatisch aus einer abstrakten Definition des Interfaces erzeugt. Der Client ruft den Client-Stub als normale Prozedur auf und übergibt die notwendigen Argumente. Im Stub erfolgt die Umsetzung auf eine netzwerktaugliche Darstellung sowie der Aufruf des RPC-Protokolls.

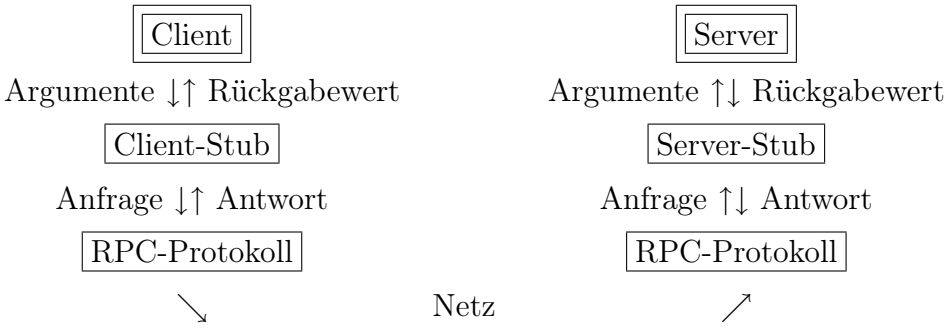


Abbildung 10.2: RPC

10.3 Beispiel

Als Beispiel soll ein Server für Anfragen nach Telefonnummern realisiert werden. Im ersten Schritt der RPC Programmentwicklung muss ein eindeutiger Bezeichner für das so genannte Interface erzeugt werden. Dazu steht das Programm `uuidgen` zur Verfügung. Eventuell muss zunächst `vcvars32.bat` ausgeführt werden, damit die Suchpfade entsprechend gesetzt sind. Der Aufruf

```
uuidgen /i /otelbu.idl
```

erzeugt dann eine Datei mit einem eindeutigen Identifikator (Universally Unique IDentifier, UUID). Auf meinem Rechner resultierte die Datei `telbu.idl` mit dem Inhalt

```
//file telbu.idl
[
uuid(49db6843-b369-4551-86f6-b3254a9b8a1a),
version(1.0)
]
interface INTERFACENAME
{

}
```

In diese IDL-Datei (IDL: Interface Definition Language) werden als nächstes ein Name für das Interface sowie die vorgesehenen Prozeduren eingetragen. Das Interface bekommt den Namen `telbu` als Abkürzung von Telefonbuch. Zunächst ist nur eine einzige Prozedur `testKomm` als Test für die Kommunikation vorgesehen. Diese Prozedur erhält als Argument eine Zeichenkette, die dann als Bestätigung für eine geglückte Übertragung auf der Seite des Servers ausgegeben wird. Damit erhält man

```
//file telbu.idl
[
uuid(49db6843-b369-4551-86f6-b3254a9b8a1a),
version(1.0)
]
interface telbu
{
void testKomm([in, string] unsigned char * testString);
}
```

In der Deklaration sind zusätzliche Attribute für den Parameter eingetragen. Die Angabe `in` legt fest, dass es sich um einen Eingabewert handelt, der von der Prozedur nicht überschrieben werden darf. Das zweite Attribut `string` deklariert den

Parameter als eine gemäß der C-Konvention mit `'\0'` abgeschlossene Zeichenkette. Damit kann der Compiler die Funktion `strlen` einsetzen, um die Länge des zu übertragenen Datenbereiches zu bestimmen. Weiterhin benötigt man ein so genanntes Application Configuration File (ACF).

```
//file: telbu.acf
[implicit_handle (handle_t telbu_IfHandle)
] interface telbu
{
}
```

Aus diesen beiden Dateien erzeugt der MIDL-Compiler (MIDL steht für Microsoft IDL) mit dem Aufruf

```
midl telbu.idl
```

C-Code für Server- und Client-Stub. Die beiden Dateien erhalten die Namen `telbu_s.c` und `telbu_c.c`. Weiterhin wird eine Header-Datei `telbu.h` angelegt.

10.3.1 Server

Nach diesen Vorbereitungen kann der Server implementiert werden. Der RPC-Mechanismus kann auf verschiedene Protokolle aufgesetzt werden. Das gewünschte Protokoll wird in Form einer Zeichenkette spezifiziert. Die Zeichenkette besteht aus einer Kennung für das RPC Protokoll: `ncacn` oder `ncadg` für Strom- oder Paketorientierte Kommunikation. Anschließend folgen die Kennungen für ein Vermittlungsprotokoll wie etwa IP und ein Transportprotokoll (z.B. TCP). Die einzelnen Bestandteile sind durch _-Zeichen getrennt. Damit ist die Spezifikation für TCP/IP `ncacn_ip_tcp`. Ein anderes Beispiel ist `ncacn_np` für die Kommunikation über *named pipes*. Das Protokoll zusammen mit dem Namen des Endpunktes – im vorliegenden Fall der Name die Portadresse – wird mit der Funktion `RpcServerUseProtseqEp` angegeben. Anschließend wird mit `RpcServerRegisterIf` das Interface registriert. Schließlich wartet mit `RpcServerListen` der Server auf Anfragen. Insgesamt erhält man folgenden Code für den Server:

```
/* file: telbus.c */
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include "telbu.h"

void main()
{
  RPC_STATUS status;
  unsigned char *pszProtocolSequence = "ncacn_ip_tcp";
```

```
  unsigned char *pszSecurity         = NULL;  /*no Security*/
  unsigned char *pszEndpoint         = "5432";
  unsigned int   cMinCalls           = 1;
  unsigned int   cMaxCalls           = 20;
  unsigned int   fDontWait           = FALSE;

  printf( "RpcServerUseProtseqEp\n");
  status = RpcServerUseProtseqEp(pszProtocolSequence,
                cMaxCalls,
                pszEndpoint,
                pszSecurity);

  if (status) exit(status);

  printf( "RpcServerRegisterIf\n");
  status = RpcServerRegisterIf(telbu_v1_0_s_ifspec,
                  NULL,
                  NULL);
  if (status) exit(status);

  printf( "RpcServerListen\n");
  status = RpcServerListen(cMinCalls,
              cMaxCalls,
              fDontWait);

  if (status) exit(status);

}  // end main()

/****************************************************/
/*         MIDL allocate and free                   */
/****************************************************/

void __RPC_FAR * __RPC_USER midl_user_allocate(size_t len)
{
  return(malloc(len));
}

void __RPC_USER midl_user_free(void __RPC_FAR * ptr)
{
  free(ptr);
}
```

Zusätzlich werden zwei Routinen zum Reservieren und Freigeben von Speicher benötigt. Die beiden Routinen sind hier lediglich Hüllen (wrapper) um die entsprechenden Funktionen `malloc` und `free` aus der Standardbibliothek von C.

10.3.2 Client

Der Client muss zunächst die RPC-Verbindung spezifizieren. Er benutzt dazu die Funktion `RpcStringBindingCompose`, um aus einer Anzahl von Parametern eine Zeichenkette zu konstruieren. Abhängig vom Protokoll müssen nicht unbedingt alle Parameter gesetzt werden. So benötigt das Protokoll über named pipes keine Netzwerkadresse, da es nur zwischen Prozessen auf einem Rechner verwendet werden kann.

Diese Zeichenkette wird dann an `RpcBindingFromStringBinding` übergeben. Bei Erfolg ist das Interface aktiviert und das zweite Argument enthält einen gültigen so genannten Handle auf das Interface. Danach kann der Client die Prozeduren aufrufen. Die Aufrufe sind im Beispiel in einen speziellen `RpcTryExcept` Block eingebettet, der eventuelle Laufzeitfehler auffängt.

Der Client ruft in diesem Beispiel 10 mal die Prozedur `testKomm`. Am Ende werden noch die belegten Ressourcen wieder frei gegeben. Das Programm hat folgende Form:

```
/* file: telbuc.c */
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include "telbu.h"

void main()
{
  RPC_STATUS status;
  unsigned char *pszUuid             = NULL;
  unsigned char *pszProtocolSequence = "ncacn_ip_tcp";
  unsigned char *pszNetworkAddress   = "127.0.0.1";
  unsigned char *pszEndpoint         = "5432";
  unsigned char *pszOptions          = NULL;
  unsigned char *pszStringBinding    = NULL;
  unsigned char *testString          = "Hallo Server";
  unsigned long ulCode;

  status = RpcStringBindingCompose(pszUuid,
                   pszProtocolSequence,
                   pszNetworkAddress,
                   pszEndpoint,
```

```
                    pszOptions,
                    &pszStringBinding);
  if (status) exit(status);
  printf("pszStringBinding: %s\n", pszStringBinding );

  status = RpcBindingFromStringBinding(pszStringBinding,
                        &telbu_IfHandle);
  if (status) exit(status);

  RpcTryExcept {
    int i;
    // mehrfach die Test-Prozedur aufrufen
    for( i=0; i<10; i++ ) {
      printf("%d ", i);
      testKomm(testString);
      Sleep( 1000 );
    }
  }
  RpcExcept(1) {
    ulCode = RpcExceptionCode();
    printf("Runtime exception 0x%lx = %ld\n", ulCode, ulCode);
  }
  RpcEndExcept

  status = RpcStringFree(&pszStringBinding);
  if (status) exit(status);

  status = RpcBindingFree(&telbu_IfHandle);
  if (status) exit(status);

  exit(0);
}  // end main()

/*****************************************************/
/*         MIDL allocate and free                    */
/*****************************************************/

void __RPC_FAR * __RPC_USER midl_user_allocate(size_t len)
{
  return(malloc(len));
}

void __RPC_USER midl_user_free(void __RPC_FAR * ptr)
```

```
{
  free(ptr);
}
```

10.3.3 Compilieren und Linken

Die Aufrufe zum Übersetzen und Linken der Programme können vorteilhaft in einer Make-Datei zusammen gefasst werden. Der folgende Text in der Datei `makefile` wird dann durch `nmake` ausgewertet und die notwendigen Aufrufe von Compiler und Linker gestartet. Das Makefile enthält auch die Abhängigkeiten von der IDL-Datei. Nach einer Änderung an dieser Datei wird durch den Aufruf von `nmake` automatisch die gesamte Verarbeitungskette gestartet. Am Ende erhält man die beiden Anwendungen `telbus` und `telbuc` für Server und Client. Die beiden Anwendungen kann man in zwei Eingabefenstern ausführen. **Hinweis:** `nmake` erwartet bei der Angabe der Abhängigkeiten im `makefile` jeweils ein Tabulator-Zeichen nach dem Doppelpunkt. Dieses Zeichen darf nicht durch Leerzeichen ersetzt werden.

```
#makefile for telbuc.exe and telbus.exe
#link refers to the linker
#conflags refers to flags for console applications
#conlibs refers to libraries for console applications

!include <ntwin32.mak>

all : telbuc.exe telbus.exe

# Make the client side application telbuc
telbuc.exe : telbuc.obj telbu_c.obj
    $(link) $(linkdebug) $(conflags) -out:telbuc.exe \
        telbuc.obj telbu_c.obj \
        rpcrt4.lib $(conlibs)

# telbuc main program
telbuc.obj : telbuc.c telbu.h
    $(cc) $(cdebug) $(cflags) $(cvars) $*.c

# telbuc stub
telbu_c.obj : telbu_c.c telbu.h
    $(cc) $(cdebug) $(cflags) $(cvars) $*.c

# Make the server side application telbus
telbus.exe : telbus.obj proc.obj telbu_s.obj
```

```
    $(link) $(linkdebug) $(conflags) -out:telbus.exe \
        telbus.obj telbu_s.obj proc.obj \
        rpcrt4.lib $(conlibsmt)

# telbu server main program
telbus.obj : telbus.c telbu.h
    $(cc) $(cdebug) $(cflags) $(cvarsmt) $*.c

# remote procedures
proc.obj : proc.c telbu.h
    $(cc) $(cdebug) $(cflags) $(cvarsmt) $*.c

# telbus stub file
telbu_s.obj : telbu_s.c telbu.h
    $(cc) $(cdebug) $(cflags) $(cvarsmt) $*.c

# Stubs and header file from the IDL file
telbu.h telbu_c.c telbu_s.c : telbu.idl telbu.acf
    midl telbu.idl
```

Mit der erfolgreichen Ausführung von `nmake` werden die beiden Dateien `telbus.exe` und `telbuc.exe` erzeugt. Nach dem Start wartet der Server auf Anfragen von Client-Prozessen. Der Client ruft in einer Schleife mehrfach die Prozedur `testKomm` auf. Bei jedem Aufruf gibt der Server die erhaltene Zeichenkette aus.

10.3.4 Abfrage-Prozedur

Nach diesen Vorbereitungen können wir die eigentliche Funktionalität Telefonbuch angehen. Zunächst wird eine Struktur mit zwei Zeichenketten zur Aufnahme der Daten Name und Nummer eingeführt. In dem folgenden Codeabschnitt wird ein Feld der Größe 100 angelegt. Die beiden ersten Einträge werden gefüllt und ein Zähler `count` wird auf die Anzahl 2 gesetzt:

```
\\ in Datei proc.c
struct {
  char *name;
  char *number;
} tb[100] = {"Maier",   "123 567",
             "Schmidt", "33 33 22" };
int count = 2;
```

Bei der Übergabe von Argumenten müssen die Besonderheiten der Netzwerkkommunikation beachtet werden. So ist ist es wenig sinnvoll, einen Zeiger zu übergeben. Die Information, dass die Daten an angegebenen Speicheradresse stehen, hat

für einen anderen Prozess auf einem anderen Rechner keine Bedeutung. Vielmehr müssen die Daten in geeigneter Form übertragen werden. Der RPC-Compiler übernimmt die Aufgabe, aus den Definitionen in der IDL-Datei entsprechenden Code zu generieren. Allerdings sind dabei die beschriebenen Beschränkungen zu beachten. Es ist konkret nicht möglich – wie sonst in C üblich – das Ergebnis einer Anfrage nach einer Telefonnummer als Zeiger auf das Resultat zurück zu geben. Eine Möglichkeit ist, das Ergebnis in einem Feld vorgegebener Größe zu schreiben. Die IDL-Datei hat dann folgende Form:

```
//file telbu.idl
[
uuid(49db6843-b369-4551-86f6-b3254a9b8a1a),
version(1.0)
]
interface telbu
{
  #define STRSIZE 100
  cpp_quote("#define STRSIZE 100")
  void testKomm([in, string] unsigned char * testString);
  int getNumber( [in, string] unsigned char * name,
                 [out ] unsigned char number[STRSIZE]);
}
```

Die Prozedur `getNumber` enthält zwei Parameter:

1. `name` mit dem Namen, für den eine Telefonnummer gesucht wird.
2. `number` als Feld, in das das Ergebnis der Suche eingetragen wird.

Der Integer-Rückgabewert ist als Ergebnismeldung vorgesehen. Das Feld `number` ist mit einer festen Größen definiert. Zur besseren Übersicht ist Größe über eine symbolische Konstante `STRSIZE` festgelegt. Die Anweisung `define` ist nur für die Auswertung durch den Generator `midl` gültig. Mit der Anweisung `cpp_quote` wird eine Kopie der Definition der Konstanten in die Datei `telbu.h` geschrieben. Dadurch ist diese Definition auch in den C-Programmen zugänglich. Die Prozedur wird durch folgende C-Funktion realisiert:

```
\\ in Datei proc.c
int getNumber( unsigned char *name, unsigned char *number ){
  int i;
  strcpy( number, "");
  for( i=0; i<count; i++ ) {
    if( strstr( tb[i].name, name ) )  {
      strcpy( number, tb[i].number);
      return i;
```

```
    }
  }
  return -1;
}
```

Mit der Bibilotheksfunktion `strstr` wird geprüft, ob die gesuchte Zeichenkette in einem der Namensfelder enthalten ist. Ist ein entsprechender Eintrag gefunden, wird die zugehörige Telefonnummer in das Ergebnisfeld kopiert und die Funktion beendet. Als Rückgabewert dient der Index des Eintrags. Kann kein passender Eintrag gefunden werden, wird dies durch den Rückgabewert `-1` signalisiert. In den Client wird eine Schleife mit Abfragen wie folgt

```
/* file: telbuc.c */
...
  unsigned char number[STRSIZE], name[STRSIZE];
...
    while( scanf("%s", name) ) {
      if( getNumber( name, number ) < 0 ) {
    printf("Kein Eintrag\n");
      } else {
    printf( "%s\n", number );
      }
    }
...
```

eingefügt. Der Client liest jeweils ein Wort ein und ruft die Prozedur zur Suche auf.

10.4 Übungen

Übung 10.1 *Erweitern Sie die Telefonbuchanwendung um:*

- *Eine Prozedur zur Eingabe neuer Einträge*
- *Eine Prozedur zur Suche aller Einträge mit einem gegebenen Namen*

Kapitel 11

Anwendungen

11.1 Einleitung

Auf die von Ende-zu-Ende Protokollen wie TCP und UDP bereit gestellten Dienste bauen Anwendungen wie email oder WWW für die Benutzer auf. Einige Anwendungen wie `telenet` nutzen direkt den Transportdienst während andere zusätzliche eigene Kommunikationsprotokolle einführen. In diesem Kapitel wird am Beispiel einiger Anwendungen diskutiert, wie diese Anwendungen die Funktionalität der Transportschicht nutzen.

11.2 WWW

Die wohl derzeit am meisten genutzte Anwendung über Rechnernetze ist das World Wide Web WWW. An diesem Beispiel lassen sich gut einige grundlegende Konzepte darstellen.

Auf Seite des Benutzers steht ein Anwendungsprogramm (User Interface), das Informationen in textueller und graphischer Form darstellt und Eingaben des Benutzers annimmt. Dieses Anwendungsprogramm ist der Web-Client oder Web-Browser wie beispielsweise Opera, Netscape, Firefox oder Microsoft Internet Explorer. Auf der anderen Seite – in der Regel über ein Netzwerk zu erreichen – steht ein Web-Server (häufig *Microsoft Internet Information Services* IIS oder Apache). Der Web-Server reagiert auf Anfragen des Clients und schickt Bestätigungen und angeforderte Informationen.

Die Kommunikation zwischen Server und Client erfolgt mittels TCP. TCP stellt einen zuverlässigen, bidirektionalen Byte-Strom bereit. Die Kommunikation hat aber die Form von Anfragen und Antworten. Daher wird ein Protokoll benötigt, das über den TCP Byte-Strom einen geeigneten Anfragen und Antwort Mechanismus realisiert. Dieses Protokoll ist HTTP: das HyperText Transport Protocol.

HTTP ist ein einheitliches Anwendungsprotokoll mit dem verschiedene Clients – d. h. Clients von unterschiedlichen Herstellern – mit unterschiedlichen Servern kommunizieren können. Die Clients und Server können auf unterschiedlichen Hard- und Software-Plattformen laufen und die Darstellung der Information durch den Client kann sich nach den jeweiligen Gegebenheiten stark unterscheiden. Aber das gemeinsame Anwendungsprotokoll ermöglicht über diese Grenzen die Interoperabilität zwischen Server und Client.

HTTP legt in seinem Anwendungsprotokoll fest, welche Anfragen möglich sind und wie die Antworten darauf aussehen. Der Inhalt der ausgetauschten Informationen ist für HTTP irrelevant. HTTP ist lediglich für den Transport zuständig. Die Informationsdarstellung ist in dem Begleitprotokoll HTML *HyperText Markup Language* geregelt. In HTML ist der Aufbau der Seite mit Formatierungsinformationen wie Textart, Textgrößen, Farben, etc. sowie die Einbindung von Elemente wie Bilder oder Videoclips beschrieben. Weiterhin kann ein Dokument Verweise (Links) auf andere Dokumente enthalten. Erweiterungen wie Java-Applets, Javascript oder Flash können in die Dokumente integriert werden.

Diese Unterteilung in Client-Anwendung, Server-Anwendung, Anwendungsprotokoll und eventuelles Begleitprotokoll findet man bei vielen Anwendungen wieder. In manchen Fällen ist die Trennung weniger deutlich sichtbar, wenn wie bei `ftp` die Anwendung den gleichen Namen wie das Anwendungsprotokoll trägt.

11.2.1 HTTP

Das Protokoll HTTP besteht aus Anfragenachrichten und Antwortnachrichten. Die Nachrichten werden in lesbarer Form als ASCII-Texte ausgetauscht. Es ist daher ohne weiteres möglich, über eine `telnet`-Verbindung selbst Anfragenachrichten an einen Server zu schicken.

Betrachten wir den Ablauf für die elementare Operation zum Lesen einer HTML-Datei. Der Standort der Datei ist als Universal Resource Locator URL spezifiziert. Ein solcher URL ist `http://www.fh-friedberg.de/index.html`. Der Ablauf ist dann wie folgt:

1. Der Client extrahiert aus dem URL den Knotennamen `www.fh-friedberg.de` des Servers.
2. Über DNS ermittelt der Client die IP-Adresse.
3. Er baut eine TCP-Verbindung zu Port 80 des Server auf.
4. Der Client schickt die Abfrage nach der Datei `index.html`.
5. Der Server antwortet und überträgt die Datei.
6. Die TCP-Verbindung wird abgebaut. (Ab der Version 1.1 besteht die Möglichkeit, die TCP-Verbindung aufrecht zu erhalten, um weitere Elemente über die gleiche Verbindung laden zu können.)

Tabelle 11.1: Anfrageoptionen in HTTP

`GET`	Lesen eines Dokuments
`HEAD`	Lesen des Headers eines Dokuments
`PUT`	Schreiben eines Dokuments
`POST`	Anhängen von Daten an bestehendes Dokument
`DELETE`	Löschen eines Dokuments
`OPTIONS`	Abfrage von verfügbaren Optionen
`TRACE`	Zu Testzwecken wird die Anfrage wie erhalten an den Client zurück geschickt.

7. Der Browser analysiert den HTML-Code der Seite und zeigt den Text an. Gegebenenfalls holt er weitere benötigte Elemente wie z.B. Bilder.

Die Anfragenachricht hat die Form:

```
GET http://www.fh-friedberg.de/index.html HTTP/1.1
```

Sie besteht aus drei Teilen:

1. dem Befehl `GET`
2. dem URL
3. einer Kennung der verwendeten Version des Protokolls.

Im allgemeinen können auf die erste Zeile der Abfrage noch mehrere weitere Zeilen mit Optionen folgen. Das Ende der Nachricht wird durch eine Leerzeile markiert. Tabelle 11.1 enthält wichtige Anfrageoptionen in HTTP. Die Optionen ermöglichen im Wesentlichen das Lesen, Schreiben und Löschen von Dokumenten auf dem Server.

In der ersten Zeile der Antwortnachricht sendet der Server seine Versionsnummer für HTTP und einen Ergebniscode mit einem erläuternden Text. Daran anschließend können optionale Informationszeilen folgen. Das Format ist *Informationsart : Informationstext*. Typische Informationen sind Beschreibung des Inhaltes oder Datum der letzten Änderung. Bei der Abfrage `GET` folgt dann der Text des angeforderten Dokuments. Als Beispiel erhält man mit der oben angegebenen Abfrage die Antwort

```
HTTP/1.0 200 OK
Server: WEBULA/1.2.3
Date: Wed, 19 Jun 2002 08:51:23 CEST
Last-Modified: Wed, 05 Jun 2002 11:00:27 CEST
MIME-Version: 1.0
Content-Type: text/html
```

Tabelle 11.2: Typen der Statuscodes in HTTP mit einigen Beispielen

Code	Typ	Erklärung
2xx:	Erfolg	Aktion empfangen, verstanden und ausgeführt
200		OK
204		No Content: Methode war erfolgreich, jedoch folgt keine Antwort im Rest der Nachricht.
3xx:	Redirection	Weitere Aktionen sind erforderlich.
301		Moved Permanently: Die angeforderte Seite ist dauerhaft umgezogen.
4xx:	Client-Fehler	Anfrage enthält fehlerhafter Syntax oder ist nicht ausführbar.
401		Unauthorized: keine Berechtigung für die angeforderte Seite.
5xx:	Server-Fehler	Eine gültige Anfrage führt zu einem Fehler im Server.
500		Internal Server Error.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
...
</HTML>
```

Die erste Zeile enthält in diesem Beispiel den Code 200 mit der Erläuterung `OK` als Bestätigung für eine erfolgreiche Transaktion. Eine Übersicht über die Statuscodes in HTTP gibt Tabelle 11.2. Die nächsten Zeilen enthalten Informationen über den Server, Datumsangaben sowie die Spezifikation des Inhalts. Getrennt durch eine Leerzeile folgt dann der HTML-Code der Seite.

11.2.2 Universal Resource Identifier

Dokumente werden über eine Adresse in Form eines Universal Resource Locators URL – eine der beiden Unterkategorien von Universal Resource Identifier URI – angesprochen. Die zweite Unterkategorie sind die Universal Resource Names URN. Dazu gehören Email oder News Adressen. URI ist ein allgemeines Schema um Ressourcen zu adressieren. Die Spezifikation ist in RFC 2396: „Uniform Resource Identifiers (URI): Generic Syntax“ beschrieben. Die allgemeine Form ist *Schema:schemaspezifischer Teil*. Im Schema-Teil wird der Dienst angegebenen. Im obigen Beispiel war dies `http`. Daneben sind u.a. noch `ftp`, `news`, `file`, `telnet` oder `mailto` möglich. Browser erlauben in der Regel auch die Eingabe

von anderen Diensten als HTTP und starten dann die entsprechende Anwendung.

Der zweite Teil hat bei HTTP die allgemeine Form *//user:password@host:port/pfad*. In den meisten Fällen werden nur ein Teil der Angaben benötigt. Weiterhin gelten Vereinbarungen wie etwa die Verwendung von `index.html` als Defaultwert für den Dateinamen. Die Abkürzung *∼username* für das Verzeichnis eines Benutzers ist aus UNIX übernommen.

Neben dem eigentlichen Namen kann man weitere Optionen an den URI anhängen. Dies ist eine einfache Möglichkeit für den Client Informationen an den Server zu schicken. Der Server entnimmt dann die Optionen dem erweiterten URI und erzeugt eine Seite mit der angeforderten Information. Als Beispiel ist der folgende URI die Anfrage an ebay nach Artikeln mit dem Schlüsselwort *Friedberg* (Zeilenwechsel und Leerzeichen sind nur zur besseren Übersicht eingefügt)[1]:

```
http://search.ebay.de/search/search.dll?MfcISAPICommand=GetResult
 &ht=1
 &shortcut=4
 &SortProperty=MetaEndSort
 &maxRecordsPerPage=50
 &st=2
 &ebaytag1code=77
 &query=friedberg
```

Der Teil bis zu dem Fragezeichen spezifiziert die Anwendung. Der Rest wird dann beim Aufruf als Argument an diese Anwendung übergeben. Im Beispiel besteht das Argument aus einer Liste mit Eigenschaften und den dazu gehörigen Werten. Die Anwendung wird vom Browser aktiviert. Sie liest die Argumente, führt eine entsprechende Aktion aus und gibt das Resultat als HTML-Text an den Browser zurück.

11.2.3 Cache

Wenn für jedes Lesen einer Web-Seite der Inhalt neu über das Netz geladen wird, entsteht sehr viel Datenverkehr. In vielen Fällen ist das neue Lesen gar nicht erforderlich, da sich der Inhalt nicht geändert hat. Daher ist es sinnvoll, die gelesenen Seiten in einem temporären Speicher – dem Cache (engl. geheimes Lager) – abzulegen. Bei einem erneuten Lesen der Seite kann dann die Kopie aus dem Zwischenspeicher verwendet werden. Dadurch wird nicht nur das Netz entlastet, sondern auch der Seitenaufbau beschleunigt.

Die Zwischenspeicherung kann an vielen Stellen erfolgen. Zunächst kann der Browser Daten lokal auf der Festplatten speichern. Weiterhin kann in einem lokalen Netz ein Knoten einen gemeinsamen Cache anbieten. Dann können mehrere

[1] Das Beispiel stammt aus einer älteren Version der ebay-Software.

Benutzer die lokale Kopie gemeinsam benutzen. Knoten mit dieser Funktionalität nennt man Proxy (engl. Stellvertreter). Neben dem Caching übernehmen Proxies weitere Aufgaben wie Regelung von Zugriffsrechten oder Umsetzung von Protokollen. So kann ein Proxy für einen Client, der nur HTTP unterstützt, eine Umsetzung auf FTP vornehmen, um damit einen FTP-Server abfragen zu können. Schließlich kann auch ein Internet Service Provider (ISP) in seinem Netz einen oder mehrere Knoten mit Cache installieren.

Das Design von HTTP unterstützt Caching durch mehrere Elemente. Bevor eine Seite aus dem Cache verwendet wird, muss sicher gestellt werden, dass die Seite noch aktuell ist. Dazu weist der Server beim Verschicken den Seiten in einem Optionsfeld `Expires` ein Gültigkeitsdatum zu. Bis zu diesem Datum kann der Cache davon ausgehen, dass die Seite noch aktuell ist. Bei späteren Zugriffen kann er über die Abfrageoption `HEAD` beziehungsweise eine spezielle Variante von `GET` prüfen, ob die Seite in der Zwischenzeit geändert wurde. Andererseits werden Seiten nach einer gewissen Zeit ohne Zugriff auch wieder aus dem Cache entfernt.

Ein Seiteneffekt des Caching-Mechanismus betrifft die häufig benutzten Zähler für Zugriffe auf Seiten. Wenn noch eine aktuelle Version in dem Cache eines Proxies liegt, wird der Zugriff eines zweiten Benutzers lokal bedient, ohne dass der Server dies zählen kann. Daher kann der Zugriffszähler einen zu niedrigen Wert enthalten.

11.3 Web-Anwendungen

Dem Protokoll HTTP funktioniert nach dem *Request-Response-Paradigma*. Der Client sendet eine Anforderung (*Request*), auf die der Server mit einer Antwort (*Response*) reagiert. Das Protokoll ist zustandslos. Jede Anfrage wird für sich alleine behandelt. Der Server behält keine Informationen aus früheren Anfragen. Im einfachsten Fall fordert der Client eine Datei an, der Server sendet diese Datei und der Client zeigt den Inhalt an.

Mittlerweile sind Anwendungen im Internet wesentlich komplexer. Moderne Anwendungen sind interaktiv und beinhalten eine Benutzerverwaltung. Gleichzeitig verschwindet die Trennung zwischen lokalen und zentralen Komponenten und Daten immer mehr. Für den Endanwender ist häufig gar nicht mehr ohne weiteres sichtbar, ob Daten lokal oder zentral abgelegt werden. Möglich wurde dies durch eine Reihe von Technologien sowohl für die Client als auch die Server-Seite.

Zunächst ist möglich, dass der Server Programme schickt, die dann im Client ausgeführt werden. Beispielsweise kann JavaScript-Code in den HTML-Text eingebettet werden. Eine typische Anwendung ist, die Benutzereingaben in Formularfeldern auf Plausibilität zu prüfen. Fehlerhafte oder fehlende Eingaben können dann schon lokal erkannt und behandelt werden. Das Netzwerk und der Server werden entlastet und - positiv für den Benutzer - die Reaktion erfolgt sofort. Komplexe Animationen können als Java Applets oder Adobe Flash Dateien übertragen

werden. In beiden Fällen benötigt der Browser zur Ausführung eine entsprechende Erweiterung (*Browser plugin*): die Java Laufzeitumgebung beziehungsweise den Adobe Flash Player. Verwendet werden diese Technologien unter anderem für interaktive Tutorials oder Spiele.

Aus Sicherheitsgründen können JavaScript-Programme und Java Applets nicht auf Dateien zugreifen. Es besteht allerdings die Möglichkeit, kleine Textdateien (Cookies) mit Informationen wie Benutzername oder Spielstand anzulegen und später wieder einzulesen. Diese Dateien werden vom Browser verwaltet. Ein vorsichtiger Benutzer kann diese Möglichkeit durch eine entsprechende Einstellung im Browser unterbinden.

Anders als bei den clientseitigen Verfahren wird bei serverseitigen Lösungen die angeforderte Seite auf dem Serversystem dynamisch erzeugt. Dazu stehen vielfältige Möglichkeiten zur Verfügung. Zunächst können die Seiten mit Skriptsprachen wie Perl oder erzeugt werden, wobei PHP derzeit wohl am meisten verbreitet ist. Der Client schickt beispielsweise eine GET-Anfrage nach der Datei `datei.php`. Der Server ruft den PHP-Interpreter mit der angegebenen Datei auf und gibt die dabei resultierende Ausgabe an den Client zurück. Mit PHP sind neben vielen anderen Anwendungen Wikipedia und die Content-Management-Systeme TYPO3 und Mambo realisiert. Eine besonders bequeme Installation für solche Fälle bietet XAMPP. Das für verschiedene Betriebssysteme (daher das *X*) vorkonfektionierte Paket beinhaltet den Webserver Apache, die Datenbank MySQL bzw. SQLite und die Skriptsprachen Perl und PHP.

Eine komplexe, mehrschichtige Architekturen für große Anwendungen ist Java EE (*Java Platform, Enterprise Edition*). Neben anderen Komponenten werden dabei Java Servlets und Java Server Pages (JSP) eingesetzt. Zur Vereinfachung der Arbeit wurden Frameworks wie Spring oder Struts entwickelt. Eine andere Möglichkeit als serverseitige Technologie für Webanwendungen ist ASP.NET (*Active Server Pages .NET*) von Microsoft auf Basis des .NET-Frameworks.

Bei großen Dateien macht sich die Übertragungszeit negativ bemerkbar. Um die Antwortzeit zu verbessern, wird bei der Technologie Ajax (Asynchronous JavaScript and XML) die Übertragung auf das tatsächlich notwendige beschränkt. Eine Anwendung, die sich dieser Technologie bedient, ist die online-Verwaltung für Bilder Flickr (www.flickr.com).

11.4 email

Elektronische Post – email – ist eine einfache aber wirkungsvolle Anwendung. Die Aufgabe ist es, ein Dokument – z.B. ein einfacher Text, ein Bild, ein Video oder eine Kombination mehrere Elemente – von einem Sender zu einem Empfänger zu schicken. Auch hier finden wir wieder die Unterteilung in Client-Anwendung, Server-Anwendung, Anwendungsprotokoll und Begleitprotokoll. Wir betrachten das Anwendungsprotokoll Simple Mail Transfer Protocol (SMTP) und das Proto-

koll Multi-Purpose Internet Mail Extension (MIME) zum Format der Nachricht.

Auf Seiten des Clients ist die Situation etwas komplizierter. Anders als bei WWW ist email eine asynchrone Anwendung. Neue mails können zu beliebigen Zeiten eintreffen. Man benötigt daher noch einen Mechanismus, um die Anwendung für das Benutzerinterface (Benutzeragent, Mail-Reader) von der Zustellung zu entkoppeln. Dabei gibt es zwei unterschiedliche Strategien.

Falls eine permanente Verbindung zum Mail-Server besteht (z.B. ein Rechner in einem lokalen Netz), dann kann ein im Hintergrund laufender Prozess als Postamt arbeiten. Dieser Mail-Daemon nimmt Mails an und legt sie im Postfach des Benutzers ab. Gleichzeitig kann er den Benutzer über die Ankunft einer neuen Mail informieren. Der Benutzer kann dann mit seinem Mail-Reader die Nachricht lesen. Umgekehrt erhält der Mail-Daemon ausgehende Nachrichten und schickt sie an den Mail-Server.

Wenn andererseits nur temporär eine Verbindung zu dem Mail-Server vorhanden ist (z.B. durch Einwahl zum ISP), benötigt man ein Protokoll, um gezielt Nachrichten abzuholen und abzugeben. Ein solches Mail-Fetching Protokoll ist das Post Office Protocol Version 3 (POP3).

Bei der Email-Anwendung ist die Trennung zwischen Client und Server weniger deutlich als bei www. Letztlich stellen beide Seiten die gleichen Funktionalität – Empfangen und Senden von Emails – bereit. Insofern ergibt sich die Unterscheidung eher aus der jeweiligen Rolle aus Sicht des Benutzers.

11.4.1 SMTP

Das Protokoll zum Austausch von Emails im Internet ist das Simple Mail Transfer Protocol SMTP. Wie HTTP ist es ein textbasiertes Protokoll auf der Basis von TCP mit Anfragen und Antworten. Ein kleines Beispiel für das Versenden einer Mail sieht wie folgt aus (Eingaben sind zur besseren Lesbarkeit mit >> gekennzeichnet):

```
>> telnet monet 25
Trying 212.201.24.18...
Connected to monet.
Escape character is '^]'.
220 monet.fh-friedberg.de ESMTP
>> HELO monet.fh-friedberg.de
250 monet.fh-friedberg.de
>> MAIL FROM <stephan.euler@mnd.fh-friedberg.de>
250 ok
>> RCPT TO <stephan.euler@t-online.de>
553 sorry, that domain isn't in my list of allowed rcpthosts (#5.7.1)
>> RCPT TO <stephan.euler@mnd.fh-friedberg.de>
250 ok
```

```
>> DATA
354 go ahead
>> Hallo
>> Hier ist eine kleine Testmail.
>>
>> .
250 ok 1024507580 qp 27898
>> QUIT
221 monet.fh-friedberg.de
```

Der Server bestätigt zunächst den erfolgreichen Verbindungsaufbau. Anschließend schickt der Client Anfragen mit Optionen. Der Server bestätigt die Anfragen mit einem dreistelligen Code und einem erläuternden Text. In dem Beispiel ist die erste Adresse nicht erlaubt und die Eingabe führt zu einem Fehler. Nach der Option `DATA` folgt der Nachrichtentext, der durch eine Zeile mit nur einem Punkt beendet wird.

11.4.2 Nachrichtenformat und MIME

Zu Beginn der Entwicklung konnte man über elektronische Post lediglich einfache Textnachrichten verschicken. Eine Nachricht besteht aus zwei Teilen: einem Kopf (Header) und einem Rumpf (Body). Jede Kopfzeile enthält einen Typ und einen dazu gehörenden Wert, getrennt durch einen Doppelpunkt. Kopfzeilen werden mit der Kombination der Zeichen für Zeilenende CR und LF (kurz <CRLF>) abgeschlossen. Eine Leerzeile trennt den Kopf vom Rumpf. Beispiele für Kopfzeilen sind:

```
Return-Path: <Manfred.Merkel@mnd.fh-friedberg.de>
Message-Id: <3C57CD5A.3020609@mnd.fh-friedberg.de>
Date: Wed, 30 Jan 2002 11:39:22 +0100
From: merkel <Manfred.Merkel@mnd.fh-friedberg.de>
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; de-DE;  ...
To: Stephan.Euler@mnd.fh-friedberg.de
Subject: [Fwd: Termin morgen]
```

Die Kopfzeilen enthalten Informationen über Sender und Empfänger, Thema, Datum, etc.. Weiterhin fügen Gateways Informationen über die Weiterleitung ein wie z.B:

```
Received: from mx0.gmx.net by merkur.hrz.uni-giessen.de for
Stephan.Euler@mnd.fh-friedberg.de; Mon, 4 Mar 2002 11:34:21 +0100
```

In der Regel steht die eigentliche Nachricht im Rumpf. Lediglich in Spezialfällen kann die gesamte Information bereits im Betreff enthalten sein (z.B. `subscribe` um sich für eine Mailing-Liste anzumelden).

Moderne Email-Systeme erlauben den Versand von nahezu beliebigen Objekten. Der Text kann formatiert sein (z.B. als HTML-Code) und Bilder, Musikstücke oder ähnliches können als Anhang (*Attachment*) mit geschickt werden. Als Standard zur Darstellung der unterschiedlichen Objekten wird Multi-Purpose Internet Mail Extension (MIME) eingesetzt. MIME umfasst

- Weitere Typen für die Kopfzeile wie `MIME-Version`.
- Definition von Inhaltstypen und -untertypen; Zwei Beispiele sind: `image/gif`, `text/richtext`.
- Einem Typ `Multipart` um mehrere Objekte zu einer gemeinsamen Email zu verbinden.

Das folgende Beispiel zeigt eine Email mit einem kurzen Text und einer MSWord-Datei.

```
MIME-Version: 1.0
Content-Type: Multipart/Mixed;
 Boundary="__Next_1024570278_Part4__"

--__Next_1024570278_Part4__
Content-Type: Text/Plain;
 Charset="ISO-8859-1"
Content-Transfer-Encoding: Quoted-Printable

Text
--__Next_1024570278_Part4__
Content-Disposition: Attachment; filename="ueb1.doc"
Content-Type: application/msword;
 Name="ueb1.doc"
Content-Transfer-Encoding: Base64

0M8R4KGxGuEAAAAAAAAAAAAAAAAAAAAAPgADAP7/CQAGAAAAAAAABAAAALgAAAAA
AAAAEAAAMAAAAAEAAAD+////AAAAAC0AAAD/////////////////////////////
////////////////////////////////////////////////////////////////
```

Die Datei wird nicht binär übertragen, um eventuelle Probleme mit der Darstellung von Binärdaten auf den diversen Gateways zu vermeiden. In dem Beispiel wird eine Base64 Kodierung benutzt. Je 3 Bytes werden zu einer 24 Bit Einheit zusammen gefasst, aus der dann 4 ASCII Zeichen mit je 6 Bit generiert werden. Die folgende Darstellung zeigt die Zuordnung der einzelnen Bit.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1. Byte								2. Byte								3. Byte							
1. Zeichen						2. Zeichen						3. Zeichen				4. Zeichen							

Die Darstellung beschränkt sich damit auf nur 64 Zeichen: Buchstaben, Ziffern und die beiden Sonderzeichen + und /. In den Kopfzeilen des Teils stehen alle Informationen, um die Form der ursprünglichen Datei wieder herzustellen.

11.4.3 Adressierung

Aus der Email-Adresse extrahiert der Mail-Daemon die Zieladresse. Mittlerweile haben die allermeisten Adressen ein klares und leicht verständliches Format in der Art *Vorname.Name@Institut*. Der Teil nach dem @ enthält den Namen des zuständigen Mail-Servers. Die zugehörige IP-Adresse kann dann über DNS ermittelt werden. Häufig werden Emails jedoch nicht direkt, sondern über spezielle Zwischenstationen (Gateways) geleitet. Diese Gateways übernehmen auch, falls erforderlich, eine Zwischenspeicherung. Anders als IP-Pakete sollen Emails nicht ohne weiteres verworfen werden, wenn die Weiterleitung momentan nicht möglich ist. Eine weitere Aufgabe von Gateways kann die Überprüfung von Emails auf verdächtigen Inhalt wie z.B. Viren sein.

Der eigentliche Zielknoten – d.h. der Rechner auf dem der Adressat seine Emails anschauen wird – ist in der Regel nicht bekannt und kann sich darüber hinaus von Fall zu Fall ändern. Daher wird die Email zunächst nur bis zum Mail-Server geschickt. Ist der Benutzer angemeldet und hat eine aktiven Mail-Daemon, so kann die Nachricht von dort aus direkt zugestellt werden. Ansonsten wird bei der nächsten Anmeldung des Benutzers an einem Rechner durch den dann gestarteten Mail-Daemon der Server informiert. Alternativ kann der Benutzer selbst aktiv die Emails abfragen. Unter Umständen (z.B. Urlaub) muss der Mail-Server die Nachrichten über entsprechend längere Zeit vorhalten.

11.5 Usenet

Neben der direkten Kommunikation über Email besteht auch die Möglichkeit, Informationen in Diskussionsforen auszutauschen. Der Ablauf ähnelt der Veröffentlichung von Zeitungsartikeln:

- Die Teilnehmer können Beiträge (Artikel, engl. *news*) schreiben und in dem Forum veröffentlichen.
- Andere Teilnehmer können auf diese Artikel antworten. Diese Antworten werden als neue Beiträge veröffentlicht, auf die wiederum reagiert werden kann. Zur besseren Übersicht werden alle Antworten auf einen Artikel als gemeinsamer Baum dargestellt. Man spricht dann von *Threads* – dem englischen Wort für Faden oder Zwirn. Dementsprechend markiert der Newsreader von Mozzila solche Diskussionen mit dem Symbol einer Zwirnrolle.

Diese Diskussionsforen (newsgroups) sind auf speziellen Newsservern abgelegt. Ein Verbund vieler Newsserver bildet das USENET (urspr. Unix User Network).

Die einzelnen Server tauschen untereinander ständig neue Artikel aus (Replikation), so dass ein einheitlicher Stand gewährleistet ist. Allerdings wird ein Server nicht alle Newsgroups bereit stellen. Daneben ist es auch möglich, mit der gleichen Technologie private oder interne Newsserver zu betreiben.

Die Vielzahl von Diskussionsgruppen ist hierarchisch unterteilt. Die einzelnen Namensbestandteile sind durch Punkte getrennt und es gilt die Regel „Vom allgemeinen zum speziellen“. Die erste Kürzel spezifiziert das Hauptthema oder ein Land. Einige Beispiele sind:

- `comp.speech.research` Ein Forum zur Forschung in der Sprachverarbeitung.
- `de.sci.informatik.ki` Eine deutsche Gruppe zum Thema *Künstliche Intelligenz*.
- `rec.games.majong` Alles über das Spiel Majong (Die Kürzel *rec* steht für *recreation*, d.h. Erholung).

Newsgroups findet man über spezielle Suchdienste wie z.B. findolin (http://www.findolin.com/). Die allermeisten Newsgroups sind unmoderiert. Allerdings wird an die Teilnehmer appelliert, bestimmte Regeln einzuhalten. Man sollte sich klarmachen, dass ein eigener Beitrag von Millionen von Menschen gelesen werden könnte. Daher ist Klarheit und Sorgfalt notwendig, um eventuellen Missverständnisse vorzubeugen. Allzu schnell wird ein Beitrag falsch verstanden und führt dann zu einem erbitterten Austausch von News. In RFC 1855 „Netiquette Guidelines‘‘ sind allgemeine Regeln für Newsgroups und andere Kommunikationsdienste zusammen gestellt.

Um an Newsgroups teilzunehmen, benötigt man einen entsprechen *Newsreader*. Dazu stehen spezielle Programme zur Verfügung. Alternativ bieten die meisten Email-Programme die entsprechende Funktionalität. Man muss dann nur noch einen News-Server auswählen. Das kann der Server eines Netzanbieters sein (z.B. `news.t-online.de` bei T-Online). Daneben gibt es kommerzielle oder frei verfügbare Server wie z.B. `www.individual.de` der Freien Universität Berlin. Der komplette Verweis auf eine Gruppe hat dann die Form

```
news://news.t-online.de/de.comm.internet.misc
```

Der Zugang ist ebenfalls über die Seiten der Firma Google (`groups.google.de`) möglich.

Der Austausch von Artikeln erfolgt über das Network News Transfer Protocol (NNTP). Dieses Protokoll spezifiziert eine Anzahl von Befehlen, um über eine strom-orientierte Verbindung Informationen und die Artikel zu übertragen. Es umfasst sowohl die Kommunikation zwischen Newsserver und Newsreader als auch den Abgleich zwischen zwei Newsservern. Für die Nachrichten wird die

gleiche Formatierung wie für Emails verwendet. Über entsprechende Kopfeinträge werden die erforderlichen Informationen übermittelt. So wird im Eintrag `Newsgroups:` festgelegt, zu welcher Gruppe oder zu welchen Gruppen ein Artikel gehört. Der Rumpf der Nachricht wird mittels MIME kodiert.

11.6 Netzwerkmanagement

In der Diskussion der verschiedenen Protokollebenen hatten wir gesehen, dass die Designer Wert auf ein weitgehend selbständiges, unüberwachtes Funktionieren des Netzes gelegt hatten. Die Protokolle enthalten Mechanismen, um mit auftretenden Fehler umgehen zu können und die Knoten passen teilweise ihr Verhalten an die aktuelle Situation an. Trotzdem benötigt man Möglichkeiten, das Verhalten eines Netzes im Detail untersuchen zu können.

Ein dazu häufig verwendetes Protokoll ist Simple Netwerk Management Protocol (SNMP). SNMP erlaubt es, zahlreiche Parameter von anderen Knoten abzufragen. Jeder Knoten, der an diesem System teilnimmt, stellt einen SNMP-Server bereit. Dann kann man von anderen Systemen aus mit einem Client Informationen abfragen SNMP realisiert dazu ein Abfrage-Antwort-Protokoll auf der Basis von UDP. Die beiden wesentlichen Operationen sind `GET` um Werte zu lesen und `SET` um Werte zu setzen.

Die Informationen werden im laufenden Betrieb gesammelt und in einer speziellen Datenbank – Management Information Base MIB – abgelegt. Über ein spezielles Identifizierungssystem kann man jede Variable im MIB abfragen. SNMP verwendet dann eine standardisierte Darstellung zur Übermittlung von Datentypen wie Integer-Zahlen oder Gleitkommazahlen.

11.7 Multimedia-Kommunikation

Mit seiner ständig wachsenden Ausdehnung und Verfügbarkeit ist das Internet auch als Plattform für Multimedia-Anwendungen interessant. Unter Multimedia im eigentlichen Sinne wird die Integration verschiedener Medien in einer Anwendung oder in einem Dokument verstanden. Dabei können die Medien zeitunabhängig (Text, Grafik, Bild) oder zeitabhängig (Audio, Video) sein. Weiterhin kann man die Anwendungen in zwei Klassen aufteilen:

1. Anwendungen zwischen zwei oder mehr Benutzern (Konferenz-Anwendungen)

2. Anwendungen zwischen einem Server und einem Client (Streaming-Anwendungen)

Multimedia-Anwendungen stellen hohe Anforderungen an die Übertragung. So benötigt etwa eine Video-Konferenz eine große Bandbreite bei gleichzeitig ge-

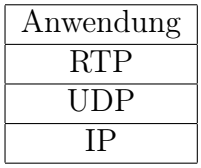


Abbildung 11.1: Protokollstack für RTP

ringer Latenz. Die Synchronisation verschiedener Datenströme (z.B. getrennte Audio- und Videokanäle) erfordert zusätzliche Maßnahmen.

11.7.1 Real-Time Transport Protocol

Ein universelles Transport Protokoll für Multimedia-Daten ist das Real-Time Transport Protocol (RTP). Das Protokoll soll die notwendigen Eigenschaften für eine Echtzeit-Kommunikation bereit stellen, dabei aber so wenig Festlegungen wie möglich treffen. Vielmehr bleibt es der Anwendung selbst überlassen, wie sie beispielsweise mit Paketverlusten umgeht. RTP setzt auf UDP auf. UDP bietet die notwendige Grundfunktionalität – Zustellung von Paketen – ohne großen Overhead. RTP ergänzt diesen Paketdienst um einige für die Echtzeit-Kommunikation notwendige Elemente. Bild 11.1 zeigt den Protokollstack.

Der Header für RTP-Pakete ist mindestens 12 Byte lang. Er enthält eine 16 bit große Sequenznummer für das Paket. Diese Nummer wird bei jedem Paket um Eins erhöht. Anhand dieser Nummern kann der Empfänger erkennen, ob er alle Pakete in der richtigen Reihenfolge erhält. Die zeitliche Beziehung wird über ein Feld mit einem Zeitstempel hergestellt. Dabei ist das exakte Format des Zeitstempels anwendungsabhängig. Die Quelle des Datenstroms wird über eine 32 bit Zahl im Feld Synchronization Source (SSRC) mitgeteilt. Über ein 7 bit großes Feld wird der Typ der Nutzdaten angeben. Damit könnte beispielsweise ein Wechsel der Kodierungsmethode angezeigt werden. Schließlich kann ein RTP-Strom Beiträge von mehreren Quellen enthalten (z.B. mehrere Mikrophonkanäle).

RTP ist optimiert auf die schnelle Übertragung. Das Protokoll enthält keinerlei Elemente zum Umgang mit verlorenen Paketen oder etwa zur Überlastkontrolle. Allerdings ist es durchaus wünschenswert, dass der Empfänger über den aktuellen Zustand der Verbindung informiert wird. Beispielsweise könnte ein Sender bei Knappheit an Bandbreite zu einer Kodierung mit niedriger Datenrate wechseln.

Zur Überwachung und Steuerung einer RTP-Sitzung dient das Protokoll RTP Control Protocol (RTCP). Eine RTP-Sitzung besteht aus einem RTP- und einem RTCP-Kanal. Für eine Sitzung wird für RTP eine gerade Portnummer und für RTCP die darauf folgende ungerade Portnummern vergeben. Über RTCP werden parallel zu RTP Pakete – in der Regel ebenfalls über UDP – ausgetauscht. Die Pakete beinhalten Sende- oder Empfangs-Berichte mit Übertragungsstatisti-

ken. Diese Berichte werden periodische verschickt. Informationen über den Sender werden mit Quellenbeschreibungspaketen übermittelt. Als Kennung dient der so genannte kanonische Name (CNAME). Das übliche Format ist *user@host*. Die Quellenbeschreibungspakete enthalten die Zuordnung zwischen SSRC und CNAME. Mit dieser Information kann der Empfänger die Daten dem Absender zuordnen. Damit können auch mehrere Quellen (z.B. Audio, Video, Grafik) von einem gemeinsamen Sender erkannt werden. Schließlich bietet RTCP die Möglichkeit, ergänzende Informationen parallel zu dem Datenstrom über RTP zu senden. Ein Beispiel hierzu sind Untertiteln für einen Videostrom. Mit verschiedenen Mechanismen wird während einer Sitzung angestrebt, den Verkehr über RTCP auf etwa 5% des RTP-Verkehrs zu beschränken.

11.7.2 Verbindungsaufbau

Vor einer Kommunikation über RTP muss eine Sitzung eingeleitet werden. Der Schwierigkeitsgrad der Aufgabe reicht dabei von der einfachen Ankündigung einer Übertragung bis zum aufwändigen Aufbau einer Videokonferenz mit Suchen der Gesprächspartner und Aushandeln der Kodierungsverfahren. Eine Arbeitsgruppe der IETF hat dafür eine Reihe von Protokollen entwickelt. Um beispielsweise eine Telefonverbindung über Internet herzustellen, werden die Protokolle Session Initiation Protocol (SIP) und Session Description Protocol (SDP) verwendet. Alternativ entwickelte ITU die Empfehlung H.323 für den Gesprächsaufbau. Die Eigenschaften des Gesprächs werden über das Call-Control Protokoll H.245 bestimmt. Geräte, die über H.323 Verbindungen aufbauen, werden als H.323-Terminals bezeichnet.

11.7.3 Sprachübertragung über IP

Eine Anwendung von großer kommerzieller Bedeutung ist die Sprachübertragung für Telefongespräche. Daher begannen verschiedene Firmen frühzeitig, Lösungen zur Sprachübertragung über IP (Voice over IP, VoIP) zu entwickeln. Die Technik verspricht einen deutlichen Kostenvorteil gegenüber dem herkömmlichen Telefonsystem. Zum einen entfällt die Installation eines eigenen Netzwerkes und zum anderen ist die Übertragung innerhalb des Internets zumindest bei Fernverbindungen wesentlich preisgünstiger. Auf der anderen Seite erwartet der Kunde einen vergleichbaren Stand bezüglich

- Einfachheit der Bedienung
- gute Sprachqualität
- hohe Zuverlässigkeit

Im folgenden werden die Grundprinzipien als Beispiel für einen Echtzeitkommunikation über das Internet beschrieben. Eine ausführliche Darstellung findet man beispielsweise in [18] und [19].

11.7.4 Sprachcodierung

Im analogen Telefonnetz wurden die Sprachsignale in dem Frequenzbereich von 300 Hz bis 3300 Hz übertragen (Telefonbandbreite). Ausgehend von dieser Qualität wurde bei der Digitalisierung eine Abtastrate von 8000 Hz gewählt. Gemäß dem Nyquist-Kriterium, demzufolge Frequenzen bis zur halben Abtastrate dargestellt werden können, sind damit Frequenzen bis maximal 4000 Hz enthalten. Bei jedem Abtastwert wird die Amplitude als Zahlenwert kodiert (Pulscodemodulation, PCM). Die Auflösung der Zahlen bestimmt die Qualität dieser Darstellung. Für die Übertragung im Telefonnetzen nutzt man aus, dass nicht über den gesamten Amplitudenbereich eine gleichmäßige Quantisierung notwendig ist. Für große Amplituden kann man eine gröbere Rasterung verwenden, ohne dass dadurch eine hörbare Verschlechterung der Sprachqualität wahrzunehmen ist. Auf diese Weise ist es möglich, mit nur 8 bit Auflösung eine gute Qualität zu erreichen. Diese Kombination – 8 kHz Abtastrate und 8 bit Auflösung entsprechend 64 kbit/s – ist der Standard in ISDN.

Die Datenrate lässt sich verringern, wenn man spezielle Eigenschaften der Signale ausnutzt. Ein Ansatz beruht auf der Annahme, dass die Amplituden aufeinander folgender Abtastwerte in der Regel nicht beliebig weit auseinander liegen. Betrachtet man anstelle der Abtastwerte deren Differenzen, so kann der dann kleinere Wertebereich mit weniger Bit quantisiert werden. Bei adaptiven Verfahren wird der Wertebereich zusätzlich an die aktuelle Lautstärke angepasst, um eine möglichst gute Übereinstimmung zwischen Wertebereich und auftretenden Amplituden zu erreichen. Ein Standard nach diesem Prinzip ist Adaptive Differentielle PCM (ADPCM) mit 32 kbit/s gemäß ITU G.726. Gegenüber dem PCM-Standard wird bei nahezu gleicher Sprachqualität nur noch die halbe Datenrate benötigt. Ein Beispiel für den Einsatz dieses Standards ist DECT (Digital European Cordless Telephony) für schnurlose Telefone .

Sprachsignale enthalten noch sehr viel mehr Strukturen. Eine Reihe von Verfahren, die diese Strukturen weitgehend ausnutzen, beruhen auf dem Basisprinzip von Code Excited Linear Predictive Coding (CELP). Diese Verfahren benutzen spezielle adaptive Filter – so genannte lineare Prädiktoren – zur Vorhersage der nächsten Abtastwerte. Das Anregungssignal wird durch systematisches Probieren von Codes aus einem Vorrat ermittelt [20]. Vertreter dieser Coderfamilie werden in GSM-Netzen verwendet. Der so genannte Full Rate Coder arbeitet mit 12,2 kbit/s. Die neue Nachfolgetechnik Half Rate benötigt bei annähernd gleicher Sprachqualität nur noch 5,6 kbit/s. Für VoIP weit verbreitet sind die Standards G.729A (CompuServe ACELP) mit 8 kbit/s und G.723.1 MultiRate Coder mit 5.3 und 6.3 kbit/s. Ein neuerer Standard speziell für das Internet ist der Internet

Low Bit Rate Codec (iLBC). Der Coder arbeitet bei 13.3 kbit/s oder 15.2 kbit/s. Er wurde speziell für Situationen, in denen Pakete verloren gehen, ausgelegt. In solchen Fällen wird mittels einer so genannten Packet Loss Concealment Einheit für eine Überbrückung der fehlenden Blöcke gesorgt.

Eine weitere Datenreduktion ist möglich, indem Sprachpausen nicht übertragen werden. Dialoge enthalten einen Anteil von bis zu 50% Pausen. Erkennt der Koder, dass ein Teilnehmer im Moment nicht spricht, so werden die Blöcke mit dem Hintergrundgeräusch nicht übertragen. Um den Eindruck einer „toten Leitung“ zu vermeiden, kann an der Gegenseite ein künstliches Rauschen eingespielt werden. Damit ist eine deutliche Reduktion der Datenkommunikation zu erreichen. Allerdings können Fehlentscheidung in der Pausedetektion oder ein zu spätes Ansprechen auf beginnende Sprachaktivität zu einem sehr unnatürlichen Spracheindruck führen.

Zu den Sprachdaten müssen bei der Berechnung des Bedarfs an Bandbreite die verschieden Paketheader hinzu addiert werden. In Tabelle 11.3 ist für einen Koder mit 6 kbit/s und einer Dauer von 30 ms für einen Sprachblock die Größe des resultierenden IP-Paketes berechnet. Bei der Übertragung über Ethernet kommen weitere 29 Byte (inklusive 3 Byte LLC Header) pro Paket hinzu. Insgesamt ergeben sich somit 92 Byte pro Paket. Mit 33,33 Paketen pro Sekunde resultiert eine Bandbreite von 24,5 kbit/s pro Sprachkanal beziehungsweise 49 kbit/s pro Telefonverbindung.

Tabelle 11.3: Größe eines IP-Paketes mit Sprachdaten

Kodierrate	6	kbit/s
Blocklänge	30	ms
Bit pro Block	180	bit
RTP-Nutzlast	23	Byte
RTP Header	12	Byte
UDP Header	8	Byte
IP Header	20	Byte
Summe	63	Byte

Die Zusammenstellung zeigt, dass die verschiedenen Header in Summe gegenüber der Nutzlast dominieren. Daher wurden Protokoll-Erweiterungen entwickelt, um die Länge der Header zu reduzieren. Ein Ansatzpunkt dazu ist die Tatsache, dass ein Teil der Headerinformationen nur zu Beginn der Verbindung benötigt wird. Während der eigentlichen Datenübertragung werden diese Informationen entweder gar nicht mehr oder nur geringfügig verändert. Indem nur noch solche Veränderungen übertragen werden, kann die Länge der Header deutlich reduziert werden. Ein Problem besteht allerdings in der Möglichkeit, dass einzelne Pakete verloren gehen können. Ohne weitere Maßnahmen würde dann unter Umständen

ein Teil der Veränderungen fehlen, so dass der Empfänger fehlerhafte Werte rekonstruiert. Speziell für dieses Einsatzgebiet wurde das Protokoll RObust Header Compression (ROHC), RFC 3095, definiert. Nach [21] lässt sich mit ROHC eine Reduktion der Header von RTP, UDP und IP von zusammen 20 Byte auf im Mittel nur noch etwa 6 Byte erzielen, ohne dass die Sprachqualität beeinträchtigt wird.

Ein zweites wichtiges Qualititätskriterium ist die Verzögerungszeit. Ab etwa 100 ms machen sich Verzögerungen deutlich störend bemerkbar. Laut ITU-T Empfehlung gelten Verzögerungen bis 200 ms noch als gut und bis 400 ms noch als gerade akzeptabel. Die Sprachkodierung führt bei den niedrigen Datenraten zu etwa 10 bis 30 ms Verzögerung. Hinzu kommt die Latenz der Verbindung. Innerhalb eines Firmennetzes ist die resultierende Gesamtverzögerung in der Regel unterhalb der kritischen Grenzen. Bei einer Übertragung im Internet können sich allerdings durch zu große Verzögerungen und Paketverlusten deutliche Qualitätseinbußen ergeben.

11.7.5 Telefon-Anwendungen

Mit den beschriebenen Mitteln kann ein Rechner die Funktion eines Telefons übernehmen. Dazu genügen ein PC, eine handelsüblichen Soundkarte, ein Mikrophon und ein Lautsprecher sowie eine entsprechende Software. Eine Anwendung ist das Telefonieren von Rechner zu Rechner. Die entsprechenden Programme ähneln Chat-Programmen. Über einen Server kann man Verbindung zu den angemeldeten Personen aufnehmen. In der Regel können sich weitere Personen an dem Gespräch beteiligen, so dass eine Konferenz entsteht. Viele Programme unterstützen die parallele Kommunikation über mehrere Medien. So sind in das Programm `NetMeeting` der Firma Microsoft neben Audio unter anderem auch Video und Whiteboard – ein gemeinsames Zeichentablett – integriert.

Neben der Kommunikation zwischen Rechnern ist auch die Verbindung zu dem öffentlichen Telefonnetz möglich. So bieten diverse Anbieter Gateways zwischen Internet und Telefonnetz an. Eine kostengünstige Verbindung besteht dann aus einer Internet-Verbindung bis zu einem Gateway in der Nähe des Ziels und einer Telefonverbindung vom Gateway bis zum Endteilnehmer. Das Gateway übernimmt die Umsetzung der Daten sowie der Signalisierungsinformationen. Bei H.323 ist ein Gatekeeper bei der Suche nach dem günstigsten Gateway behilflich.

Das Konzept lässt sich leicht zum Telefonieren zwischen zwei konventionellen Telefonen erweitern. Ein Teilnehmer ruft bei seinem lokalen Gateway an. Von dort wird eine RTP-Sitzung zu dem passenden Gateway in der Nähe des Ziels aufgebaut.

11.8 Übungen

Übung 11.1

Verwenden Sie `telnet` *um mit SMTP-Befehlen eine email zu schreiben und zu versenden. Testen Sie, welche Adressen als Absender akzeptiert werden.*

Übung 11.2 *Erweitern Sie den Socket-Client aus Übung 9.1 so, dass er von einem www-Server mittels HTTP die Seite index.html liest. Suchen Sie in dem HTML-Text nach Links in der Form* `<a href="Name des Links" ...>` *und geben Sie diese Links aus.*

Übung 11.3 *Über welchen URI kann man den selbst entwickelten Server ansprechen? Wie lässt sich der Server zu einem Besucherzähler ausbauen?*

Übung 11.4 *Geben Sie den Verweis auf eine Newsgroup in Ihren Browser ein. Wie zeigt er die entsprechende Gruppe an? Unter Windows können Sie den Verweis auch direkt auf dem Desktop anlegen. Was passiert dann bei Aktivieren?*

Kapitel 12

Java

12.1 Einleitung

In den Bibliotheken von Java finden sich eine ganze Reihe von Klassen zur Unterstützung von Anwendungen über Netzwerke und insbesondere im Internet. Die Klassen bieten ein unterschiedliches Einstiegsniveau. Einerseits besteht die Möglichkeit, mittels Sockets auf einer relativ niedrigen Ebene zu arbeiten. Anderseits stehen auch Klassen wie URL zur Verfügung, die bereits einen Großteil der Netzwerkfunktionalität beinhalten. Im folgenden werden exemplarisch einige Möglichkeiten der beiden Klassen Socket und URL vorgestellt.

12.2 Socket

12.2.1 Client-Socket

In der Klassenbibliothek von Java stehen mehrere nach Funktionalitäten getrennte Klassen für Sockets zur Verfügung. Die Klasse `Socket` realisiert einen Client-Socket. Verschiedene Konstruktoren ermöglichen die Angabe des Servers über den Namen oder die IP-Adresse. Als Beispiel sei die Form

