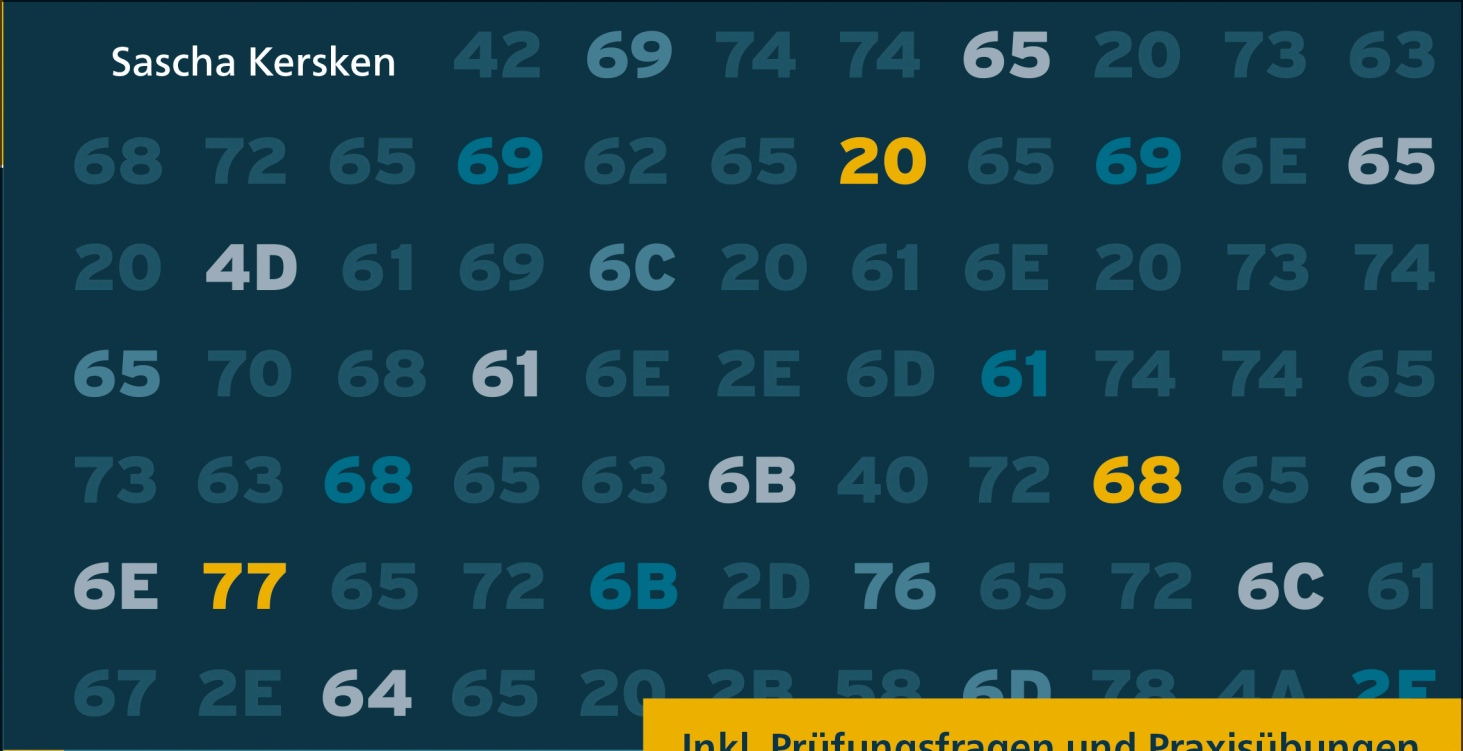
Sascha Kersken



Inkl. Prüfungsfragen und Praxisübungen

IT-HANDBUCH

für Fachinformatiker*innen

Der Ausbildungsbegleiter

- IT-Grundlagen, Netzwerk- und Servertechnik, Programmierung
- Praxisorientiertes Lehr- und Nachschlagewerk
- Alle Fachrichtungen: Anwendungsentwicklung, Systemintegration, Daten- und Prozessanalyse, Digitale Vernetzung

11., aktualisierte und überarbeitete Auflage

Rheinwerk Computing

Kapitel 5
Netzwerkgrundlagen

Jeder wandle für sich und wisse nichts von dem andern.
Wandern nur beide gerad', finden sich beide gewiss.
– Johann Wolfgang Goethe/Friedrich Schiller, Xenien

Internet und lokale Netzwerke haben die Bedeutung des Computers in den letzten zwei bis drei Jahrzehnten revolutioniert. Viele Anwendungsprogramme kooperieren über das Netzwerk miteinander. Der Datenaustausch in und zwischen Unternehmen findet fast ausschließlich per Vernetzung statt, und immer mehr Geschäftsabläufe erfolgen online. Da die Netzwerkfähigkeit zudem eine Grundfunktionalität aller modernen Betriebssysteme geworden ist, steht diese Einführung noch vor dem Kapitel über allgemeine Systemkonzepte.

Nach einer historischen und technischen Einführung erfahren Sie in diesem Kapitel das Wichtigste über gängige Netzwerkhardware; somit wird die Betrachtung der Hardware aus dem vorangegangenen Kapitel hier vervollständigt. Anschließend werden die Netzwerkprotokolle mit dem Hauptaugenmerk auf die seit Jahren dominierenden Internetprotokolle (TCP/IP) beleuchtet.

5.1 Einführung

In diesem Abschnitt erfahren Sie zunächst einmal, was Netzwerke eigentlich sind und was verschiedene Netzwerktypen voneinander unterscheidet. Anschließend wird die Entstehungsgeschichte lokaler Netze, der Datenfernübertragung und des Internets betrachtet.

5.1.1 Was ist ein Netzwerk?

Ein *Netzwerk* (englisch: *Network*, auf Deutsch manchmal auch kurz »Netz«) ist eine Verbindung mehrerer Computer zum Zweck des Datenaustauschs, für verteilte Anwendungen oder auch für die Kommunikation zwischen ihren Benutzer*innen.

Im Lauf der Computergeschichte haben sich viele verschiedene Möglichkeiten der Verkabelung und der Kommunikationsstrukturen sowie zahlreiche Anwendungsgebiete entwickelt:

- Die Verkabelung oder allgemein die Hardwaregrundlage reicht von der Verwendung gewöhnlicher Telefonleitungen mit besonderen Verbindungsgeräten, den DSL-Routern,

über speziell für die Anwendung in lokalen Netzwerken entwickelte Netzwerkkarten und Netzwerkkabel bis hin zu Hochgeschwindigkeitsnetzen, etwa über Glasfaserkabel. Auch die diversen Möglichkeiten der drahtlosen Übertragung werden immer wichtiger.

- Kommunikationsstrukturen, definiert durch sogenannte *Netzwerkprotokolle*, gibt es unzählige. Viele sind von einem bestimmten Hersteller, einer Plattform oder einem Betriebssystem abhängig, andere – wie die Internetprotokollfamilie TCP/IP – sind offen, unabhängig und weit verbreitet. In der Praxis spielt heutzutage nur noch TCP/IP eine wichtige Rolle und wird daher in diesem Kapitel exklusiv behandelt.
- Was die Anwendungsgebiete angeht, reichen diese vom einfachen Dateiaustausch in Arbeitsgruppen über die gemeinsame Nutzung teurer Hard- und Software bis hin zu hochkomplexen, spezialisierten und verteilten Anwendungen.

Paketvermittelte Datenübertragung

Ein wesentliches Merkmal der meisten Netzwerkformen ist die Übertragung von Daten mithilfe sogenannter *Datenpakete*.

Um die *Paketvermittlung* (*Packet Switching*) zu verstehen, sollten Sie zunächst ihr Gegenteil betrachten, die *Schaltkreisvermittlung* (*Circuit Switching*) der herkömmlichen Telefonleitungen. (Hinweis: Inzwischen verwenden Festnetztelefonverbindungen nicht mehr zwingend Circuit Switching; durch die Einführung neuer Technik laufen auch diese hinter den Kulissen immer häufiger paketvermittelt ab – per *Voice over IP*, kurz *VoIP*, sogar bis zum individuellen Telefonanschluss. Mithilfe geeigneter Kommunikationsprotokolle wird aber dafür gesorgt, dass es in der Praxis keine sichtbaren Unterschiede gibt.) Durch das Wählen einer bestimmten Rufnummer (oder früher durch die Handvermittlung) werden bestimmte Schalter geschlossen, die für die gesamte Dauer des Telefongesprächs eine feste Punkt-zu-Punkt-Verbindung zwischen beiden Stellen herstellen. Über diese dauerhafte Leitung können Sprache oder Daten in Echtzeit und in der korrekten Reihenfolge ohne Unterbrechung übertragen werden. Nachdem die Übertragung beendet ist, wird die Verbindung wieder abgebaut, und die betroffenen Leitungen stehen für andere Verbindungen zur Verfügung.

Ganz anders sieht es bei der Paketvermittlung aus: Zu keinem Zeitpunkt der Datenübertragung wird eine direkte Verbindung zwischen den beiden beteiligten Stellen hergestellt. Stattdessen sind beide nur indirekt über ein loses Netz von Vermittlungsstellen, *Router* genannt, miteinander verbunden. Damit auf diesem Weg Daten übertragen werden können, wird folgender Mechanismus verwendet:

- Die Daten werden in kleinere Einheiten unterteilt, die Datenpakete.
- Jedes einzelne Datenpaket wird mit der Absender- und der Empfängeradresse versehen.
- Der Absender übergibt jedes Datenpaket an den nächstgelegenen Router.
- Jeder beteiligte Router versucht, das Paket anhand der Empfängerangabe an den günstigsten Router weiterzuleiten, damit es letztlich an seinem Ziel ankommt.

- Die empfangende Stelle nimmt die Datenpakete entgegen und interpretiert sie je nach Daten- und Übertragungsart auf irgendeine zwischen den beiden Stellen vereinbarte Art und Weise.

Zur reinen Paketvermittlung gehört zunächst einmal kein Mechanismus, der die vollständige Auslieferung aller Datenpakete garantiert. Es wird standardmäßig weder der Erfolg noch das Ausbleiben einer Paketlieferung gemeldet. Im Übrigen wird auch keine verbindliche Reihenfolge festgelegt. Da jedes einzelne Paket einen beliebigen Weg durch das Netzwerk nehmen kann, kommt mitunter ein später abgesendetes Paket noch vor einem früher versandten am Empfangsort an.

Um die potenziell unsichere Datenübertragung per Paketvermittlung für bestimmte Anwendungen zuverlässiger zu machen, wird zusätzlich eine Erfolgskontrolle implementiert. Außerdem werden die Pakete oft durchnummeriert (beziehungsweise mit Informationen über den Byte-Offset und die Größe des jeweiligen Pakets versehen), um die korrekte Reihenfolge wiederherzustellen. Allerdings haben solche Maßnahmen nichts mit der eigentlichen Paketvermittlung zu tun und müssen in diesem Zusammenhang nicht beachtet werden. In der Regel sind die Softwarekomponenten, die sich um die Übertragung der Datenpakete kümmern, gar nicht in der Lage, diese zusätzlichen Kontrollinformationen selbst auszuwerten, da sie aus ihrer Sicht Nutzdaten darstellen.

5.1.2 Entstehung der Netzwerke

Wenn Sie sich die Geschichte der Computer anschauen, die in Kapitel 1, »Einführung«, skizziert wurde, fällt auf, dass die Verwendung von Netzwerken anfangs keinen Sinn ergeben hätte: Bei den frühen Großrechnern gab es keine standardisierte Software, die miteinander hätte kommunizieren können. Darüber hinaus wurden sie zunächst über Schalttafeln und später über Lochkarten bedient. Es gab also keine Echtzeitinteraktion zwischen Menschen und Programmen, sodass es zunächst recht abwegig war, verschiedene Computer miteinander interagieren zu lassen. Frühestens als der Dialogbetrieb über Terminals (siehe Kapitel 4, »Hardware«, und Kapitel 6, »Betriebssysteme«) eingeführt wurde, war an eine Vernetzung zu denken.

Geschichte des Internets

Der Anstoß für die Entwicklung eines Computernetzwerks kam aus einer eher unerwarteten Richtung: Die atomare Bedrohung des Kalten Krieges schürte die Angst der Verantwortlichen in Politik und Militär in den USA, im Fall eines Atomkriegs handlungsunfähig zu werden, weil die Übermittlung von Informationen nicht mehr gewährleistet sein könnte. Es war schlichtweg zu riskant, sich auf einen einzigen Zentralcomputer mit Terminals zu verlassen. Deshalb begann 1969 der Betrieb eines experimentellen Netzes aus vier Computern an ver-

schiedenen US-amerikanischen Universitäten. Federführend für das Projekt war die (Defense Department's) Advanced Research Projects Agency (*ARPA*, später auch *DARPA*), eine Forschungskommission des amerikanischen Verteidigungsministeriums, die 1957 als Reaktion auf den ersten sowjetischen Satelliten Sputnik gegründet worden war. Die USA wollten den Anschluss auf verschiedenen wichtigen Gebieten der Wissenschaft nicht verpassen – und neben der Raumfahrt gehörte auch die Computertechnik zu diesen Gebieten. Folgerichtig hieß dieses erste Netzwerk *ARPANET*.

Allgemein sind bei der Betrachtung von Netzwerken immer mindestens zwei Ebenen zu unterscheiden: zum einen der Anwendungszweck des Netzwerks, zum anderen dessen technische Realisierung. Bei näherem Hinsehen sind sogar noch weitere solcher Ebenen auszumachen; diese sogenannten *Schichtenmodelle* werden in Abschnitt 5.2, »Funktionsebenen von Netzwerken«, besprochen. Interessanterweise stellt sich im Entwicklungsverlauf von Netzwerken manchmal heraus, dass der gewünschte Anwendungszweck technisch anders realisierbar ist, aber auch oft, dass eine bestimmte technische Realisation völlig anderen Anwendungen als der ursprünglich geplanten dienlich sein kann. Besonders in der Geschichte des Internets, dessen Vorläufer das ARPANET war, ist dies häufig festzustellen.

Die ursprüngliche Anwendung dieses Netzes bestand lediglich darin, Datenbestände auf den unterschiedlichen angeschlossenen Computern automatisch zu synchronisieren, also einfach aus Sicherheitsgründen den gleichen Informationsbestand auf mehreren Rechnern bereitzuhalten.[1]

Grundgedanke der Vernetzung selbst war dabei besonders die Fähigkeit jedes beteiligten Computers, Daten, die nicht für ihn selbst bestimmt waren, sinnvoll weiterzuleiten. Daraus ergeben sich zwei unschätzbare organisatorische und technische Vorteile:

- Ein Computer muss nicht direkt mit demjenigen verbunden sein, mit dem er Daten austauschen soll.
- Der Ausfall oder die Überlastung eines bestimmten Verbindungswegs kann durch Alternativen kompensiert werden.

Auf diese Weise konnte das ursprüngliche Ziel, nämlich die Angriffs- und Ausfallsicherheit des Netzes zu gewährleisten, erreicht werden.

Schon unmittelbar nach der Einrichtung des ARPANET begann die eingangs erwähnte Weiterentwicklung. Man stellte schnell fest, dass die technische Infrastruktur dieses Netzes für weit mehr Anwendungen zu nutzen war als das vergleichsweise langweilige automatische Synchronisieren von Datenbeständen. So kam bald eine benutzerorientierte Möglichkeit des Dateiaustauschs hinzu. Außerdem war es schon für gewöhnliche Konfigurationsaufgaben unerlässlich, einem entfernten Computer unmittelbar Anweisungen erteilen zu können.

1 Auch heutige Serversysteme vervielfältigen wichtige Daten auf diese Weise automatisch. Das Verfahren wird *Replikation* genannt und kommt insbesondere bei Datenbank- oder Verzeichnisdienstservern zum Einsatz. In Kapitel 13, »Datenbanken«, wird es am Beispiel von MySQL beschrieben.

Dies war der Ausgangspunkt für die Entwicklung der *Terminal-Emulation*, also der Benutzung des eigenen Terminals für einen Computer, an den es nicht unmittelbar, sondern nur indirekt über das Netzwerk angeschlossen ist. Auch wenn diese Anwendungen noch nicht sofort ihre späteren Namen – FTP und Telnet – erhielten und die technischen Details ihrer Implementierung sich weiterentwickelten, sind sie dennoch nach wie vor wichtige Nutzungsschwerpunkte des Internets.

Alles in allem wurde dieses Netzwerk schnell populär. Zwei Jahre nach seiner Einrichtung, im Jahr 1971, waren bereits 40 Computer an verschiedenen Universitäten und staatlichen Forschungseinrichtungen angeschlossen, und es war bei Weitem nicht nur die militärische Nutzung von Interesse. Auch akademisch hatte das Netz viel zu bieten: Wissenschaftler sind darauf angewiesen, Daten auszutauschen; hier ergab sich eine sehr schnelle und effektive Möglichkeit dazu.

1972 wurde dann der bis dahin bedeutendste Dienst dieses Netzes erfunden: *Ray Tomlinson*, ein Mitarbeiter eines Ingenieurbüros in Kalifornien, verschickte die erste *E-Mail*.[2] Bis heute zählt die E-Mail zu den erfolgreichsten und verbreitetsten Anwendungen des Netzes; sie kann sich nach dem viel jüngeren World Wide Web noch immer auf einem guten zweiten Platz in puncto Beliebtheit von Internetdiensten halten, und es ist auch nicht zu sehen, warum sich das in absehbarer Zeit ändern sollte. Zwar gibt es offensichtliche Probleme wie das massenhafte Aufkommen von Spam (unerwünschten Werbemails) und Phishing (Mails, die Empfänger*innen vorgaukeln, sie seien von bekannten Firmen, und sie zum Beispiel zur Passworteingabe verleiten), aber es gibt weiterhin keine allgemein verbreitete Alternative.

Das ursprüngliche ARPANET wuchs immer weiter. Zudem wurden nach dem gleichen Prinzip andere, ähnliche Netze konstruiert. Dies ist nicht zuletzt der Tatsache zu verdanken, dass alle Schritte, die zur Entwicklung des Netzes beigetragen haben, von Anfang an sorgfältig dokumentiert und der Öffentlichkeit zugänglich gemacht wurden. Dieser Dokumentationsstil ist bis heute beibehalten worden, die entsprechenden Dokumente heißen *RFC* (*Request For Comments*, etwa »Bitte um Kommentare«).

Es gibt bis heute über 9.400 solcher RFC-Dokumente, die alle online zur Verfügung stehen, zum Beispiel unter *https://www.rfc-editor.org/*. Die meisten sind technische Beschreibungen von Entwürfen, Protokollen und Verfahrensweisen; nur wenige (in der Regel mit dem Datum 1. April) nehmen sich nicht ganz so ernst – zum Beispiel RFC 2324, in dem das Protokoll HTCPCP zur Steuerung vernetzter Kaffeemaschinen vorgeschlagen wird, oder RFC 1300, ein nettes Gedicht über Namen und Begriffe, die im Zuge der Computer- und Netzwerkentwicklung ihre ursprüngliche Bedeutung verändert haben.

Alle Personen, Institutionen und Unternehmen, die etwas Entscheidendes zum ARPANET und zum späteren Internet beitrugen, haben dies in solchen Dokumenten erläutert. Das er-

2 Gemeint ist eine Internet-E-Mail im heutigen Sinne mit *Username@Domain*-Adressierung. Andere Formen des elektronischen Briefs – insbesondere zwischen verschiedenen Terminalkonten auf demselben Computer – wurden schon etwas früher eingeführt.

möglicht jeder beliebigen Person oder Firma, die Hard- oder Software herstellt, mit ihren Produkten diese Standards zu unterstützen, denn sie gehören keinem einzelnen Unternehmen und keiner bestimmten Person, und niemand kann den Zugriff darauf beschränken oder Lizenzgebühren fordern – ein entscheidender Grund dafür, dass die Protokolle des Internets heute vom Personal Computer bis zum Großrechner überall dominieren.

In den 1980er-Jahren schließlich wurde der militärisch genutzte Teil des ARPANET als *MilNet* von ihm abgetrennt. Das restliche ARPANET wurde mit dem *NSFNet*, dem Netz der *National Science Foundation*, und einigen anderen Netzwerken zum Internet zusammengeschlossen. Die kommerzielle Nutzung, heute Hauptverwendungsgebiet des Internets, ließ danach aber noch fast 15 Jahre auf sich warten, denn die Anwendungen des Internets waren zwar robust und wenig störanfällig, aber alles andere als benutzerfreundlich. Abgesehen davon waren die ersten Personal Computer, die in der zweiten Hälfte der 1970er-Jahre auftauchten, weder konzeptionell noch von der Leistung her in der Lage, mit den Internetprotokollen etwas anzufangen.

Recht früh wurde dagegen die *Datenfernübertragung* (*DFÜ*), also der Datenaustausch über Telefonleitungen, für Home- und Personal Computer eingeführt. Ab Ende der 1970er-Jahre wurden sogenannte *Akustikkoppler* verwendet: Geräte, die an den Computer angeschlossen wurden und auf die einfach der Telefonhörer gelegt werden musste. Diese langsamen und störanfälligen Apparate wurden bald durch Modems ersetzt, die eine direkte Verbindung zwischen Computer und Telefonleitung zuließen und im Laufe der Jahre allmählich schneller und zuverlässiger wurden. Hauptanwendungsgebiete waren auf der einen Seite die sogenannten *Mailboxen*, also Informations- und Datenangebote für Computer, die eine direkte Telefonverbindung zum Mailboxrechner herstellten. Auf der anderen Seite entstanden in den 1980er-Jahren die meisten kommerziellen *Onlinedienste* wie CompuServe, AOL oder in Deutschland BTX (Vorläufer von T-Online), das zunächst über spezielle Terminals anstelle von PCs mit einer bestimmten Software genutzt wurde.

Die Entwicklung des Internets vom exklusiven Wissenschaftsnetz zum Massenmedium nahm ihren Anfang erst 1989 in der Schweiz am Europäischen Forschungsinstitut für Kernphysik (CERN) in Genf. Dort machte sich der britische Informatiker *Tim Berners-Lee* Gedanken darüber, wie man Netzwerke, besonders das Internet, für den einfachen und effizienten Zugriff auf wissenschaftliche Dokumente nutzen könnte. Ergebnis dieser Arbeit war die Grundidee des *World Wide Web*, eines hypertextbasierten Informationssystems, das die Infrastruktur des Internets zur Datenübermittlung nutzen sollte.

Hypertext ist nichts anderes als Text mit integrierten Querverweisen, die automatisch funktionieren. Mit anderen Worten: Durch Anklicken des Querverweises, der in diesem Zusammenhang *Hyperlink* heißt, stellt der Text selbst – beziehungsweise das System, das diesen darstellt – die Verbindung mit dem verknüpften Dokument her.

Nun war Hypertext 1989 gewiss nichts Neues. Versuche damit reichen zurück bis in die 1950er-Jahre, in Hilfesystemen war er in den 1980er-Jahren bereits Alltag.[3] Neu war nur seine Nutzung über ein Netzwerk, genauer gesagt, über das Internet.

So entstand ein äußerst effektives Informationssystem für die Wissenschaft, über das Beteiligte aus aller Welt ihre Forschungsergebnisse miteinander austauschten. Der Prototyp des World Wide Web umfasste im Einzelnen die folgenden Bestandteile:

- ein spezielles neues Anwendungsprotokoll, das *Hypertext Transfer Protocol* (*HTTP*),
- einen Serverdienst, der in der Lage ist, Anfragen, die in der Sprache des HTTP formuliert sind, auszuliefern,
- eine neu geschaffene Formatierungs- und Beschreibungssprache für solche Hypertext-Dokumente, die *Hypertext Markup Language* (*HTML*) sowie
- ein Anzeigeprogramm für entsprechend formatierte Dokumente, den *Browser*.

1991 wurde das System der Öffentlichkeit vorgestellt. Es wurde praktisch von Anfang an nicht nur zu ernsthaften wissenschaftlichen Zwecken genutzt, sondern allgemein zur Veröffentlichung von Text, Bildern und anderen Informationsformen zu den verschiedensten Themen. Zunächst war die Nutzung des Systems beschränkt auf wissenschaftliches Personal sowie interessierte Studierende. Sie störten sich nicht am mangelnden Komfort der ersten Browser oder den geringen Layoutfähigkeiten der ersten HTML-Versionen. Als jedoch immer mehr Privatpersonen dazukamen – was durch das allmähliche Entstehen kommerzieller Internetprovider und Browser für PC-Betriebssysteme wie Windows oder Mac OS gefördert wurde –, änderte sich dies. Der berühmt gewordene »Browserkrieg« zwischen Netscape und Microsoft um die Jahrtausendwende schuf letztlich Fakten, die niemand für möglich oder auch nur wünschenswert gehalten hätte, die jedoch lange das Wesen des World Wide Web bestimmten.

Zwei Merkmale sind hier besonders wichtig:

- Die Seitenbeschreibungssprache HTML wurde immer mehr für die Definition des Seitenlayouts statt nur für die Struktur genutzt. Für Websites, die ein möglichst großes Publikum erreichen sollen, das weniger technisch interessiert ist als inhaltlich, ist das Layout wichtiger als die Struktur. (Inzwischen kommt für das Layout allerdings praktisch nur noch CSS zum Einsatz, und HTML konzentriert sich wieder – wie ursprünglich beabsichtigt – auf die Dokumentstruktur.)
- Der Anteil kommerzieller Websites am gesamten Bestand wurde immer größer und überwiegt heute bei Weitem; das Angebot im Web ist den Rundfunkmedien wie etwa dem Fernsehen ähnlicher geworden (ganz davon abgesehen, dass praktisch alle Radio- und Fernsehsender Livestreams und Mediatheken über das Internet bereitstellen und dass es

3 Ironischerweise wurden solche Hilfesysteme, zum Beispiel innerhalb von Anwendungsprogrammen, in den 1980ern bis in die 90er-Jahre als »Online-Hilfe« bezeichnet, obwohl sie aus Netzwerkperspektive offline, das heißt auf dem lokalen Computer, gespeichert waren und ausgeführt wurden.

netzexklusive Medien-Streamingdienste wie Netflix oder Disney+ gibt). Während Tim Berners-Lee sich ursprünglich ein Netz vorgestellt hatte, in dem alle Beteiligten Inhalte sowohl anbieten als auch konsumieren sollten, wird das Web heutzutage von vielen weitgehend passiv als Medium genutzt. Erst die kollaborativen *Web-2.0-Tools* wie Blogs, Wikis, soziale Netzwerke und andere kommen Berners-Lees eigentlichen Ideen näher, wobei die gleichzeitig zu beobachtende Kommerzialisierung samt Aufkauf der wichtigsten Sites durch große Medienkonzerne sicherlich nicht in seinem Sinne ist – und natürlich erst recht nicht die Algorithmen der sozialen Medien, die möglichst sensationalistische, kontroverse und allzu oft schlecht recherchierte bis offen gelogene Posts in die Aufmerksamkeit rücken, statt informierte Diskussionen zu fördern.

Lokale Netze

Einen vollkommen anderen Anstoß zur Entwicklung von Netzwerken gab das Aufkommen des sogenannten *Outsourcings* in der Computertechnik, also der Verlagerung der Rechenleistung von einem Zentralcomputer auf den einzelnen Schreibtisch.

Die fortschreitende Ausstattung von Büros mit Personal Computern führte mangels anderer Optionen zunächst zur Blüte des *Turnschuhnetzwerks* (englisch: *Sneakernet*): Menschen liefen mit Datenträgern bewaffnet durch das ganze Gebäude, um Daten miteinander auszutauschen oder zum Beispiel einen speziellen Drucker zu verwenden. Auch zwischen verschiedenen Unternehmen erfreute sich der sogenannte *Datenträgeraustausch* großer Beliebtheit: Die Datensätze von Geschäftsvorfällen wurden auf Disketten oder Magnetbändern zwischen den einzelnen Unternehmen hin und her gereicht.

Lokale Firmennetzwerke wurden zwar bereits Mitte der 1970er-Jahre bei *Xerox* PARC erfunden, aber erst Ende der 1980er-Jahre rückten sie stärker ins allgemeine Interesse. Es war ein Bedürfnis der PC-Anwender*innen, miteinander Daten auszutauschen, einfach deshalb, weil die meisten Vorgänge der Datenverarbeitung in Firmen von mehreren Personen erledigt werden. So entstanden viele verschiedene Arten der Netzwerkhardware. Neben Ethernet mit seinen vielfältigen Varianten gab es beispielsweise Token Ring von IBM, ARCnet oder auch einfache serielle Direktverbindungen zwischen Computern über die sogenannten *Nullmodemkabel*.

Was die Software angeht, wurden die eigentlich nicht dafür geeigneten PC-Betriebssysteme um Netzwerkfähigkeiten erweitert. Hinzu kamen spezielle Betriebssysteme für Server, also solche Rechner, die anderen im Netzwerk verschiedene Ressourcen zur Verfügung stellen. Bekannt sind hier etwa Novell NetWare, IBM OS/2 oder später auch Windows NT Server.

Wenn Sie in diesem Zusammenhang Linux und andere Unix-Varianten vermissen, liegt das daran, dass Unix als PC-Betriebssystem und als Serversystem für PC-Netzwerke erst einige Jahre später populär wurde. Ein gewisses Grundverständnis für Unix ist übrigens unerlässlich, um die Funktionsweise der Internetprotokolle nachvollziehen zu können. Einige Grundlagen dieses Systems werden in Kapitel 6, »Betriebssysteme«, erläutert.

5.2 Funktionsebenen von Netzwerken

Wie bereits in der Einleitung mehrfach angedeutet wurde, besteht ein gewisses Problem beim Verständnis von Netzwerken darin, dass einige sehr verschiedene Aspekte zu ihrem Funktionieren beitragen. Schon ganz zu Beginn haben Sie eine grobe Unterteilung in die drei Ebenen Verkabelung oder allgemein Netzwerkhardware, Kommunikationsstrukturen oder Netzwerkprotokolle und schließlich Anwendungen eines Netzwerks kennengelernt.

Eine so ungenaue Einteilung lässt die grundsätzliche Schwierigkeit erkennen, reicht aber nicht ganz aus, um Netzwerke in all ihren Bestandteilen zu begreifen, und schon gar nicht, um verschiedene Arten von Netzwerken miteinander zu vergleichen. Auch die Tatsache, dass ein und dieselbe Komponente auf einer bestimmten Ebene wahlweise mit mehreren unterschiedlichen Elementen einer anderen Funktionsebene zusammenarbeiten kann, macht die Einteilung noch nicht transparent genug.

5.2.1 Das OSI-Referenzmodell

Um die Ebenen, die ein Netzwerk ausmachen, ganz genau auseinanderhalten zu können, bedient man sich sogenannter *Schichtenmodelle* (*Layer Models*). Das bekannteste und verbreitetste von ihnen ist das OSI-Referenzmodell der internationalen Standardisierungsorganisation ISO. OSI steht für *Open Systems Interconnect*, also etwa »Verbindung zwischen offenen Systemen«. Das Modell wurde 1978 entworfen und besteht aus sieben übereinander angeordneten Schichten, die jeweils einen Aspekt der Netzwerkkommunikation beschreiben. Ganz unten ist die Hardware angesiedelt, ganz oben befindet sich die Anwendung des Netzes.

Überblick

Hier zunächst die Schichten des OSI-Modells im Überblick, die Beschreibung folgt im Anschluss daran:

1. Bit-Übertragungsschicht (*Physical Layer*)
2. Sicherungsschicht (*Data Link Layer*)
3. Vermittlungsschicht (*Network Layer*)
4. Transportschicht (*Transport Layer*)
5. Kommunikationssteuerungsschicht (*Session Layer*)
6. Darstellungsschicht (*Presentation Layer*)
7. Anwendungsschicht (*Application Layer*)

Die Bezeichnung *OSI-Referenzmodell* deutet bereits darauf hin, dass es sich nicht um einen Standard handelt, der konkrete Netzwerkprotokolle definiert. Das OSI-Modell definiert nur die Funktionen der einzelnen Schichten und ist somit ein Schema zur Definition solcher Standards, beispielsweise für die im weiteren Verlauf des Kapitels vorgestellten IEEE-802.3-Standards. Jeder Standard deckt dabei immer nur Teilaspekte des OSI-Modells ab.