```
Socket s = new Socket( "www.fh-friedberg.de", 80 );
```

angegeben. An den so erzeugten Socket können mit den Methoden

```
getInputStream()
getOutputStream()
```

Ein- und Ausgangsströme angehängt werden. Die Anwendung kann dann diese Ströme in der gleichen Art und Weise wie z.B. Ströme von und zu Dateien benutzen. Der Socket übernimmt die Details der Netzwerkkommunikation.

Als Beispiel folgt ein Programm, das über eine Socket-Verbindung eine Email mit einer Klausurnote verschickt. Zunächst öffnet das Programm eine Socket-Verbindung zu einem Mail-Server an dessen Port 25. Anschließend werden die

Befehle gemäß SMTP gesendet. Zur besseren Übersicht wird im Beispiel auf eine Erfolgskontrolle verzichtet. Dazu müssten über einen `InputStream` die zurück geschickten SMTP-Statuscodes gelesen und ausgewertet werden.

```
import java.net.*;
import java.io.*;

public class TestEmail {

    public static void main( String args[]) {
       String host   = "mailto.t-online.de";
       String from   = "stephan.euler@t-online.de";
       String to     = "stephan.euler@t-online.de";
       String betreff= "Note der Informatikklausur III";
       String inhalt = "Sie haben die Note 3 \n";
       try
       {
          Socket mailSocket = new Socket(host, 25 );
          PrintWriter out = new PrintWriter(
          mailSocket.getOutputStream(), true );

          out.println( "HELO " + host);
          out.println( "MAIL FROM: <" + from + ">");
          out.println( "RCPT TO: <" + to + ">");
          out.println( "DATA" );
          out.println( "SUBJECT: " + betreff );
          out.println( inhalt );
          out.println( ".");
          out.println( "QUIT" );

          out.close();
          mailSocket.close();
       }
       catch( IOException e ) { System.err.println( e ); }
   }
}
```

12.2.2 Server-Sockets

Ein Server für verbindungsorientierte Sockets wird mit der Klasse `ServerSocket` realisiert. Die Klasse enthält eine Reihe von Konstruktoren und Methoden, um einen Server-Socket zu erzeugen und anzumelden. Eine kompakte Form ist

```
ServerSocket socket = new ServerSocket(port, backlog);
```

mit der Angabe der Portnummer `port` und der Größe `backlog` der Warteschlange für anstehende Clients. Nach dem Aufruf der Methode `accept()` wartet der Socket auf einen Client. Meldet sich ein Client an, so kehrt die Methode mit einem „normalen“ Socket als Rückgabewert zurück. Mit diesen Elementen ist in der Klasse `SocketServer` ein einfacher Server realisiert. Der Server wartet in einer Endlosschleife auf Anfragen. Ein Client erhält bei der Kontaktaufnahme eine kurze Begrüßung. Anschließend wird die Verbindung wieder abgebaut.

```
import java.net.*;
import java.io.*;

public class SocketServer {
   static int backlog = 10;  // Laenge der Warteschlange
   static int port    = 1234;

   public static void main( String args[]) {

      try
      {
       ServerSocket socket =
                  new ServerSocket(port, backlog);
       for( ;; ) {
          System.out.println( "Warte auf Verbindung... " );
          Socket sockConnected = socket.accept();
          System.out.println( "Verbunden mit "
                        + sockConnected);
          PrintStream ps = new PrintStream(
                 sockConnected.getOutputStream() );
          ps.println( "Hallo" );
          sockConnected.close();
       }

   }
   catch( IOException e ) { System.err.println( e ); }
   }
}
```

12.2.3 UDP-Socket

Der beschrieben Client-Socket verwendet eine stromorientierte Verbindung (TCP). Paketorientierte Verbindungen (UDP) werden mit der Klasse `DatagramSocket` realisiert. So erzeugt die Zeile

```
DatagramSocket s = new DatagramSocket(8888);
```

einen neuen Socket für Datagramm-Kommunikation über Port 8888. Der Datenaustausch erfolgt dann mit Objekten der Klasse `DatagramPacket`. Zum Versenden werden in diese Pakete die Daten und die Zieladressen eingetragen.

Das Zusammenspiel der Klassen soll am Beispiel eines Chat-Servers dargestellt werden. Der Server hat folgende Funktionalitäten:

- Anmeldung von Clients
- Die Nachricht eines Clients wird an alle andere Clients weiter geleitet
- Abmeldung von Clients

An- und Abmeldung erfolgen über Pakete, die eine entsprechende Kennung enthalten. Der Server führt eine Liste mit allen angemeldeten Clients. Erhält er von einem der Clients eine Nachricht, so wird diese an alle anderen Clients kopiert.

Der folgende Code realisiert einen solchen Server[1]. Die angemeldeten Clients werden in einem Objekt der Klasse `Vector` gespeichert. Die Fehlerprüfung beschränkt sich auf die Ausgabe von eventuell auftretenden Ausnahmefehlern (Exceptions).

```
import java.net.*;
import java.io.*;
import java.util.*;

public class DatagrammServer {
   // Kennungen für An- und Abmeldung
   static final String  ANMELDUNG ="ANMELDUNG";
   static final String  ENDE      ="ENDE";
   static int port    = 1234;
   static int length  = 256;   // Länge eines Pakets

   public static void main( String args[]) {

      DatagramPacket paket =
           new DatagramPacket( new byte[length], length);
      Vector clients = new Vector();   // Liste der Clients

      try {
        DatagramSocket socket = new DatagramSocket(port);
        for( ;; ) {
```

[1]Einige der verwendeten Methoden sind erst ab Java Version 1.4 vorhanden. Falls notwendig lassen sich diese Methoden leicht durch etwas wenige elegante Methoden aus den älteren Versionen ersetzen.

```
            // Warten auf nächstes Paket
            socket.receive( paket );
            InetSocketAddress add =
                (InetSocketAddress)paket.getSocketAddress();

            // Text aus Paket extrahieren
            String text =
               new String(paket.getData(), 0, paket.getLength());
            System.out.println( add +">" + text);

            // Paket auswerten
            if( text.equals( ANMELDUNG ) ) {
               clients.add( add );
               System.out.println( "Anzahl Clients: "
                                   + clients.size() );
            }  else if( text.equals( ENDE ) ){
               clients.remove( add );
               System.out.println( "Anzahl Clients: "
                                   + clients.size() );
            }  else {
               // Versenden von Kopien an alle anderen Clients
               for( int i=0; i<clients.size(); i++ ) {
                 InetSocketAddress dest =
                     (InetSocketAddress) clients.get(i);
                 if( ! dest.equals( add ) ) {
                    paket.setSocketAddress( dest );
                    socket.send( paket );
                    System.out.println( "Kopie an " + dest );
                 }
               }
            }
          }
       }
       catch( IOException e ) {
          System.err.println( "Ausnahmefehler: " + e );
       }
    }
}
```

Als passendes Gegenstück kann der folgende Client benutzt werden. Bei dem Aufruf kann der Name des Servers als Argument übergeben werden. Fehlt diese Angabe, sucht der Client auf dem eigenen Rechner nach dem Server. Der Client meldet sich dann mit einem entsprechenden Paket beim Server an. In einer Schlei-

fe wartet er auf Eingaben. Jede Eingabe wird als Zeichenkette (String) gelesen, in ein Feld von Bytes umgewandelt und dann als Paket verschickt. Besteht die Eingabe aus dem Text `ENDE`, so wird dies als Abmeldenachricht an den Server geschickt. Anschließend wird die Schleife verlassen und das Programm beendet.

Anders als bei dem Server erfolgt der Nachrichtenaustausch asynchron. Der Server wartet auf eine Nachricht und reagiert darauf. Demgegenüber kann beim Client zu jedem Zeitpunkt vom Server eine neue Nachricht eintreffen. Es gibt keine zeitliche Koppelung zwischen Eingaben und Ausgaben.

In der Programmiersprache Java bzw. in der virtuellen Maschine zur Ausführung von Java-Anwendungen sind Möglichkeiten für parallele Verarbeitung integriert. Eine Anwendung kann in mehrere eigenständige Programmfragmente – so genannte Threads – aufgeteilt werden. Im vorliegenden Fall wird die Aufgabe auf zwei solche Threads aufgeteilt. Auch die Hauptanwendung mit der Schleife für Benutzereingaben läuft in einem Thread. Die einkommende Nachrichten werden in einem eigenen Verarbeitungsstrang in einem zweiten Thread behandelt. Die beiden Threads laufen parallel.

Programmtechnisch wird der zweite Thread in einer eigenen Klasse realisiert. Diese Klasse implementiert das Interface `Runnable`. Damit verpflichtet sie sich, eine Methode `run` zu realisieren. Bei der Instanzierung eines Objektes der Klasse wird dann automatisch diese Methode aufgerufen und in einem eigenen Thread ausgeführt. Die Hauptanwendung braucht daher nur ein Objekt der Klasse anzulegen um den zweiten Thread zu starten. Beim Anlegen wird im Konstruktor der Socket übergeben.

```
import java.net.*;
import java.io.*;

public class DatagrammClient {
   static final String  ANMELDUNG = "ANMELDUNG";
   static final String  ENDE      = "ENDE";
   static int port    = 1234;
   static int length  = 256;    // Länge eines Pakets

   public static void main( String args[]) {
      String servername = "localhost";
      String text = null;
      DatagramPacket packet;
      byte[] ba = ANMELDUNG.getBytes();

      // Namen des Servers von Kommandozeile übernehmen
      if( args.length > 0 ) servername = args[0];

      try {
```

```
        DatagramSocket socket = new DatagramSocket();
        InetAddress ia =  InetAddress.getByName( servername );
        packet = new DatagramPacket( ba, ba.length, ia, port);
        // sende Anmeldung
        socket.send( packet );

        // Lesen der empfangenen Pakete erfolgt in eigenem Thread
        LeseThread lt = new LeseThread( socket );

        // Eingaben von Tastatur an Server schicken
        BufferedReader br = new BufferedReader(
             new InputStreamReader( System.in )  );
        do {
           text = br.readLine();
           ba = text.getBytes();
           packet.setData( ba, 0, ba.length );
           socket.send( packet );
        } while( ! text.equals("ENDE") );

        // alles beenden
        System.exit(0);

      }
      catch( IOException e ) {
        System.err.println("Ausnahmefehler: " +  e );
      }
   }
}
```

Dazu gehört die Klasse `LeseThread` zum Empfang von Nachrichten.

```
class LeseThread implements Runnable {
   static int length  = 256;
   DatagramSocket socket;

   LeseThread(DatagramSocket socket ) {
      this.socket = socket;
      Thread t = new Thread(this,"Lesen");
      t.start();
   }

   public void run()  {
      DatagramPacket packet =
```

```
         new DatagramPacket( new byte[length], length);
      while( true ) {
        try {
           socket.receive( packet );
           InetSocketAddress add =
              (InetSocketAddress)packet.getSocketAddress();
           String text =
             new String(packet.getData(), 0, packet.getLength());
           System.out.println( add +">" + text);
           //System.out.println( ">" + text);
        }
        catch( IOException e ) {
           System.err.println("Ausnahmefehler: " +  e );
        }
      }
   }
}
```

12.2.4 Die Klasse URL

Während einerseits die Klassen zu Sockets Netzwerkkommunikation auf einer relativ niedrigen Ebene behandeln, gibt es in Java auch zahlreiche Möglichkeiten auf höherer Ebene anzusetzen. Eine solche Möglichkeit besteht über die Klasse `URL`. Ein URL-Objekt wird im einfachsten Fall durch den Konstruktor mit dem Namen des Links als Parameter erzeugt:

```
URL url = new URL("http://www.fh-friedberg.de");
```

Das URL-Objekt beinhaltet zunächst nur die Information über den Link. In einen weiteren Schritt kann darüber eine Verbindung aufgebaut werden. Eine Möglichkeit dazu bietet die Methode `openConnection()`. Sie liefert ein Objekt der Klasse `URLConnection`, über das die weitere Kommunikation läuft. Im folgenden Beispiel wird dieses Schema benutzt, um Informationen über einen Link abzufragen. Insbesondere verwendete die Anwendung die Methode `getContentType()` um den Typ des Dokumentes zu erfragen. Der Rückgabewert ist eine Zeichenkette mit der MIME-Angabe des Typs. Die Methode `getContent()` liefert ein passendes Objekt, um den Inhalt des Links zu lesen. Bei HTML-Dateien sind dies Varianten von `InputStream`. Für gif-Dateien als anderes Beispiel erhält man ein Objekt der Klasse `URLImageSource`.

```
import java.net.*;
import java.io.*;

public class ExamineURL {
```

```
   static int keyWidth = 20;

   public static void main( String args[]) {
       // Kopieren des Argumentes beim Aufruf
      String link = args[0];

      try {
         System.out.println("Trying URL " + link );
         URL url = new URL( link  );
         URLConnection conn = url.openConnection();
         int n = 1;

         // Ausgabe der Elemente des http-Headers
         String key;
         System.out.println("Header fields:");
         while((key=conn.getHeaderFieldKey(n) ) != null ) {
            key += ": ";
            while( key.length() < keyWidth ) key += " ";
            System.out.println( n + " " + key
                        + conn.getHeaderField(n) );
            ++n;
         }

         // Objekt für Zugriff auf Inhalt
         Object o =  conn.getContent();
         System.out.println( "Content:     " +  o );
         System.out.println( "ContentType: "
                             + conn.getContentType() );
         System.out.println( "Permission:  "
                             + conn.getPermission() );

      } catch( Exception ex ) {
            System.out.println("Cannot create URL "
                             + link + " Excption:" + ex);
      }
   }
}
```

Bei der Anwendung auf die Homepage der FH Friedberg erhält man:

```
java ExamineURL http://www.fh-friedberg.de
Trying URL http://www.fh-friedberg.de
Header fields:
1 Date:               Mon, 16 Dec 2002 08:56:52 GMT
```

```
2 MIME-Version:       1.0
3 Content-Type:       text/html
Content:     java.io.PushbackInputStream@3169f8
ContentType: text/html
Permission:  (java.net.SocketPermission
    www.fh-friedberg.de:80 connect,resolve)
```

12.2.5 Mailto-Links

Ein URL-Objekt kann nicht nur auf Dateien verweisen sondern auch andere Dienste ansprechen. Ein Beispiel sind Verweise mit dem Protokoll `mailto`. Damit können über ein URL-Objekt auch Emails verschickt werden. Die Klasse `URLemail` verwendet diesen Mechanismus, um eine Notenemail zu verschicken. Zu beachten ist, dass vor dem Verbinden der Java-VM der zu verwendende Mail-Server bekannt gemacht werden muss. Dies erfolgt durch Setzen der entsprechenden Systemeigenschaft mittels

```
System.setProperty("mail.host", "mailto.t-online.de");
```

In entsprechender Art und Weise können auch andere Eigenschaften wie z.B. der Name eines eventuell vorhandenen Proxy-Servers festgelegt werden.

```
import java.io.*;
import java.net.*;

public class URLemail
{
   public static void main(String args[])throws Exception {
      String from      = "stephan.euler@t-online.de";
      String to        = "stephan.euler@t-online.de";
      String subject   = "Klausurergebnis IN3";
      String nachricht = "Sie haben die Note 2.";

      System.setProperty("mail.host",
                    "mailto.t-online.de");
      URL u = new URL("mailto:" + to);
      URLConnection c = u.openConnection();
      c.setDoOutput(true);
      System.out.println("Connecting...");
      c.connect();

      PrintWriter out = new PrintWriter(
             new OutputStreamWriter(c.getOutputStream()));
      out.println("Subject: " + subject);
```

```
        out.println(nachricht);
        out.close();
        System.out.println("Nachricht versendet.");
    }
}
```

12.3 Übungen

Übung 12.1 *Ergänzen Sie die Socket-Anwendung zum Email-Versand um eine Kontrolle der zurück gegebenen Statusmeldungen.*

Übung 12.2 *Erweitern Sie das Beispiel eines Chat-Servers:*

1. *Ein Client wird automatisch mit der ersten Nachricht angemeldet.*
2. *Wenn ein Client nach einer vorgegebenen Anzahl von Nachrichten selbst keine geschickt hat, wird er aus der Liste der aktiven Clients gelöscht. Ergänzen Sie das Protokoll, so dass Clients ansonsten leere Nachrichten als „Lebenszeichen“ schicken können.*
3. *Was passiert, wenn ein eingegebener Text länger als die Paketlänge 256 ist? Erweitern Sie die Programm, um auch solche langen Texte behandeln zu können.*
4. *Wenn sich der Client hinter einem Firewall befindet, kann der Sendeport von Paket zu Paket variieren. Verändern Sie das Protokoll dahingehend, dass ein Client bei der Anmeldung eine eindeutig Kennung erhält. Mit dieser Meldung markiert er dann alle Nachrichten, so dass sie vom Client wieder eindeutig zugeordnet werden können.*

Übung 12.3 *Testen Sie die Anwendung* `ExamineURL` *mit verschiedenen Dateitypen als Ziel. (Hinweis: über das Protokoll* `file:` *können auch lokale Dateien angesprochen werden.)*

Kapitel 13

XML

13.1 Einleitung

Mit der starken Zunahme von Netzwerk-Anwendungen besteht ein großer Bedarf an einheitlichen Formaten zum Datenaustausch. Traditionell wurden häufig proprietäre Standards eingesetzt. Dies funktioniert problemlos, solange die stets die Anwendungs-Software des Herstellers verwendet wird. Die Übergabe der Daten an Anwendungen anderer Hersteller war in vielen Fällen schwierig. Die dadurch gegebene Schutzfunktion für die eigene Software mit einer langfristigen Kundenbindung war durchaus nicht immer ungewollt.

Die Prinzipien des Internets bedingen demgegenüber den Einsatz offener, wohl dokumentierter Standards. Die Sprache HTML für die einzelnen Webseiten ist ein Beispiel dafür. Dank des offenen Standards existiert eine Vielzahl von Editoren und Browsern. Allerdings kennen Entwickler die Probleme, die sich aus den zahllosen proprietären Erweiterungen und der damit in der Realität eingeschränkten Kompatibilität ergeben.

In HTML wird im wesentlichen die Formatierung beschrieben. Ein HTML-Dokument enthält nur wenige Informationen, die die Inhalte klassifizieren. So besagt der Tag `<h1>`, dass der Inhalt eine Hauptüberschrift ist - eine wichtige Hilfe bei der Analyse eines Dokumentes. Aber es ist keineswegs zwingend vorgeschrieben, tatsächlich alle Hauptüberschriften in dieses Tag einzubetten. Die inhaltliche Analyse eines HTML-Dokumentes ist zwar prinzipiell möglich aber aufwändig und fehleranfällig. So erfordert es einigen Programmieraufwand, um aus Seiten typischer Anbieter wie ebay oder google die gesuchte Informationen zu extrahieren.

Sehr viel einfacher ist die Situation, wenn die Informationen in einer strukturierten Form, zusätzlich getrennt nach Inhalt und Format angeboten werden. Vor diesem Hintergrund entstand die *EXtensible Markup Language* oder kurz XML als Standard des World Wide Web Consortium (W3C). Für das Arbeiten XML entstand eine Vielzahl von weiteren Standards und Werkzeugen, unter anderem

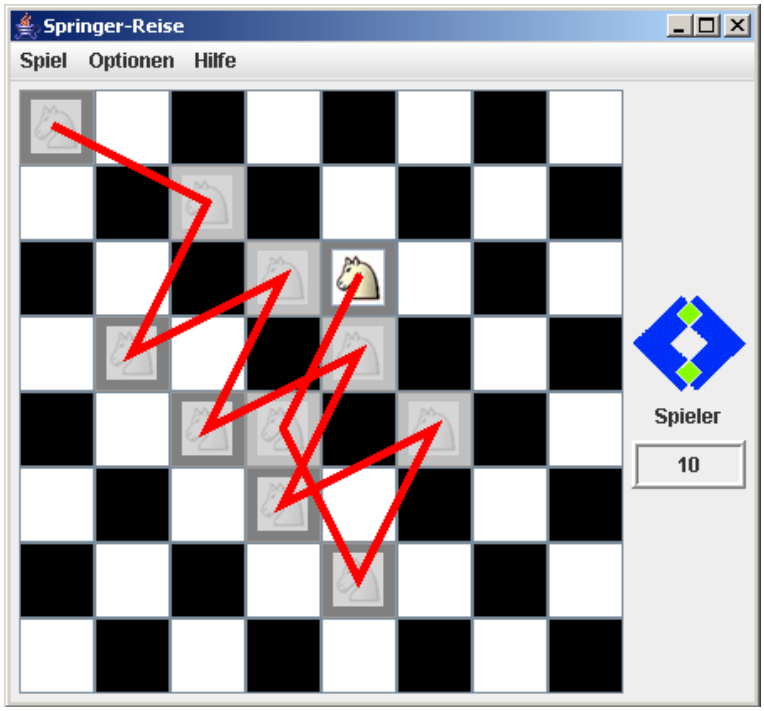


Abbildung 13.1: Beispielanwendung Springer-Reise

für:

- Analyse und Generierung von XML-Dokumenten
- Transformation in andere Darstellungen
- Auswahl von Teilen aus einer XML-Struktur

Im folgenden werden die Grundprinzipien beschrieben. Als Beispiel-Anwendung werden wir ein einfaches Spiel *Springer-Reise* betrachten. Ziel dieses Spiels ist es, mit einem Springer möglichst viele Felder eines Schachbrettes zu besuchen, ohne ein Feld mehrfach zu betreten. Bild 13.1 zeigt die Anwendung. Die Anwendung soll so erweitert werden, dass die Informationen über den aktuellen Stand gespeichert beziehungsweise später wieder geladen werden können. Als Format soll XML eingesetzt werden.

13.2 Grundlagen

Beginnen wir mit dem Beispiel einer Sicherungsdatei. Als Information soll

- die Größe des Spielfeldes
- die Information zu einem Spieler:

- Name (optional)
- Position
- Punktestand

gespeichert werden. In Bild 13.2 ist der entsprechende XML-Code dargestellt. Die einzelnen Bestandteile werden jeweils durch spitze Klammern <> begrenzt. Das Dokument beginnt mit einer optionalen XML-Deklaration in der Form

```
<?xml Attribut-Wert-Paare ?>
```

wobei die Attribute als Name, Gleichheitszeichen und dem Wert in Anführungszeichen

```
Attribut="Wert"
```

angeben sind. In dem Beispiel sind die XML-Version und die verwendete Zeichenkodierung spezifiziert. Nach einer Kommentarzeile folgt der eigentliche Inhalt. Die Daten werden durch ineinander geschachtelte Elemente beschrieben. Ein Element wird durch ein Paar aus öffnendem Start-Tag (`<Tag-Name>`), Inhalt und schließendem End-Tag (`</Tag-Name>`) definiert. Text wird direkt – d. h. ohne Anführungszeichen – eingetragen. Längere Textabschnitte können in CDATA-Abschnitte (*Character Data*)

<[CDATA[! *Inhalt - kann auch über mehrere Zeilen gehen und XML-Zeichen wie <> enthalten*]]>

eingetragen werden. Dies hat den Vorteil, dass eventuelle Sonderzeichen wie >-Zeichen nicht interpretiert werden. Dadurch ist es leicht, längere Programmabschnitte oder selbst anderen XML-Text einzubinden.

In einfachen Fällen ohne Inhalt kann statt des Tag-Paares ein so genannter Empty-Element-Tag (`<Tag-Name />`) verwendet werden. Elemente können im Start-Tag Attribute enthalten. In dem Beispiel wird diese Möglichkeit für die Angabe der Position verwendet. Nach dem gleichen Muster hätte man auch die Breite und Höhe als Attribute angeben können statt als Inhalt.

Jedes XML-Dokument darf genau ein Hauptelement enthalten. Man kann sich dieses auch als Wurzel eines Baumes vorstellen. Die weiteren Elemente sind dann die Knoten und Blätter des Baumes. Eine entsprechende Darstellung zeigt Bild 13.3

13.3 Transformation

In XML-Dokumenten werden die Inhalte angegeben. Häufig möchte man dazu eine passende Formatierung angeben. Dazu muss aus dem XML-Dokument eine Präsentation in beispielsweise HTML oder PDF (*Portable Document Format*) erzeugt werden. Man spricht allgemein von einer Transformation. Wir betrachten zwei Techniken:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Springer-Reise Tue Sep 12 11:22:36 CEST 2006 -->

<Springerspiel>
    <Breite>8</Breite>
    <Hoehe>8</Hoehe>
    <Spieler>
        <Name>Aljechin</Name>
        <Position Spalte="3" Zeile="4"/>
        <Punkte>20</Punkte>
    </Spieler>
</Springerspiel>
```

Abbildung 13.2: Beispiel einer Sicherungsdatei

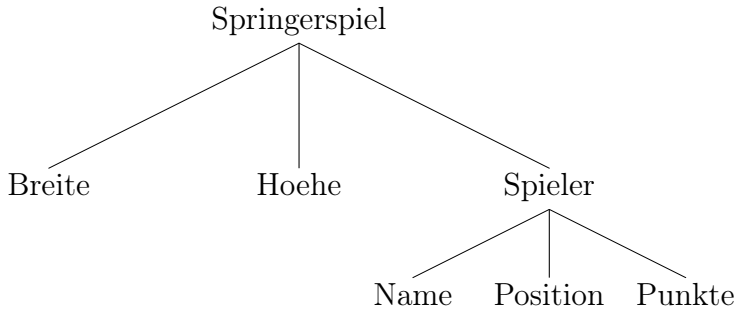


Abbildung 13.3: Baumdarstellung der Sicherungsinformation

1. Formatierung mittels Cascading Style Sheets (CSS)

2. Allgemeine Transformation mit Extensible Stylesheet Language Transformation (XSLT)

Obwohl häufig als Alternativen dargestellt, haben beide Ansätze unterschiedliche Zielsetzungen. Mit CSS kann man aus einem XML-Dokument eine HTML-Präsentation erzeugen. Demgegenüber ist XSLT wesentlich umfangreicher. Es unterstützt unterschiedliche Zielformate und ermöglicht eine Verarbeitung der Daten wie etwa Sortieren oder Auswählen von Teilmengen. Geht es nur um die Darstellung eines XML-Dokumentes, ist in der Regel CSS ausreichend. Dies ist mit XSLT ebenso möglich. Aber der damit verbundene Aufwand lohnt sich erst, wenn komplexere Verarbeitungen erforderlich sind.

13.3.1 Cascading Style Sheets

Die Cascading Style Sheets (CSS) beziehungsweise Stylesheets generell sind eine bereits in Kombination mit HTML bekannte Technik, um Daten und Formatierung auseinander zu halten. Das Grundprinzip von Stylesheets ist, in einem eigenen Bereich festzulegen, wie bestimmte Elemente formatiert werden sollen. Die gleiche Technik wird in Systemen zur Textverarbeitung verwendet. Den Stylesheets entsprechen bei Microsoft Word die Formatvorlagen und bei LaTeX die Dokumentenklassen.

Die Vorteile der Formatinformationen an zentraler Stelle liegen auf der Hand. Änderungen an der Formatierung müssen nur noch einmal vorgenommen werden und wirken sich automatisch auf alle Dokumente aus. Einmal gewählte Formatierungen können leicht auf andere Dokumente angewandt werden.

Für unser Beispiel sollen die Formate in einer eigenen Datei `springer.css` abgelegt werden. Dann wird das XML-Dokument um die Verarbeitungsanweisung

```
<?xml-stylesheet type="text/css" href="springer.css"?>
```

ergänzt. In der Datei wird die Formatierung über Regeln in der Form

```
Selektor { Eigenschaft:Wert; }
```

festgelegt. Dabei bezeichnet `Selektor` das Element, dessen Eigenschaften festgelegt werden. Als Beispiel wird durch die Regel

```
Breite {
  color:blue;
  font-size:x-large;
}
```

```
Springerspiel:before{
    content:"Ein Springerspiel";
    display:block;
    text-decoration:underline;
    font-size: xx-large;
}

Breite:before{
  content:"Breite:"  }

Hoehe:before{
  content:"Höhe:"  }

Breite {
  color:blue;
  font-size:x-large }

Hoehe {
    color:blue;
    font-size: x-large; }

Name {
  text-transform:uppercase;
  display:block;
  font-size:12 }
```

Abbildung 13.4: CSS Stylesheet

für das Element `Breite` Farbe und Schriftgröße definiert. Die Regeln in Bild 13.4 zeigen verschiedene Möglichkeiten von CSS am Beispiel der Springer-Reise. Ein Pseudo-Element wie `Breite:before` wird vor dem entsprechenden Element eingefügt. Öffnet man nun die XML-Datei mit einem Browser, so wird dieser die CSS-Formatierung auf die einzelnen Elemente anwenden und das Ergebnis entsprechend anzeigen. In Bild 13.5 zeigt das Result mit dem Browser Firefox.

13.3.2 XSLT

Nach der Definition auf der W3C Seite umfasst der Standard XSL – die Abkürzung steht für *Extensible Stylesheet Language* – drei Teile:

- XSL Transformations (XSLT)
- XML Path Language (XPath)

Abbildung 13.5: Darstellung der Sicherungsdatei im Browser Springer-Reise mit CSS

- XSL Formatting Objects (XSL-FO)

XSL selbst beschreibt Formatierungen, über XPath werden Teile aus einem Dokumentenbaum spezifiziert und mit XSLT schließlich können Transformationen durchgeführt werden. Wir werden uns im folgenden darauf beschränken, an unserem Beispiel die Verwendung von XSLT zur Darstellung unserer XML-Dateien zu verwenden. Die Transformation lassen sich von dem im Browser eingebauten XSLT-Prozessor durchführen. Als Alternative kann man Prozessoren wie Xalan von der Apache Software Foundation verwenden.

Ein XSL-Dokument ist wiederum in XML-Format geschrieben. Vergleichbar mit den Regeln in CSS werden über so genannte *Templates* Anweisungen für die Elemente definiert. Als Beispiel legt der folgende Abschnitt fest, dass aus dem Element Spieler der Name zusammen mit dem Text `Spieler:` ausgegeben wird:

```
<xsl:template match="Spieler">
  Spieler: <em><xsl:value-of select="Name" /></em>
</xsl:template>
```

Der Name des Spielers wird durch den HTML-Tag `em` kursiv gesetzt. Die vollständige XSL-Datei ist in Bild 13.6 wiedergegeben. Hauptelement ist ein Tag vom Typ `xsl:stylesheet`. Der Präfix `xsl` zeigt an, dass der Tag zu dem entsprechenden Namensraum (*name space*) gehört. Namensräume sind ein allgemeines Konzept, um eindeutige Namen zu gewährleisten. In einem XML-Dokument können Elemente aus mehreren Namensräumen gemischt werden. Die Namensräume werden durch Attribute `xmlns:namensraum` und einem eindeutigen Wert definiert. Der Wert hat die Form einer URL. Allerdings muss eine solche Datei gar nicht existieren - nur der Name wird verwendet. In dem Beispiel wird ein Namensraum `xsl` definiert.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" />

<xsl:template match="Spieler">
    <h2>Spieler</h2>
    <p style="font-size:larger">
    Name:  <xsl:value-of select="Name" /><br/>
    Punkte: <xsl:value-of select="Punkte" /><br/>
    Position:
    Spalte <xsl:value-of select="Position/@Spalte" />,
    Zeile <xsl:value-of select="Position/@Zeile" />
  </p>
</xsl:template>

<xsl:template match="Springerspiel">
  <html>
    <head><title>Springer-Reise</title></head>
    <body>
      <h1>Ein Spiel der Springer-Reise</h1>
      <p style="font-size:larger">
        Spielfeld:
        <xsl:value-of select="Breite" />  x
        <xsl:value-of select="Hoehe" /> Felder
      </p>
      <xsl:apply-templates select="Spieler"/>
   </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Abbildung 13.6: XSLT Stylesheet

Abbildung 13.7: Darstellung der Sicherungsdatei im Browser Springer-Reise mit XSLT

Die Ausgabe wird durch zwei Templates festgelegt. Das Template für das Hauptelement `Springerspiel` enthält die HTML-Befehle für die Seitenstruktur, eine Überschrift und die Größenangaben zum Spielfeld. Über `value-of`-Tags wird der Inhalt einzelner Elemente eingebaut. Schließlich wird mittels `apply-templates` das - in unserem Fall einzige - Spieler-Element mit einem eigenen Template aufgerufen. Die Attribute zur Position des Spielers können in der Form

Elementname `@` Attributname

angesprochen werden. Dies ist ein einfacher Fall von Adressierung mittels XPath (XML *Path Language*). XPath [22] ist eine Anfragesprache, um Teile eines XML-Dokumentes zu auszuwählen. Die resultierende Darstellung im Browser zeigt Bild 13.7

Wie bereits gesagt beinhaltet XSLT sehr viel als die Formatangabe. Aus dem Dokumentbaum können Elemente gezielt ausgewählt oder neu sortiert werden. Als kleinen Einstieg in diese Möglichkeiten wollen wir eine Abfrage realisieren. Nach unserer Definition ist das Element `Name` im Springerspiel optional. Wenn der Spieler keinen eigenen Namen vergeben hat, so wird dieses Element weggelassen. Im Browser wird dann nur der Text `Name:` aber ohne weiteren Inhalt dargestellt.

```
Name:
<xsl:choose>
  <!-- Test, ob Name definiert,
       ansonsten werden die Striche ausgegeben -->
  <xsl:when test="Name">
    <xsl:value-of select="Name" />
    <!-- Jetzt noch ein Bild dazu -->
    <img><xsl:attribute name="src">
    <xsl:value-of select="Name" /> <xsl:text>.jpg</xsl:text>
  </xsl:when>
  <xsl:otherwise>
    -------
  </xsl:otherwise>
</xsl:choose>
```

Abbildung 13.8: Abfrage XSLT

Stattdessen sollen jetzt in diesem Fall zur Verdeutlichung einige Striche `----` erscheinen.

In XSLT [23] sind mehrere Elemente für eine bedingte Ausführung vorgesehen. Die Auswahl aus mehreren Alternativen ermöglicht das `choose`-Element. Ähnlich wie die `switch`-Anweisung in C oder Java können mehrere Bedingungen (`xsl:when`) angegeben werden. Das erste Element mit einer zutreffenden Bedingung (und nur dieses) wird ausgeführt. Falls kein Element passt, wird das optionale Element `xsl:otherwise` verwendet. Mit dieser Konstruktion lässt sich unser Abfrage wie in Bild 13.8 gezeigt realisieren. Als weitere Ergänzung wurde noch ein Bild des Spielers eingefügt. Der Dateiname wird nach dem Muster `name.jpg` gebildet.

13.4 Grammatik

Bisher hatten wir keinerlei Einschränkungen bezüglich der Elemente getroffen. Für die Anwendung als Sicherungsdatei, bei der die XML-Dateien nur von der eigenen Software geschrieben und gelesen werden, ist dies auch nicht notwendig. Anders ist die Situation, wenn das Format als offene Schnittstelle genutzt werden soll. So können Anwender direkt den XML-Text bearbeiten oder andere Anwendungen können - beispielsweise als Aufgabensammlung - entsprechende Dateien erzeugen.

Bisher können wir nur verlangen, dass die Dateien korrektes XML enthalten. So darf es nur ein Wurzelelement geben und alle Tags müssen immer abgeschlossen sein. Ein Dokument, das in diesem Sinne korrekt ist, gilt als wohlgeformt. Eine

weitergehende Prüfung, ob auch alle Kriterien einer Darstellung des Springerspiels erfüllt sind, verlangt demgegenüber die Gültigkeit. Diese Prüfung wird als Validierung bezeichnet. Dazu ist erforderlich, den Aufbau des XML-Dokumentes vollständig zu spezifizieren. Wie das geht, ist Inhalt der nächsten Abschnitte.

13.4.1 Dokumenttypdefinition

Eine Möglichkeit zur Definition von Regeln für XML-Dateien ist die Dokumenttypdefinition (*Document Type Definition* DTD). Wie der Name besagt, definiert man dabei einen Typ von Dokument. Über Regeln wird festgelegt, welche Elemente es gibt und welchen Inhalt sie jeweils haben. Die Regeln können entweder in einem eigenen Abschnitt innerhalb der XML-Datei oder in einer eigenen Datei stehen. Wir verwenden die flexiblere Lösung einer eigenen Datei `springer.dtd`. Diese wird in der XML-Datei durch die Anweisung

```
<!DOCTYPE Springerspiel SYSTEM "springer.dtd">
```

bekannt gemacht. In der DTD werden die einzelnen Elemente beschrieben. Mit der Regel

```
<!ELEMENT Name (#PCDATA)>
```

wird ein Element `Name` definiert. Inhalt kann beliebiger Text (*Parsed Character DATA*) sein. Enthält ein Element eine Folge von anderen Elementen, so werden diese aufgelistet:

```
<!ELEMENT Spieler ( Name?, Position, Punkte) >
```

Das Fragezeichen hinter `Name` besagt, dass `Name` nicht oder einmal vorkommen darf. Mit solchen nachgestellten Symbolen kann die Häufigkeit festgelegt werden. Ein Pluszeichen bedeutet mindestens einmalünd ein Stern beliebig oft oder gar nicht. Die anderen Elemente ohne Häufigkeitszeichen müssen genau einmal vorkommen. Das Element Position hat keinen Inhalt aber dafür zwei Attribute. Dies wird wie folgt spezifiziert:

```
<!ELEMENT Position EMPTY>
<!ATTLIST Position
   Spalte CDATA #REQUIRED
   Zeile  CDATA #REQUIRED
>
```

Beide Attribute enthalten Text (Character DATA) und sind durch `#REQUIRED` zwingend erforderlich. Optionale Attribute werden durch `#IMPLIED` markiert oder mit einen Standardwert

```
   Zeile  CDATA "0"
```

```
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Punkte (#PCDATA)>
<!ELEMENT Position EMPTY>
<!ATTLIST Position
   Spalte CDATA #REQUIRED
   Zeile  CDATA #REQUIRED
>
<!ELEMENT Breite (#PCDATA)>
<!ELEMENT Hoehe (#PCDATA)>
<!ELEMENT Spieler ( Name?, Position, Punkte) >
<!ELEMENT Springerspiel ( Breite, Hoehe, Spieler) >
```

Abbildung 13.9: DTD

angegeben. Schließlich kann man die möglichen Werte eines Attributes aufzählen. Als Beispiel definiert

```
Farbe  ( Weiss | Schwarz ) "Weiss"
```

ein Attribut, das nur einen der beiden angegebenen Wert annehmen darf. Fehlt die Angabe, so wird der Standardwert `Weiss` verwendet. Der senkrechte Strich trennt bei DTDs generell Alternativen. Den gesamten Inhalt der DTD-Datei zeigt Bild 13.9

Mit einer DTD lässt sich die Struktur eines XML-Dokumentes festlegen. Allerdings sind die Möglichkeiten beschränkt. Insbesondere ist eine genauere Einschränkung der Inhalte nicht möglich. So ist das folgende durchaus gültig gemäß der DTD:

```
<Breite>8</Breite><Hoehe>-8</Hoehe>
<Spieler>
<Position Spalte="35" Zeile="Zehn"/>
<Punkte>20</Punkte>
</Spieler>
```

Weder der Datentyp noch der Wertebereich lassen sich auf einfache Art und Weise einschränken. Auch die Häufigkeit kann nur grob angegeben werden. Die realistische Forderung ein oder zwei Spieler ist durch die vorhandenen Häufigkeitszeichen nicht abgedeckt.

13.4.2 XML-Schema Definition

In einem XML-Schema werden ähnliche wie in einem DTD die Struktur von XML-Dokumenten beschrieben. Ein XML-Schema ist selbst wieder eine XML-Datei. Dies ist ein praktischer Vorteil gegenüber DTD, da keine weitere Sprache

gelernt und behandelt werden muss. Allerdings ist eine längere Einarbeitungszeit notwendig, um den großen Leistungsumfang von XML-Schema ausnutzen zu können.

Das XML-Schema soll in eine Datei `springer.xsd` stehen. Dann wird in der Sicherungsdatei der Verweis auf die Datei als Attribut

```
<Springerspiel
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="springer.xsd">
```

eingefügt. Den Inhalt der Datei zeigt Bild 13.10. Dort wird zunächst das Element `Springerspiel` von dem neuen Typ `SpringerspielType` definiert. Dieses wiederum enthält zunächst die beiden Elemente `Breite` und `Hoehe`, deren Inhalt über den vordefinierten XSD-Typ `positiveInteger` auf ganze Zahlen größer oder gleich Null beschränkt ist. Das weitere Element `Spieler` hat wieder einen selbst definierten Typ.

Die Möglichkeit, auf den Namen zu verzichten, wird durch eine Mindestanzahl von Null realisiert. Allgemein gilt für jedes Element, dass es genau einmal auftreten muss. Diese Vorgabe lässt sich über die Attribute `minOccurs` und `maxOccurs` verändern. Eine Besonderheit ist noch das leere Element `Position`. Hier ist die zugehörige Typbeschreibung leer, lediglich die beiden Attribute sind angegeben. Auf der Adresse `www.w3.org/2001/03/webdata/xsv` besteht die Möglichkeit, Dokumente validieren zu lassen. Mit dem angegebenen Schema wird die Sicherungsdatei mit der Meldung

```
No schema-validity problems were found in the target
```

als gültig akzeptiert. Demgegenüber führen die oben angegebenen unsinnigen Einträge zu den Hinweisen

```
2 schema-validity problems were found in the target

Invalid per cvc-complex-type.1.2.2:
element content failed type check: -8<1
Invalid per cvc-attribute.1.2:
attribute type check failed for {None}:
Zeile: Zehn is not a valid decimal literal
```

Durch die vordefinierten Typen lassen sich die möglichen Eingaben bereits gut eingrenzen. Darüber hinaus ist es möglich, neue Typen mit verfeinerten Bedingungen einzuführen. Im Listing 13.11 wird als Beispiel ein neuer Typ `brettDimension` definiert, dessen Wertbereich auf einen für die Breite und Höhe des Brettes sinnvollen Bereich einschränkt (Beispiel nach [24]).

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="Springerspiel" type="SpringerspielType"/>

<xsd:complexType name="SpringerspielType">
  <xsd:sequence>
      <xsd:element name="Breite"   type="xsd:positiveInteger"/>
      <xsd:element name="Hoehe"    type="xsd:positiveInteger"/>
      <xsd:element name="Spieler"  type="SpielerType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="SpielerType">
  <xsd:sequence>
    <xsd:element name="Name"     type="xsd:string" minOccurs="0"/>
    <xsd:element name="Position" type="PositionType"/>
    <xsd:element name="Punkte"   type="xsd:positiveInteger"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PositionType">
        <xsd:attribute name="Spalte" type="xsd:positiveInteger" />
        <xsd:attribute name="Zeile"  type="xsd:positiveInteger" />
</xsd:complexType>

</xsd:schema>
```

Abbildung 13.10: XML-Schema Definition

```
<xsd:simpleType name="brettDimension">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="4"/>
    <xsd:maxInclusive value="100"/>
  </xsd:restriction>
</xsd:simpleType>
```

Abbildung 13.11: Typ mit Wertebereich $[4, 100]$

13.5 Programmier-Schnittstellen

Nachdem wird die am Beispiel der Sicherungsdatei des Springerspiels den Aufbau der XML-Datei sowie die Möglichkeiten zur Darstellung und Validierung kennen gelernt haben, wollen wir uns der Implementierung zuwenden. Dazu sind zwei Aufgaben zu erledigen:

1. Generierung einer Sicherungsdatei mit dem aktuellen Spielstand
2. Analyse einer Sicherungsdatei und Übernahme der gespeicherten Informationen

Wir werden dabei zwei Ansätze betrachten:

1. Document Object Model (DOM)
2. Simple API for XML (SAX)

Es handelt sich dabei jeweils um eine allgemeine Programmier-Schnittstelle (*Application Programming Interface*, API). Die Definitionen sind unabhängig von der konkreten Programmiersprache. Wir behandeln die Realisierung in Java.

13.5.1 DOM

Generierung

Mit DOM wird aus dem XML-Dokument eine interne Repräsentation als Baum aufgebaut. Damit ist eine große Flexibilität gewährleistet. Man kann einzelne Knoten ansprechen, Inhalte verändern oder sogar die Struktur verändern. Der interne Baum spielt die Rolle eines Vermittlers zwischen dem XML-Dokument und den Java-Klassen. Für die Umwandlung zwischen Baum und Dokument stehen Bibliotheken bereit. Häufig ist für die weitere Verarbeitung der vollständige Baum zu unhandlich und man bevorzugt eine flachere Darstellung mit eigenen Klassen. Wir wollen den Aufbau des Baumes anhand der Generierung verfolgen. Zunächst wird ein noch leeres Dokument erzeugt:

```
DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();

DocumentBuilder builder =  factory.newDocumentBuilder();
Document document = builder.newDocument();
```

Zur besseren Übersicht ist hier und im folgenden die Ausnahmebehandlung mit try-catch weggelassen. An die Wurzel werden drei Knoten angehängt:

1. Ein Kommentar mit der aktuellen Uhrzeit

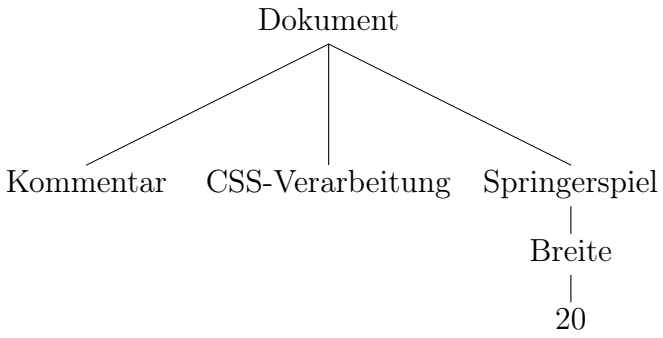


Abbildung 13.12: DOM-Baum mit Element Breite

2. Eine Verarbeitungsanweisung mit der CSS-Datei
3. Der Hauptknoten `Springerspiel`

Im Java-Programm werden die drei verschiedenen Knoten mit passenden Methoden `create...` erzeugt und mit `appendChild` eingehängt:

```
document.appendChild
( document.createComment(" Springer-Reise " +
        Calendar.getInstance().getTime().toString() +" ")
  );
String css = "type=\"text/css\" href=\"springer.css\"";
document.appendChild(
     document.createProcessingInstruction("xml-stylesheet", css)
     );

Element root =
  (Element) document.createElement("Springerspiel");
document.appendChild(root);
```

DOM unterscheidet zwischen verschiedenen Typen von Knoten wie beispielsweise Dokumentknoten für das gesamte Dokument und Elementknoten für einzelne XML-Elemente. In Java ist dies über eine Hierarchie von Interfaces, abgeleitet von dem Interface `Node` realisiert. Alle Instanzen von Knoten implementieren damit das Interface `Node`. An das Wurzelelement `Springerspiel` werden nach dem gleichen Muster die weiteren Elemente angefügt. Als Beispiel zeigt der folgende Codeabschnitt, wie das Element `Breite` eingefügt wird.

```
// erzeuge Element Breite, Wert steht in Variable spalten
Element breite = (Element) document.createElement("Breite");
breite.appendChild( document.createTextNode( ""+ spalten  ) );
root.appendChild( breite );
```

Der eigentliche Inhalt - der Wert der Variablen `spalten` - wird als `TextNode` an das Element `Breite` angehängt. Bild 13.12 zeigt den bisher aufgebauten Baum. Entsprechend ist das Vorgehen für alle anderen Elemente. Das Element `Spieler` wird als Teilbaum mit Knoten für die zugehörigen Elemente dargestellt. Attribute werden nicht als Kinderknoten, sondern als Eigenschaften eingetragen.

```
Element spielerE = (Element) document.createElement("Spieler");
...
Element position = (Element) document.createElement("Position");
position.setAttribute("Spalte", ""+ spieler[0].getX()  );
position.setAttribute("Zeile",  ""+ spieler[0].getY()  );
spielerE.appendChild( position );
...
root.appendChild( spielerE );
```

Nachdem der Baum intern vollständig aufgebaut ist, wird er mittels einer Instanz der Klasse `Transformer` in eine XML-Datei geschrieben:

```
File f = new File( "save.xml" );

TransformerFactory tFactory =
    TransformerFactory.newInstance();

Transformer transformer = tFactory.newTransformer();
transformer.setOutputProperty(OutputKeys.METHOD, "xml");
transformer.setOutputProperty(OutputKeys.INDENT, "yes");
transformer.setOutputProperty(OutputKeys.ENCODING, "ISO-8859-1");

DOMSource source    = new DOMSource(document);
StreamResult result = new StreamResult( new FileOutputStream( f ) );
transformer.transform(source, result);
```

Analyse

Im Prinzip verläuft die Analyse des Dokuments ähnlich zur Generierung. Ausgehend vom Wurzelknoten werden die relevanten Knoten aufgesucht und ihre Inhalte entnommen. Der so genannte Parser analysiert den Inhalt der XML-Datei und baut daraus den Baum auf. Entsprechende Java-Befehle sind:

```
DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.parse( new File( "save.xml" ) );
```

Wie der Name besagt wird in der Methode `parse` die Analyse durchgeführt. Eventuelle Fehler werden als Exceptions gemeldet. Mit `getDocumentElement()` erhält man das Hauptelement. Eine Liste der Kinderknoten eines Knotens liefert dessen Methode `getChildNodes()`. Im Java-Programm wird die Liste mit allen Kinderknoten des Hauptelementes durchlaufen. Jeder Knoten des Typs `ELEMENT_NODE` wird in eine Element-Instanz gewandelt und der Tag-Name wird in die Variable `tag` kopiert. Anhand des Names wird das weitere Vorgehen entschieden. Bei `Breite` und `Hoehe` wird direkt der Inhalt verwertet während bei `Spieler` dessen Kinderknoten geholt und weiter analysiert werden.

```
Element root =  document.getDocumentElement();

NodeList nlroot = root.getChildNodes( );
for( int k=0; k<nlroot.getLength(); k++ ) {
    Node nvc = nlroot.item(k);
    if( nvc.getNodeType() == Node.ELEMENT_NODE ) {
    Element e = (Element) nvc;
    String tag =  e.getTagName();
    if( tag.equals( "Breite" ) ) {
        spalten = Integer.parseInt( e.getTextContent() );
    } else  if( tag.equals( "Hoehe" ) ) {
...
```

Beim Element `Position` steht die Information nicht im Inhalt sondern in den Attributen. Diese werden über die Methode

```
element.getAttribute( AttributName )
```

abgefragt. Dementsprechend werden die Koordinaten mit

```
spieler[0].setX
    ( Integer.parseInt
      ( e2.getAttribute( "Spalte" ) ) );
spieler[0].setY
    ( Integer.parseInt
      ( e2.getAttribute( "Zeile" ) ) );
```

gelesen.

13.5.2 SAX

Häufig ist der Aufbau des vollständigen Baumes überflüssig. In solchen Fällen kann man auf das einfachere *Simple API for XML* (SAX) zurückgreifen [25]. SAX beruht auf einer sequentiellen Verarbeitung. Die XML-Daten werden als

Strom betrachtet. Markante Punkte innerhalb des Stromes wie Anfang oder Ende eines Elementes werden als Ereignisse gemeldet und es dafür bereitgestellte Methoden aufgerufen. Diese Methoden werden als *callback* oder *Event handler* bezeichnet. Der Anwender kann die vordefinierten Methoden überlagern und damit das gewünschte Verhalten realisieren.

Als praktische Anwendung dient wieder die Analyse der Sicherungsdatei des Springerspiels. Allerdings wird das Modul nicht in das Spiel integriert. Daher werden die relevanten Information nur ausgegeben. Ausgangspunkt unserer Implementierung ist die Klasse `DefaultHandler` in der Java-Bibliothek. Daraus leiten wir unsere eigene Klasse `SAXReader` ab:

```
public class SAXReader extends DefaultHandler
```

Damit erbt `SAXReader` alle in der Oberklasse definierten Methoden. Wir können nun darangehen, einige Methoden zu überlagern. Zunächst wird in der Hilfsmethode `characters`, die bei Textabschnitten innerhalb eines Elementes aufgerufen wird, der gefundene Text in eine Variable `textContent` kopiert.

```
public void characters(char buf[], int offset, int len)
throws SAXException {
   // String textContent is defined in SAXReader
   textContent = new String(buf, offset, len);
}
```

Die Verwendung dieses Textes zeigt die Methode `endElement`, die bei jedem schließenden Tag ausgeführt wird:

```
public void endElement(String namespaceURI,
                       String sName, // simple name
                       String qName  // qualified name
           )
   throws SAXException {
   String eName = sName; // element name
   if ("".equals(eName)) eName = qName; // namespaceAware = false

   if( eName.equals( "Breite" ) ) {
     System.out.println( "Breite: " + textContent );
   }else if( eName.equals( "Hoehe" ) ) {
     System.out.println( "Hoehe: " + textContent );
   }else if(
 ...
```

Die Methode erhält als Argument den Namen des Elementes. In dem Codeabschnitt ist die Abfrage auf die Elemente `Breite` und `Hoehe` gezeigt. Der jeweilige Inhalt, der vorher in `textContent` abgelegt wurde, wird hier nur ausgegeben. Für

diese beiden Elemente genügt es, nach dem erkannten Ende-Tag den Inhalt auszuwerten. Anders ist die Situation bei `Spieler` und `Position`. Beim Beginn eines Elementes `Spieler` ist es sinnvoll, eine entsprechende Datenstruktur anzulegen, damit bei den folgenden eingebetteten Elementen die Informationen bereits an die passende Stelle geschrieben werden können. In dem ansonsten leeren Element `Position` sind die Attribute relevant. Attribute werden als weiteres Argument der Methode `startElement` übergeben. Der folgende Abschnitt zeigt die Implementierung von `startElement`, bei der ein neuer Spieler erkannt wird und die Positionsangaben ausgegeben werden.