Die Bedeutung der einzelnen Schichten des OSI-Modells

1. Die *Bit-Übertragungsschicht* oder auch *physikalische Schicht* beschreibt nur, wie die reine Übertragung der Daten elektrisch beziehungsweise allgemein physikalisch erfolgt. OSI-basierte Netzwerkstandards beschreiben in dieser untersten Schicht die Struktur der Signale. Dazu gehören unter anderem die folgenden Aspekte:
   - zulässiger Amplitudenbereich
   - Versand- und Empfangsmethoden für Bit-Folgen
   - Operationen zur Umwandlung dieser Bit-Folgen in Daten für die nächsthöhere Schicht (und umgekehrt)
   - Verarbeitungsgeschwindigkeit der Bit-Folgen
   - Start- und Stoppsignale
   - Erkennung beziehungsweise Unterscheidung der Signale bei gemeinsam genutzten Medien
   - Übertragungseigenschaften der Medien (Kabel, Lichtwellenleiter, Funk oder Ähnliches)

   Die Medien selbst sowie Netzwerkkarten oder Onboard-Netzwerkchips sind keine Bestandteile der Definitionen auf der ersten Schicht. Die Herstellerfirmen müssen selbst dafür Sorge tragen, dass ihre Produkte den Spezifikationen genügen.
2. Die *Sicherungsschicht* beschreibt alle Vorkehrungen, die dafür sorgen, dass aus den einzelnen zu übertragenden Bits, also dem reinen physikalischen Stromfluss, ein verlässlicher Datenfluss wird. Dazu gehören die beiden Teilaufgaben *Media Access Control* (*MAC*) – die Regelung des Datenverkehrs, wenn mehrere Geräte denselben Kanal verwenden – sowie *Logical Link Control* (*LLC*), bei der es um die Herstellung und Aufrechterhaltung von Verbindungen zwischen den Geräten geht.

   Viele Protokolle dieser Schicht implementieren eine Fehlerkontrolle, bei Ethernet wird zum Beispiel das einfache Prüfsummenverfahren *Cyclic Redundancy Check* (*CRC*) verwendet. In manchen Fällen wird auch *Quality of Service* (*QoS*), eine Art Prioritätsinformation, genutzt. In der Sicherungsschicht werden die Bit-Folgen in Einheiten einer bestimmten Größe unterteilt und mit einem Header aus Metainformationen versehen. Je nach Standard werden auf dieser Ebene unterschiedliche Namen für diese Datenpakete verwendet. Bei Ethernet und Token Ring ist beispielsweise von *Frames* die Rede, bei ATM von *Zellen*. Der Payload (Nutzdateninhalt) eines Frames beziehungsweise einer Zelle beginnt in aller Regel mit dem Header eines hineinverschachtelten Pakets der nächsthöheren Schicht. Es kann aber auch vorkommen, dass Pakete verschiedener Protokolle der zweiten Schicht ineinander verschachtelt werden. Dies ist zum Beispiel bei *PPP over Ethernet* (Betrieb des klassischen Modem-Einwahlprotokolls über Ethernet, vor allem für DSL) der Fall.
3. Die *Netzwerkschicht* oder *Vermittlungsschicht* definiert diejenigen Komponenten und Protokolle des Netzwerks, die an der indirekten Verbindung von Computern beteiligt

sind. Hier ist sogenanntes *Routing* erforderlich, das Weiterleiten von Daten in andere logische oder auch physikalisch inkompatible Netzwerke. So gehören zum Beispiel alle diejenigen Protokolle zur Netzwerkschicht, die die logischen Computeradressen der höheren Schichten in die physikalischen Adressen umsetzen, bei Ethernet zum Beispiel ARP. Auch auf der Netzwerkschicht werden die Daten in Pakete unterteilt, deren Namen sich je nach konkretem Protokoll unterscheiden. Das mit Abstand verbreitetste Protokoll dieser Ebene, das im weiteren Verlauf des Kapitels noch ausführlich vorgestellte IP-Protokoll, bezeichnet sie als *IP-Datagramme*.

4. Die Protokolle der *Transportschicht* lassen sich in verbindungsorientierte Protokolle wie TCP und verbindungslose Protokolle wie etwa UDP unterteilen. Auf dieser Schicht werden vielfältige Aufgaben erledigt. Ein wichtiger Aspekt sind Multiplexmechanismen, die die Anbindung der Datenpakete an konkrete Prozesse auf den kommunizierenden Rechnern ermöglichen, bei TCP und UDP beispielsweise über Portnummern, bei SPX über Connection-IDs. Verbindungsorientierte Transportprotokolle wie TCP oder SPX sind zudem meist mit einer Fluss- und Fehlerkontrolle ausgestattet, um zu gewährleisten, dass Pakete vollständig am Ziel ankommen und dort in der richtigen Reihenfolge verarbeitet werden. Auch auf der vierten Schicht verwenden verschiedene Protokolle jeweils eigene Bezeichnungen für die Datenpakete; so ist etwa von *UDP-Datagrammen*, *TCP-Sequenzen* und *SPX-Paketen* die Rede.
5. Die *Kommunikationssteuerungsschicht* oder *Sitzungsschicht* stellt die kontinuierliche, das heißt einen Kontext wahrende Kommunikation zwischen kooperierenden Anwendungen oder Prozessen auf verschiedenen Rechnern sicher.
6. Die *Darstellungs-* oder *Präsentationsschicht* dient der Konvertierung und Übertragung von Datenformaten, Zeichensätzen, grafischen Anweisungen und Dateidiensten.
7. Die *Anwendungsschicht* schließlich definiert die unmittelbare Kommunikation zwischen den Benutzeroberflächen der Anwendungsprogramme, kümmert sich also um die Verwendung derjenigen Dienste über das Netzwerk, die Menschen unmittelbar zu Gesicht bekommen.[4]

Da das OSI-Modell eine Zusammenstellung von möglichen Fähigkeiten für viele verschiedene Arten von Netzwerken darstellt, kann es natürlich vorkommen, dass eine Schicht in einem bestimmten Netzwerk wichtiger ist als eine andere oder dass zum Beispiel ein Protokoll Funktionen zweier Schichten abdeckt oder auch nur eine Teilfunktion einer Schicht erbringt. Um diese Umstände deutlich zu machen, wendet man häufig anders aufgeteilte Schichtenmodelle mit meist weniger, selten mehr Schichten an – dies gerade dann, wenn es um die konkrete Beschreibung einer bestimmten Art von Netzwerk geht.

4 Halb scherzhaft werden die Personen, die vernetzte Rechner nutzen, manchmal als *achte Schicht* bezeichnet, und Probleme, die durch fehlerhafte Benutzung entstehen, nennt man entsprechend *Layer-8-Fehler*.

5.2.2 Das Schichtenmodell der Internetprotokolle

Im Bereich der TCP/IP-Netzwerkprotokolle, die unter dem Betriebssystem Unix und im Internet den Standard darstellen, wird zum Beispiel häufig ein Modell aus nur vier Schichten verwendet. Dies wird dem Wesen dieser Protokolle auch wesentlich eher gerecht als das OSI-Modell, denn die Internetprotokolle sind bereits einige Jahre älter als OSI. Es handelt sich um das Schichtenmodell des ursprünglichen ARPANET, das vom US-Verteidigungsministerium finanziert wurde. Deshalb wird es meist als *DoD-Modell* (*Department of Defense*), manchmal auch als *DDN-Modell* (*Department of Defense Network*) bezeichnet.

Überblick

Die vier Schichten bei TCP/IP-Netzwerken nach dem DDN Standard Protocol Handbook sind:

1. Netzzugangsschicht (*Network Access Layer* oder *Link Layer*)
2. Internetschicht (*Internet Layer*)
3. Host-zu-Host-Transportschicht (*Host-to-Host Transport Layer* oder einfach *Transport Layer*)
4. Anwendungsschicht (*Application Layer*)

Diese vier Schichten sind den konkreten Gegebenheiten von TCP/IP-Netzwerken angepasst, bei denen es zum Beispiel nur theoretisch möglich ist, von einer separaten Sitzungsschicht zu sprechen.

Das OSI-Referenzmodell kann mit dem DDN-Schichtenmodell deshalb nur grob in Beziehung gesetzt werden. Tabelle 5.1 zeigt, wie ein solcher Vergleich ungefähr aussehen könnte.

OSI-Modell	DDN-Modell
7. Anwendungsschicht	4. Anwendungsschicht
6. Darstellungsschicht	
5. Sitzungsschicht	
4. Transportschicht	3. Host-zu-Host-Transportschicht
3. Vermittlungsschicht	2. Internetschicht
2. Sicherungsschicht	1. Netzzugangsschicht
1. Bit-Übertragungsschicht	

Tabelle 5.1 Vergleich zwischen dem OSI-Referenzmodell und dem DDN-Schichtenmodell der Internetprotokolle

Die Bedeutung der Schichten des DDN-Modells

1. Die Netzzugangsschicht (*Network Access Layer* oder *Link Layer*) beschreibt, wie die physikalische Datenübertragung erfolgt. Die Aufgaben, die auf dieser Schicht zu erledigen sind,

werden durch viele recht unterschiedliche Protokolle erbracht, einfach deshalb, weil es kaum eine Sorte von Netzwerkhardware gibt, auf der die Internetprotokolle noch nicht implementiert worden wären. Die eigentlichen Kernprotokolle, zu denen besonders die Namensgeber der Protokollfamilie gehören – also das *Transmission Control Protocol* (TCP) und das *Internet Protocol* (IP) –, kümmern sich überhaupt nicht um die physikalischen Verhältnisse. Damit das möglich ist, müssen auf dieser untersten Schicht die Bit-Übertragung und die Transportsicherung zuverlässig zur Verfügung gestellt werden.

Auf diese Weise entspricht die Netzzugangsschicht der Internetprotokolle den beiden untersten Schichten von OSI.

2. Die Internetschicht (*Internet Layer*), die im Wesentlichen der Vermittlungsschicht des OSI-Modells ähnelt, kümmert sich um die logische Adressierung der Rechner im Netz, durch die die grundsätzliche Identifizierbarkeit des jeweiligen Rechners sichergestellt wird. Eine weitere wichtige Aufgabe auf dieser Ebene ist das Routing, also die Weiterleitung von Daten über verschiedene physikalisch und/oder logisch getrennte Netze hinweg. Grundlage dieser Tätigkeiten ist das *IP-Protokoll* (*Internet Protocol*). Es definiert die IP-Adressen, 32 (in einer neueren Version 128) Bit breite Nummern, die den einzelnen Rechnern zugewiesen werden und die der Unterscheidung der einzelnen Netzwerke und der Rechner in diesen Netzen dienen. Außerdem versieht es jedes Datenpaket mit einem Header, also einer Zusatzinformation, die insbesondere die Quelladresse des sendenden Rechners und die Zieladresse des empfangenden Hosts enthält. Ein Datenpaket dieser Ebene wird als *Datagramm* bezeichnet.
3. Die Host-zu-Host-Transportschicht (*Host-to-Host Transport Layer*) kümmert sich um den zuverlässigen Datenaustausch zwischen den kommunizierenden Rechnern. Im Wesentlichen sind hier zwei verschiedene Protokolle verantwortlich (neben anderen, selten verwendeten). Diese beiden Protokolle werden niemals gleichzeitig, sondern immer alternativ verwendet. Das einfachere und weniger robuste *UDP* (*User Datagram Protocol*) stellt einen schlichten und wenig datenintensiven Mechanismus zur Verfügung, der die direkte Nutzung der IP-Datagramme für die Host-zu-Host-Kommunikation erlaubt. Dabei wird keine virtuelle Verbindung zwischen den beiden Rechnern hergestellt; es findet also keine Kontrolle über einen kontinuierlichen Datenstrom statt. Das erheblich komplexere *TCP* (*Transmission Control Protocol*) hat zwar einen deutlich größeren Overhead (Datenmehraufwand durch Verwaltungsinformationen) als UDP, stellt aber dafür einen zuverlässigen Transportdienst dar: Es wird eine virtuelle Verbindung zwischen den beiden Hosts hergestellt. Diese besteht darin, dass über Länge und Byte-Offset jedes Datenpakets Buch geführt wird (was einer Nummerierung der Pakete gleichkommt) und eine Übertragungskontrolle sowie eine eventuelle Neuübertragung jedes einzelnen Pakets stattfinden. Ob eine Anwendung nun UDP oder TCP verwendet, ist ihre eigene Entscheidung. Allgemein benutzen Dienste, die kontinuierlich größere Datenmengen transportieren müssen, eher TCP, während etwa Verwaltungs- und Konfigurationsdienste zu UDP tendieren.

Im Vergleich zum OSI-Modell nimmt die Host-zu-Host-Transportschicht insbesondere die Aufgaben der OSI-Transportschicht wahr; je nach konkretem Protokoll können auch Funktionen der Sitzungsschicht ausgemacht werden.

Der Begriff *Host* (Gastgeber) bezeichnet übrigens jeden Computer, der an ein Netzwerk angeschlossen ist und mit anderen Geräten kommuniziert. Es ist keine Bezeichnung für einen expliziten Dienstleistungsrechner, dieser (oder vielmehr die darauf ausgeführte Software) wird *Server* genannt. Der Host muss lediglich vom Router abgegrenzt werden, der Pakete nicht für sich selbst entgegennimmt, sondern an andere Netze weiterleitet. Da das Routing jedoch eine Ebene weiter unten stattfindet, ist es ein auf dieser Schicht unsichtbares Detail – die Transportschicht ist nur für Rechner relevant, die Daten für den Eigenbedarf benötigen.[5]

4. Die Anwendungsschicht (*Application Layer*) schließlich definiert die Kommunikation zwischen den Anwendungsprogrammen auf den einzelnen Rechnern; hier arbeiten Protokolle wie HTTP für Webserver, FTP zur Dateiübertragung oder SMTP für den E-Mail-Versand. Die Schicht entspricht im Wesentlichen der gleichnamigen obersten Schicht des OSI-Referenzmodells, wobei auch einige Komponenten von dessen Darstellungsschicht mit hineinspielen. Beispielsweise bedarf HTML-Code, der von einer Webserveranwendung ausgeliefert wird, der Interpretation durch einen Browser; hier entspräche der HTML-Code selbst eher der Darstellungsschicht, die Browseranwendung aber der OSI-Anwendungsschicht. Sitzungsmanagement ist dagegen vom ursprünglichen Design her gar nicht vorgesehen; falls es benötigt wird, muss es durch die Anwendungen selbst bereitgestellt werden. In Kapitel 19, »Webserveranwendungen«, erfahren Sie beispielsweise, wie Sie mithilfe der Programmiersprache PHP Websessions verwalten.

5.2.3 Netzwerkkommunikation über die Schichten eines Schichtenmodells

In diesem Abschnitt wird erläutert, wie die Kommunikation über die Schichten von Schichtenmodellen funktioniert. Dazu werden zwei Beispiele gegeben: Das erste ist ein Alltagsbeispiel, das mit Computernetzwerken nichts zu tun hat, während das zweite ein einfaches Beispiel der Netzwerkkommunikation darstellt.

Ein Alltagsbeispiel

Neben der Datenübertragung im Netzwerk lassen sich auch völlig andere Arten der Kommunikation in Schichten gliedern. Beispielsweise kann die Kommunikation zwischen zwei Personen am Telefon folgendermaßen unterteilt werden:

5 Router verständigen sich allerdings auch mithilfe der im weiteren Verlauf dieses Kapitels vorgestellten Routing-Protokolle miteinander; diese Protokolle werden durchaus auf der Host-zu-Host-Transportschicht ausgeführt. Trotzdem ist dieses technische Detail für die eigentlichen Anwendungshosts uninteressant und unsichtbar.