```
public void startElement( String namespaceURI,
                          String lName, // local name
                          String qName, // qualified name
                          Attributes attrs) {
...
   if( eName.equals( "Spieler" ) ) {
       System.out.println( "Spieler:  " );
       // Hier muesste ein Spieler initialisiert werden
   }else if( eName.equals( "Position" ) ) {
       System.out.println( " Position:" );
       for (int i = 0; i < attrs.getLength(); i++) {
          String aName = attrs.getLocalName(i); // Attr name
          if ("".equals(aName)) aName = attrs.getQName(i);
          System.out.println("   " + aName + "\t"
               + attrs.getValue(i) + "");
       }
   } else {
...
```

Mit diesen Vorarbeit kann der eigentliche Parser wie folgt als Methode realisiert werden:

```
public void parse( String fileName ) {
   DefaultHandler handler = this;
   SAXParserFactory factory = SAXParserFactory.newInstance();
   try {
      SAXParser saxParser = factory.newSAXParser();
      saxParser.parse( new File(fileName), handler );
   } catch (Throwable t) {
      t.printStackTrace();
   }
}
```

Im Hauptprogramm wird eine Instanz von `SAXReader` erzeugt und dessen Methode `parse` aufgerufen:

```
public static void main(String[] args) {
    (new SAXReader()).parse( "save.xml" );
}
```

Bei der Ausführung erhält man die Ausgabe

```
>java SAXReader
Breite: 8
Hoehe: 8
Spieler:
 Name: Aljechin
 Position:
   Spalte       3
   Zeile        5
   Farbe        Weiss
 Punkte: 20
```

13.5.3 Vergleich DOM und SAX

Die kurze Vorstellung der beiden APIs DOM und SAX zeigte die grundsätzlichen Unterschiede:

- In DOM wird ein kompletter Baum mit der vollständigen Information aufgebaut. Der Anwender kann über Zugriffsmethoden einzelne Elemente lesen und ändern.
- In SAX wird beim Parsen aus dem XML-Dokument ein Ereignisstrom generiert. Der Anwender kann auf die für ihn relevanten Ereignisse reagieren.

Eine Implementierung mit SAX verbraucht weniger Speicher und Rechenzeit. Dies kann bei Zielsystemen mit eingeschränkten Ressourcen wie etwa Smartphones ein entscheidender Vorteil sein. Aber auch bei Server-Anwendungen ermöglicht eine effiziente Realisierung mehr Anfragen gleichzeitig zu bedienen. Daher kann man in Fällen, in denen Rechenzeit und Speicherbedarf eine Rolle spielen, SAX empfehlen. Die Stärken von DOM zeigen sich, wenn ein freier Zugriff auf die Elemente notwendig wird. Änderungen können am DOM-Baum leicht durchgeführt werden.

DOM ist ein programmiersprachen-unabhängiges API. Dementsprechend sind die Repräsentation und die Zugriffsmethoden neutral gehalten. In einer Java-Anwendung wird man in der Regel nicht mit dem DOM-Baum arbeiten, sondern die Inhalte in eigene Datenstrukturen kopieren. Es ist naheliegend, diese Zusatzarbeit zu sparen und statt dessen gleich einen Baum mit den reichhaltigen Möglichkeiten von Java und der Klassenbibliothek zu implementieren. Zwei solche Ansätze sind DOM4J und JDOM. Noch weitergehend ist das Konzept des *Data binding*, bei dem zu den XML-Komponenten direkt passende Java-Objekte

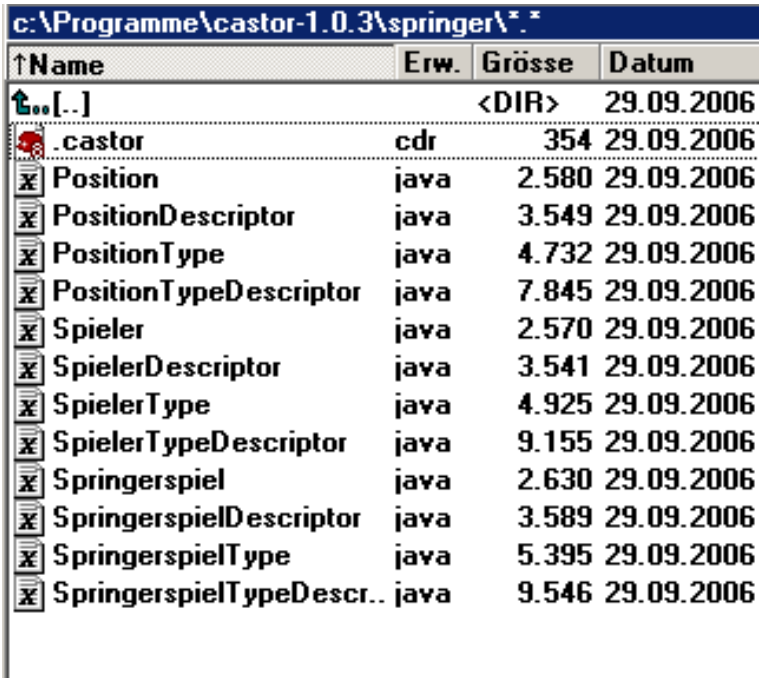


Abbildung 13.13: Die mit dem Programm Castor aus dem Schema generierten Java-Klassen

verwendet werden. Im Projekt *Castor* beispielsweise wurde ein entsprechender Konverter entwickelt. Aus den Definitionen in unserem XML-Schema werden mit dem Aufruf

```
java  org.exolab.castor.builder.SourceGenerator
    -i springer.xsd -package springer
```

passende Java-Klassen generiert (Bild 13.13).

13.6 Übungen

Übung 13.1 *Überarbeiten Sie eine der Formatierung aus Abschnitt 13.3 mittels CSS oder XSLT, so dass ein ansprechendes Design entsteht.*

Übung 13.2 *Erweitern Sie die Sicherungsdatei des Springerspiels um folgende Informationen:*

- *zwei Spieler*
- *bisherige Züge*
- *Positionen der optionalen Sperrfelder*

Führen Sie die Ergänzungen jeweils in

- *CSS oder XSLT*
- *DTD oder XML-Schema*
- *DOM oder SAX*

aus.

Kapitel 14

Web-Services

14.1 Einleitung

Die in Kapitel 10 vorgestellten Remote Procedure Calls (RPC) ermöglichen es, dass eine Anwendung über eine Netzwerkkommunikation eine Prozedur einer anderen Anwendung ausführt. Der RPC-Mechanismus sorgt für eine eindeutige Identifikation der Prozedur und die Datenübergabe. Ähnliche Möglichkeiten bietet in Java die Remote Methode Invocation (RMI) sowie die Common Object Request Broker Architecture (CORBA). Ein neuerer Ansatz ist SOAP. Ursprünglich stand die Abkürzung für Simple Object Access Protocol. Da mittlerweile SOAP wesentlich mehr als Datenaustausch beinhaltet, wird der Name als eigenständige Bezeichnung verwendet.

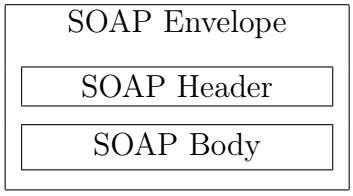
SOAP basiert auf den etablierten Web-Technologien. Die Daten und Prozeduraufrufen werden mittels XML (EXtensible Markup Language) in Textform beschrieben. Als Transportprotokoll dient üblicherweise HTTP. Man spricht daher von Web-Services. Durch die Verwendung von offenen Standards und textbasierten Darstellungen entsteht eine flexible und sprachneutrale Architektur. Da die Dienste auf HTTP basieren und damit über den in aller Regel freien Port 80 arbeiten, werden die bei anderen Diensten oft auftretenden Einschränkungen durch Firewalls vermieden. Für zahlreiche Programmiersprachen stehen Schnittstellen-Implementierungen zur Verfügung. Besonders zu nennen sind hier Microsoft .NET mit Visual Basic und C#, Java und Perl.

Anwendung finden Web-Services als Schnittstelle zu Diensten von Anbietern wie z.B. eBay oder Google. Beispielsweise bietet Google die Möglichkeit an, über einen Web-Service Suchanfragen zu stellen. Über diese Schnittstelle erhält man das Suchergebnis in einer strukturierten Form. Die Alternative wäre, die Analyse der HTML-Seite mit Suchergebnissen. Mühsam müssten dann aus den Formatierungsinformationen, die sich auf die Darstellung der Daten beziehen, die gewünschten Inhalte extrahiert werden. Demgegenüber ist – wie wir im Abschnitt 14.6 sehen werden – die Programmierung eines Clients für den Web-Service sehr

viel einfacher.

14.2 Nachrichtenformat

Eine Nachricht in SOAP besteht aus einer Hülle (Envelope), einem optionalen Kopf (Header) sowie dem Nachrichtentext im Rumpf (Body):



Im Kopf stehen Blöcke mit Informationen, die den Transport der Nachricht betreffen. Jeder SOAP-Knoten (SOAP intermediary) auf dem Weg vom Sender (Initial SOAP sender) zum Empfänger (Ultimate SOAP receiver) interpretiert die Angaben im Header und führt gegebenenfalls entsprechende Aktionen aus. Dabei können auch einzelne Blöcke aus dem Header gelöscht und neue eingefügt werden. Die Informationen im Rumpf sind für den eigentlichen Empfänger bestimmt.

14.3 Beispiel mit SOAPLite

14.3.1 Grundstruktur

Wir beginnen mit einem Beispiel in der Programmiersprache Perl. Dazu steht mit SOAPLite (`http://www.soaplite.com/`) eine Reihe von Modulen für einen einfachen Zugriff auf SOAP zur Verfügung [26]. Der Namensteil *Lite* bezieht sich nicht etwa auf einen eingeschränkten Leistungsumfang, sondern die einfache Verwendbarkeit. Wir wollen die Grundstruktur eines Notenservers implementieren. Über entsprechende Anfragen kann man bei dem Server die Noten abfragen. Dazu wird eine Datei mit folgendem Inhalt erstellt:

```
#!perl -w
# Datei noten.cgi
# Noten Server

use SOAP::Transport::HTTP;

SOAP::Transport::HTTP::CGI
  -> dispatch_to('Notenserver')
  -> handle;
```

```
package Notenserver;

# Verbindungstest
sub test {
  return "Friedberger Notenserver";
}
```

Die Endung cgi steht für Common Gateway Interface. Dabei handelt es sich um einen allgemeiner Mechanismus, über den ein Webserver Programme ausführen kann. Diese Programme erzeugen HTML-Code, der dann an den Client geschickt wird. Bei Apache-Webservern stehen die cgi-Dateien üblicherweise in einem Verzeichnis `cgi-bin`. Die erste Zeile des Beispielprogramms gibt an, dass der folgende Code mit Perl interpretiert wird. Im weiteren wird

- festgelegt, dass der Transport über HTTP erfolgen soll.
- ein Paket mit dem Namen `Notenserver` definiert.
- eine Prozedur `test` eingeführt. Diese Prozedur hat kein Argument und gibt einen kurzen Begrüßungstext zurück.

Mit diesen Vorbereitungen kann dann der Client wie folgt geschrieben werden:

```
#!perl -w
# Datei noten.pl
# Noten Client

use SOAP::Lite;

my $s = SOAP::Lite
  -> uri('http://localhost/Notenserver')
  -> proxy('http://localhost/cgi-bin/noten.cgi');

print $s
  -> test()
  -> result;
```

Hier wird zunächst ein Kontaktpunkt auf dem lokalen Rechner mit dem Namen `Notenserver` und der zugehörigen Datei angelegt. Dann wird das Ergebnis nach Aufruf der Prozedur `test()` ausgegeben. Bei der Ausführung des Clients erscheint dann in der Tat der Begrüßungstext

```
Friedberger Notenserver
```

Die Module in SOAPLite übernehmen die Generierung und Analyse der SOAP-Nachrichten. Man kann sich die übermittelten Nachrichten mit dem Test-Tool

ProxyTrace von Simon Fell (`http://pocketsoap.com/`) anschauen. Das Tool wird zwischen Client und Server geschaltet. Dazu muss im Client ein zusätzlicher Proxy angegeben werden:

```
-> proxy('http://localhost/cgi-bin/noten.cgi',
             proxy => ['http' => 'http://localhost:9090/']);
```

In diesem Fall ist Port 9090 gewählt. ProxyTrace liest alle Paket an Port 9090, protokolliert sie und leitet sie dann an den Server auf Port 80 weiter. Der einfache Test ergibt folgende Anfrage:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <namesp1:test xmlns:namesp1="http://localhost/Notenserver"/>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Im Rumpf steht nur ein Eintrag mit dem Namen der Prozedur sowie des Paketes. Die zugehörige Antwort lautet:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <namesp1:testResponse xmlns:namesp1="http://localhost/Notenserver">
      <s-gensym3 xsi:type="xsd:string">
         Friedberger Notenserver
      </s-gensym3>
    </namesp1:testResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

14.3.2 Erweiterungen

Auf dieser Basis können Erweiterungen vorgenommen werden. Zunächst führen wir die Prozedur `note2` mit einem Argument für die Matrikelnummer ein. Die

übergebenen Argumente stehen in der speziellen Variablen `@_`, wobei der erste Wert den Namen des Paketes (in diesem Fall Notenserver) enthält. Mit der Anweisung

```
  my ($class, $mat) = @_;   # kopieren der Argumente
```

werden die beiden ersten Argumente in die zwei Variablen `$class` und `$mat` kopiert. Insgesamt ergibt sich:

```
# in noten.cgi anhaengen
# Funktion mit Argument Matrikelnummer
sub note2 {
  my ($class, $mat) = @_;   # kopieren der Argumente
  return "Matrikelnummer $mat, Ergebnis:  Mathematik 1.3";
}
```

In dem Beispiel wird die Matrikelnummer lediglich zurück gegebenen. In einer realistischen Anwendung würde die zugehörige Note aus einer Datenbank ermittelt. Der Aufruf im Client hat dann die Form

```
# in noten.pl
print $s
  -> note2( "123456" )
  -> result;
```

Als Ausgabe resultiert dann:

```
Matrikelnummer 123456, Ergebnis:  Mathematik 1.3
```

Die SOAP-Anfrage enthält das Argument im Rumpf. Die Matrikelnummer wird wie alles andere als Text übertragen. Allerdings wurde der Typ als Integer erkannt und entsprechend markiert.

```
<SOAP-ENV:Body>
  <namesp3:note2
    xmlns:namesp3="http://localhost/Notenserver">
    <c-gensym5 xsi:type="xsd:int">
      123456
    </c-gensym5>
  </namesp3:note2>
</SOAP-ENV:Body>
```

Betrachten wir abschließend noch die Rückgabe einer komplexeren Datenstruktur. Die Prozedur `note3` gibt eine Liste mit Fachnamen und einzelnen Noten zurück.

```
# Rueckgabe mit Feld
sub note3 {
  return {
     'Mathematik', '1.3',
     'BWL',        '2.0',
     'Informatik', '1.0'
  };
}
```

Dies wird in der SOAP-Antwort wie folgt als Struktur abgebildet:

```
<s-gensym3 xsi:type="namesp2:SOAPStruct">
  <Informatik xsi:type="xsd:float">1.0</Informatik>
  <BWL xsi:type="xsd:float">2.0</BWL>
  <Mathematik xsi:type="xsd:float">1.3</Mathematik>
</s-gensym3>
```

Das Ergebnis ist demnach eine Struktur mit den Elementen Informatik, BWL und Mathematik, die wiederum jeweils einen Wert vom Typ float haben. In der Sprache Perl gibt es keinen Datentyp für Strukturen. Man kann allerdings ein ähnliches Verhalten mit Hash-Tabellen nachbilden. Der Name eines Elementes dient dann als Schlüssel zum Zugriff auf den Wert. Dementsprechend ruft der folgende Code im Client die Prozedur `note3` auf, speichert einen Verweis auf das Ergebnis in der Variablen `$result` und gibt dann alle Elemente der Tabelle aus.

```
my $result = $s
  -> note3()
  -> result;

foreach my $key (keys %{$result}) {
  print $key, ": ", $result->{$key} || '', "\n";
}
```

Es resultiert dann die Ausgabe:

```
Informatik: 1.0
Mathematik: 1.3
BWL: 2.0
```

Die Information ist in der Tat korrekt angekommen, lediglich die Reihenfolge der einzelnen Elemente wurde durch die interne Verwaltung der Hash-Tabelle verändert.

14.4 Java-Client

Eine direkte Möglichkeit zur Generierung von SOAP-Nachrichten bietet das SOAP with Attachments API for Java (SAAJ). Im wesentlichen übernehmen die Klassen und Methoden dieses APIs die Aufgabe, die SOAP-Nachrichten mit XML-Text zu füllen. Ausgehend von dem Beispiel in [27] lässt sich ein Client für den Aufruf der Prozedur `test()` unseres Notenservers wie folgt realisieren:

```
import java.net.*;
import javax.xml.soap.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;

public class NotenClient {

   public static void main(String args[]) {
      try {
         // Verbindung
         SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
         SOAPConnection connection = scf.createConnection();
         SOAPFactory sf = SOAPFactory.newInstance();

         // Nachricht erzeugen
         MessageFactory mf   = MessageFactory.newInstance();
         SOAPMessage message = mf.createMessage();

         // SOAP Objekte anlegen
         SOAPPart     soapPart = message.getSOAPPart();
         SOAPEnvelope envelope = soapPart.getEnvelope();
         SOAPBody     body     = envelope.getBody();

         // Body fuellen
         Name bodyName =
               sf.createName("test", "namespace1",
                   "http://localhost/Notenserver");
         SOAPBodyElement bodyElement = body.addBodyElement(bodyName);

         // Kontrollausgabe der SOAP-Nachricht
         System.out.println("********************************");
         System.out.println("SOAP Anfrage:");
         message.writeTo(System.out);
         System.out.println();
```

```
            // Ziel festlegen
            URL endpoint = new URL("http://localhost/cgi-bin/noten.cgi");

            // Schliesslich Nachricht schicken,
            // Antwort abholen und dann Verbindung schliessen
            SOAPMessage response = connection.call(message, endpoint);
            connection.close();

            // Ausgabe der Antwort
            System.out.println("********************************");
            System.out.println("SOAP Antwort:");

            TransformerFactory tf = TransformerFactory.newInstance();
            Transformer transformer = tf.newTransformer();
            Source content = response.getSOAPPart().getContent();
            StreamResult result = new StreamResult(System.out);
            transformer.transform(content, result);
            System.out.println();

        } catch(Exception e) {
            System.out.println("Exception: " +e.getMessage());
            e.printStackTrace();
        }
    }
}
```

14.5 Web Service Description Language

SAAJ hilft, die SOAP-Nachrichten zu erstellen und zu analysieren. Ansonsten bleibt es in der Verantwortung des Programmierers, die Prozeduren mit ihren Parametern und Rückgabewerten richtig zu verwenden. Einen anderen Weg verfolgt man mit der Web Service Description Language (WSDL) [28]. Hier wird zunächst der Web-Service formal beschrieben. Dazu werden – selbstverständlich wieder im XML-Format – die Eigenschaften des Services definiert. Daraus können dann automatisch entsprechende Implementierungen in den verschiedenen Programmiersprachen generiert werden. Im Einzelnen beinhaltet eine WSDL-Datei folgende Elemente

- Types: Definition von Datentypen.
- Message: Abstrakte Definition von Nachrichten.
- Operation: Abstrakte Definition von Aktionen.

- Port Type: Operationen, die von einem oder mehreren Endpunkten unterstützt werden.
- Binding: Protokoll und Datenformat für einen Port Type
- Port: ein einzelner Endpunkt.
- Service: Eine Menge von zusammen gehörenden Endpunkten.

```
<?xml version="1.0"?>
<!-- WSDL Beschreibung fuer Notenserver -->

<definitions name="Notenserver"
             targetNamespace="urn:Notenserver"
             xmlns:typens="urn:Notenserver"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
             xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

  <!-- Definition eines komplexen Typs -->
  <types>
    <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
                targetNamespace="urn:Notenserver">
      <xsd:complexType name="NotenserverResult">
        <xsd:all>
          <xsd:element name="Mathematik"       type="xsd:float"/>
          <xsd:element name="BWL"              type="xsd:float"/>
          <xsd:element name="Informatik"       type="xsd:float"/>
        </xsd:all>
      </xsd:complexType>
    </xsd:schema>
  </types>

  <!-- ************  Nachrichten ***************  -->
  <message name="test">  </message>

  <message name="testAntwort">
    <part name="return"         type="xsd:string"/>
  </message>

  <message name="note3">
    <part name="matrikel"       type="xsd:string"/>
```

```
</message>

<message name="note3Antwort">
  <part name="return"         type="typens:NotenserverResult"/>
</message>

<!--Port Definition "Notenserver" -->
<portType name="NotenserverPort">

  <operation name="test">
    <input message="typens:test"/>
    <output message="typens:testAntwort"/>
  </operation>

 <operation name="note3">
    <input message="typens:note3"/>
    <output message="typens:note3Antwort"/>
  </operation>

 </portType>

<!-- Binding for RPC, SOAP over HTTP -->
<binding name="NotenserverBinding" type="typens:NotenserverPort">
  <soap:binding style="rpc"
                transport="http://schemas.xmlsoap.org/soap/http"/>

 <operation name="test">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="encoded"
         namespace="urn:Notenserver"
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <output>
      <soap:body use="encoded"
         namespace="urn:Notenserver"
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
  </operation>
</binding>

 <operation name="note3">
    <soap:operation soapAction=""/>
```

```
      <input>
        <soap:body use="encoded"
           namespace="urn:Notenserver"
           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
           namespace="urn:Notenserver"
           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
   </operation>

  <!-- Endpoint Notenserver -->
  <service name="NotenserverService">
    <port name="NotenserverPort" binding="typens:NotenserverBinding">
      <soap:address location="http://localhost/cgi-bin/noten.cgi"/>
    </port>
  </service>

</definitions>
```

Aus dieser Datei werden mit dem Tool WSDL2Java aus dem AXIS-Projekt mit dem Aufruf

```
java org.apache.axis.wsdl.WSDL2Java noten.wsdl
```

entsprechende Java-Klassen erzeugt. Im Beispiel werden folgende 5 Klassen angelegt:

```
NotenserverBindingStub.java
NotenserverPort.java
NotenserverResult.java
NotenserverService.java
NotenserverServiceLocator.java
```

Als Beispiel sei der Beginn der Klassendatei für das Result angegeben:

```
/**
 * NotenserverResult.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis 1.2RC1 Sep 29, 2004 (08:29:40 EDT) WSDL2Java emitter.
 */

package Notenserver;
```

```
public class NotenserverResult  implements java.io.Serializable {
  private float mathematik;
  private float BWL;
  private float informatik;

  public NotenserverResult() {}

  public NotenserverResult( float mathematik, float BWL,
                            float informatik) {
    this.mathematik = mathematik;
    this.BWL = BWL;
    this.informatik = informatik;
  }
...
```

Der Client hat dann folgende Form:

```
import  Notenserver.*;

public class Tester
{
  public static void main(String [] args) throws Exception {
    // Service und Port anlegen
    NotenserverService service = new NotenserverServiceLocator();
    NotenserverPort port = service.getNotenserverPort();

    System.out.println( "Methode test:" );
    System.out.println( port.test(  ) );

    System.out.println( "Methode note3:" );
    NotenserverResult nr =  port.note3( "123456" );
    System.out.println( "Mathematik: " + nr.getMathematik() );
    System.out.println( "Informatik: " + nr.getInformatik() );
    System.out.println( "BWL:        " + nr.getBWL() );

  }
}
```

Die Ausführung ergibt wieder das bekannte Resultat

```
Methode test:
Friedberger Notenserver
Methode note3:
```

```
Mathematik: 1.3
Informatik: 1.0
BWL:        2.0
```

14.6 Beispiel Google

Wie bereits erwähnt bietet Google eine SOAP-Schnittstelle an. Man muss sich für die Nutzung dieses Dienstes bei Google registrieren lassen (http://www.google.com/apis/) und erhält dann einen Zugangsschlüssel für eine begrenzte Anzahl von kostenlosen automatischen Anfragen pro Tag.

Weiterhin stellt Google Beispielprogramme in diversen Programmiersprachen bereit. Davon abgeleitet ist das folgende Perl-Programm für eine automatische Suchanfrage. In dem Beispiel ist als Suchbegriff `informatik friedberg` fest eingestellt. Das Programm gibt die Gesamtzahl der Treffer sowie die 10 besten Ergebnisse aus.

```
use SOAP::Lite;
# Hier Zugangsschluessel fuer Google eintragen
my $key='00000000000000000000000000000000';
my $query="informatik friedberg";
my $googleSearch = SOAP::Lite -> service("file:GoogleSearch.wsdl");
my $result = $googleSearch -> doGoogleSearch($key, $query, 0, 10,
      "false", "", "false", "", "latin1", "latin1");

print "$result->{'estimatedTotalResultsCount'} Treffer\n";

print "Top Treffer:\n";
print "-------------------------------------------------\n";
$i = 1;
foreach my $el (@{$result->{'resultElements'}})
  {
    print "$i $el->{'URL'} \n";
    $i++;
  }
print "-------------------------------------------------\n";
```

Die Ausführung lieferte folgendes Resultat:

```
5820 Treffer
Top Treffer:
-------------------------------------------------
1 http://www.fh-giessen.de/fh/script/
```

```
2 http://www.fh-giessen.de/fachschaft/mni/
3 http://www.fh-giessen.de/studium/info-pdf/2002-10-informatik....
4 http://www.fh-giessen.de/fachbereich/mni/download/MH_1.pdf
5 http://www.stern.de/CHE5/CHE5?module=Fachbereich&do=show&id=4...
6 http://www.stern.de/CHE5/CHE5?module=Fachbereich&do=show&id=4...
7 http://www.studieren.de/details1.asp?id1=338&id2=457&id3=8080...
8 http://www.zeit.de/anzeigen/stellen/2004/1/200401_anz_st_39789
9 http://sirius.fh-friedberg.de/MND_MS_Forum/Windows_2000_Techni...
10 http://sirius.fh-friedberg.de/MND_MS_Forum/Windows_2000_AD_te...
-------------------------------------------------
```

Kapitel 15

Datensicherheit und Verschlüsselung

15.1 Einleitung

Mit der zunehmenden Bedeutung vernetzter Rechnersysteme wächst der Bedarf an Sicherheit. Insbesondere die Verlagerung von kritischen Abläufen wie

- Einkaufen und Verkaufen
- Abwickeln von Bank- und Börsengeschäften
- Bewerbungen auf Ausschreibungen

hin zu elektronischen Formen verlangt eine sichere und vertrauliche technische Grundlage. Darüber hinaus gilt es, die direkt oder indirekt an das universelle Netz angeschlossenen Systeme gegen unberechtigte Zugriffe zu sichern.

15.2 Firewall

Wie bei der namengebenden Anwendung als Schutz gegen die Ausbreitung von Bränden, hat ein Firewall die Aufgabe, ein privates Netz vor unkontrollierten Zugriffen aus anderen Netzen zu schützen. Oder – um ein anderes Bild zu gebrauchen – so wie eine mittelalterliche Burg mit Mauer, Tor und Zugbrücke wird ein Rechnernetz durch einen kontrollierten Zugang vor Angriffen von außen geschützt. Insofern versteht man unter Firewall das allgemeines Konzept einer strikten Trennung zwischen internen und externen Netzen. Nach vorgegebenen Regeln wird entschieden, welche Daten durchgelassen werden.

Eine Firewall kann durch ein eigenständiges Gerät (Rechner mit entsprechender Software oder spezielle Hardware-Firewall) realisiert werden. Diese Firewall wird in der Regel als Absicherung zwischen einem lokalen Netz und dem Internet

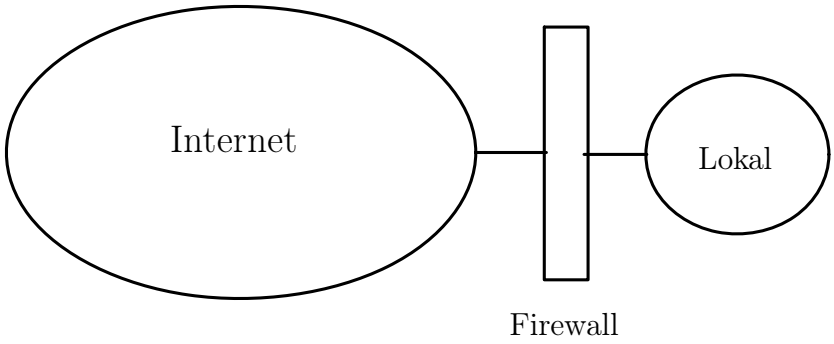


Abbildung 15.1: Firewall zwischen Internet und lokalem Netz

geschaltet. Man kann dies auch als einen Router zwischen den beiden Netzen sehen, der allerdings zusätzlich den Datenverkehr kontrolliert.

Am einfachsten aufgebaut sind filterbasierte Firewalls (Paketfilter). Die Regeln beziehen sich dann auf bestimmte Adressen. Eine Möglichkeit ist, eine Tabelle mit erlaubten Quell- und Zieladressen zu führen. Als Prinzip gilt dann: *Alles, was nicht ausdrücklich erlaubt ist, ist verboten* . Dann würde eine Zeile

169.254.107.238, 123, 212.125.100.86, 80

bedeuten, dass alle Pakete vom Rechner 169.254.107.238 und Port 123 zu Rechner 212.125.100.86, Port 80 passieren dürfen. Als Alternative kann man auch das liberale Konzept: *Alles, was nicht ausdrücklich verboten ist, ist erlaubt* einsetzen. In diesem Fall würde der Eintrag genau diese Kombination verbieten.

Ein Paketfilter arbeitet auf den unteren Protokollebenen. In dem beschriebenen Beispiel werden Pakete bis zur Schicht 4 analysiert, um dann anhand der IP-Adresse und der Port-Nummer über die Weiterleitung zu entscheiden. Daher werden sie auch als Network Level Firewall bezeichnet. Die Wirksamkeit einer Firewall lässt sich erhöhen, indem nicht nur auf Basis eines einzelnen Paketes entschieden wird, sondern eine Verbindung als ganzes betrachtet wird. Am weitesten gehen in diesem Sinne Application Level Firewalls (Proxy-basierte Firewalls).

In diesem Fall kommuniziert eine Anwendung – z.B. ein Browser – nicht direkt mit dem Internet. Vielmehr wird der gesamt Verkehr über einen Stellvertreter – den Proxy – abgewickelt. Nur der Proxy hat die Berechtigung, Daten mit dem Internet auszutauschen. Dieses Konzept erlaubt weit gehende Kontroll- und Protokollfunktionen und gilt als sehr sicher. Negativ ist eine gewisse Einschränkung der Flexibilität und verringerte Transfergeschwindigkeit. Als Alternative werden daher auch Mischformen zwischen den Filtern auf Paketebene und den Proxys auf Anwendungsebene eingesetzt.

Von außen ist der Firewall der einzige Angriffspunkt auf das lokale Netz. Daher wird bei der Administration besondere Sorgfalt auf die Sicherheit dieses Systems gelegt. Ist allerdings diese Hürde doch genommen, so befindet sich der

Eindringling bereits im relativ ungeschützten internen Netz. Um diese Gefahr zu verringern, kann das interne Netz durch eine zweite Firewall geschützt werden. Hinter der ersten Firewall sind dann nur Rechner angeordnet, die einen externen Zugang benötigen. Typischerweise werden hier Webserver und ähnliches zu finden sein. Der Zugang aus diesem Netz zu dem internen Netz mit den kritischen Daten wird dann durch eine zweite Firewall geschützt. Die Zone zwischen den beiden Firewalls bildet ein Grenznetz (auch De-Militarized Zone, DMZ) .

Zur Absicherung von einzelnen Rechnern werden so genannte *Personal* oder *Desktop Firewall* Lösungen eingesetzt. Solche Systeme werden häufig von Privatanwendern verwendet, um ihren direkt am Internet angeschlossenen Rechner zu schützen. Die Personal Firewall wirkt dann als Paketfilter, der unerwünschten Datenverkehr zwischen dem Rechner und dem Internet blockiert.

Allerdings darf man den Gewinn an Sicherheit nicht überschätzen. Man muss leider davon ausgehen, dass ein z.B. über Email eingedrungener Virus intelligent genug ist, um die Firewall von innen heraus zu überwinden. Beispielsweise könnte er eine Regel für eine weitere Verbindung eintragen und diese dann für eigene Zwecke nutzen. Eine Personal Firewall ist allerdings gut geeignet, um einen Überblick über die Netzwerk-Aktivitäten zu gewinnen.

15.3 Virtuelle private Netze

Firewalls dienen dazu, das eigene Netz gegenüber dem Internet abzuschotten. Nun gibt es allerdings Situationen, in denen gerade ein externer Zugang benötigt wird. Ein typischer Fall ist ein Mitarbeiter auf Dienstreise, der Zugriff auf seine internen Daten benötigt. Ein recht sicherer Weg besteht in der Verwendung von Wählleitungen. Die Verbindung geht dann über ein Modem und eine Telefonleitung zu einem entsprechenden Firmenanschluss. Dieser Anschluss wird durch ein Passwort geschützt.

Wesentlich eleganter ist es, auf die Infrastruktur des Internets zurück zu greifen. Der Mitarbeiter nutzt dann beispielsweise den WLAN-Zugang in seinem Hotel, um sich mit dem Internet zu verbinden. Umgekehrt muss die Firma einen Zugang vom Internet zu den internen Daten bereit stellen. Für den Weg dazwischen wird das Internet genutzt.

Das Problem ist nun, wie diese Verbindung sicher gestaltet werden kann. Der Weg durch das Internet ist nicht geschützt und es besteht die Gefahr, dass der Inhalt der IP-Pakete von Unberechtigten eingesehen oder gar manipuliert wird. Betrachten wir ein Szenario, bei dem ein externer Mitarbeiter (Client) auf einen Rechner X im internen Netz zugreifen möchte. Dazu muss er Daten an die Adresse X schicken. Da es sich um vertrauliche Informationen handelt, müssen die Daten verschlüsselt werden. Aber auch die Adresse X soll nicht öffentlich bekannt werden.

Eine in diesem Sinne sichere Verbindung wird durch einen so genannten *Tun-*

nel durch das Internet realisiert. Dazu wird das eigentliche Paket – Kopf und Rumpf – als Einheit verschlüsselt. Die entstehenden Daten werden dann als Nutzlast in einem IP-Paket an einen entsprechenden Server (Remote-Tunnel-Server) im Firmennetz geschickt. Aus Sicht des Internets handelt es sich also um eine IP-Verbindung zwischen Client und diesem Server. Die Daten in dem IP-Paket sind durch Verschlüsselung geschützt. Der Server nimmt das Paket entgegen und entschlüsselt die Daten. Damit rekonstruiert er das eigentliche Paket, das er dann in das interne Netz einspeisen kann.

Durch diese Technik kann eine Endanwendung sich so verhalten, als wäre der Rechner direkt an das Firmennetz angeschlossen (Virtuelles privates Netz, VPN). Die Pakete durchqueren das dazwischen liegende Internet eingebettet in ein IP-Paket. Auf diese Art und Weise kann eine sichere Kommunikation über das Internet erreicht werden. Dies ist nicht der einzige Vorteil. Der Inhalt des Paketes spielt für den Transport durch den IP-Tunnel keine Rolle. Es ist daher durchaus möglich, dass im firmeninternen Netz ein anderes Protokoll oder private IP-Adressen verwendet werden.

Eine komplette Lösung für den Aufbau eines Tunnels besteht aus den Komponenten für die einzelnen Funktionen wie z.B. Benutzer-Authentifikation oder Daten-Verschlüsselung. Einige verbreitete Tunneling-Protokoll sind:

- Point to Point Tunneling Protocol (PPTP)
- Layer 2 Tunneling Protocol (L2TP), RFC 2661
- IP Security Protokoll (IPSec)

Sie wurden für verschiedene Anwendungsmöglichkeiten entwickelt und unterscheiden sich dem entsprechend in ihren Möglichkeiten.

15.4 Verschlüsselung

15.4.1 Einleitung

Der Inhalt der Pakete im Internet ist relativ leicht einsehbar. Mit wenig Aufwand kann man auf einem Rechner beispielsweise alle ankommenden und abgehende IP-Pakete lesen. Ohne weitere Maßnahmen sind daher kritische Daten wie Kennwörter oder die Nummern von Kreditkarten nicht vor Unberechtigten geschützt. Im Internet hat man als Anwender keinen Einfluss auf den Übertragungsweg. Man muss vielmehr im Allgemeinen davon ausgehen, dass die Übertragung nicht abgesichert ist und der Inhalt der Pakete auch Fremden zugänglich ist.

Um trotzdem sensible Daten übermitteln zu können, muss man diese Daten durch eine Verschlüsselung schützen. Im Prinzip handelt es sich dabei um eine Art von Kodierung ähnlich wie der Einbau von Paritätsinformationen. Durch einen Verschlüsselungsalgorithmus wird die Nachricht – der Klartext – in eine

andere Form gebracht. Diese kodierte Nachricht wird im Netz übertragen. Der Empfänger kann den Verschlüsselungsalgorithmus umkehren und den Klartext rekonstruieren.

Die Besonderheit bei diesem Vorgehen ist allerdings, dass der Algorithmus geheim gehalten wird. Nur Sender und Empfänger kennen den Algorithmus (oder zumindest die verwendeten Parameter). Dritte können zwar während der Übertragung die Nachricht abfangen, aber ohne Kenntnis des Verschlüsselungsalgorithmus können sie die Nachricht nicht verstehen oder gar manipulieren. Anschaulich kann man sich die Nachricht wie ein Paket mit einem Schloss vorstellen. Der Sender verschliesst das Paket und nur mit dem richtigen Schlüssel kann es wieder geöffnet werden. Die Methoden der Verschlüsselung sind Gegenstand der Kryptographie. Demgegenüber bezeichnet Kryptoanalyse die Wissenschaft von der Entschlüsselung ohne Kenntnis des Schlüssels.

Im folgenden werden zunächst anhand eines einfachen Beispiels die Grundregeln eingeführt. Anschließend folgt der Übergang zu digitalen System mit einer Beschreibung einiger wesentlichen Verfahren. Darauf aufbauen werden einige Anwendungsbeispiele vorgestellt. In der Literatur zur Kryptographie werden häufig die Beteiligten als Personen mit den Namen *Bob* und *Alice* behandelt. Im Rahmen dieses Textes werden aber weiterhin die Bezeichnungen *Sender* und *Empfänger* verwendet. In symmetrischen Fällen, bei denen diese Unterscheidung nicht sinnvoll ist, wird durch die Abkürzungen A und B ersetzt.

15.4.2 Monoalphabetisch Verschlüsselung

Betrachten wir ein einfaches Beispiel für einen Algorithmus zur Verschlüsselung von Texten:

1. Die Buchstaben des Schlüsselwortes werden in umgekehrter Reihenfolge aufgeschrieben. Eventuelle doppelt vorkommende Buchstaben werden dabei nur einmal berücksichtigt.

2. Alle anderen Buchstaben des Alphabets werden ebenfalls in umgekehrter Reihenfolge angehängt.

Nach dieser Vorschrift wird aus dem Schlüsselwort HAMLET folgende Tabelle zur Verschlüsselung erzeugt:

A	B	C	D	E	F	G	H	I	J	K	L	M
T	**E**	**L**	**M**	**A**	**H**	Z	Y	X	W	V	U	S
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
R	Q	P	O	N	K	J	I	G	F	D	C	B

Anhand dieser Tabelle wird der zu übermittelnde Text umgesetzt. Jeder einzelne Buchstabe wird gemäß dieser Zuordnung durch einen anderen ersetzt. Als Beispiel ergibt sich folgende Verschlüsselung:

S	E	I	N		O	D	E	R		N	I	C	H	T	S	E	I	N
K	A	X	R		Q	M	A	N		R	X	L	Y	J	K	A	X	R

Die Leerzeichen verraten noch relativ viel über die Satzstruktur. Es ist daher eine gute Idee, entweder das Leerzeichen als weiteres Zeichen in die Tabelle aufzunehmen oder einfach alle Leerzeichen zu entfernen. In aller Regel wird der Empfänger nach der Entschlüsselung aller Buchstaben die Wortgrenzen leicht rekonstruieren können. Ohne Leerzeichen erhält man in dem Beispiel die verschlüsselte Nachricht

KAXRQMANRXLYJKAXR

Für den Uneingeweihten handelt es sich hier um eine sinnlose Buchstabenfolge. Der Empfänger kann demgegenüber - sofern er das Schlüsselwort kennt - den Prozess der Verschlüsselung umkehren. Dazu konstruiert er zuerst nach der obigen Vorschrift ebenfalls die Zuordnungstabelle. Dann ersetzt er die empfangenen Zeichen wieder durch die ursprünglichen Zeichen. Aus der resultierenden Buchstabenfolge wird er leicht durch Einfügen der Leerzeichen den ursprünglichen Text rekonstruieren können.

Da nur ein Alphabet zur Verschlüsselung benutzt wird, bezeichnet man derartige Verfahren als monoalphabetisch. Weiterhin handelt es sich um ein symmetrisches Verfahren: Es gibt genau einen Schlüssel, der sowohl zum Verschlüsseln als auch zum Entschlüsseln benötigt wird. Sender und Empfänger benötigen die gleichen Information. Kennt ein Unbefugter den Schlüssel, so kann er ebenfalls jede Nachricht entschlüsseln.

Solche Verfahren waren in unterschiedlichen Varianten jahrhundertelang in Gebrauch. Allerdings bieten sie nur einen geringen Schutz. Die Verschlüsselung kann relativ leicht durch den Einsatz der Häufigkeitsanalyse gebrochen werden. Dabei nutzt man die unterschiedliche Häufigkeit für das Auftreten der einzelnen Buchstaben aus. So ist beispielsweise in deutschen Texten im Mittel E der häufigste Buchstabe. Damit kann man davon ausgehen, dass der häufigste Buchstabe in einem derart verschlüsselten Text mit recht hoher Wahrscheinlichkeit eigentlich das E darstellt. Weiterhin kann man nach häufig vorkommenden Buchstabenpaaren suchen. So sind die Kombinationen LL, MM, NN, EE oder TT gute Kandidaten für häufig doppelt vorkommende Zeichen.

Auch immer wiederkehrende längere Folgen wie SCH bieten Ansatzpunkte für die Entschlüsselung. Auf diese Art und Weise kann durch systematisches Raten unter Berücksichtigung der bekannten Häufigkeiten von Buchstaben und Buchstabenfolgen eine monoalphabetische Verschlüsselung relativ leicht entschlüsselt werden. Eine besondere Hilfe ist dabei, wenn man einige der Wörter im Klartext kennt. Handelt es sich beispielsweise um einen Wetterbericht, so kommt darin mit einiger Wahrscheinlichkeit auch das Wort WETTER vor. Man kann dann gezielt nach der Folge ETTE beziehungsweise einem Muster XYYX suchen. Findet man ein Muster XYYX, so kann man die Zuordnung X=E und Y=T ausprobieren.

Derartige einfache Verschlüsselungsverfahren sind damit nicht vollkommen nutzlos geworden – aber sie bieten nur einen geringen Schutz. Ob dieser Schutz

als ausreichend erachtet werden kann, hängt von der Anwendung ab. Soll nur eine sehr kurze Nachricht geschützt werden oder ist der Schutz nur für eine kurze Zeit erforderlich, so kann eine solche einfache Verschlüsselung durchaus ausreichen. Die verschlüsselte Nachricht ist zunächst gegenüber Unbefugten geschützt. Jemand, der zufällig oder absichtlich die Nachricht liest, erkennt nicht direkt den Inhalt.

15.4.3 Digitale Verschlüsselung

Mit dem Aufkommen von Rechenautomaten und Computern änderte sich zunächst nichts am Prinzip. Allerdings können sowohl die Verschlüsselung als die Versuche zum Aufbrechen des Codes durch den Einsatz von Rechnern dramatisch beschleunigt werden. Damit sind einerseits aufwändigere Verfahren möglich und andererseits verkürzt sich die Zeitspanne, für die ein Code Sicherheit bietet. Die ersten Rechner wurden während des zweiten Weltkrieges in den britischen Geheimlaboren als Hilfsmittel zur Entschlüsselung des deutschen Funkverkehrs eingesetzt [29]. Genauer gesagt, wurde diese Rechner speziell für diesen Zweck entwickelt.

Mit der beginnenden Digitalisierung wurden entsprechende neuartige Verschlüsselungsverfahren benötigt. Diese Verfahren arbeiten nicht mehr mit Zeichen sondern mit einzelnen Bits. Die Nachricht wird unabhängig von ihrem Inhalt als ein Bitstrom behandelt. Bei den im folgenden vorgestellten Verfahren handelt es sich um so genannte Blockcodes. Dabei wird die Nachricht in entsprechende viele einzelne Blöcke mit jeweils einer gegebenen Anzahl von Bit unterteilt. Jeder Block wird dann durch Bit-Manipulationen verschlüsselt. Ein solches Verfahren ist der Data Encryption Standard (DES). Der Algorithmus wurde Mitte der 70er Jahre entwickelt und später vom amerikanischen National Institute of Standards and Technology (NIST) als Standard eingeführt [30].

Der Algorithmus beruht auf der Verwürfelung von Gruppen von jeweils 64 Bit. In einem mehrstufigen Verfahren werden die einzelnen Bit miteinander verknüpft und mit den Ergebnissen überschrieben. In DES werden insgesamt 16 Iterationen – auch als Runden bezeichnet – durchgeführt. Die Details der Verknüpfungsfunktion sind durch den Schlüssel festgelegt. Die Länge eines Schlüssels beträgt 56 Bit. In jeder Runde wird aus diesem Hauptschlüssel ein spezifischer Schlüssel berechnet.

Im Grunde ist DES in dieser Basisversion ein monoalphabetisches Verfahren. Es bildet Wörter der Länge 64 auf andere Wörter der gleichen Länge ab. Kennt man die Struktur der ursprünglichen Daten, so eröffnet diese Eigenschaft die Möglichkeit zur Manipulation. Handelt es sich beispielsweise um Datensätze für Mitarbeiter, so kann man einen Block von 64 Bit einfach durch einen anderen Block ersetzen. Auf diese Art und Weise könnte man beispielsweise das eigene Gehalt durch das eines anderen - besser verdienenden - Kollegen ersetzen, indem man den Block mit den entsprechenden Feldern aus dem fremden Datensatz kopiert. Um derartige Manipulationen auszuschließen, besteht die Möglichkeit, in

die Verschlüsselung eine Abhängigkeit von vergangenen Blöcken einzubauen. Damit ergibt ein bestimmter Datenblock in Abhängigkeit vom Kontext unterschiedliche Codes und die Daten sind gegen einfachen Austausch geschützt. Generell führt die Verkettung zu einem besseren Schutz gegen Kryptoanalyse.

Durch den Schlüssel der Länge 56 sind insgesamt 2^{56} verschiedene Abbildungen möglich. Kennt man ein kurzes Stück Klartext, so kann man durch erschöpfende Suche alle 2^{56} möglichen Schlüssel ausprobieren. Geht man davon aus, dass ein handelsüblicher PC in der Größenordnung eine Million Schlüssel pro Sekunde ausprobieren kann, so berechnet sich eine Zeit von etwa 2300 Jahren für die Suche über alle möglichen Schlüssel. Im Mittel wird der richtige Schlüssel dann nach 1250 Jahren gefunden.

Diese Zeit klingt zunächst beruhigend. Allerdings lässt sich die Aufgabe in einfachster Art und Weise parallelisieren und auf viele Rechner aufteilen. Mit entsprechend vielen Rechnern lässt sich somit die Verschlüsselung in überschaubarer Zeit aufbrechen. Weiterhin kann durch spezielle Hardware-Bausteine die Suche wesentlich beschleunigt werden. Insgesamt kann man daher DES zumindest für kritische Anwendungen nicht mehr als hinreichend sicher betrachten.

Eine ausreichende Sicherheit kann durch einen längeren Schlüssel oder die dreifache Anwendung von DES mit 2 oder 3 verschiedenen Schlüsseln wieder hergestellt werden. Daneben wurden eine Reihe anderen Verfahren vorgeschlagen. Das wohl wichtigste darunter ist IDEA (International Data Encryption Algorithm), entwickelt von Xuejia Lai und James L. Massey von der ETH Zürich [31]. IDEA basiert auf einem Schlüssel der Länge 128 Bit. Die erschöpfende Suche auf einem einzelnen PC würde etwa 10^{25} Jahre benötigen. Damit besteht ausreichend Sicherheit auch gegen Angriffe mit optimierten Hardware-Bausteinen und parallelen Einsatz vieler Rechner.

15.4.4 Gemeinsame Schlüsselvereinbarung

Grundsätzlich bieten die beschriebenen Verfahren eine ausreichende Sicherheit. Sind die verwendeten Schlüssel groß genug, so können nach derzeitigem Stand die Verschlüsselungen nicht aufgebrochen werden. Das Problem dieser Verfahren liegt in der Verwaltung der Schlüssel. Will ein Sender eine verschlüsselte Nachricht verschicken, muss er dem Empfänger seinen Schlüssel mitteilen oder einen vom Empfänger zuvor erhaltenen Schlüssel verwenden. Neben der eigentlichen Nachricht muss also auch der Schlüssel übertragen werden. Diese Übertragung ist besonders sensibel. Sowie ein Dritter in Besitz des Schlüssels gelangt, kann er ebenfalls die Nachricht lesen. Daher muss der Schlüssel auf einem besonders sicheren Weg verschickt werden. Beispielsweise kann eine Bank die Schlüssel mit einem zuverlässigen Boten an ihre Kunden ausliefern.

Mit dem zunehmenden Nachrichtenverkehr stellte die Schlüsselverteilung ein immer größeres logistisches Problem dar. Um so wichtiger war die Entdeckung, dass eine Schlüsselvereinbarung auch über einen unsicheren Kanal möglich ist.

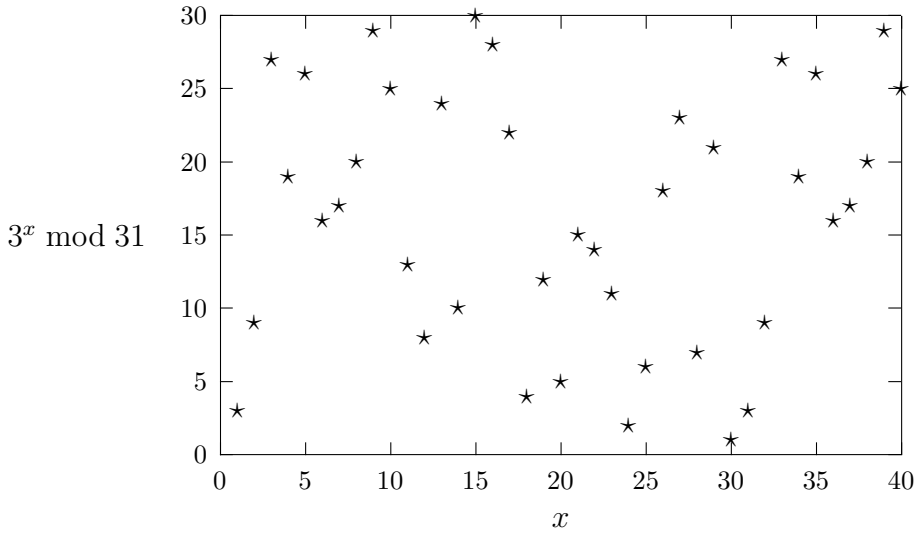


Abbildung 15.2: Beispiel für eine Modulo-Funktion.

Sender und Empfänger vereinbaren dabei einen Schlüssel, der dann wie gehabt für eine symmetrische Verschlüsselung verwendet wird. Die Vereinbarung kann über einen offenen Kanal erfolgen. Ein Dritter kann aus den übertragen Informationen nicht den Schlüssel rekonstruieren. Diese zunächst verblüffende Möglichkeit wurde zuerst von Whitfield Diffie und Martin Hellmann publiziert [32]. Der Kerngedanke ist, eine mathematische Funktion zu verwenden, die praktisch nicht umkehrbar ist. Eine solche „Einwegfunktion“ kann zwar für ein Argument leicht ausgewertet werden. Aber die Umkehrung – die Berechnung des Argumentes aus dem Funktionswert – ist nicht in vernünftiger Zeit möglich. In der Schlüsselvereinbarung nach Diffie und Hellmann (manchmal auch als Diffie-Hellmann-Merkle-Verfahren bezeichnet) und in vielen weiteren Ansätzen wird die Modulo-Funktion verwendet, um den Einweg-Charakter zu erzielen.

Die Modulo-Funktion berechnet den Rest nach ganzzahliger Division. Der Ausdruck $17 \bmod 5$ als Beispiel ergibt den Wert 2, da $17 = 3 * 5 + 2$ gilt. Bei arithmetischen Ausdrücken werden die Operanden zunächst wie gewohnt verknüpft und auf das Ergebnis wird die Modulo-Funktion angewandt. So ergibt beispielsweise $3 * 3 \bmod 5 = 4$. Das Resultat der Exponentialfunktion $3^x \bmod 31$ für verschiedene Werte von x zeigt Bild 15.2. Anders als bei herkömmlichen Funktionen ist hier keine klare Struktur zu erkennen. Aufgrund der Modulo-Funktion schwanken die Ergebnisse selbst für direkt aufeinander folgende Werte von x stark.

Der Einweg-Charakter der Modulo-Funktion resultiert aus der Schwierigkeit, die Funktion zu invertieren. Es ist zwar relativ leicht, für ein gegebenes x die Exponentialfunktion zu berechnen und die Modulo-Operation anzuwenden. Aber

Tabelle 15.1: Schlüsselvereinbarung nach Diffie und Hellmann zwischen Partner A und Partner B

A	öffentlich	B
	$Y = 7$, $P = 11$	
$A = 3$		$B = 6$
$Y^A \bmod P$ $7^3 \bmod 11 = 2$		$Y^B \bmod P$ $7^6 \bmod 11 = 4$
	$\alpha = 2$, $\beta = 4$	
$\beta^A \bmod P$ $4^3 \bmod 11 = 9$		$\alpha^B \bmod P$ $2^6 \bmod 11 = 9$

für die Umkehrung ist kein schnelles Verfahren bekannt. Die Aufgabe, für einen Wert a ein x zu finden, so dass $3^x \bmod 31 = a$ erfüllt wird, erfordert im Allgemeinen einen sehr viel höheren Rechenaufwand. Bei großen Zahlen mit mehreren hundert Stellen lässt sich mit geschickten Algorithmen die Exponentialfunktion immer noch in überschaubarer Zeit auswerten. Aber die Umkehrung erfordert dann selbst bei leistungsfähigen Systemen so viel Zeit, dass sie als praktisch unmöglich anzusehen ist.

Betrachten wir die Schlüsselvereinbarung nach Diffie und Hellmann an einem einfachen Beispiel. In Tabelle 15.1 sind die einzelnen Schritte dargestellt. Zunächst vereinbaren die beiden Partner A und B zwei Zahlen Y und P. Diese Zahlen müssen bestimmte Bedingungen erfüllen. So müssen Y, $(Y-1)/2$ und P Primzahlen sein. Diese Zahlen können öffentlich bekannt sein. Dann wählt jeder Partner ein geheime Zahl aus. Jeder Partner berechnet dann aus seiner Zahl den Wert der Funktion $Y^x \bmod P$. Das Resultat α beziehungsweise β schickt er an sein Gegenüber. Schließlich berechnet er aus dem gerade erhaltenen Wert und seiner Geheimzahl den Schlüssel. Durch die geschickte Konstruktion der Funktion erhalten beide Partner den gleichen Wert – im Beispiel die Zahl 9. Diese Zahl können sie als symmetrischen Schlüssel verwenden. In der Praxis werden die Zahlenwerte sehr viel größer sein, um einen Schlüssel ausreichender Länge zu erhalten.

Die Wirkungsweise dieses Verfahrens lässt sich leicht nachvollziehen. Ausgeschrieben ergeben die beiden Berechnungen $(Y^B \bmod P)^A \bmod P$ und $(Y^A \bmod P)^B \bmod P$. Nach den Regeln der modularen Arithmetik kann der zweite Exponent in den ersten Ausdruck gezogen werden. Damit sind beide Ausdrücke gleichwertig mit $Y^{AB} \bmod P$.

Ein Dritter kann über die öffentliche Leitung die Zahlen Y, P, α und β erfahren. Um den Schlüssel zu berechnen, fehlt ihm die Zahl A (oder gleichwertig B). Daher müsste die Funktion

$$Y^A \bmod P = \alpha \tag{15.1}$$

nach A aufgelöst werden. Dafür ist wie gesagt kein effizientes Lösungsverfahren

bekannt. In dem Beispiel mit den kleinen Zahlen könnte aus der Gleichung

$$7^A \bmod 11 = 2 \tag{15.2}$$

der gesuchte Wert $A = 3$ noch relativ leicht durch Probieren bestimmt werden. Aber für große Zahlenwerte führt dies zu so langen Zeiten, dass die Umkehrung als praktisch nicht möglich angesehen werden kann.

Das Verfahren von Diffie und Hellmann ermöglicht es, über eine Leitung einen gemeinsamen Schlüssel zu vereinbaren ohne dass die übertragenen Informationen von Dritten verwendet werden können. Allerdings ist es noch nicht sicher genug für unsichere Kanäle. Wenn ein Dritter X die Möglichkeit hat, Nachrichten zu verändern, kann er sich in das Austausch-Protokoll einschalten. Er wählt eine eigene Zufallszahl und führt damit das Protokoll mit den Partnern A und B durch. Dann entstehen sichere Verbindungen zwischen A und X sowie B und X. A und B sind im besten Glauben, ihre Nachrichten direkt aneinander zu senden. In Wirklichkeit erreichen die Nachrichten immer zuerst X, der sie dann nach Belieben manipulieren kann bevor er sie weiter gibt. Eine Möglichkeit sich gegen solche aktiven Angriffe zu schützen, ist die Verwendung fester Schlüssel, die bei einer zuverlässigen Stelle hinterlegt werden.

15.4.5 Öffentliche Schlüssel

Ein praktischer Nachteil bleibt bei dem Verfahren der Schlüsselvereinbarung: beide Partner müssen gleichzeitig aktiv sein, um das Protokoll durchführen zu können. Damit ist das Verfahren beispielsweise zum Versand einer Email nur schlecht einsetzbar. Einfacher zu verwenden sind Verfahren, bei denen der Absender seine Nachricht verschlüsseln kann, ohne immer vorher mit dem Empfänger direkt Informationen austauschen zu müssen. Dies ermöglichen Verfahren mit öffentlichen Schlüssel (public key) [33]. Hierbei stellt der Empfänger einen Schlüssel bereit, der allgemein bekannt gemacht wird. Der Sender verwendet diesen Schlüssel, um seine Nachricht zu verschlüsseln. Der öffentliche Schlüssel hilft nicht zu Entschlüsselung. Benötigt wird vielmehr ein zweiter Schlüssel. Diesen Schlüssel kennt nur der Empfänger, so dass auch nur er die Verschlüsselung aufheben kann. Solange er ihn geheim hält, kann niemand sonst die Nachricht verstehen.

Das bekannteste derartige Verfahren ist RSA, so benannt nach den Entwicklern Ronald Rivest, Adi Shamir und Leonhard Adleman. RSA basiert ebenfalls auf der Modulo-Arithmetik. Der öffentliche Schlüssel besteht aus zwei Zahlen, die wie folgt gewählt werden:

- Der potentielle Empfänger A wählt zwei Primzahlen p und q.
- Aus den beiden Primzahlen berechnet er das Produkt $n = p \cdot q$.
- Weiterhin wählt er eine Zahl $e < n$.

- e und n bilden den öffentlichen Schlüssel und werden entsprechend publiziert.

Damit kann ein Sender eine Nachricht an A nach der folgenden Vorschrift verschlüsseln

$$C = M^e \bmod n \tag{15.3}$$

Die Zahl $M < n$ repräsentiert einen Block mit der Länge $\ln_2 n$ Bit. Eine längere Nachricht muss in entsprechen viele Teile aufgespalten werden. Die resultierende Zahl C ist der geheime Text für diesen Block. Aufgrund der Eigenschaften der Modulo-Funktion ist es sehr schwer, aus C und e wieder auf M zu schließen. Bei hinreichend großen Zahlen n und e kann man die Umkehrung als praktisch unmöglich betrachten.

Damit ist die Verschlüsselung sicher, aber es stellt sich die Frage, wie der Empfänger die Nachricht lesen kann. Hier kommen die beiden wohlweislich geheim gehaltenen Zahlen p und q ins Spiel. Mit diesen beiden Zahlen kann eine weiter Zahl d mit der Eigenschaft

$$e \cdot d = 1 \bmod ((p-1) \cdot (q-1)) \tag{15.4}$$

berechnet werden. Details dazu sind im Anhang A beschrieben. An dieser Stelle genügt es festzustellen, dass die Bestimmung von d relativ einfach ist. Bei dieser Wahl des Produktes $e \cdot d$ gilt weiterhin die Beziehung

$$C^d = M^{e \cdot d} \bmod n = M \tag{15.5}$$

Auch hier sei für Details auf den Anhang A verwiesen. Mit anderen Worten: der Empfänger braucht lediglich einen Exponenten zu berechnen und auf das Ergebnis die Modulo-Funktion anzuwenden, um gemäß

$$M = C^d \bmod n \tag{15.6}$$

den ursprünglich Wert M zu erhalten. Die Sicherheit von RSA beruht auf den beiden Zahlen p und q. Kennt man sie oder zumindest das Produkt $(p-1)\cdot(q-1)$, so ist es leicht daraus d zu bestimmen. Öffentlich bekannt ist aber nur der Wert $n = p \cdot q$. Der Rückschluss von n auf die beiden Faktoren – die so genannte Faktorzerlegung – ist ein enorm zeitaufwändiges Verfahren. Solange niemand eine schnelle Methode zur Suche nach den Primfaktoren findet – ein Problem an dem die Mathematiker seit mehr als 2000 Jahren arbeiten – ist RSA ein sicheres Verfahren. Für sicherheitskritische Anwendungen werden Zahlen größer als 10^{300} eingesetzt. Die weiteren Fortschritte in der Rechengeschwindigkeit können durch Erhöhung von n leicht kompensiert werden.

Der Ausdruck (15.5) weist eine interessante Symmetrie bezüglich e und d auf. In dem Produkt spielt die Reihenfolge keine Rolle. Man kann also durchaus auch mit dem geheimen Schlüssel d verschlüsseln und dann mit dem öffentlichen

Schlüssel e den Vorgang umkehren. Damit ist keine Vertraulichkeit möglich, da jeder Zugang zu dem öffentlichen Schlüssel hat. Aber diese Eigenschaft kann zum Signieren von Nachrichten genutzt werden. Wenn eine Nachricht zu dem öffentlichen Schlüssel passt, muss sie mit dem zugehörigen privaten Schlüssel erzeugt worden sein. Damit kann nur der Besitzers des Schlüssels der Absender sein.

15.5 Anwendungen

Ausgehend von den Algorithmen lassen sich Anwendungen realisieren. Für die Praxis spielen neben den theoretischen Eigenschaften auch Fragen wie der benötigte Rechenaufwand oder die sichere Verwaltung der Schlüssel eine entscheidende Rolle. So hat RSA bei allen Vorzügen den Nachteil eines relativ hohen Rechenbedarfs. Damit ist es nicht praktikabel, alle Nachrichten mit RSA zu schützen. Weiterhin sind legale Aspekte wie z.B. Patentsituation oder Ausfuhrgesetze zu beachten.

15.5.1 PGP

Eine freie, weit verbreitete Software für die Verschlüsselung von Emails ist PGP (Pretty Good Privacy) [34]. Phil Zimmermann entwickelte mit PGP ein einfach zu bedienendes Programm, in dem mehrere Verschlüsselungsverfahren kombiniert werden. Die eigentliche Nachricht wird mit einem aufwandsgünstigen symmetrischen Verfahren wie DES kodiert. Dazu verwendet wird ein einmaliger Schlüssel, der wiederum mit dem öffentlichen Schlüssel des Empfängers geschützt wird. Der Empfänger – und nur der Empfänger – kann den Einmalschlüssel lesen und damit die gesamte Nachricht entschlüsseln.

Der Sender fügt noch eine mit seinem eigenen privaten Schlüssel kodierte Signatur an, die eine Prüfsumme über die Nachricht enthält. Damit kann der Empfänger sicher stellen, dass die Nachricht tatsächlich vom angegebenen Sender stammt und der Text nicht verändert wurde. In PGP integriert ist die Verwaltung der öffentlichen Schlüssel in einem so genannten Key-Ring (Schlüsselbund), wobei die Schlüssel nach ihrer Vertrauenswürdigkeit bewertet werden.

15.5.2 Sichere Transportschicht

Eine flexible Möglichkeit für verschiedenste sicherheitskritische Anwendungen wird durch eine sichere Transportschicht bereit gestellt. Bild 15.3 zeigt diesen Ansatz am Beispiel von HTTP. Ein zusätzliches Protokoll wird zwischen TCP und Anwendungsprotokoll eingeschoben. Das Protokoll verwendet die Funktionalitäten von TCP und ergänzt sie um eine Sicherung. Aus Sicht der Anwendung ändert sich praktisch nichts, außer dass jetzt die Transportschicht sicher

<table>
<tr><td>HTTPS</td><td>Anwendung</td></tr>
<tr><td>Sichere Transportschicht</td><td rowspan="2">Transport</td></tr>
<tr><td>TCP</td></tr>
<tr><td>IP</td><td>Netzwerk</td></tr>
<tr><td>...</td><td></td></tr>
</table>

Abbildung 15.3: Sichere Transportschicht im Protokollstapel

ist. Zur Kennzeichnung hängt man an die Abkürzung des Anwendungsprotokoll den Buchstaben S für *Secure* an (z.B. HTTPS für HTTP Secure).

Der IETF-Standard dazu ist TLS (Transport Layer Security), eine Weiterentwicklung des ursprünglich von der Firma Netscape Communications entwickelten Protokolls SSL (Secure Sockets Layer). Der Ablauf entspricht wieder den bereits besprochenen Grundprinzipien. Der Client benutzt den öffentlichen Schlüssel des Servers, um den einen so genannten session-key zu übermitteln. Dieser session-key ist ein einmaliger Schlüssel, gültig nur für die aktuelle Verbindung, für eine symmetrische Verschlüsselung.

15.6 Übungen

Übung 15.1 *Der folgende Text wurde mit dem beschriebenen, einfachen monoalphabetischen Verfahren verschlüsselt. Können Sie mit der Häufigkeitsanalyse die Verschlüsselung aufheben? Wie lautet das Schlüsselwort?*