1. Die beiden Telefone sind physikalisch über eine Telefonleitung miteinander verbunden.
2. Die Verbindung zwischen den Telefonanschlüssen kommt dadurch zustande, dass eine der beiden Personen die eindeutige Nummer der anderen wählt und die andere das Gespräch annimmt.
3. Über die Telefonleitung werden Informationen in Form von elektromagnetischen Impulsen übertragen, beim klassischen Telefonnetz analog, bei ISDN, Mobilfunk oder VoIP dagegen digital.
4. An den beiden Endpunkten der Kommunikation sprechen die Beteiligten in ihre jeweilige Sprechmuschel hinein; die akustischen Signale werden in elektromagnetische Impulse umgewandelt (bei den digitalen Varianten kommt noch die Analog-Digital-Wandlung hinzu). Umgekehrt hören die Beteiligten aus der Hörmuschel wiederum akustische Signale, die aus den übertragenen Impulsen zurückverwandelt wurden.
5. Die akustischen Signale, die die Personen miteinander austauschen, werden zu Silben, Wörtern und schließlich Sätzen kombiniert.
6. Aus den einzelnen Bestandteilen der Sprache ergibt sich schließlich der eigentliche Inhalt der Nachrichten, die miteinander ausgetauscht werden.

Möglicherweise besteht dieses Kommunikationsmodell sogar aus noch mehr Schichten: Angenommen, die beiden Personen wären leitende Angestellte oder gar Vorstandsmitglieder eines großen Unternehmens. Dann wird in aller Regel auf beiden Seiten ein Sekretariat die Gesprächsvermittlung vornehmen, möglicherweise ist sogar noch eine Telefonzentrale involviert. Noch komplizierter wird es, wenn etwa Dolmetscher*innen[6] mitwirken.

Dieses einfache Beispiel zeigt deutlich, dass alle Ebenen, die sich oberhalb der untersten, also der physikalischen Ebene befinden, nur Abstraktionen darstellen, durch die den übertragenen Signalen jeweils ein neuer Sinnzusammenhang zugeordnet wird. Die eigentliche Verbindung erfolgt nämlich stets nur auf dieser untersten Ebene! Für jedes Schichtenmodell gilt daher zusammenfassend Folgendes:

Die Daten, die über einen Kommunikationskanal übertragen werden sollen, werden auf der Seite des Senders zunächst Schicht für Schicht nach unten weitergereicht und jeweils mit den spezifischen Zusatzinformationen für diese Schicht versehen. Schließlich werden sie über die unterste Schicht, die eigentliche physikalische Verbindung, übertragen. Auf der Empfängerseite werden sie dann wieder schichtweise nach oben weitergeleitet. Jede Schicht ermittelt die für sie bestimmten Informationen und regelt die Weiterleitung an die nächsthöhere Schicht.

Was in dem Telefonbeispiel geschieht, wird schematisch in Abbildung 5.1 dargestellt: Während die beteiligten Personen den Eindruck haben, in einem direkten Gespräch miteinander

6 Allmählich werden KI-basierte Systeme häufiger, bei denen die Simultanübersetzung nicht von Menschen, sondern von Software durchgeführt wird.

zu kommunizieren, geschieht in Wirklichkeit etwas erheblich Komplexeres: Die gesprochene Sprache (die eigentlich aus Silben beziehungsweise einzelnen Lauten besteht, die letztlich einfach nur Schallwellen sind) wird in eine andere Form von Information umgewandelt, über eine elektrische Leitung übertragen und auf der Empfängerseite wieder zusammengesetzt.

Wichtig ist außerdem, dass jede Schicht immer nur die für sie selbst bestimmten Informationen auswertet. Beim Empfang von Daten haben die niedrigeren Schichten überhaupt erst dafür gesorgt, dass die Informationen auf der entsprechenden Schicht angekommen sind, die Spezialinformationen der höheren Schichten sind der aktuellen Schicht dagegen unbekannt. Sie muss lediglich anhand ihrer eigenen Informationen dafür sorgen, dass die Daten an die korrekte Stelle einer höheren Schicht ausgeliefert werden. Auf diese Weise ist jede Schicht virtuell mit der Schicht der gleichen Stufe auf der anderen Seite verbunden; das tatsächliche Zustandekommen dieser Verbindung ist für die jeweilige Schicht dagegen absolut unsichtbar.

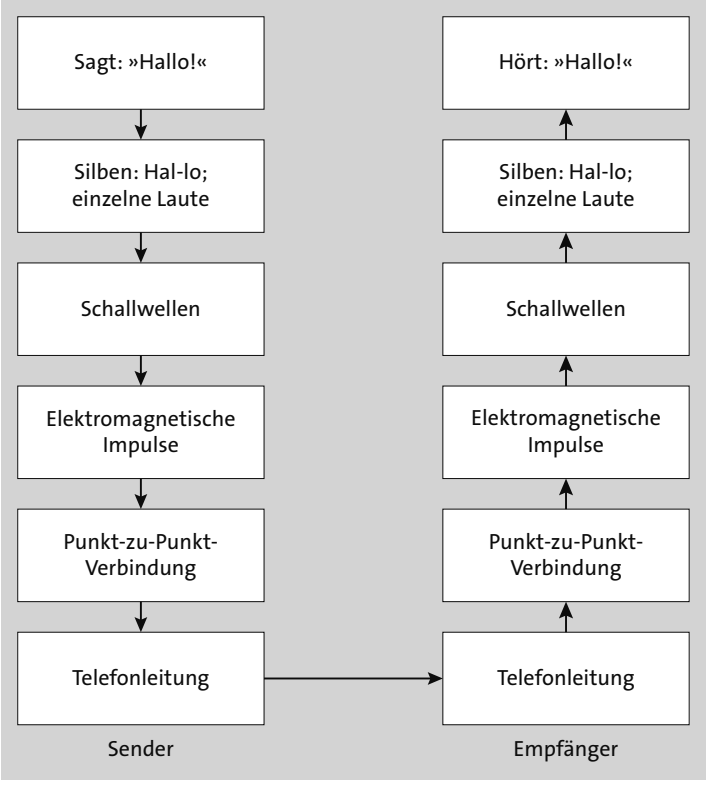


Abbildung 5.1 Schichtenmodell eines Telefongesprächs. Die tatsächliche Verbindung besteht nur auf der Ebene der Telefonleitung!

Ein Netzwerkbeispiel

Dieses zweite Beispiel – der Versand einer E-Mail an den Rheinwerk Verlag – zeigt, wie sich die beim Telefonbeispiel erläuterten Sachverhalte wieder auf Netzwerke übertragen lassen. Schematisch geschieht Folgendes:

1. In meinem E-Mail-Programm, zum Beispiel Mozilla Thunderbird, verfasse ich den eigentlichen Inhalt der Mail, als Empfängeradresse setze ich *info@rheinwerk-verlag.de* ein. Nachdem ich alles fertig geschrieben habe, drücke ich auf den Absendebutton.
2. Da eine E-Mail eine in sich geschlossene Dateneinheit darstellt, die in ihrer ursprünglichen Reihenfolge beim Empfänger ankommen muss, wird der Transport durch das TCP-Protokoll übernommen, dessen eingebaute Datenflusskontrolle dafür sorgt, dass alle Daten vollständig und in der richtigen Reihenfolge übertragen werden.
3. Die Datenpakete, die durch das TCP-Protokoll angelegt wurden, werden nun durch das IP-Protokoll mit der korrekten Absender- und Empfängeradresse versehen. Diese Adressen haben nichts mit den nur auf der Anwendungsebene wichtigen E-Mail-Adressen zu tun. Vielmehr geht es darum, dass mein Rechner die Daten an den zuständigen Mailserver beziehungsweise an einen Vermittlungsrechner versendet.
4. Die fertig adressierten Datenpakete werden jetzt dem eigentlichen physikalischen Netzwerk anvertraut und entsprechend übertragen.

Auf der Empfängerseite – also auf dem Serverrechner, der das Postfach *info@rheinwerk-verlag.de* verwaltet – kommen die Daten dann folgendermaßen an:

1. Über die physikalische Netzwerkverbindung des Serverrechners treffen Datenpakete ein.
2. Auf der Ebene des IP-Protokolls werden die Datenpakete nach den zuständigen Transportdiensten sortiert und an diese weitergereicht – die E-Mail wird dem TCP-Dienst übergeben.
3. Der TCP-Dienst stellt fest, dass die entsprechenden Datenpakete für den Mailserver (gemeint ist das Programm, nicht der Rechner selbst) bestimmt sind, und reicht sie an diesen weiter.
4. Der Mailserver wertet die E-Mail-Adresse der Empfängerseite aus und speichert die Mail in dem Postfach *info@rheinwerk-verlag.de*. Dort kann sie jederzeit von berechtigten Personen abgeholt werden.

Die Übertragung der E-Mail vom empfangenden Server an das E-Mail-Programm des eigentlichen Empfängers funktioniert im Großen und Ganzen genauso, obwohl auf der Anwendungsebene ein anderes Protokoll zum Einsatz kommt.

Abbildung 5.2 zeigt noch einmal schematisch, wie die Übertragung funktioniert. Mehr über die grundlegenden E-Mail-Protokolle erfahren Sie in Abschnitt 5.6.5, »Verschiedene Internetanwendungsprotokolle«.

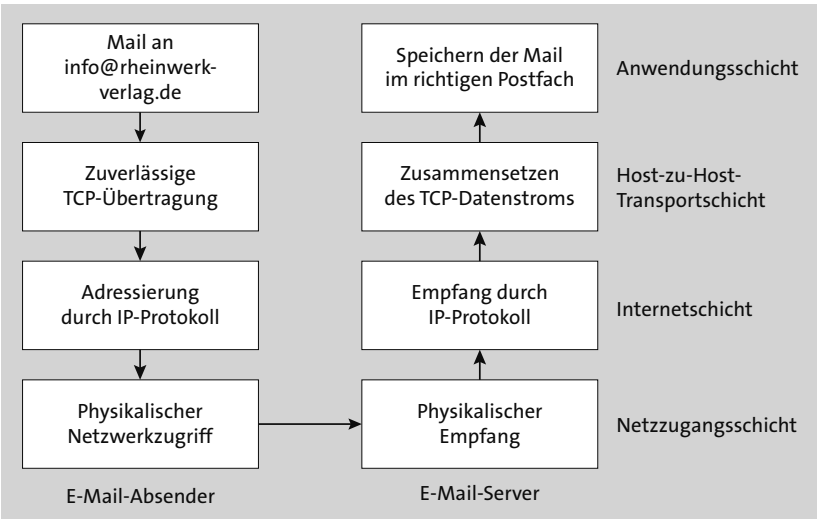


Abbildung 5.2 Übertragung einer E-Mail vom E-Mail-Programm des Absenders auf den Postfachserver des Empfängers. In der Regel sind allerdings mehrere Zwischenstationen beteiligt.

5.3 Klassifizierung von Netzwerken

Nachdem Sie nun mithilfe der Schichtenmodelle eine Möglichkeit kennengelernt haben, unterschiedliche Netzwerke in ihren Funktionen miteinander zu vergleichen, sollten Sie auch verstehen, worin sie sich unterscheiden. Es gibt diverse Unterscheidungsmerkmale, die zwar nicht genau den Schichten der Modelle entsprechen, aber doch ebenfalls mehrere Aspekte der einzelnen Netzwerke betreffen. Es handelt sich um die Unterscheidung nach der Reichweite des Netzwerks, der physikalischen Grundstruktur oder Topologie und zuletzt nach der zentralen oder dezentralen Verwendung des jeweiligen Netzes.

5.3.1 Die Reichweite des Netzwerks

Bei der Unterteilung der Netzwerke entsprechend der Reichweite – also nach der geografischen Größenordnung, die das Netzwerk überbrückt – werden insgesamt vier Stufen unterschieden:

- Das *Local Area Network* (LAN) – das lokale Netzwerk – beschreibt ein Netzwerk, das an ein einzelnes zusammenhängendes Areal gebunden ist, also etwa einen Raum, ein Gebäude oder maximal ein zusammenhängendes (Firmen-)Gelände. LANs sind heutzutage in Wirtschaftsunternehmen, Schulen und Universitäten sowie anderen Organisationen und Instituten weit verbreitet.
- Das *Metropolitan Area Network* (MAN) – das Stadtgebietsnetzwerk – bezeichnet ein Netz, das eine Stadt, eine Gemeinde oder auch eine Region umfasst. Ein Beispiel wären die ver-

schiedenen eigenen Netze von NetCologne in Köln. Die Ausdehnung eines MAN betrifft einen Umkreis von 100 km und mehr.

- Das *Wide Area Network* (WAN) – das Fernnetzwerk – ist ein Netz, das mehrere Städte, eine ganze Region oder sogar ein ganzes Land umfasst. In Deutschland gibt es beispielsweise das Deutsche Forschungsnetz (DFN).
- Das *Global Area Network* (GAN) – das weltweite Netzwerk – ist über mehrere Länder, einen ganzen Kontinent oder sogar die ganze Welt verbreitet. Das bei Weitem größte GAN ist heutzutage natürlich das Internet – im engeren Sinne ist ein GAN allerdings ein homogenes Netzwerk, während das Internet aus zahllosen Einzelnetzen mit unterschiedlichen Architekturen zusammengesetzt ist.

Es sei noch angemerkt, dass die drei Netzwerkarten, die größere Entfernungen überbrücken – also MAN, WAN und GAN –, oftmals einfach unter dem Sammelnamen WAN zusammengefasst werden. Dies umso mehr, als alle drei Typen von Fernnetzen im Wesentlichen die gleiche Art von Technologie verwenden – oder genauer gesagt: Alle Arten von Technologien für Fernnetze werden von allen drei Netzarten verwendet.

Früher gab es viele Fernnetze, die Wählleitungen, also einfache Telefonverbindungen, verwendeten – sowohl das klassische Analog- als auch das digitale ISDN-Netz. In größeren Städten sind heute die diversen DSL-Dienste am verbreitetsten, bei denen durch die Verwendung besonders hochfrequenter Signale über die normalen Kupferdrähte der Telefonleitungen wesentlich höhere Datenübertragungsraten erzielt werden – oder noch höhere über Glasfaserkabel, falls vorhanden. Zu einer besonderen Form der Wählleitung zählen Verbindungen über die digitalen GSM-Mobilfunknetze und deren Nachfolger GPRS, EDGE, UMTS, LTE und 5G. Daneben existieren unterschiedliche Arten von Standleitungen, die für besonders häufig beanspruchte oder besonders zuverlässige Leitungen verwendet werden. Dabei gibt es unter anderem spezielle DSL-Standleitungen oder Glasfasernetze. Auch drahtlose Übertragung, etwa über Funk- oder Satellitenverbindungen, spielt eine immer größere Rolle.

Zu beachten ist allerdings, dass DSL, Wireless LAN und Mobilfunknetze im Wesentlichen die Technologien für den Zugang einzelner Hosts zu einem MAN oder WAN darstellen. Im Backbone-Bereich, also in der eigentlichen Netzwerkinfrastruktur, kommen vor allem Zeitmultiplexing-Verfahren über Glasfasernetze zum Einsatz. Sprache, Video und sonstige Daten werden dabei zum Beispiel über Gigabit-Ethernet übertragen, während ältere Standards wie SDH/SONET, ATM oder Frame-Relay kaum noch eine Rolle spielen. Eine zunehmende Bedeutung erlangten in den letzten Jahren darüber hinaus DWDM-Verfahren (*Dense Wavelength Division Multiplexing*). Dabei werden über ein und denselben Lichtwellenleiter mehrere Signale mit unterschiedlicher Wellenlänge gleichzeitig versandt, was für extrem hohe Datenraten sorgt.

Lokale Netzwerke nutzen ebenfalls viele unterschiedliche technische Übertragungsarten. Allein für Ethernet, die häufigste Form der lokalen Vernetzung, wurden und werden unterschiedliche Arten von Koaxial-, Twisted-Pair- oder Glasfaserkabeln verwendet. Diese zeich-

nen sich durch verschiedene Übertragungsgeschwindigkeiten, mögliche maximale Entfernungen und natürlich auch unterschiedliche Kosten aus. Wireless LAN – der Betrieb von lokalen Netzwerken ohne Kabel über Funk, Infrarot oder Mikrowellen – erfreut sich ebenfalls zunehmender Beliebtheit. Neben Ethernet gab es traditionell viele andere Formen kabelgebundener lokaler Netzwerke, die jedoch kaum noch eine Rolle spielen.

Die technologischen Grundlagen der Verkabelung, der Signalübermittlung und des Netzzugangs werden in Abschnitt 5.4, »Netzwerkkarten, Netzwerkkabel und Netzzugangsverfahren«, ausführlicher besprochen.

5.3.2 Die Netzwerktopologie

Die *Topologie* eines Netzwerks beschreibt, in welcher physikalischen Grundform die einzelnen Geräte organisiert sind. Manche Arten von Netzwerkhardware setzen eine bestimmte Topologie voraus, andere überlassen den Einrichtenden die Entscheidung zwischen mehreren Möglichkeiten. Topologie ist normalerweise eine Eigenschaft lokaler Netzwerke oder gar einzelner Netzsegmente. Die meisten Fernnetze verbinden ohnehin nicht einzelne Rechner, sondern ganze Netzwerke an unterschiedlichen Orten miteinander.

Es werden im Wesentlichen folgende Grundformen unterschieden:

- Die *Bustopologie* beschreibt ein Netzwerk, bei dem die einzelnen Knoten (Anschlüsse) hintereinander an einem einzelnen Kabelstrang angeschlossen sind, dessen Enden nicht miteinander verbunden werden dürfen (sonst würde es sich um eine Ringtopologie handeln!). Häufig werden die beiden Enden des Kabelstrangs durch Abschlusswiderstände (Terminatoren) abgeschlossen. Ein Beispiel für echte busförmige Netzwerke ist das klassische Ethernet über Koaxialkabel.
- Die *Sterntopologie* ist die Form eines Netzes, bei dem alle Knoten mit jeweils eigenem Kabel an einem zentralen Gerät miteinander verbunden werden. Dieses zentrale Bindeglied heißt, je nach seiner genauen Funktionsweise, *Hub* oder *Switch*. Die Sterntopologie wird zum Beispiel von Ethernet über Twisted-Pair-Kabel verwendet.
- Die *Ringtopologie* ähnelt der Bustopologie insofern, als auch hier alle Knoten an einem zentralen Strang aufgereiht sind. Dieser zentrale Kabelstrang bildet jedoch einen geschlossenen Ring. Daraus ergibt sich automatisch eine Datenstromrichtung, in die die Datenpakete grundsätzlich weitergereicht werden. Bekanntestes Beispiel der ringförmigen Vernetzung ist *Token Ring*.
- Die *Baumtopologie* schließlich ist eher ein Standard für den Zusammenschluss verschiedener Netzsegmente. Von einem zentralen Kabelstrang, gewissermaßen dem »Stamm« des Baums, gehen in beliebige Richtungen einzelne Verästelungen ab, an denen entweder eine einzelne Station oder ein ganzes Netz hängt.

Wichtig ist zu guter Letzt, dass ein Unterschied zwischen einer physikalischen und einer logischen Topologie bestehen kann, denn die äußere Form der Verkabelung (physikalische To-

pologie) kann einfach aus praktischen Erwägungen heraus gewählt worden sein, obwohl von der Funktion her eine völlig andere Struktur herrscht, nämlich die logische Topologie.

Ein gutes Beispiel für eine unterschiedliche physikalische und logische Struktur sind neuere Token-Ring-Varianten: Die eigentliche Vernetzung erfolgt sternförmig, logisch betrachtet handelt es sich jedoch um einen Ring. Auch Ethernet über Twisted-Pair-Kabel verwendet – physikalisch gesehen – die Sterntopologie, die logische Funktionsweise hängt von der Art des zentralen Verteilers ab: Ein Hub erzeugt letztlich die Funktion eines busförmigen Netzes, da es einen durchgehenden Strang enthält, an dem alle Stationen angeschlossen sind. Ein Switch dagegen stellt jeweils eine gesonderte Verbindung zwischen zwei Stationen her, die miteinander Daten austauschen; mithin handelt es sich hier auch logisch um die echte Sternform.

5.3.3 Der Zentralisierungsgrad des Netzwerks

Ein weiteres wichtiges Kriterium bei der Einteilung von Netzwerken in unterschiedliche Gruppen ist die Frage nach der Arbeitsaufteilung in ihnen. Kleine Arbeitsgruppen, die jeweils mit ihren Arbeitsplatzrechnern untereinander Dateien austauschen möchten, haben hier sicherlich andere Bedürfnisse als riesige Organisationen, in denen Tausende von Personen auf bestimmte Datenbestände zugreifen müssen. Deshalb werden die sogenannten Client-Server-Netzwerke, in denen zentrale Dienstleistungsrechner, die Server, arbeiten, von den Peer-to-Peer-Netzwerken unterschieden, in denen die einzelnen Computer gleichberechtigt Ressourcen freigeben und verwenden können.

- Das *Client-Server-Netzwerk* unterscheidet generell zwei Arten von beteiligten Rechnern: Der Server (Dienstleister) ist ein Computer, der den Arbeitsstationen an zentraler Stelle Ressourcen und Funktionen zur Verfügung stellt; der Client (Kunde) nimmt diese Dienstleistungen in Anspruch. Die Dienste, die von Servern angeboten werden, sind sehr vielfältig: Sie reichen vom einfachen Dateiserver, der Dateien im Netzwerk verteilt oder Festplattenplatz für andere freigibt, über Druckserver, Mail- und andere Kommunikationsserver bis hin zu ganz speziellen Diensten wie Datenbank- oder Anwendungsservern.
- Das *Peer-to-Peer-Netzwerk* besteht aus prinzipiell gleichberechtigten Arbeitsplatzrechnern (ein *Peer* ist so etwas wie ein »Kollege«). Jeder lokale Rechner ist in der Lage, ausgewählte Ressourcen an andere im Netzwerk freizugeben. Das heißt, dass alle Rechner im Netz bis zu einem gewissen Grad Serverdienste wahrnehmen.

In der Praxis sind allerdings Mischformen häufiger anzutreffen als reine Client-Server- oder absolute Peer-to-Peer-Netze. Beispielsweise könnte man sich in einem Unternehmen die folgende Situation vorstellen: Aufgaben wie die direkte Kommunikation (E-Mail), der Zugang zum Internet (über einen Proxyserver oder einfach einen Router) oder Lösungen zum Backup (Datensicherung) werden durch zentrale Server zur Verfügung gestellt; der Zugang

zu Dateien innerhalb der Abteilungen oder auf Drucker der Teammitglieder innerhalb eines Büros wird dagegen im Peer-to-Peer-Verfahren unter Umgehung von Servern geregelt.

Wichtig ist außerdem, zu verstehen, dass die Begriffe *Client* und *Server* im engeren Sinne nicht unbedingt spezifische Rechner, sondern besondere Softwarekomponenten bezeichnen.

Ein Server ist einfach ein Programm, das meist automatisch gestartet wird und im Hintergrund darauf »lauert«, irgendeine Dienstleistung zur Verfügung zu stellen. Allgemeiner werden solche Programme zum Beispiel im Unix-Umfeld als *Daemon* bezeichnet, unter Windows 11 und seinen Vorgängern in der Windows-NT-Familie heißen sie Dienst (*Service*). Grundsätzlich kann ein solcher Serverdienst auf jedem beliebigen Rechner laufen – vorausgesetzt natürlich, er ist für die Hardwareplattform und das Betriebssystem dieses Rechners bestimmt. Der Grund für den Einsatz besonders leistungsfähiger Hardware (eben der Serverhardware) und spezialisierter Betriebssysteme liegt einfach in ihrer höheren Belastbarkeit, wenn ihre Dienste vielfach gleichzeitig benötigt werden.

Ein Client ist zunächst eine Software, die in der Lage ist, mit der Serversoftware zu kommunizieren; üblicherweise stellt sie auch eine Benutzerschnittstelle zur Verfügung, um diese Kommunikation in Anspruch zu nehmen. So ist beispielsweise ein Webbrowser ein Client für das HTTP-Anwendungsprotokoll, er kommuniziert also mit HTTP-Servern. Interessanterweise können Webserver und Browser auch beide auf demselben Rechner laufen. Dies ist nützlich, um Webanwendungen zunächst lokal auszuprobieren.

Arten von Servern

Im Folgenden sollen einige Serverarten genauer vorgestellt werden. Sie sollten auf jeden Fall das zuvor Gesagte im Hinterkopf behalten: Es spielt überhaupt keine Rolle für die allgemeine Funktion, ob ein Serverdienst, also die Software, die diesen Dienst zur Verfügung stellt,

- mit anderen Diensten zusammen auf demselben Rechner läuft,
- allein auf einem separaten Serverrechner ausgeführt wird oder
- sogar auf mehrere Serverrechner verteilt ist, weil ansonsten die Belastung zu groß wäre.

Letzteres ist insbesondere im Bereich öffentlicher WWW-Server sehr häufig zu finden, da populäre Sites wie etwa Suchmaschinen, Nachrichtenportale oder große Webshops sehr viel Datenverkehr zu verkraften haben. Hier werden sogenannte *Load-Balancing-Systeme* eingesetzt, die die hereinstürmenden Anfragen automatisch möglichst gerecht auf mehrere physikalische Server verteilen.

Im Wesentlichen gibt es die folgenden wichtigen Arten von Serverdiensten:

- Fileserver
- Printserver
- Mailserver

- Webserver
- Verzeichnisdienstserver
- Anwendungsserver und Serveranwendungen

In den folgenden Abschnitten wird jeder dieser Servertypen kurz vorgestellt; in späteren Kapiteln erhalten Sie auch konkrete Beispiele für viele von ihnen.

Fileserver

Der Fileserver (*Dateiserver*) stellt anderen Rechnern im Netzwerk freigegebene Verzeichnisse zur Verfügung. Auf diese Weise können sich mehrere Personen über einen zentralen Austauschpunkt gegenseitig Dateien zukommen lassen. Der Fileserver ist relativ stark an ein bestimmtes Betriebssystem oder eine Plattform gebunden. Erst allmählich setzen sich neuere Möglichkeiten durch, die in der Lage sind, auch unterschiedliche Rechner gleichzeitig zu bedienen. Denn die Besonderheit eines Fileservers besteht darin, dass er völlig transparent genauso benutzt werden kann wie das lokale Dateisystem eines Arbeitsplatzrechners. In einem idealen (lokalen) Netzwerk sollte es vollkommen egal sein, ob Dateien am Arbeitsplatz oder auf einem Fileserver zu finden sind.

Sehr wichtig ist im Zusammenhang mit Fileservern die Verwaltung von Zugriffsrechten, da nicht jede Datei allgemein freigegeben werden soll.

Der Internetdienst FTP (*File Transfer Protocol*) ist übrigens kein vollwertiger Fileserver, sondern dient lediglich der einfachen Dateiübertragung. Die Informationen über die Dateien des entfernten Rechners sind nicht vollständig genug, um das Äquivalent eines Dateisystems abzubilden.

Printserver

Der Printserver (oder *Druckserver*) erlaubt mehreren Arbeitsstationen den gemeinsamen Zugriff auf einen Drucker. Die größte Herausforderung besteht darin, den einzelnen Arbeitsstationen automatisch den passenden Druckertreiber für ihr jeweiliges Betriebssystem zur Verfügung zu stellen, sodass diese den Drucker einfach verwenden können, ohne dass der Treiber zuvor noch einmal lokal installiert werden müsste.

Der Betrieb von Printservern ist besonders in Windows-Netzwerken sehr verbreitet, da hier der Drucker gewöhnlich über ein USB-Kabel an einen einzelnen Rechner angeschlossen wird. Dieser Rechner wird dann so eingerichtet, dass er den Zugriff auf den Drucker auch den anderen Computern erlaubt.

Bei anderen Plattformen gibt es das Problem in dieser Form seltener. In klassischen Macintosh-Netzwerken war es beispielsweise schon lange üblich – und viel bedienungsfreundlicher –, den Drucker unmittelbar per Ethernet ans Netzwerk anzuschließen, denn damit ist er automatisch für alle freigegeben. Fast alle modernen Drucker sind inzwischen mit dieser

Fähigkeit ausgestattet – sie können mit Ethernet oder WLAN verbunden werden und üblicherweise mit Windows-, macOS- und auch Linux-Rechnern zusammenarbeiten.

Mailserver

Ein Server für elektronische Post (E-Mail) muss nicht immer bei einem Internetprovider installiert sein, sondern kann auch im lokalen Netz seinen Dienst verrichten. Denn erstens ist es in Unternehmen oder Organisationen oft von Vorteil, wenn die Teams untereinander per E-Mail kommunizieren können, und zweitens ist es manchmal schon allein deshalb erforderlich, einen internen Mailserver zu betreiben, weil der Zugang zum Internet aus Sicherheitsgründen stark eingeschränkt ist und etwa die Kommunikation eines Arbeitsplatzrechners mit einem externen Mailserver gar nicht zulässt.

Obwohl im Lauf der Netzwerkentwicklungsgeschichte verschiedene Formen der elektronischen Post entstanden sind, gibt es heute eigentlich keine Alternative mehr zu Internet-E-Mail. Diese verwendet verschiedene Serverdienste zum Senden und Empfangen der E-Mail: Das SMTP-Protokoll (*Simple Mail Transport Protocol*) bestimmt, wie zu versendende E-Mails zu transportieren sind; POP3 (*Post Office Protocol Version 3*) oder das modernere, komfortablere IMAP (*Internet Message Access Protocol*) beschreiben ein Benutzerkonto (Postfach) für eingehende E-Mails sowie den Vorgang der »Abholung«.

Rein theoretisch kann Internet-E-Mail direkt zum einzelnen Host gesendet werden. Das ist aber insofern problematisch, als normale Arbeitsplatzrechner gelegentlich ausgeschaltet werden und private Einzelplatzrechner manchmal nur temporär über Wählleitungen mit dem Internet verbunden sind. Dies ist überhaupt der wichtigste Grund dafür, dass sich Posteingangsserver etabliert haben, auf denen die Mail für eine bestimmte Empfangsadresse im Prinzip vorgehalten wird, bis sie abgerufen wird.

Da die gewöhnliche Form der E-Mail auf den Standard-Internetprotokollen aufsetzt, gibt es übrigens kein Problem, sie plattform- und betriebssystemübergreifend zu verwenden.

Webserver

Ein Webserver (die exakte Bezeichnung ist eigentlich *HTTP-Server*) liefert auf Anfrage Webseiten über ein Netzwerk aus. In der Regel ist dieses Netzwerk das Internet. In den lokalen Netzen von Unternehmen und Institutionen ist diese Form der Informationsübermittlung ebenfalls sehr verbreitet. Ein solches lokales Netz, das Technologien und Dienste der Internetprotokolle verwendet, wird *Intranet* genannt. Die Clientsoftware ist ein Anzeigeprogramm für Webseiten, der sogenannte *Browser*. Er wird verwendet, um Webseiten anzufordern, zu betrachten und um den enthaltenen Hyperlinks – also den Verknüpfungen zu anderen Dokumenten auf dem gleichen oder einem anderen Server – per Mausklick folgen zu können.

Webseiten sind prinzipiell Textdokumente, die in der Strukturierungssprache HTML geschrieben werden. Viele dieser Dokumente liegen statisch auf dem Server und werden ein-

fach auf Anfrage ausgeliefert. Eine wachsende Anzahl solcher Dokumente wird aber auch aus Vorlagen und dynamischen Daten, etwa aus einer Datenbank, kombiniert und dann an den anfragenden Host geschickt. Diese Entwicklung ist unvermeidlich für Websites mit umfangreichem, schnell wechselndem Inhalt, etwa Online-Tageszeitungen, für Kataloge in E-Commerce-Sites oder für soziale Medien.

Webserver sind im Übrigen schon von ihrer Grundidee her dafür gedacht, Clients unter vielen verschiedenen Betriebssystemen zu bedienen. Sollte es Inkompatibilitäten geben, liegt das höchstens daran, dass bei der Erstellung des HTML-Codes Steuerbefehle verwendet wurden, die nicht jeder Browser versteht.

Der praktische Einsatz eines Webservers wird in Kapitel 14, »Server für Webanwendungen«, am wichtigsten Beispiel Apache beschrieben; die Kapitel 18, »Webseitenerstellung mit HTML und CSS«, bis Kapitel 20, »JavaScript und Ajax«, kümmern sich dagegen um die Erstellung von Webinhalten und -anwendungen.