```
SRNAZNKERTPNRLMZWZYDMNZYMXNLAVSNBKXPWZCP
NWJVWNRWEZCANWJNOMCAYKNMMNYKWTNWAZLMVTZO
NRWTZWTRWSRNYRLNOZLKOTNEKWSNWSNOZXNORPZW
RMCANMCAORELMLNYYNONSTZOZYYNWUVNHNMCAONR
HLRWSNONOBZNAYKWTSNOTVYSPZNENOGRNSKOCASR
NMNXNLAVSNNRWTNANRXNOUYZWEKNONRWNWTVYSMC
AZLBNWLMCAYKNMMNYLGROS
```

Anhang A

RSA im Detail

A.1 Mathematische Grundlagen

Ausgangspunkt ist die Eulersche[1] φ-Funktion (engl. totient function). Für eine natürliche Zahl n bezeichnet $\varphi(n)$ die Anzahl von Zahlen kleiner n, die teilerfremd zu n sind. Teilerfremd bedeutet, dass zwei Zahlen keinen gemeinsamen ganzzahligen Faktor enthalten. Als Beispiel betrachten wir die Zahl 10 mit den beiden Faktoren 2 und 5. Dann ergibt $\varphi(10) = 4$, da nur die Zahlen 1, 3, 7 und 9 teilerfremd zu 10 sind. Alle anderen Zahlen enthalten ebenfalls einen der beiden Teiler.

Für zwei natürliche Zahlen a und m, die ebenfalls teilerfremd sind, gilt der Satz von Euler (auch als Satz von Euler-Fermat[2] bezeichnet)

$$a^{\varphi(m)} \bmod m = 1 \tag{A.1}$$

wobei die Modulo-Funktion $\bmod$ den Rest bei ganzzahliger Division angibt. Gleichung (A.1) besagt demnach, dass bei der Division

$$a^{\varphi(m)}/m$$

der Rest 1 übrig bleibt. Anders formuliert, es gibt eine natürliche Zahl L, die die Bedingung

$$a^{\varphi(m)} = L \cdot m + 1 \tag{A.2}$$

erfüllt. Der Exponent kann mit einer weiteren natürlichen Zahl k in der Form

$$a^{k \cdot \varphi(m)+1} \bmod m = a \bmod m \tag{A.3}$$

erweitert werden. Die Gleichheit lässt sich mit der Rechenregel

$$(a \cdot b) \bmod n = [(a \bmod n) \cdot (b \bmod n)] \bmod n \tag{A.4}$$

[1] Leonhard Euler, Schweizer Mathematiker 1707-1783
[2] Pierre de Fermat, französischer Mathematiker 1601-1665

leicht zeigen. Ausführlich geschrieben erhält man

$$(a^{\varphi(m)} \cdot \ldots \cdot a^{\varphi(m)} \cdot a) \bmod m \tag{A.5}$$

und nach A.4

$$[(a^{\varphi(m)} \bmod m) \cdot \ldots \cdot (a^{\varphi(m)} \bmod m) \cdot (a \bmod m)]) \bmod m \tag{A.6}$$

Nach dem Satz von Euler sind die Exponenten alle gleich 1. so dass sich in der Tat A.3 auf $a \bmod m$ reduziert.

Der Ausdruck $k \cdot \varphi(m) + 1$ kann als $1 \bmod \varphi(m)$ gelesen werden. Bei der ganzzahligen Division durch $\varphi(m)$ bleibt gerade der Rest 1 übrig. Im RSA-Algorithmus werden daher die Zahlen e und d genau so gewählt, dass sie die Bedingung $e \cdot d = 1 \bmod \varphi(m)$ erfüllen. Damit stellen sich zwei Probleme:

1. Welchen Wert hat $\varphi(m)$?

2. Wie kann bei gegebenen e der Wert von d berechnet werden?

Für beliebige Werte von n ist die Berechnung von $\varphi(n)$ zeitaufwändig. Allerdings gibt es einige Sonderfälle. Insbesondere gilt für jede Primzahl p die Regel $\varphi(p) = p - 1$, denn eine Primzahl enthält per Definition keine ganzzahligen Faktoren. Weiterhin lässt sich zeigen, dass für das Produkt zweier Primzahlen p und q die einfache Beziehung

$$\varphi(p \cdot q) = \varphi(p) \cdot \varphi(q) = (p - 1) \cdot (q - 1) \tag{A.7}$$

gilt. Wenn man also m als Produkt zweier Primzahlen wählt, so reduziert sich die Berechnung von $\varphi(m)$ auf die Multiplikation zweier Zahlen. Bleibt die Bestimmung von d. Es gilt

$$e \cdot d = 1 \bmod \varphi(m) = k \cdot \varphi(m) + 1 \tag{A.8}$$

und damit

$$e \cdot d - k \cdot \varphi(m) = 1 \tag{A.9}$$

Es gilt dann, zwei ganze Zahlen d und k zu finden, die diese Beziehung erfüllen. Die Zahl k hat für RSA keine Bedeutung, aber d ist der gesuchte private Schlüssel. Die Lösung von (A.9) basiert auf dem Euklidischen[3] Algorithmus zur Bestimmung des größten gemeinsamen Teilers (ggT) zweier Zahl. Berechnet wird die größte ganze Zahl, die beide Zahlen ohne Rest teilt. Das Verfahren beruht darauf, dass ein gemeinsamer Teiler d zweier Zahlen x und y auch deren Differenz teilt:

$$(x - y)/d = x/d - y/d = k_x - k_y \tag{A.10}$$

[3]Euklid, griechischer Mathematiker ca. 325-ca. 270 v. Chr.

Da sowohl k_x als auch k_y ganze Zahlen sind, ist auch die Differenz wieder eine ganze Zahl. Um den ggT zu ermitteln, werden im Euklidischen Algorithmus iterativ Differenzen gebildet, wobei immer der kleinere Wert vom größeren abgezogen wird. Das neue Zahlenpaar $x - y, y$ hat aufgrund von (A.10) wieder den gleichen ggT. Irgendwann wird ein Wert 0 erreicht und der verbleibende Wert ist dann der ggt von x und y. Man kann das Verfahren beschleunigen, in dem man den kleineren Wert in einem Schritt so oft möglich abzieht und nur den verbleibenden Rest weiter verwendet. Dies entspricht wiederum einer Modulo-Operation. Eine beispielhafte Implementierung als Methode in Java hat folgende Form:

```
public static int ggt(int x, int y ) {
    if( y > x ) return ggt( y, x);   // x > y sichern
    while( y > 0 ) {
        System.out.println( "x,y = " + x + " " + y);
        int r = x % y;
        x = y;
        y = r;
    }
    return x;
}
```

Mit den Werten 696 und 532 als Beispiel ergibt sich

```
x,y = 696 532
x,y = 532 164
x,y = 164 40
x,y = 40 4
ggT = 4
```

Die Methode berechnet in 5 Iterationen den ggT 4. In jedem Schritt wird der alte Wert von y nach x geschoben. Als neuer Wert für y wird der Rest der Division x/y eingesetzt. Dieser Rest kann damit in der Form $x - k \cdot y$ geschrieben werden, wobei k das Ergebnis der ganzzahligen Division von x durch y ist. Betrachten wir die Folge der entstehenden Restwerte. Der erste Wert ist

$$y_1 = x - k_1 \cdot y$$

Dann berechnet sich der zweite Wert als

$$\begin{aligned} y_2 &= x_1 - k_2 \cdot y_1 \\ &= y - k_2 \cdot (x - k_1 \cdot y) \\ &= -k_2 \cdot x + (1 + k_1 k_2) \cdot y \end{aligned}$$

Der Wert y_2 kann als eine Summe der Form $a \cdot x + b \cdot y$ geschrieben werden. Die Koeffizienten a und b entstehen durch Addition und Multiplikation von ganzen

Zahlen und sind damit selbst auch wiederum ganze Zahlen. Diese Argumentation lässt sich auf die weiteren Restwerte verallgemeinern, so dass insbesondere auch für den letzten Restwert - den ggT - die Darstellung

$$\text{ggT} = a \cdot x + b \cdot y \quad a, b \in \mathcal{Z}$$

resultiert. Damit haben wir den Ansatz zur Lösung der Beziehung (A.9). Wir haben e so gewählt, dass der ggT von e und $\varphi(m)$ gerade den Wert 1 hat. Damit ist sicher gestellt, dass ganzzahlige Lösungen für d und k existieren. Die tatsächlichen Werte lassen sich mit einer Erweiterung des Euklidschen Algorithmus bestimmen. Im Prinzip wird dabei die oben skizzierte Berechnung der Koeffizienten aus den Vorgängerwerten vorgenommen.

Insgesamt ist damit ein einfacher Weg zur Berechnung des privaten Schlüssels eröffnet. Die Funktion $\varphi(m)$ wird als einfaches Produkt zweier Zahlen berechnet und dann kann mit dem erweiterten Euklidschen Algorithmus aus $\varphi(m)$ und dem öffentlichen Schlüssel e der private Schlüssel d berechnet werden.

A.2 Implementierung

Nach der Darstellung der mathematischen Grundlagen sollen noch einige Aspekte der Implementierung betrachtet werden. Dazu dient die in Java realisierte Klasse `RSADemo`. Beim Aufruf kann man die Werte für p, q und e vorgeben. Das Programm berechnet dann das zugehörige d und bestimmt für alle Zeichen den verschlüsselten Wert.

Die Implementierung des erweiterten Euklidschen Algorithmus verwendet eine rekursive Methode `erweiterterEuklid`. Diese Methode ruft sich immer wieder auf, bis der ggT erreicht ist. Dann werden davon ausgehend die Werte von u und v ermittelt.

Eine praktische Schwierigkeit besteht in der Berechnung der Potenzen. Zur Verschlüsselung muss die Potenz

$$C = M^e \bmod n \tag{A.11}$$

berechnet werden. Die direkte Berechnung führt schnell zu einem Überlauf durch zu große Zahlen. Glücklicherweise gibt es eine einfache Methode, den Überlauf zu vermeiden. Betrachten wir dazu die Darstellung

$$M = k \cdot n + r$$

bei der M in ein ganzzahliges Vielfaches von n und den Rest r aufgespalten wird. Dann gilt

$$\begin{aligned} M^e &= M^{e-1} \cdot M \\ &= M^{e-1} \cdot (k \cdot n + r) \\ &= M^{e-1} \cdot kn + M^{e-1} \cdot r \end{aligned}$$

Der erste Summand ist wiederum ein ganzzahliges Vielfaches von n und spielt damit bei der Modulo-Bildung keine Rolle. Nutzt man weiterhin die Beziehung $r = M \bmod n$, so kann man schreiben

$$M^e \bmod n = M^{e-1} \cdot M \bmod n \tag{A.12}$$

Die Berechnung von M^e lässt sich damit auf die Bestimmung von $r = M \bmod n$ und M^{e-1} zurück führen. Die gleiche Argumentation kann auf die Auswertung von M^{e-1} angewandt werden. Insgesamt ergibt sich dann eine Folge von Multiplikationen und Restwertbildungen, bei denen die Zahlenwerte im darstellbaren Bereich bleiben. Die Methode `aHochbModn` stellt eine entsprechende Implementierung dar. Es sei angemerkt, dass die Berechnung wesentlich beschleunigt werden kann, wenn man anstelle der fortgesetzten Multiplikation mit Quadrieren arbeitet. Die Anzahl der benötigten Rechenoperationen wächst dann nur noch linear mit den Anzahl der Bits des Exponenten.

```
public class RSADemo
{
    public static int u, v;

    private static int erweiterterEuklid(int a, int b) {
    int g;
    if (b==0) {
        g=a;
        u=1;
        v=0;
    } else  {
        g = erweiterterEuklid(b, a%b);
        int t=u-a/b*v;
        u=v;
        v=t;
    }
    return g;
    }


    public static int aHochbModn( int a, int b, int n ) {
    int res = a % n;
    for( int i=0; i<b-1; i++ ) {
        res = (res * a) % n;
    }
    return res;
    }
```

```
    public static void main( String[] args) {
    int p = 17;
    int q = 11;
    int e = 9;

        for( int i=0; i<args.length; i++ ) {
            if( args[i].equals("-p") ) {
        p =  Integer.parseInt( args[++i] );
        } else if( args[i].equals("-q") ) {
        q =  Integer.parseInt( args[++i] );
        } else if( args[i].equals("-e") ) {
        e =  Integer.parseInt( args[++i] );
            } else {
                System.out.println("unknown Argument: " + args[i]);
                System.exit(1);
            }
        }

    int n = p*q;
    int ggt = erweiterterEuklid( (p-1)*(q-1), e);
    System.out.println( "ggt = " + ggt);
    if( ggt != 1 ) {
        System.out.print( "e="+e+" und (p-1)*(q-1)="+(p-1)*(q-1) );
        System.out.println( " sind nicht teilerfremd!");
        System.exit(1);
    }
    System.out.println( "u,v: " + u + " " + v);
    int d = v;
    if( d < 0 ) d += (p-1)*(q-1);  // negative Werte korrigieren

    // Schleife über alle Zeichen
    for( int m = 0; m<n; m++ ) {
        System.out.print( "Zeichen = " + m);
        int c = aHochbModn( m, e, n );
        System.out.print( " G = " + c);
        System.out.println( " D = " +  aHochbModn(c,d,n) );
    }
    }
}
```

Anhang B

Endliche Automaten und Maschinen

B.1 Einleitung

Die Realisierung von Netzwerk-Anwendungen stellt eine Reihe von Anforderungen. Zum einen sind Anwendung asynchron. Zum anderen muss man stets mit Infrastruktur-Problemen rechnen. Verschärft wird das Problem durch fehlende Debug-Möglichkeiten. Oft ist es nicht möglich, einen Fehler nachzuvollziehen oder gar zu reproduzieren.

Eine wesentliche Hilfe bei der Entwicklung ist das Konzept von Zustandsautomaten. Im Kapitel 8 ATM hatten wir das Verfahren zur Synchronisation der Zellen mit einem Zustandsdiagramm übersichtlich dargestellt. Dieses Beispiel zeigt die Grundidee: die Anwendung wird durch ein Modell mit einzelnen Zuständen beschrieben. Zu jedem Zeitpunkt befindet sich das System in einem dieser Zustände. Auftretende Ereignisse lösen dann in Abhängigkeit vom aktuellen Zustand Aktionen aus und führen zu einem neuen Zustand.

Mit diesem Konzept lässt sich das Verhalten detailliert und anschaulich modellieren. Gleichzeitig ist die spätere Umsetzung in Programm-Code sehr einfach. Angesichts dieser Vorteile sind in diesem Kapitel die Grundlagen zu endlichen Automaten und Maschinen zusammengestellt. An einem einfachen Beispielen wird die Einfachheit der Umsetzung demonstriert.

B.2 Endliche Automaten

Ein Automat besteht aus einer endlichen Anzahl von Zuständen, die durch Übergänge untereinander verbunden sind. Zu Beginn der Verarbeitung befindet sich der Automat in dem Startzustand. Dann wird das erste Zeichen gelesen. In Abhängigkeit von diesem Zeichen geht der Automat in einen anderen Zustand über. Dieser Vorgang wird für jedes weitere eingegebene Zeichen wiederholt. Die Ana-

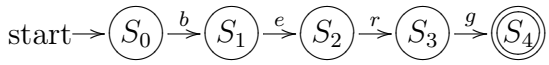


Abbildung B.1: Automat für Suche nach `berg`

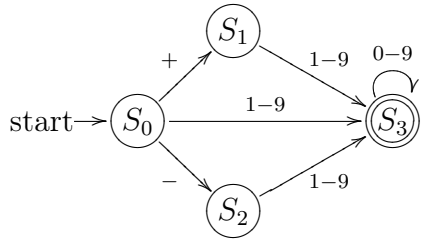
lyse wird abgebrochen, wenn ein Zeichen auftritt, für das im aktuellen Zustand kein weiterer Übergang vorgesehen ist. Ist das Ende der Eingabefolge erreicht, so wird noch geprüft, ob ein als mögliches Ende markierter Zustand erreicht wurde. In diesem Fall gilt die Folge als akzeptiert.

Der einfache Automat in Bild B.1 prüft auf das Wort `berg`. Die Zustände sind als Kreise dargestellt. An den Übergangspfeilen steht jeweils, welches Zeichen zu dazu gehört. Endzustände werden durch einen doppelten Kreis hervorgehoben.

Im Startzustand S_0 ist nur die Eingabe b vorgesehen. Dann erfolgt der Übergang zu dem Zustand S_1. Ansonsten wird die Analyse bereits hier als gescheitert abgebrochen. Entsprechend wird in S_1 nur e erkannt. Dieser Vorgang wird für 4 Eingabezeichen durchgeführt. Bei Übereinstimmung befindet sich der Automat dann im Zustand S_3 – dem einzigen erlaubten Endzustand.

Übung B.1 *Wie kann der Automat erweitert werden, so dass er auch* `burg` *akzeptiert?*

Ein etwas komplizierteres Beispiel ist der folgende Automat für die Darstellung von ganzen Zahlen mit optionalem Vorzeichen:



Bei der Analyse der Folge `+342` ergibt sich folgender Ablauf:

Zustand	Eingabe
S_0	+
S_1	3
S_3	4
S_3	2
S_3	*Ende*

Der Automat befindet sich nach der Verarbeitung der Eingangsfolge in einem Enzustand. Damit ist die Eingabe akzeptiert.

Definition Automat:

Gegeben sei eine Eingabemenge E, eine Zustandsmenge Z, ein Anfangszustand $z_a \in Z$, eine Menge von Endzuständen $F \subseteq Z$ und eine Abbildung $f : E \times Z \to Z$. Dann bildet das 5-Tupel (E, Z, z_a, f, F) einen Automaten A. Für endliche E und Z ist A ein endlicher Automat.

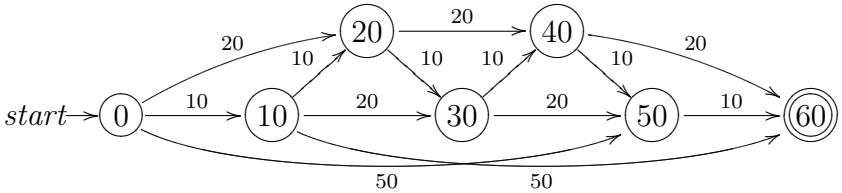
Die Abbildung $f : E \times Z \to Z$, die für jede Kombination von Zustand und Eingangswert den Folgezustand festlegt, kann übersichtlich als Matrix geschrieben werden.

Beispiel B.1 *Zahlen*

- $E = \{+, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $Z = \{S_0, S_1, S_2, S_3\}$
- $z_a = S_0$
- $F = \{S_3\}$
- *Abbildung f:*

	S_0	S_1	S_2	S_3
$+$	S_1			
$-$	S_2			
0				S_3
$1-9$	S_3	S_3	S_3	S_3

Beispiel B.2 *Kaffeautomat:*
Der folgende Automat akzeptiert alle Zahlungen mit 10, 20 und 30 Cent Münzen, die zum Endbetrag von 60 Cent führen.

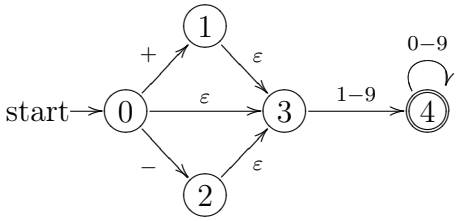


Beispiel B.3 *Implementieren Sie in C einen endlichen Automaten zur Erkennung von Zahlen. Lesen Sie jeweils die nächste Zahl mit* `getch` *ein. Verwenden Sie eine Variable zum Speichern des aktuellen Zustandes. Die verschiedenen Zustände sollen dann als case-Marken einer switch-Auswahl auf dieser Variablen realisiert werden.*

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>

int main(int argc, char* argv[])
{
    char z;
    int  state = 0;

    for( ;; ) {
      z = getch();
      printf("%c %d\n", z, state);
      switch( state ) {
      case 0:
          if( z == '+' ) state = 1;
          else if( z == '-' ) state = 2;
          else if( isdigit( z ) && z != '0' ) state = 3;
          else state = -1;
          break;
      case 1:
          if( isdigit( z ) && z != '0' ) state = 3;
          else state = -1;
          break;
      case 2:
          if( isdigit( z ) && z != '0' ) state = 3;
          else state = -1;
          break;

      case 3:
          if( ! isdigit( z )  ) state = -1;
          break;
      }

      printf("new state: %d\n", state);
      if( state == -1 ) {
          printf("Fehler\n");
          return 0;
      }

    }
}
```

B.2.1 ε-Übergänge

In vielen Fällen sind Übergänge nützlich, bei denen kein Eingangssymbol gelesen wird. Man bezeichnet solche Übergänge als ε-Übergänge. Ein entsprechender Automat für die Darstellung von ganzen Zahlen mit optionalem Vorzeichen hat folgende Form:



Die ε-Übergänge vereinfachen den Aufbau der Automaten. Sie sind aber keine wirkliche Erweiterung der Möglichkeiten. Man kann zeigen, dass sich jeder Automat mit ε-Übergängen in einen gleichwertigen Automat ohne derartige Übergänge umformen lässt.

B.2.2 Endliche Maschinen

Fügt man zu einem Automaten eine Ausgabe hinzu, so erhält man eine Maschine. Man unterscheidet:

- Moore-Maschine: Ausgabe erfolgt im Zustand
- Mealy-Maschine: Ausgabe erfolgt bei Übergängen

Wir betrachten als Beispiel eine Moore-Maschine zur Addition von Binärzahlen.

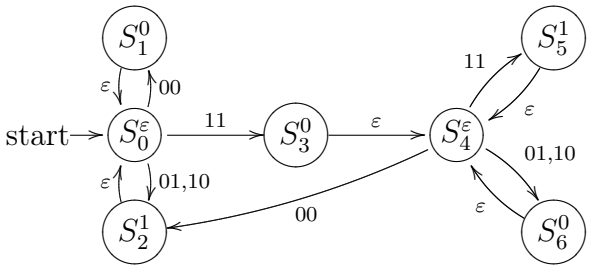
- Eingang: die Werte der beiden Zahlen an einer Stelle. $E = \{00, 01, 10, 11\}$
- Ergebnis der jeweiligen Addition unter Berücksichtigung eines eventuellen Übertrags.

Beispiel:

	0	1	1	0	1	1	0	1
	1	0	0	0	1	1	0	1
Übertrag				1	1		1	
	1	1	1	1	1	0	1	0

- Eingangsfolge: $11, 00, 11, 11, 00, 10, 10, 01$
- Ausgangsfolge: $0, 1, 0, 1, 1, 1, 1, 1$

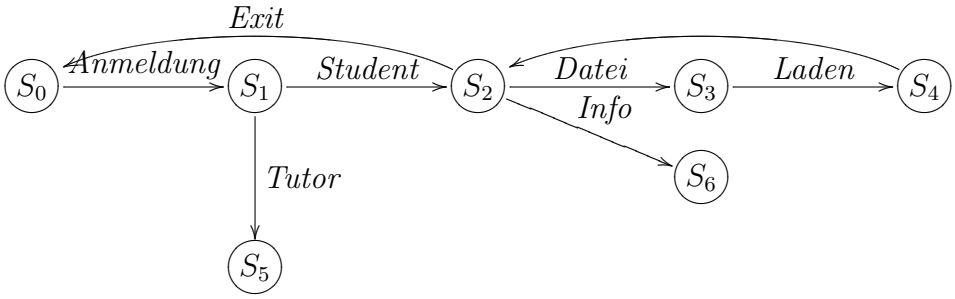
Wir verwenden dabei folgende Notation: S_n^x ist der Zustand n mit der Ausgabe x. Dabei bezeichnet S_n^ε einen Zustand ohne Ausgabe.



Für das Beispiel ergibt sich folgende Zustandsfolge:

$$S_0, S_3, S_4, S_2, S_0, S_3, S_4, S_5, S_4, S_2, S_0, S_2, S_0, S_2, S_0, S_2, S_0$$

Beispiel B.4 *Ein einfacher Automat zur (recht unvollständigen) Modellierung eines Systems zur Abgabe und Kontrolle von Aufgaben:*



Anhang C

Ressourcen

C.1 Tools

C.1.1 Ethereal

Ethereal (`www.ethereal.com/`) ist ein frei erhältlicher Netzwerk-Analysator. Implementierungen sowohl für Unix als auch Windows Betriebssysteme sind verfügbar. Damit können Daten von einer aktiven Netzverbindung oder vorab auf Festplatte protokollierte Daten in verschiedenster Art und Weise dargestellt und untersucht werden. Dabei können gesamte Abläufe wie z.B. der Datenstrom einer TCP-Verbindung oder auch Details eines einzelnen Paketes untersucht werden.

C.1.2 Snort

Ein System zur gezielten Suche nach Einbruchsversuchen ist Snort (www.snort.org). Das System ist frei erhältlich. Ursprünglich für die verschiedenen Unix Betriebssysteme entwickelt, ist es mittlerweile auch für Windows Systeme verfügbar.

C.1.3 OpenH323 Gatekeeper - The GNU Gatekeeper

Einen frei verfügbaren H.323 Gatekeeper für VoIP Anwendungen findet man unter http://www.gnugk.org/. Die Implementierung basiert auf dem OpenH323 Protokollstapel (www.openh323.org). OpenPhone ist ein dazu gehörende Realisierung eines Endgerätes mit graphischer Benutzeroberfläche.

C.2 Nützliche Links

`www.transtec.de/D/d/kompendium` Das IT-Kompendium der Firma transtec, auch in Buchform erhältlich [35].

`www.networks.siemens.de/solutionprovider/_online_lexikon` Das Siemens Online Lexikon „mit ca. 6.000 Begriffserläuterungen, mehr als 10.000 Fachwörtern und zahlreichen Fotos, Grafiken und Tabellen.“

`www.meisermedia.com/glossar/` Das Internet - Glossar der Firma Meisermedia

`www.linktionary.com/index.htm` Startseite für *The Encyclopedia of Networking* mit Online-Erklärungen und Möglichkeiten zum Download einer älteren Ausgabe des Buches von Tom Sheldon [36].

`www.cs.columbia.edu/~hgs/rtp/` Eine umfangreiche Seite zum Thema Realtime Protocol RTP

`www.w3.org` Die Homepage des World Wide Web Consortium (W3C) mit Informationen, Dokumenten, Software u.s.w. rund um das World Wide Web.

`www.mediacrypt.com/indexExplorer.htm` IDEA Verschlüsselung

`theory.lcs.mit.edu/~rivest/homepage.html` Die Homepage von Ronald L. Rivest u.a. mit einer ausführlichen Link-Sammlung zum Thema Kryptographie.

`web.mit.edu/network/pgp.html` MIT Distribution Center for PGP (Pretty Good Privacy)

Anhang D

Abkürzungen

3GPP 3rd Generation Partnership Project

AAL ATM Adaptation Layer

ACF Application Configuration File

ACK Acknowledge. ASCII-Zeichen 6.

ADPCM Adaptive Differentielle Pulscodemodulation

ADSL Asymmetric Digital Subscriber Line

AIMD Additive Increase Multiplicative Decrease

AP Access Point

APNIC Asia Pacific Network Information Centre

ARIN American Registry for Internet Numbers

ARP Address Resolution Protocol

ARPANET Advanced Research Projects Agency Network

AS Autonomes System. In sich abgeschlossenes System mit einheitlichem Routing.

ASCII American Standard Code for Information Interchange

ATM Asynchronous Transfer Mode

BACnet Building Automation Control Network

BGP Border Gateway Protocol. Protokoll für das Routing zwischen autonomen Systemen.

CAN Controller Area Network

CCITT Comité Consultatif International Téléphonique et Télégraphique

CELP Code Excited Linear Predictive Coding

CEPT Conférence Européenne des Administrations des Postes et des Télécommunications

CIDR Classless InterDomain Routing

CGI Common Gateway Interface

CORBA Common Object Request Broker Architecture

CRC Cyclic Redundancy Check

CSS Cascading Style Sheets

CTS Clear To Send

CSMA/CD Carrier Sense Multiple Access with Collision Detection

DEC Digital Equipment Corporation. Computerfirma, entwicklete u.a. PDP und VAX Systeme und das Betriebssystem VMS, später von Compaq übernommen.

DECT Digital European Cordless Telephony. Standard für digitales schnurloses Telefon.

DES Data Encryption Standard

DHCP Dynamic Host Configuration Protocol

DLE Data Link Escape

DMZ De-Militarized Zone

DNS Domain Name System.

DOM Document Object Model

DSL Digital Subscriber Line

DTD Document Type Definition

EIB Europäischer Installationsbus

EIGRP Enhanced Interior Gateway Routing Protocol

ETSI European Telecommunications Standards Institute

ETX End of TeXt. ASCII-Zeichen 3.

FDDI Fiber Distributed Data Interface

FTP File Transport (Transfer) Protocol

GAN Global Area Network

GPRS General Packet Radio Service

GSM Global System for Mobile Communications. Standard für digitales Funkttelefon.

HDLC High Level Data Link Control

HTML HyperText Markup Language

HTTP HyperText Transport Protocol

IAB Internet Architecture Board

IANA Internet Assigned Numbers Authority

ICANN Internet Corporation for Assigned Names and Numbers

ICMP Internet Control Message Protocol

IDEA International Data Encryption Algorithm

IDL Interface Definition Language.

IEEE Institute of Electrical and Electronics Engineers

IETF Internet Engineering Task Force

IESG Internet Engineering Steering Group

IGRP Interior Gateway Routing Protocol

IHL Internet Header Length

IKE Internet Key Exchange

iLBC Internet Low Bit Rate Codec

IM Instant Messenger

IMAP Internet Message Access Protocol

IP Internet Protocol

IPsec IP Security Protokoll

IPTV IP Television

IRC Internet Relay Chat

IRTF Internet Research Task Force

ISDN Integrated Services Digital Network

ISO International Standards Organization

ISO-IEC/JTC1 Joint Technical Committee of the International Organization for Standardization and International Electrotechnical Commission

ISOC Internet SOCiety

ISP Internet Service Provider

ITU International Telecommunications Union

JPEG Joint Photographic Experts Group

L2F Layer 2 Forwarding

L2TP Layer 2 Tunneling Protocol

LACNIC Latin American and Caribbean Internet Addresses Registry

LAN Local Area Network

LCP Link Control Protocol. Protokoll zur Verhandlung der Parameter einer Verbindung mit PPP.

LDAP Lightweight Directory Access Protocol

LLC Logical Link Control

LON Local Operating Network

MACA Multiple Access with Collision Avoidance

MAN Metropolitan Area Network

MAP Manufacturing Automation Protocol

MIB Management Information Base

MIME Multi-Purpose Internet Mail Extension

MPEG Motion Picture Experts Group

MSS Maximum Segment Size. Die maximale Größe eines TCP-Segments.

NAT Network Address Translation

NAK Negative Acknowledge. ASCII-Zeichen 21.

NetBEUI NetBIOS Extended User Inferface

NetBIOS Network Basic Input Output System

NIC Network Information Center

NIST National Institute of Standards and Technology

NNI Network node interface. Netzwerk-Interface bei ATM.

NNTP Network News Transfer Protocol

NRZ Non-Return to Zero

NRZI Non-Return to Zero Inverted

NTP Network Time Protocol

OSI Open Systems Interconnection

PAN Personal Area Network

PARC Palo Alto Research Center. Das Forschungslabor der Firma XEROX in Palo Alto.

PAT Port and Address Translation

PCM Pulscodemodulation

PGP Pretty Good Privacy

POP3 Post Office Protocol Version 3

PPP Point–to–Point Protocol

PPTP Point–to–Point Tunneling Protocol

PTI Payload Type Identifier. Kennung für Zellentyp bei ATM.

PVC Permanent Virtual Circuit. Permanente virtuelle Leitung über Paketvermittlung.

QoS Quality of Service

RARP Reverse Address Resolution Protocol

RFC Request For Comment

RIP Routing Information Protocol

RIPE Réseaux IP Européens

RIPE NCC RIPE Network Coordination Centre

RMI Remote Methode Invocation

ROHC RObust Header Compression

RPC Remote Procedure Call.

RTP Real-Time Transport Protocol

RTCP RTP Control Protocol

RTS Request To Send

RTT Round-Trip Time

SAN Storage / System Area Network

SAP Session Announcement Protocol

SAX Simple API for XML

SDP Session Description Protocol

SEAL Simple Efficient Adaptation Layer. Bezeichnung für AAL-5.

SIP Session Initiation Protocol

SMTP Simple Mail Transfer Protocol

SNMP Simple Network Management Protocol

SOAP Simple Object Access Protocol oder Service Oriented Access Protocol

SSL Secure Sockets Layer

SSRC Synchronization Source. Quelle eines RTP-Stroms.

STP Spanning Tree Protocol

STX Start of TeXt. ASCII-Zeichen 2.

SVC Switched Virtual Circuit. Vermittelte virtuelle Leitung über Paketvermittlung.

SWS Send Window Size. Die maximale Anzahl von ausstehenden Rahmen beim Sliding Window Algorithmus.

TCP Transmission Control Protocol

TFRC TCP-Friendly Rate Control

TFTP Trivial File Transport Protocol

THT Token Hold Time. Die Zeit, für die ein Knoten beim Token-Ring den Token behalten darf.

TLS Transport Layer Security

TOS Type of Service. Spezifikation im IP-Header.

TRT Token Rotation Time. Die Umlaufzeit für den Token bei Token-Ring-Netzen.

TTL Time To Live. Zähler für die maximale Lebensdauer eines IP-Datagramms.

TTP Time Triggered Protocol. Echtzeitfähiger Bus zur Vernetzung im Kfz-Bereich.

UBE Unsolicited Bulk Email. Unverlangte Massen-E-Mails.

UCE Unsolicited Commercial Email. Unverlangte kommerzielle E-Mails.

UDDI Universal Description, Discovery and Integration

UDP User Datagram Protocol

UNI User node interface. Endpunkt-Interface bei ATM.

URI Universal Resource Identifier

URL Universal Resource Locator

URN Universal Resource Name

USB Universal Serial Bus

UUID Universally Unique IDentifier.

VC Virtual Circuit. Virtuelle Leitung über Paketvermittlung.

VCI Virtual Circuit Identifier

VCI Virtual Channel Identifier. Kennung für virtuellen ATM-Kanal.

VPI Virtual Path Identifier. Kennung für virtuellen ATM-Pfad.

VPN Virtual Private Network

VRRP Virtual Router Redundancy Protocol

W3C World Wide Web Consortium

WAN Wide Area Network

WEP Wireless Equivalent Privacy. Ursprünglicher Sicherheitsmechanismus für WLAN.

Wi-Fi Wireless Fidelity.

WLAN Wireless LAN

WMAN Wireless Metropolitan Area Networks

WPA Wireless Protected Access. Verbesserter Sicherheitsmechanismus für WLAN.

WPAN Wireless Personal Area Network

WSDL Web Service Description Language

WWW World Wide Web

XDR eXchange Date Representation

XML EXtensible Markup Language

XSL ExXtensible Stylesheet Language

XSLT XSL Transformation

Anhang E

Ausgewählte Lösungen

Lösung 1.2
In dem folgenden C-Programm wird eine Liste von Rechnern getestet. Dazu wird jeweils über die Funktion `system` der Befehl `ping` aufgerufen. Die Option `-n 1` besagt, dass nur jeweils ein Paket geschickt wird. Die Ausgabe des Aufrufs wird in eine Datei mit dem Namen `tmp.log` umgeleitet. Von dort wird sie wieder von dem Testprogramm gelesen. Dann wird untersucht, ob die 7. Zeile die Zeichenfolge `(0% Verlust` enthält.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

void main(int argc, char* argv[])
{
    FILE *fp;
    char *befehl="ping -n 1 ";
    char *rechner[] = {
        "www.t-online.de",
        "www.gibtsnicht.nix",
        "128.30.52.25",
        "www.vw.com"
    };
    char *nach = " > tmp.log";
    char aufruf[200];
    char zeile[200];
    int i;

    for( i=0; i<(sizeof rechner) / (sizeof (char*)); i++ ) {
        printf("%-20s", rechner[i] );
        sprintf(aufruf, "%s%s%s", befehl, rechner[i], nach );
```

```
        system( aufruf  );
        fp = fopen( "tmp.log", "r" );
        for( int i=0; i<7; i++ ) {
            fgets( zeile, sizeof zeile, fp );
        }
        if( strstr( zeile, "(0% Verlust" ) ) {
            printf("\tOkay\n");
        } else {
            printf("\tFehler\n" );
        }
        fclose( fp );
    }
}
```

Als Ausgabe resultiert

```
www.t-online.de         Okay
www.gibtsnicht.nix      Fehler
128.30.52.25            Okay
www.vw.com              Okay
```

Lösung 2.1

1. $650 \cdot 2^{20} \cdot 8/(75 \cdot 60) = 1,212$ MBit / s
2. $320 \cdot 240 \cdot 8 \cdot 10 = 6,144$ MBit / s
3. $640 \cdot 480 \cdot 24 \cdot 30 = 221,2$ MBit / s
4. Kompressionsfaktoren: 1.) 18,9 2.) 9,6 3.) 3456

Lösung 2.2

Sprache: $t_a = 70000/300000 = 0,233$ s

Bild: $t_a = 5000/300000 = 0,0167$ s

Die Laufzeitdifferenz beträgt $0,216$ Sekunden.

Lösung 2.3

1. $l_{max} = c \cdot t_A = 300000\text{km/s} \cdot 10\text{ms} = 3000$ km
2. $t_U = 1024 \cdot 8/1\text{Mbit/s} = 8,192$ ms
   Damit bleibt für die Ausbreitungsverzögerung $t_A = 10 - 8,192 = 1,808\text{ms}$.
   Schließlich ergibt sich $l_{max} = 300000\text{km/s} \cdot 1,808\text{ms} = 542,4$ km

Lösung 2.4

1. Ausbreitungsverzögerung: $t_a = 385000\text{km}/300000\text{km/s} = 1,283\text{s}$
   Minimale Antwortzeit: $2 \cdot t_a = 2,566\text{s}$

2. Ausbreitungsverzögerung: $t_a = 54 \cdot 10^6\text{km}/300000\text{km/s} = 180\text{s}$
   Minimale Antwortzeit: $2 \cdot t_a = 360\text{s}$

3. Verzögerungs-Bandbreiten Produkt: $1,283\text{s} \cdot 1\text{Mbit/s} = 1,283 \cdot 10^6\text{bit} \approx 157\text{kByte}$

4. $t_U = 2\text{MByte}/1\text{Mbit/s} = 2 \cdot 8 \cdot 2^{20}/10^6 = 16,78\text{s}$
   Mindestdauer: $2 \cdot t_a + t_U = 19,35$ Sekunden.

Lösung 3.1

Querparität	Binär-Darstellung	Zeichen
1	100 0110	F
0	111 0010	r
0	110 1001	i
0	110 0101	e
1	110 0100	d
1	110 0010	b
0	110 0101	e
0	111 0010	r
1	110 0111	g
0	100 1110	Längsparität

Lösung 3.2
Zwei gültige Zeichen unterscheiden sich in mindestens vier Positionen. Demnach beträgt der Hamming-Abstand 4.

Lösung 3.3
Als Rest erhält man die Bitfolge 110.

Lösung 3.5 Die folgende Funktion zeigt eine mögliche Implementierung der Paritätsprüfung für ein Zeichen.

```
// Funktion parity
// Zählt in dem übergebenen Zeichen die Anzahl der Bits mit Wert 1
// Rückgabewert:   1  bei ungerader Anzahl
```

```
//                  0  bei gerader Anzahl
int parity( char zeichen ) {
    int j;
    unsigned int pos;
    int count = 0;

    pos = 1;
    for( j=0; j<8; j++ ) {  /* alle bits */
        if( zeichen & pos ) ++count;
        pos <<= 1;
    }
    return count % 2;
}
```

Abhängig vom Resultat der Prüfung muss das Paritätsbits (höchstwertige Bit) gesetzt werden. Eine entsprechende Anweisung ist:

```
rahmen[i] |= 0x80; // Setzen des Paritätsbits durch
                   // bit-weises ODER mit 1000 0000
```

Schließlich muss der Empfänger die Paritätsbits wieder löschen, um die ursprünglichen Zeichen zu erhalten. Der Programmcode dazu ist:

```
rahmen[i] &= 0x7f; // Löschen des Paritätsbits durch
                   // bit-weises UND mit 0111 1111
```

Aus der gegebenen Bitfehler-Wahrscheinlichkeit kann die Wahrscheinlichkeit für fehlerhafte Rahmen berechnet werden. Am einfachsten betrachte man dazu die Wahrscheinlichkeit für fehlerfreie Rahmen. Damit ein Rahmen fehlerfrei ist, muss jedes einzelne Bit korrekt sein. Mit der Bezeichnung P_{bit} für die Wahrscheinlichkeit eines korrekten Bits ist die Wahrscheinlichkeit, dass N Bit korrekt sind, die Potenz P_{bit}^N. Umgeschrieben auf die Fehlerwahrscheinlichkeiten ergibt sich dann

$$P_{RF} = 1 - (1 - P_{BF})^N$$

wobei P_{BF} die Bitfehler- und P_{RF} die Rahmenfehler-Wahrscheinlichkeit bezeichnen.

Bild E.1 zeigt die Simulationsergebnisse für Rahmen mit 62 Zeichen. Zusätzlich ist der Verlauf der Funktion

$$P_{RF} = 1 - (1 - P_{BF})^{8 \cdot 62}$$

eingetragen.

Lösung 3.6 Ablauf für einen Block:

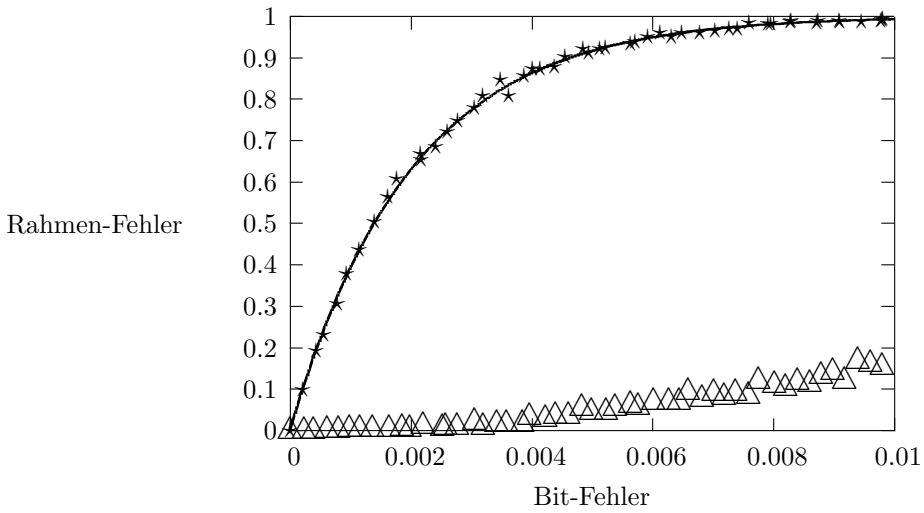


Abbildung E.1: Wahrscheinlichkeit für fehlerhafte Rahmen nach Übertragung bei gegebener Bitfehler-Wahrscheinlichkeit bei Rahmen mit 62 Zeichen. ($\star$ ohne Prüfung, $\triangle$ mit Querparität)

Von Frankfurt bis zum Knoten ($t_U + t_A$)	0,02153 s
Paritätsprüfung	0,00100 s
Versand (t_U)	0,00819 s
Bestätigung ($t_U + t_A$)	0,01334 s
Summe pro Block	0,04406 s

Insgesamt resultiert dann bei $3 \cdot 1024$ Blöcken eine Übertragungsdauer von 135,35 Sekunden.

Lösung 5.2
In einem Store-and-Forward Switch wird ein Rahmen erst nach erfolgter Prüfung weitergeleitet. Damit muss der ganze Rahmen (1000 Bytes Nutzdaten plus 29 Byte Header) abgewartet werden. Bei beispielsweise 100 Mbit/s dauert dies

$$\frac{1029 \cdot 8\text{bit}}{100\text{Mbit/s}} = 823,2\mu s$$

Demgegenüber benötigt ein Cut-Through Switch nur die Zieladresse. Er kann daher bereits nach 14 Byte beziehungsweise

$$\frac{14 \cdot 8\text{bit}}{100\text{Mbit/s}} = 11,2\mu s$$

mit der Weiterleitung beginnen.

Lösung 7.1
Die Segmentnummer ist 32 bit groß. Es können damit 2^{32} Bytes geschickt werden, bis sich die Nummer wiederholt. Dann gilt für die Übertragungsverzögerung

$$t_U = 2^{32} * 8/100/10^6 = 343,6sec = 5,73min$$

Lösung 8.1
Die Datenrate beträgt 64 kbit/s. Bei 48 Byte Nutzdaten pro Zelle folgt damit

$$\frac{64000\text{bit/s}}{48\,\text{cot}\,8\text{bit}} \approx 167\text{Zellen/s}$$

Das Verhältnis zwischen Nutzdaten und Headerdaten beträgt bei ATM stets 48 : 5.

Lösung 9.2
Mit der Funktion `getsockopt` können diverse Optionen abgefragt werden. Die maximale Länge erhält man über den Namen `SO_MAX_MSG_SIZE`.

Literaturverzeichnis

[1] Larry L. Peterson and Bruce S. Davie. *Computernetze*. dpunkt.verlag, Heidelberg, 1. edition, 2000.

[2] Douglas E. Comer. *Computernetzwerke und Internets*. Prentice Hall.

[3] C. Lindemann, C. Immler, and F. Harms. *Internet Intern*. Data Becker, 2000.

[4] Erich Stein. *Taschenbuch Rechnernetze und Internet*. Fachbuchverlag Leipzig im Carl Hanser Verlag, 2001.

[5] Andrew S. Tanenbaum. *Computernetzwerke*. Prentice Hall, 2000.

[6] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in C*. Cambridge University Press, 1992.

[7] M. Schwartz. *Telecommunication Networks: Protocols, Modeling and Analysis*. Addison-Wesley, 1988.

[8] Echelon. *Introduction to the LONWORKS® System 1.0*, 1999.

[9] Y. Yeh, M. G. Hluchyj, and A. S. Acampora. The knockout switch: A simple, modular architecture for high-performance packet switching. *IEEE J. Sel. Areas Commun.*, 5(8):1274 –1283, October 1987.

[10] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jonathan B. Postel, Lawrence G. Roberts, and Stephen S. Wolff. The past and future history of the internet. *Communications of the ACM*, 40(2):102–108, 1997.

[11] Daniel Karrenberg, Gerard Ross, Paul Wilson, and Leslie Nobile. Development of the regional internet registry system. *The Internet Protocol Journal,* `www.cisco.com/warp/public/759/ipj_4-4/ipj_4-4_regional.html`, 2001.

[12] H. Häckelmann, H.J. Petzold, and S. Strahringer. *Kommunikationssysteme: Technik und Anwendung*. Springer, Berlin, 2000.

[13] Emmanuel Chimi. *High-Speed Networking: Konzepte, Technologien, Standards.* Hanser, München Wien, 1998.

[14] Sonia Fahmy, Raj Jain, Shivkumar Kalyanaraman, Rohit Goyal, Bobby Vandalore, and Xiangrong Cai. A survey of protocols and open issues in ATM multipoint communication. OSU Technical Report, Ohio State University, August 21 1997.

[15] W. R. Stevens. *UNIX network programming.* Prentice Hall, Englewood Cliffs, 1990.

[16] L. Wall and R.L. Schwartz. *Programmieren mit Perl.* O'Reilly Verlag, 2001.

[17] Microsoft. *RPC Programmer's Guide and Reference.*

[18] Rolf-Dieter Köhler. *Voice over IP.* mitp-Verlag, Bonn, 2002.

[19] B. Goode. Voive over Internet Protocol (VoIP). *Proc. IEEE*, Vol. 90:1495–1517, 2002.

[20] P. Noll. Speech and audio coding for multimedia communications. In *Proceedings International Cost 254 Workshop on Intelligent Communication Technologies and Applications*, pages 253–263, Neuchatel, 2000.

[21] Stephan Rein, Frank Fitzek, and Martin Reisslein. Voice quality evaluation for wireless transmission with ROHC. In *Proceedings of the 7th IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA)*, pages 461–466, Honolulu, August 2003.

[22] James Clark and Steve DeRose. *XML Path Language (XPath) Version 1.0.* W3C Recommendation, November 1999.

[23] James Clark. *XSL Transformations (XSLT) Version 1.0.* W3C Recommendation, November 1999.

[24] David C. Fallside and Priscilla Walmsley. *XML Schema Part 0: Primer Second Edition.* W3C Recommendation, October 2004.

[25] David Brownell. *SAX2.* O'Reilly, 2002.

[26] Randy J. Ray and Pavel Kulchenko. *Programming Web Services with Perl.* O'Reilly, 2002.

[27] Qusay H. Mahmoud. Accessing and interacting with remote SOAP-enabled services. *Sun Technical Articles and Tips*, 2004.

[28] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. *W3C Note*, 2001.

[29] S. Singh. *Geheime Botschaften.* Carl Hanser Verlag, München, 2000.

[30] National Institute of Standards and Technology (NIST). *FIPS Publication 81: DES Modes of Operation*, December 2, 1980.

[31] X. Lai and J. Massey. A proposal for a new block encryption standard. In I. B. Damgård, editor, *Proc. EUROCRYPT 90*, pages 389–404. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 473.

[32] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22:644–654, November 1976.

[33] W. Diffie. The first ten years of public-key cryptography. *Proc. IEEE*, 76(5):560–577, May 1988.

[34] Philip Zimmermann, Christopher Creutzig, and Andreas Buhl. *PGP - Pretty Good Privacy - Der Briefumschlag für Ihre elektronische Post.* Art d'Ameublement, 1999.

[35] Dieter Weißhaar, editor. *IT-Kompendium - IT-Know-how von A-Z.* Eigenverlag transtec, 2003.

[36] Tom Sheldon. *The Encyclopedia of Networking and Telecommunications.* Osborne/McGraw-Hill, 2001.

Index

Öffentliche Schlüssel, 201
Überlastkontrolle, 79, 83
Übertragungsverzögerung, 7

accept(), 99
Additive Increase, 84
Adobe Flash, 126
Adreßauflösungsprotokoll, 67
Ajax, 127
Aloha, 41
Apache, 121
appendChild(), 168
Applet, 126
ARP-Cache, 67
ARP-Tabelle, 67
ATM Adaptation Layer, 92
ATM-Zelle, 90
Ausbreitungsverzögerung, 7
AXIS-Projekt, 187

Banyan-Netzwerke, 58
Base64 Kodierung, 130
Batcher-Netzwerke, 58
Baud, 22
Baudot, Jean-Maurice-Émile, 23
Best-Effort Prinzip, 61
BGP-Sprecher, 73
bind(), 97
Bluetooth, 48
Border Gateway Protocol, 73
Bridge, 52
ByteFlight, 49

Cache, 125
callback, 171
Cascading Style Sheets, 157
Castor, 173
CDATA, 155
characters(), 171
Chat-Server, 144
CNAME, 135
Common Gateway Interface, 179
connect(), 102
Controller Area Network, 49
Cookie, 127
CORBA, 177
CSMA/CD, 41, 44
Cut-Through, 58

Data Encryption Standard, 197
DatagramSocket, 143
DECT, 136
DefaultHandler, 171
Desktop Firewall, 193
Dienstgüte, 88
Distanzvektor-Routing, 70
Document Object Model, 167
Dokumenttypdefinition, 163
DOM, 167
DOM4J, 173
Domain Name System, 73
DTD, 163

Echokompensator, 90
Empfangsfensters, 32
Empty-Element-Tag, 155
endElement(), 171
Endliche Automaten, 211
Endliche Maschinen, 215
Ethereal, 217
Event handler, 171
exponential backoff, 41

Extranet, 12

Feldbusse, 49
Fiber Distributed Data Interface, 48
Firewire, 49
FlexRay, 49
Flußkontrolle, 79, 83

Gültigkeit, 163
Gatekeeper, 138
Gateway, 52
Gebäudevernetzung, 49
Generator–Polynom, 30
gethostname(), 100
Grenznetz, 193
GSM, 10, 136

H.245, 135
H.323, 135
Häufigkeitsanalyse, 196
Hamming-Abstand, 27
Hotspots, 46
HTML, 153
htons(), 98
Hub, 42, 52
HyperText Markup Language, 122
HyperText Transport Protocol, 121

IBM–Token-Ring, 44
IDEA, 198
IEEE, 39
IEEE 1394, 49
IEEE 802.11, 46
IEEE 802.16, 46
IEEE 802.1D, 54
IEEE 802.4, 48
IEEE 802.5, 44
IETF, 135
Interface Definition Language, 111
Internet Architecture Board, 75
Internet Assigned Numbers Authority, 75
Internet Control Message Protocol, 68
Internet Corporation for Assigned Names and Numbers, 75
Internet Draft, 75
Internet Engineering Steering Group, 75
Internet Engineering Task Force, 75
Internet Society , 75
Internet Standard, 75
Intranet, 12
IP Security Protokoll, 194
ISDN, 10, 87, 136
ITU, 135

JavaScript, 126
JDOM, 173
Jitter, 10

kanonischer Name, 135
Key-Ring, 203
Knockout-Switch, 58
Kollisionsdomäne, 42
Kryptoanalyse, 195
Kryptographie, 195
kumulative Bestätigung, 33

LAN-Emulation, 94
Latenz, 7
Layer 2 Tunneling Protocol, 194
Leitungsvermittlung, 5
Lernende Bridges, 53
Lichtgeschwindigkeit, 7
Link-State-Routing, 72
listen(), 99
Logical Link Control, 40

Management Information Base, 133
Manchester–Kodierung, 22
Manets, 47
Manufacturing Automation Protocol, 48
Mealy-Maschine, 215
MIME, 130
Monoalphabetisch Verschlüsselung, 195
Moore-Maschine, 215

Multi-Purpose Internet Mail Extension, 130
Multiplicative Decrease, 84

Namensraum, 159
Network News Transfer Protocol, 132
Newsserver, 132
Next-Hop-Router, 66
Nyquist-Kriterium, 136

On The Fly, 58
OpenH323 Gatekeeper, 217

Paketfilter, 192
Paketvermittlung, 5
Parsed Character DATA, 163
Parser, 169, 172
Perlmann, R., 54
Personal Firewall, 193
PHP, 127
Piconetz, 12
Ping, 68
Point to Point Tunneling Protocol, 194
Pretty Good Privacy, 203
Profibus, 49
Protokollstapel, 15
Proxy, 126
ProxyTrace, 180
public key, 201
Push-Operation, 80

Quality of Service, 88

Real-Time Transport Protocol, 134
recv(), 100
recvfrom(), 104
Redundanz, 25
Regional Internet Registries, 75
Remote Methode Invocation, 177
Remote Procedure Call, 109
Remote Procedure Calls , 177
Repeater, 42, 52
Request for Comments, 75
Request-Response, 126
Reverse Address Resolution Protocol, 68
RObust Header Compression, 138
Round–Trip Time, 8
Router, 52
Routing, 70
Routing Information Protocol, 72
RSA, 201
RTP Control Protocol, 134

SAX, 167, 170
SCSI, 49
selektive Bestätigung, 32
send(), 100
Sendefenster, 32
sendto(), 104
Sequenznummer, 33
Session Description Protocol, 135
Session Initiation Protocol , 135
Simple API for XML, 167, 170
Simple Netwerk Management Protocol, 133
Slow-Start, 84
Snort, 217
SOAP, 177
SOAPLite, 178
Socket, 95, 141
socket(), 96
SOMAXCONN, 99
Spanning tree, 54
startElement(), 172
Sternkoppler, 42
Store-and-Forward, 57
Stub, 110
Subnetzmaske, 63
Switch, 52

Taktregenerierung, 22
Thread, 131
Time To Live, 65
Token, 44
Token Bus, 48

Tunnel, 194
twisted pair, 42

Universal Resource Identifier, 124
Universal Resource Locator, 122
USB, 49
USENET, 132

Verzögerung–Bandbreite–Produkt, 9
Virtual Circuit Identifier, 55
Virtuelle Leitungen, 55
virtueller Kanal, 89
Voice over IP, 135

W3C, 153
Web Service Description Language, 184
Weiterleitungstabelle, 53
Windows Sockets, 96
WSAStartup(), 96

Xalan, 159
XAMPP, 127
XML, 153
XPath, 158, 161
XSL-FO, 159
XSLT, 158

Zeichenstopfen, 24
Zeitstrahl, 30
Zimmermann, Phil, 203
zweidimensionale Parität, 27