Verzeichnisdienstserver

Verzeichnisdienste (*Directory Services*) gewinnen in der IT seit längerer Zeit stark an Bedeutung. Ein Verzeichnis ist in diesem Zusammenhang kein Dateisystem, sondern ein datenbankähnlicher standardisierter Katalog von User-Accounts, Computern, Peripheriegeräten, Diensten und Berechtigungen in einem Netzwerk. Durch den Eintrag in das Verzeichnis können diese Informationen netzwerkweit abgerufen werden, sodass Verzeichnisdienste eine praktische Grundlage für zahlreiche Dienste legen, die in einer größeren Netzwerkumgebung die Arbeit von Administrationen und Teams erleichtern. Hier nur einige Beispiele:

- automatisierte Softwareverteilung und -installation
- mobile Benutzerprofile (Roaming User Profiles)
- zentralisierte Anmeldedienste (Single-Sign-on)
- rechner-, benutzer- und eigenschaftsbasierte Rechtekontrolle

In Kapitel 15, »Weitere Internet-Serverdienste«, wird OpenLDAP als Praxisbeispiel für einen Verzeichnisdienst vorgestellt.

Anwendungsserver und Serveranwendungen

Ein Anwendungsserver (*Application Server*) erlaubt die Benutzung von Anwendungsprogrammen, die sich eigentlich auf dem Server befinden, über das Netzwerk.

Bei der einfachsten Form des Anwendungsservers liegt der Datenbestand der Anwendung auf den Datenträgern des Servers, die Anwendung wird über das Netzwerk in den Arbeitsspeicher des Clients geladen und dort lokal ausgeführt. Der Unterschied zum Fileserver ist hier minimal: Es muss der Anwendung lediglich klar sein, dass eventuell notwendige Zusatzkomponenten oder Konfigurationsdaten nicht auf dem Rechner liegen, auf dem sie ausgeführt wird, sondern auf der Maschine, von der sie geladen wurde.

Bei vielen normalen Einzelplatz-Anwendungsprogrammen kann eine solche Einstellung vorgenommen werden. Diese Verwendung von Software hat vor allem zwei Vorteile: Erstens kann es weniger Arbeit bedeuten, ein Programm einmal auf dem Server statt auf mehreren Arbeitsplatzrechnern zu installieren, und zweitens können Kosten gespart werden – die meisten Softwarelizenzen gelten jeweils pro Rechner, auf dem das jeweilige Programm installiert ist. Wird eine Anwendung auf mehreren Rechnern genutzt, aber nicht gleichzeitig, kann die Software auf dem Server installiert werden; damit werden die Lizenzgebühren dann nur einmal fällig.

Bei komplexeren Formen von Anwendungsservern werden Teile des Programms – oder unter Umständen auch das ganze Programm – direkt auf dem Server ausgeführt. Die möglichen Gründe dafür sind im Einzelfall genauso vielfältig wie die verschiedenen Formen der Umsetzung. Beispielsweise ist es bei großen Datenbanken üblich, dass der Datenbestand als solcher auf einem Server liegt und ebenso die grundlegende Datenverwaltungssoftware. Auf den Clients existieren dann in der Regel sogenannte *Frontends*, also Softwarekomponenten, die eine Bedienoberfläche für die eigentliche Datenbank bereitstellen. (Das Pendant zum Frontend bildet das *Backend*, wobei es sich um einen nur für spezielle Verwaltungs-Accounts zugänglichen Teil des Clients handelt, der der Verwaltung des Servers dient.)

Noch einen Schritt weiter gehen die sogenannten *verteilten Anwendungen* oder *Enterprise-Anwendungen*. Sie basieren in der Regel auf einem oder mehreren Datenbankservern für den Datenbestand, einem Anwendungsserver für die Geschäftsabläufe und diversen Client-Frontends (sowohl nativen Programmen für bestimmte Betriebssysteme als auch Webanwendungen).

Eine andere Form der Serveranwendung existiert bei der Verwendung der sogenannten *Terminalserver*. Die einfachste Form, der Internetdienst Telnet – häufiger dessen sichere Weiterentwicklung SSH –, stellt eine Konsolenoberfläche zur Verfügung, über die sich von fern auf dem Server selbst mithilfe von Kommandoeingaben arbeiten lässt. Das heißt, die Ein- und Ausgabe zeilenorientierter Kommandos und Anwendungsprogramme erfolgt auf dem Client, die eigentliche Ausführung auf dem Server – der eigene Rechner wird so zu einem Terminal für den entfernten Server (Terminal-Emulation).

Eine sehr merkwürdige Form der Serversoftware ist in diesem Zusammenhang der aus dem Unix-Bereich stammende *X-Window-Server* oder einfach X-Server (siehe Kapitel 6, »Betriebssysteme«). Die Bezeichnung *Server* für diese Software erscheint zunächst sehr irreführend, handelt es sich doch einfach um die Grundlage der grafischen Benutzeroberfläche (GUI) unter Unix. Der X-Server stellt den Anwendungsprogrammen seine Dienste zur Verfügung, die darauf zugreifen, um Fenster und andere Komponenten des GUI darzustellen.

Übrigens rechtfertigt die Tatsache, dass hier ein Dienst verfügbar gemacht wird, den Begriff *Server*. Dabei müssen Anwendung und X-Server auch nicht unbedingt auf demselben Rechner laufen. Erstaunlicherweise läuft aber der X-Server auf dem Anwendungsclient! Denn da

die Programmausführung auf dem entfernten Rechner stattfindet, aber die grafische Darstellung auf dem lokalen Rechner, muss dieser Dienst hier angeboten werden.

Terminalserver gibt es ebenfalls unter Windows Server 2022 und anderen Microsoft-Systemen; auch hier läuft die eigentliche Anwendung auf dem Server, der Client erlaubt deren Bedienung und Anzeige. Das Angebot solcher Anwendungsdienste über das Internet wird allmählich beliebter. Ein *ASP* (*Application Service Provider*) lässt Anwendungen wie beispielsweise Bürosoftware auf seinen Servern laufen; über einen gewöhnlichen Webbrowser (oder manchmal über spezielle Clientsoftware) können Benutzer*innen darauf zugreifen und die angebotene Software von der ganzen Welt aus benutzen. Eine wesentlich einfachere Form solcher Serveranwendungen, die über das Web verwendet werden und die Sie wahrscheinlich gut kennen, ist der weitverbreitete webbasierte E-Mail-Dienst mit diversen Zusatzfunktionen, wie ihn Google Mail und GMX anbieten.

Die Quintessenz der Verwendung von Anwendungsservern war die von einigen Firmen (zum Beispiel Oracle) seit Jahrzehnten angestrebte Abschaffung der gewöhnlichen Personal Computer und deren Ersatz durch sogenannte *Thin Clients* – Rechner ohne Festplatte, die ihr Betriebssystem und die Anwendungsprogramme vollständig aus dem Netzwerk oder aus dem Internet beziehen. Allerdings konnte sich das Konzept nie recht durchsetzen. Das Hauptargument der entsprechenden Unternehmen, nämlich die geringeren Kosten, ließ sich angesichts des massiven Preisverfalls bei den »ausgewachsenen« PCs nicht aufrechterhalten.

Inzwischen sind es auf der Clientseite eher die Tablets, die den gewöhnlichen PCs den Rang ablaufen. Zudem bieten immer mehr Hardware- und Betriebssystemhersteller bereits ab Werk Cloud-Dienste zur Daten- und Anwendungsspeicherung. Näheres zu Cloud Computing erfahren Sie in Kapitel 14, »Server für Webanwendungen«.

5.4 Netzwerkkarten, Netzwerkkabel und Netzzugangsverfahren

Im Laufe der Entwicklungsgeschichte der Netzwerke, die in diesem Kapitel bereits skizziert wurde, haben sich viele verschiedene Formen der Netzwerkhardware entwickelt. Jede von ihnen hatte zum Zeitpunkt ihrer Entstehung ihre Berechtigung, und dennoch haben sich einige auf breiter Front durchgesetzt, während andere schnell wieder vom Markt verschwunden sind. Die verbreitetste Art der Netzwerkhardware ist heute Ethernet in seinen vielfältigen Varianten.

Analog zu den zuvor beschriebenen Schichtenmodellen – vor allem dem standardisierten OSI-Referenzmodell – gibt es auch Standards, die speziell die Netzwerkhardware und den Netzzugang betreffen, also die beiden untersten Ebenen des OSI-Modells. Die umfangreichste Sammlung ist IEEE 802 des *Institute of Electrical and Electronical Engineers*. Die Nummer 802 bezeichnet Jahr und Monat der ursprünglichen Festlegung, nämlich den Februar 1980. Innerhalb dieser Sammlung existiert eine Reihe verschiedener Unterstandards bezie-

hungsweise Arbeitsgruppen. Zu den wichtigsten gehören 802.1 (allgemeine Netzwerkstandards), 802.3 (Netzzugangsverfahren CSMA/CD, besonders Ethernet) und 802.11 (drahtlose Netze). Tabelle 5.2 zeigt eine Liste, in der alle direkten Unterpunkte von IEEE 802 mit noch aktiven Arbeitsgruppen und einige exemplarische Einzelspezifikationen innerhalb dieser Unterpunkte angegeben sind. Ein paar dieser Standards werden im Folgenden näher beschrieben.

IEEE-Gruppe	Bezeichnung
802.1	Higher Layer LAN Protocols Working Group
802.3	CSMA/CD, Ethernet
802.3u	Fast Ethernet
802.3z	Gigabit Ethernet über Glasfaser
802.3ab	Gigabit Ethernet über Twisted Pair
802.4	Token-Bus-Zugriffsverfahren
802.11	drahtlose Netze
802.15	Wireless Personal Area Network (WPAN)
802.16	Broadband Wireless Access (BWA)
802.18	Radio Regulatory Technical Advisory Group (RRTAG)
802.19	Coexistence TAG
802.21	medienunabhängiges Handover
802.22	drahtlose Regionalnetze (WRAN)

Tabelle 5.2 Die IEEE-802-Hauptarbeitsgruppen und einige Unterstandards im Überblick

5.4.1 Die verschiedenen Ethernet-Standards

Ethernet ist heute der verbreitetste Standard für verkabelte lokale Netze (LANs). Zehntausende von Herstellern weltweit unterstützen diese Art von Netzwerken mit ihrer Hard- und Software.

Jede Ethernet-Schnittstelle, also die Netzwerkkarte oder der fest eingebaute Anschluss, ist mit einer weltweit einmaligen Identifikationsnummer ausgestattet, der sogenannten *MAC-Adresse* (für *Media Access Control*, einen der beiden Bestandteile der OSI-Netzzugangsschicht). Es handelt sich um eine 48 Bit lange Zahl, die in sechs hexadezimalen Blöcken zwischen 0 und 255 (00 bis FF hex) geschrieben wird, zum Beispiel `00-A0-C9-E8-5F-64`.

Die Datenpakete – auf der Netzzugangsschicht *Frames* genannt – werden mit den MAC-Adressen der sendenden und der empfangenden Station versehen und in der Regel an alle Stationen im Segment versandt. Jede Station überprüft daraufhin, ob die Daten für sie bestimmt sind. Im Übrigen kann man Ethernet-Schnittstellen auch in den *Promiscuous Mode* schalten, in dem sie ohne Unterschied alle Daten entgegennehmen. Auf diese Weise kann der gesamte Datenverkehr in einem Netzsegment überwacht werden.

Die MAC-Adresse wird normalerweise nicht über das jeweilige Teilnetz hinaus weiterverbreitet.[7] Nach außen ergäbe ihre Verwendung auch keinen Sinn, da das nächste Teilnetz auf einer Route womöglich noch nicht einmal zum Ethernet-Standard gehört.

Das Netzzugangsverfahren CSMA/CD

Es ist wichtig, zu verstehen, dass mit dem Namen *Ethernet* gar keine einheitliche Netzwerkhardware bezeichnet wird. Vielmehr handelt es sich um einen Sammelnamen für diverse Netzwerkstandards, die ein bestimmtes Netzzugangsverfahren verwenden. Insofern sind alle Ethernet-Varianten auf der OSI-Schicht 2 identisch, unterscheiden sich aber auf der untersten Schicht.

Als der Vorläufer von Ethernet Ende der 60er-Jahre des letzten Jahrhunderts an der Universität von Hawaii konzipiert wurde (anfangs unter dem geografisch passenden Namen ALOHANet), handelte es sich zunächst um Datenfunk. Diesem Umstand ist übrigens auch der endgültige Name zu verdanken: *Ether*, zu Deutsch Äther, ist das gedachte Medium, durch das sich Funkwellen fortpflanzen. Erst in den 1970er-Jahren wurde dasselbe Netzzugangsverfahren auch für die Datenübertragung per Kabel eingesetzt, und zwar zunächst über Koaxialkabel.

Das gemeinsame Netzzugangsverfahren aller Ethernet-Formen trägt den Namen CSMA/CD: *Carrier Sense Multiple Access with Collision Detection*. Schematisch gesehen, funktioniert dieses Verfahren wie folgt:

1. Ein Gerät, das Daten senden möchte, lauscht den Netzabschnitt ab, um festzustellen, ob dieser gerade frei ist, ob also gerade kein anderes Gerät sendet (*Carrier Sense*).
2. Wurde in Schritt 1 festgestellt, dass der Netzabschnitt frei ist, beginnt die Station mit dem Senden der Daten. Möglicherweise hat auch eine andere Station festgestellt, dass das Netz frei ist, und beginnt gleichzeitig ebenfalls mit dem Senden (*Multiple Access*).
3. Falls auf die beschriebene Art und Weise zwei Stationen gleichzeitig mit dem Senden begonnen haben, findet eine sogenannte *Datenkollision* statt, die von den beteiligten Stationen entdeckt wird (*Collision Detection*). Eine Station, die eine Kollision bemerkt, stellt das Senden von Nutzdaten ein und versendet stattdessen eine Warnmeldung (*Jam Signal*).
4. Eine Station, die wegen einer Datenkollision das Senden abgebrochen hat, beginnt nach einer zufällig gewählten Zeitspanne von wenigen Millisekunden erneut mit dem Senden.

7 Ausnahme: Die IP-Weiterentwicklung IPv6 benutzt die MAC-Adresse als Teil der 128 Bit langen IP-Adresse.

Genau diese Zufälligkeit der Zeitspanne, die nach einem komplizierten Verfahren berechnet wird, ist enorm wichtig, damit die beiden Stationen beim nächsten Versuch nicht wieder genau gleichzeitig mit dem Senden beginnen.

Das große Problem von Ethernet besteht darin, dass das CSMA/CD-Verfahren umso ineffektiver wird, je frequentierter der jeweilige Netzabschnitt ist: Ab einem gewissen Grenzwert überschreitet die Anzahl der Datenkollisionen die Menge der Nutzdaten. Heutzutage umgeht man dieses Problem in der Regel durch die Verwendung sogenannter *Switches*, die für zwei miteinander kommunizierende Stationen jeweils eine exklusive Punkt-zu-Punkt-Verbindung einrichten. Wenn diese Möglichkeit aufgrund veralteter, inkompatibler Hardware nicht zur Verfügung steht, muss ein Netz mit viel Datenverkehr stattdessen segmentiert, also in kleinere Abschnitte unterteilt werden.

Ethernet-Hardware

Die Bezeichnungen der verschiedenen Arten der Hardware, die für Ethernet-Netzwerke verwendet werden, setzen sich aus der Übertragungsgeschwindigkeit des jeweiligen Netzes in MBit/s und einer spezifischen Bezeichnung für den Kabeltyp oder die maximal zulässige Kabellänge zusammen.

Wie bereits erwähnt, waren Koaxialkabel die ersten für Ethernet verwendeten Kabel. Der Aufbau dieser Kabel ist folgender: Im Zentrum befindet sich ein leitender Draht, der von einer Isolationsschicht umgeben ist, darüber befindet sich ein weiterer Ring aus leitendem Metall und außen natürlich wiederum eine Isolationsschicht. Das bekannteste Alltagsbeispiel für ein Koaxialkabel ist ein handelsübliches Fernsehantennenkabel. Da diese Art der Verkabelung in der Praxis keine Rolle mehr spielt, wird hier, anders als in früheren Auflagen, nicht mehr näher darauf eingegangen.

Heutzutage wird Ethernet fast immer über Twisted-Pair-Kabel betrieben. Bei dieser Kabelsorte handelt es sich um einen verdrillten Kupferzweidrahtleiter: Je zwei isolierte Kupferdrähte werden umeinandergewickelt. Dies verhindert die gegenseitige Beeinträchtigung der Signalqualität, die bei parallel zueinander verlaufenden Kabeln durch die elektromagnetischen Felder aufträte. In einem Twisted-Pair-Kabel verlaufen üblicherweise vier, manchmal auch acht solcher Doppeladern nebeneinander. Sie enden auf beiden Seiten in einem RJ-45-Stecker, der auch für ISDN-Anschlüsse verwendet wird. Bekannt sind solche Kabel vor allem durch ihre Verwendung als Telefonleitungen.

Man unterscheidet zwei verschiedene Grundarten von Twisted-Pair-Kabeln: UTP (*Unshielded Twisted Pair*) ist ein nicht abgeschirmter, STP (*Shielded Twisted Pair*) ein abgeschirmter Zweidrahtleiter, der eine höhere Signalqualität aufweist, sodass er zum Beispiel größere Entfernungen überbrücken kann.

Außerdem werden Twisted-Pair-Kabel in verschiedene Kategorien unterteilt, die unterschiedliche Bandbreiten (gemessen in Megahertz, MHz) und entsprechend verschiedene maximale Übertragungsraten zulassen. Diese sind in Tabelle 5.3 aufgelistet.

Kategorie	Bandbreite	Verwendungszweck
1	nicht festgelegt	Telefonie
2	4 MHz	ISDN
3	10 MHz	Ethernet, Token Ring
4	16 MHz	verschiedene
5	100 MHz	Fast Ethernet, allgemeiner Standard
6	200 MHz	verschiedene
7	600 MHz	verschiedene

Tabelle 5.3 Die verschiedenen Kategorien von Twisted-Pair-Kabeln

Sämtliche über Twisted Pair verkabelten Arten von Ethernet weisen eine sternförmige Topologie auf, zumindest im physischen Sinn: Alle Stationen werden jeweils über ein eigenständiges Kabel an einen zentralen Verteiler angeschlossen. Der Vorteil dieser Form der Vernetzung besteht grundsätzlich darin, dass der Ausfall einer einzelnen Verbindung zwischen einem Rechner und dem Verteiler nicht zur Unterbrechung des gesamten Netzes führt, wie es beim busförmigen Koaxialkabel-Ethernet der Fall ist.

Der zentrale Verteiler wird in seiner einfacheren Form *Hub* genannt, die etwas teurere, aber leistungsfähigere Bauweise heißt *Switching Hub* oder kurz *Switch*. Die innere Struktur des Hubs ist letztlich busförmig, sodass es genau wie bei der Vernetzung über Koaxialkabel zu Datenkollisionen kommen kann. Ein Switch stellt dagegen für zwei Stationen, die miteinander kommunizieren möchten, eine exklusive Punkt-zu-Punkt-Verbindung bereit. Dies geschieht dadurch, dass ein Switch die MAC-Adressen aller Schnittstellen zwischenspeichert, an die er bereits Daten ausgeliefert hat, und auf diese Weise die restlichen Stationen nicht mehr mit Daten behelligen muss, die gar nicht für sie bestimmt sind. Da die Preise für Netzwerkzubehör in den letzten Jahren stark gesunken sind, gibt es eigentlich keinen Grund mehr, etwas anderes als einen Switch einzusetzen.

Bei einem Hub teilen sich alle Stationen die gesamte Übertragungsgeschwindigkeit, beim Switch steht sie dagegen jeder einzelnen Verbindung zur Verfügung.

Im Übrigen gibt es besondere Hubs, die als *Bridges* bezeichnet werden. Sie verbinden Ethernet-Netzwerke verschiedenen Typs miteinander. Beispielsweise gab es früher Bridges mit einer Reihe von RJ-45-Ports für Twisted-Pair-Kabel und zusätzlich einem Anschluss für Koaxialkabel. Neuere Bridges unterstützen einfach verschiedene maximale Übertragungsgeschwindigkeiten.

Hubs oder Switches weisen in der Regel 5 bis 24 Anschlüsse (Ports) auf, an die jeweils ein Gerät angeschlossen werden kann. Um Netzwerke mit mehr Geräten zu betreiben, sind diese

Geräte kaskadierbar: Die meisten Hubs oder Switches besitzen einen speziellen Port, den sogenannten *Uplink-Port*, der über ein Kabel mit einem normalen Port eines weiteren Verteilers verbunden werden kann. Bei vielen Hubs/Switches kann ein einzelner Port über einen Schalter zwischen *Normal* und *Uplink* umgeschaltet werden.

Die einzige Ausnahme von der allgemeinen Regel, dass ein Hub oder Switch benötigt wird, bildet der Sonderfall, in dem nur zwei Rechner miteinander vernetzt werden sollen: Die beiden Stationen können unmittelbar über ein sogenanntes *Crosslink-Kabel* verbunden werden. Dieses spezielle Kabel besitzt überkreuzte Anschlusspaare anstelle der geradlinig verlaufenden bei normalen Twisted-Pair-Kabeln.

Virtual LAN

Standardmäßig gehören alle Stationen, die an einen einzelnen Hub oder Switch angeschlossen sind, zum selben Netzsegment, und alle, die an einem anderen Hub oder Switch hängen, bilden ihr eigenes Segment. *Virtual LAN* oder *VLAN* ist eine Möglichkeit, diese starre Einteilung durch intelligente Switches zu überwinden: Entweder kann derselbe Switch mehrere getrennte Netzsegmente verwalten (und bei Bedarf auch gleich als Router zwischen ihnen fungieren), oder es können mehrere Switches ein gemeinsames logisches Netzsegment bilden. Es sind zwei Arten von Virtual LAN zu unterscheiden:

- *Portbasiertes Virtual LAN*: Bei dieser rein hardwaregesteuerten Variante gehören verschiedene Ports (gemeint sind hier die physischen Anschlüsse) des Switches zu unterschiedlichen Netzsegmenten. Gegebenenfalls besitzt der Switch eine Konfigurationsoberfläche, über die sich die konkrete Einteilung modifizieren lässt, aber danach wählt man das Netzsegment für eine bestimmte Station, indem man sie mit einem bestimmten Port verkabelt.
- *Tagged Virtual LAN*: Hierbei handelt es sich um die modernere und flexiblere, aber auch komplexere Variante. Statt die Ports des Switches bestimmten Netzsegmenten zuzuordnen, werden die Datenpakete selbst gekennzeichnet (englisch: *tagged*). Je nach verwendetem Tag gehört ein Datenpaket damit zu einem anderen Segment. Diese Variante wird vor allem verwendet, um mehrere Switches zu einem gemeinsamen größeren Segment zusammenzuschalten.

Historisch betrachtet, existieren zwei Arten von Ethernet über Twisted Pair, die unterschiedliche Übertragungsgeschwindigkeiten unterstützen:

- 10BaseT: Die Datenübertragungsrate beträgt 10 MBit/s.
- 100BaseT (auch *Fast Ethernet* genannt): Daten werden mit bis zu 100 MBit/s übertragen; dazu sind mindestens UTP-Kabel der Kategorie 5 erforderlich. Genauer gesagt, gibt es zwei Unterarten: 100BaseTX ist voll kompatibel mit 10BaseT, sodass das Netz schrittweise umgerüstet werden kann. 100BaseT4 verwendet dagegen alle vier Kupferdrahtpaare eines Twisted-Pair-Kabels und ist mit den anderen Standards inkompatibel; in der Praxis spielt es keine Rolle mehr.

Die meisten Netzwerkkarten, Hubs und Switches, die heute verkauft werden, unterstützen beide klassischen Übertragungsraten sowie noch höhere. Der zu verwendende Wert kann bei vielen Netzwerkkarten per Software eingestellt werden, häufiger wird er automatisch gewählt.

Noch neuere Formen von Ethernet erreichen Übertragungsraten von 1.000 MBit/s (Gigabit-Ethernet), entweder über Lichtwellenleiter (1000BaseFL für *Fiber Logic*) oder über Twisted-Pair-Kabel (1000BaseTX). Inzwischen sind Ethernet-Varianten mit 10 GBit/s, 100 GBit/s oder mehr möglich und üblich – anfangs nur über verschiedene Arten von Lichtwellenleitern, aber inzwischen ebenfalls über Twisted Pair.

5.4.2 Drahtlose Netze

Schon seit sehr langer Zeit werden über drahtlose Technologien wie Funk, Mikrowellen, Satellit oder Infrarot nicht nur Sprache, Radio- und Fernsehsignale, sondern auch Daten übertragen. Die digitale (!) Datenübertragung per Funk war sogar die erste Anwendung der drahtlosen Nachrichtentechnik überhaupt: Der Funkpionier Guglielmo Marconi erfand die drahtlose Telegrafie mithilfe des binären Morsealphabets[8] lange vor dem Sprechfunk.

Im Bereich der Netzwerke gibt es immer mehr Anwendungsfälle, bei denen sich der Einsatz drahtloser Techniken anbietet. Die folgenden Beispiele können als Anhaltspunkte dienen:

- In Privathaushalten wird WLAN inzwischen häufiger eingesetzt als kabelbasierte Netze. Da viele Menschen Laptops und/oder WLAN-fähige Mobiltelefone und Tablets besitzen, ist das viel praktischer. Für den Internetzugang kommen entsprechend oft WLAN-DSL-Router zum Einsatz, die eine Verbindung zwischen dem Internet und den Endgeräten vermitteln.
- In einem Unternehmen wird viel im Außendienst oder im Homeoffice gearbeitet. Die Belegschaft ist mit Notebooks ausgestattet und kommt nur gelegentlich in die Firmenzentrale.
- Eine Firma zieht in ein denkmalgeschütztes Haus ein, an dessen Bausubstanz nichts geändert werden darf – an das Verlegen von Kabelkanälen oder gar das Aufstemmen von Wänden für die Vernetzung ist nicht zu denken.
- Zwischen zwei Gebäuden eines Unternehmens verläuft eine öffentliche Straße; für die Überbrückung durch ein Kabel müsste ein langfristiges Genehmigungsverfahren mit ungewissem Ausgang eingeleitet werden.
- Auf LAN-Partys (Treffen für Netzwerkspiele), Messen, Kongressen oder ähnlichen Veranstaltungen müssen Unmengen von Computern für kurze Zeit vernetzt werden.

8 Ganz und gar binär ist das Morsealphabet übrigens nicht: Neben *Lang* und *Kurz* muss die Pause zwischen zwei Zeichen als drittes mögliches Signal betrachtet werden, da die einzelnen Zeichen (übrigens gemäß ihrer Häufigkeit in englischen Texten) aus unterschiedlich vielen Einzelsignalen bestehen.

Für den Betrieb drahtloser Netzwerke kommen die verschiedensten Übertragungsmethoden zum Einsatz. Sie lassen sich nach folgenden Kriterien unterscheiden oder für den praktischen Einsatz auswählen:

- Welche maximale Entfernung zwischen zwei Stationen muss überbrückt werden?
- Besteht zwischen den einzelnen Standorten Sichtkontakt, oder befinden sich Wände oder andere Hindernisse zwischen ihnen?
- Soll eine freie Funkfrequenz genutzt werden, oder kann es auch eine lizenzpflichtige sein (Letztere kann teuer werden)?
- Sind die vernetzten Geräte selbst stationär oder mobil?

Diese diversen Auswahlkriterien zeigen bereits, dass es so etwas wie »das« drahtlose Netz nicht gibt. Für jeden Anwendungszweck bieten sich verschiedene Lösungen an, die sorgfältig geprüft werden müssen.

Genau wie bei der verkabelten Konkurrenz lassen sich auch hier verschiedene Kategorien von Reichweiten unterscheiden. Das *WLAN* (Wireless LAN, auch *WiFi* genannt) nach IEEE 802.11 ist ein drahtloses Netz für den Nahbereich, also für die Vernetzung innerhalb einer einzelnen Institution. Das *WWAN* (Wireless Wide Area Network) dagegen ist ein drahtloses Fernnetzwerk. Dazu zählen unter anderem Satellitenverbindungen.

In diesem Abschnitt wird nur das 802.11-kompatible WLAN beschrieben, da es sich seit seiner Einführung 1997 sehr schnell verbreitet hat und heute von allen Wireless-Technologien am häufigsten eingesetzt wird. 802.11 besteht aus mehreren Unterstandards, die sich in den Punkten Frequenzspektrum, Übertragungsrate und Funktechnologie unterscheiden. Sie alle werden jedoch über Funk betrieben; eine ursprünglich ebenfalls spezifizierte Infrarotvariante hat sich nicht durchgesetzt. Infrarot wird größtenteils für den drahtlosen Anschluss von Peripheriegeräten wie Mäusen oder Tastaturen verwendet. Tabelle 5.4 zeigt eine Übersicht über die wichtigsten gebräuchlichen 802.11-Varianten.

Standard	Frequenzbereich	Übertragungsrate	Funktechnik
802.11	2,4 GHz	1 oder 2 MBit/s	FHSS/DSSS
802.11a	5 GHz	bis zu 54 MBit/s	OFDM
802.11b	2,4 GHz	5,5/11/22 MBit/s	HR/DSSS
802.11g	2,4 GHz	bis zu 54 MBit/s	OFDM
802.11n	2,4 und 5 GHz	bis zu 600 MBit/s	MIMO
802.11ac	5 GHz	1 GBit/s	MU-MIMO

Tabelle 5.4 Verschiedene Varianten von IEEE 802.11

Die Trägerfrequenz von 2,4 GHz wird vor allem deshalb am häufigsten verwendet, weil sie nicht lizenzpflichtig ist. Es handelt sich nämlich um diejenige Frequenz, mit der Mikrowellenherde arbeiten, da diese Wellenlänge Wassermoleküle am effektivsten erhitzt.

Die diversen Funkverfahren arbeiten alle mit verschiedenen Varianten der Frequency-Hopping-Methode, die auch im Mobilfunk eingesetzt wird: Nach einem bestimmten Schema werden die Funkwellen über mehrere Frequenzen übertragen, die mehrmals in der Sekunde wechseln. Dies ist erheblich weniger störanfällig als die Verwendung einer einzelnen Frequenz. Die grundlegende Technik wurde Mitte der 1930er-Jahre von der österreichischen Schauspielerin *Hedy Lamarr* erfunden. Ihr damaliger Ehemann war Rüstungsfabrikant, und diese Funktechnik sollte helfen, Torpedos der Alliierten fernzusteuern, ohne dass die Signale abgefangen und verfälscht werden konnten. Im Einzelnen werden folgende Verfahren unterschieden:

- *FHSS* (*Frequency Hopping Spread Spectrum*): Die Frequenzen wechseln nach einem zufälligen Muster.
- *DSSS* (*Direct Sequence Spread Spectrum*): Es werden erheblich mehr Einzelfrequenzen verwendet; die Verteilung erfolgt nach einem komplexen mathematischen Verfahren.
- *HR/DSSS* (*High Rate/Direct Sequence Spread Spectrum*): Entspricht DSSS mit speziellen Erweiterungen, die eine höhere Übertragungsrate ermöglichen.
- *OFDM* (*Orthogonal Frequency Division Multiplexing*): Jeder Kanal wird in mehrere Teilkanäle unterteilt, die Signale werden über alle Teilkanäle parallel übertragen. Aus diesem Grund ist OFDM das Übertragungsverfahren mit der höchsten Datenrate, andererseits aber auch das aufwendigste, sodass die entsprechende Hardware noch vor wenigen Jahren vergleichsweise teuer war.
- *MIMO* (*Multiple Input/Multiple Output*): Im Wesentlichen eine nochmals verbesserte OFDM-Variante, die wiederum erheblich höhere Übertragungsraten ermöglicht. Die Datenübertragung kann gleichzeitig über mehrere Frequenzbänder erfolgen. Eine neuere Variante namens *MU-MIMO* (*Multi-User MIMO*) begünstigt den gleichzeitigen Netzwerkzugriff mehrerer User durch etwas komplexere Signalverarbeitungsvorgänge.

Der größte Teil der Wireless-LAN-Hardware, der momentan verkauft wird, basiert auf den Standards 802.11ac, 802.11n, 802.11b und 802.11g (die meisten Geräte unterstützen wahlweise mehrere). Die Preise für Hardware dieser Variante sind in den letzten Jahren stark gefallen. Ein WLAN-Adapter ist inzwischen ab etwa 20 € erhältlich, sowohl als PCI-Karte als auch als PCMCIA- oder USB-Adapter. Außerdem sind Notebooks (und meist auch Desktop-PCs) ab Werk standardmäßig mit einer WLAN-Schnittstelle ausgestattet. Vorreiter dürften das PowerBook und das iBook von Apple gewesen sein; Apple fördert diese Technologie unter dem Namen AirPort seit vielen Jahren.

Als Netzzugangsverfahren in 802.11-Netzen kommt CSMA/CA zum Einsatz (*Carrier Sense Multiple Access with Collision Avoidance*) – wie der Name vermuten lässt, werden Datenkolli-

sionen von vornherein vermieden. Anders als bei CSMA/CD sendet eine Station, die ein freies Übertragungsmedium (in diesem Fall den entsprechenden Funkkanal) vorfindet, nicht einfach ihre Daten, sondern eine Sendeanforderung (RTS). Daraufhin warten andere sendebereite Stationen, und die erste Station, die das RTS gesendet hat, sendet ihre Daten, nachdem ihr die Empfängerstation ihre Empfangsbereitschaft (CTS) signalisiert hat. Abgeschlossen wird die Datenübertragung durch ein ACK-Signal, und daraufhin kann die nächste Station ihren Sendewunsch bekannt geben.

Das einfachste denkbare 802.11-WLAN besteht nur aus mehreren Rechnern mit entsprechender Schnittstelle, die auf direktem Weg miteinander kommunizieren. Ein solcher Aufbau wird als *Basic Service Set* (BSS) bezeichnet. Die Entfernung zwischen zwei beliebigen Stationen darf die maximale Reichweite des Funksignals nicht überschreiten, weil jede Station die Signale nur senden und empfangen, aber nicht verstärken und weiterleiten kann. Da ein solches Netzwerk nicht mit anderen Netzen kommunizieren kann, wird es als *unabhängiges BSS* (Independent BSS oder kurz IBSS) bezeichnet. Derartige Netzwerke sind sinnvoll für die sogenannte *Ad-hoc-Vernetzung* temporärer Zusammenkünfte wie Messen oder LAN-Partys.

Ein wenig komplexer wird der Aufbau eines BSS, wenn ein *Access Point* hinzugefügt wird. Im Grunde funktioniert ein Access Point wie ein Ethernet-Hub, denn sobald er vorhanden ist, kommunizieren die Stationen nicht mehr direkt miteinander, sondern senden die Frames an den Access Point, der sie an den gewünschten Empfänger weitergibt. Die Identifikation der einzelnen Stationen erfolgt wie bei Ethernet anhand einer 48 Bit langen MAC-Adresse. Ein BSS mit einem Access Point wird als *Infrastruktur-BSS* bezeichnet. Für die Reichweite des Netzes ist nur noch die Entfernung zwischen einer Station und dem Access Point ausschlaggebend.

Die wichtigste Aufgabe eines Access Point besteht in seiner Funktion als Bridge. Er verbindet das WLAN mit einem Backbone-Netzwerk – meistens Twisted-Pair-Ethernet. Auf diese Weise kann das WLAN mit stationären Teilen des Netzes verbunden werden oder Zugang zu Servern und Routern erhalten, ohne dass diese selbst mit WLAN-Schnittstellen ausgestattet werden müssten.

Im Übrigen bildet ein Verbund aus miteinander vernetzten Access Points (entweder ebenfalls über Funk oder über Ethernet) ein sogenanntes *Extended Service Set* (ESS). Eine Station kann sich innerhalb eines ESS frei bewegen, weil die Access Points einander darüber auf dem Laufenden halten, welche Stationen sich gerade in ihrem Bereich befinden. Eine Station kann immer nur genau mit einem Access Point verbunden sein; sobald das Signal eines anderen Access Point stärker wird als das des bisherigen, meldet die Station sich bei ihrem alten Access Point ab und bei dem neuen an. Auf diese Weise werden Frames immer über den jeweils aktuellen Access Point an eine Station gesendet.

Ein zusätzlicher Nutzen von Access Points besteht darin, dass sie in der Lage sind, Frames zu puffern, die an bestimmte Stationen adressiert sind. Gerade Notebooks schalten im Stand-

by-Modus oft auch die WLAN-Schnittstelle ab, um Strom zu sparen; sobald die Verbindung wieder aufgebaut wird, werden die zwischengespeicherten Frames ausgeliefert.

Das ESS-Modell wird immer häufiger für öffentlich verfügbare Netzwerkzugänge eingesetzt. In Bahnhöfen, Flughäfen oder Gaststätten stehen öffentlich zunehmend WLAN-Access-Points (auch *Hotspots* genannt) zur Verfügung, in die sich Notebooks und Mobilgeräte ohne Weiteres einwählen können. Mittlerweile werden sogar die ersten Innenstädte fast flächendeckend mit einander überlappenden Access Points ausgestattet. Irgendwann könnte ein ähnlich dichtes Netz entstehen, wie es die Mobilfunkzellen inzwischen bilden.

Eine der größten Herausforderungen beim Einsatz von Wireless-Technologien bleibt die Sicherheit. Es ist zwar auch nicht weiter schwierig, das Signal von Ethernet-Kabeln abzuhören, aber immerhin ist es vergleichsweise einfach, den physikalischen Zugang zu ihnen zu kontrollieren. Bei WLAN kann dagegen im Grunde genommen jeder die Signale mit einer kompatiblen Antenne auffangen und analysieren, um unberechtigt Informationen zu erhalten oder gar zu manipulieren. Das gilt umso mehr, als man die Grenzen der Funkreichweite niemals ganz genau auf die Größe des zu vernetzenden Gebäudes oder Geländes abstimmen kann; es ist also durchaus möglich, die Funkwellen außen zu empfangen.

Um ein Mindestmaß an Sicherheit zu gewährleisten, bot die ursprüngliche 802.11-Spezifikation eine optionale Verschlüsselung der Frames an. Allerdings ist diese Methode nicht besonders sicher; Sicherheitsexperten haben bereits bewiesen, dass die Verschlüsselung verhältnismäßig leicht zu knacken ist. Schon der Name dieser Technik, *WEP* (*Wired Equivalent Privacy*), sagt allzu deutlich aus, dass es nicht um mehr geht, als etwa dasselbe Maß an Sicherheit zu gewährleisten wie beim rein physikalischen Schutz verkabelter Netzwerke. Der Hauptverwendungszweck besteht auch gar nicht in der Geheimhaltung, sondern in der Abgrenzung eines Wireless-Netzes von benachbarten Netzen: Es ist ärgerlich, wenn jedes vorbeifahrende Fahrzeug, in dem sich zufälligerweise ein Laptop mit 802.11-Schnittstelle befindet, diesen vorübergehend automatisch ins Netz einbucht und wieder daraus verschwindet. Das lässt sich allerdings zuverlässiger verhindern, indem der Access Point mit einer Whitelist zugelassener MAC-Adressen konfiguriert wird.

Inzwischen stehen mit WPA und WPA2 (*WiFi Protected Access*) stark verbesserte WLAN-Verschlüsselungsverfahren zur Verfügung.

5.5 Datenfernübertragung

Nachdem im vorangegangenen Abschnitt die verschiedenen Formen der LAN-Vernetzung und der WANs über Standleitungen beschrieben wurden, sollen nun diverse Verfahren der Datenfernübertragung (DFÜ) geschildert werden. Wie bereits angesprochen, wurde DFÜ bereits eingesetzt, als sie lediglich der Punkt-zu-Punkt-Kommunikation zwischen einzelnen Rechnern über eine direkte Telefonverbindung diente. Heute geht es in der Regel darum, den

Zugang zu einem bestehenden Netzwerk oder (über einen kommerziellen Provider) zum Internet herzustellen.

Die erste Generation der DFÜ-Hardware, der umständliche und störanfällige Akustikkoppler, muss hier nicht mehr beschrieben werden. Auch Wählverbindungen über Modems oder digital über ISDN spielen eine immer kleinere Rolle und werden daher in diesem Buch nicht mehr behandelt. Die beiden wesentlichen Technologien sind heute die verschiedenen DSL-Dienste für den stationären Internetzugang und diverse mobile Netzwerkverbindungen. Diese beiden Zugangsverfahren werden im Folgenden dargestellt.

Eine Gemeinsamkeit aller DFÜ-Netzwerkverbindungen besteht in der Notwendigkeit, die Datenübertragung über diese Leitungen zu standardisieren und bestimmte Grundlagen für die Protokolle der Vermittlungsschicht zu schaffen. Dafür werden spezielle Protokolle verwendet, die den Netzzugang über relativ langsame serielle Leitungen ermöglichen. Das traditionelle Protokoll für die Vernetzung über Wählleitungen war SLIP (*Serial Line Interface Protocol*). Allerdings besitzt es eine Reihe organisatorischer und technischer Mängel und wurde deshalb weitgehend durch PPP (*Point-to-Point Protocol*) ersetzt.

PPP kümmert sich um die Authentifizierung nach der Einwahl, indem Username und Passwort übermittelt werden; anschließend verhandeln die beiden direkt miteinander verbundenen Punkte die eigentlichen Netzwerkdetails. Eine der wesentlichsten Fähigkeiten des Protokolls für Internetverbindungen besteht darin, dass der Einwahlknoten dem anwählenden Rechner automatisch eine IP-Adresse zuweisen kann, über die diese Netzwerkschnittstelle im gesamten Internet identifiziert wird.

Im Einzelnen erfolgen bei PPP also diese Schritte:

- Wird eine Wählleitung (analog oder ISDN) verwendet, stellt der lokale Rechner über die entsprechende Schnittstelle eine Telefonverbindung her; dies spielt in der Praxis keine große Rolle mehr. Bei DSL-Leitungen wird ebenfalls die Verbindung aktiviert, auch wenn man dies nicht als *Wählen* im klassischen Sinne bezeichnen kann.
- Der Einwahlknoten verlangt eine Authentifizierung, in der Regel in Form von Username und Passwort. Die meisten PPP-Implementierungen in modernen Betriebssystemen übermitteln diese Daten nach einmaliger Konfiguration automatisch, ohne dass Sie etwas tun müssen.
- Nachdem die Daten überprüft wurden, folgt entweder die Ablehnung des Zugriffs und der Verbindungsabbau, oder die Netzwerkparameter werden ausgehandelt. Auch wenn PPP als Netzzugangsgrundlage für alle möglichen Protokolle der Vermittlungsschicht dienen kann, wird heute fast nur noch TCP/IP aufgesetzt. Zu diesem Zweck weist der PPP-Knotenpunkt des Internetproviders der seriellen Verbindung auf der Einwahlseite eine IP-Adresse zu, eine im gesamten Internet einmalige Identifikationsnummer. Ihr Konzept wird im nächsten Abschnitt genau beschrieben.

5.5.1 DSL-Dienste

DSL ist die Abkürzung für *Digital Subscriber Line* (etwa »digitale Abonnement-Leitung«). Der Name soll verdeutlichen, dass es sich de facto um eine Standleitung anstelle einer Wählleitung handelt. Zur Einführung von DSL kam es, da es durch die allmähliche Verbesserung der Qualität von Telefonleitungen möglich wurde, Signale hoher Frequenz zu übertragen. Die meisten DSL-Dienste verwenden die klassischen Kupferleitungen der Telefongesellschaften, die allerdings immer häufiger durch Glasfaserleitungen ersetzt oder ergänzt werden.

DSL-Varianten

Es existieren zwei grundsätzliche Varianten von DSL: Bei *Symmetric DSL* (SDSL) sind die Übertragungsraten für ankommende und ausgehende Daten identisch, bei *Asymmetric DSL* (ADSL) ist die ankommende Übertragungsrate höher als die ausgehende.

Übliche DSL-Angebote wie T-DSL der Deutschen Telekom stellten ursprünglich eine Download-Rate von 1.024 KBit/s und eine Upload-Rate von 128 KBit/s zur Verfügung, teilweise sogar noch weniger. Aktuelle ADSL-Anschlüsse des klassischen Typs sind dagegen mit Download-Geschwindigkeiten von 2 bis 8 MBit/s ausgestattet. Auch die umgekehrte Datenrate wurde entsprechend vervielfacht. Die neueren Typen ADSL2 und ADSL2+ schaffen 16 beziehungsweise 25 MBit/s im Download, und die nochmals verbesserten Technologien VDSL und VDSL2 bringen es auf bis 52 MBit/s im Download und 16 MBit/s im Upload beziehungsweise 100 MBit/s in beide Richtungen. Technisch gesehen, handelt es sich bei VDSL2 also um einen SDSL-Standard.

Die Gebühren für DSL-Anschlüsse werden in der Regel nicht wie beim Telefonieren nach der Nutzungsdauer berechnet, sondern als sogenannte *Flatrate* für beliebig lange Onlinezeiten. Einige Provider verwenden allerdings eine Volumenbeschränkung, das heißt, ohne Aufpreis darf monatlich nur eine bestimmte Datenmenge transferiert werden.

Neben den DSL-Angeboten, die über normale Telefonleitungen laufen, werden seit einiger Zeit auch spezielle Lösungen angeboten. Eine davon ist die Internetverbindung über das Glasfaserkabel des Kabelfernsehens. Da dieses Kabel in Deutschland seit den 1980er-Jahren vor allem für das Passivmedium Fernsehen eingesetzt wurde, besaß es in seiner ursprünglichen Version keine Rückkanalfähigkeit. Es konnten Daten empfangen, aber nicht gesendet werden; noch nicht einmal die Anforderung einer URL konnte abgesetzt werden. Bei neueren Glasfaserleitungen, die speziell für den Anwendungszweck Internet und Kommunikation gelegt werden, ist dies natürlich anders.

Zum Anschließen von DSL wird an den TAE-Anschluss einer DSL-Verbindung ein sogenannter *Splitter* angeschlossen – eine Frequenzweiche, die die hochfrequenten DSL-Signale und die niedrigfrequenten normalen Telefonsignale voneinander trennt. Den Ausgang für die Telefonsignale bietet wiederum ein TAE-Anschluss, an den entweder ein Analogtelefon oder

ein NTBA angeschlossen wird, je nachdem, ob DSL mit einem Analog- oder mit einem ISDN-Telefonanschluss kombiniert wird.

Den Ausgang für die speziellen DSL-Signale bietet eine Twisted-Pair-Buchse vom Typ RJ-11. An diesen Anschluss wird in der Regel ein *DSL-Modem* angeschlossen, das dann über USB oder Twisted-Pair-Ethernet mit dem Computer verbunden wird. Natürlich ist die Bezeichnung DSL-Modem technisch gesehen Unfug. Bei DSL findet keinerlei Analog-Digital-Umwandlung statt. Dennoch ist der Begriff *Modem* für das Gerät sehr verbreitet, weil es den Computer mit einer seriellen Fernleitung verbindet. Beim Anschluss über eine Ethernet-Schnittstelle kommt eine spezielle PPP-Variante namens *PPPoE* (*PPP over Ethernet*) zum Einsatz.

Anstelle der reinen DSL-Modems zum Anschluss eines einzelnen Rechners werden inzwischen meist *DSL-Router* verwendet, die gleich einem gesamten Netzwerk per Ethernet, WLAN oder beidem den Internetzugang bereitstellen. Inzwischen erlauben auch die meisten Provider den Einsatz solcher Router, früher waren die günstigsten Tarife dagegen vielfach auf einen Einzelrechner beschränkt. Die jüngste Generation von DSL-Routern verzichtet auch gleich auf einen externen Splitter und bringt diese Funktion selbst mit; eventuelle Analog- oder ISDN-Telefone werden direkt an den Router angeschlossen.

5.5.2 Internetzugänge über Mobilfunk

Neben den hier behandelten stationären DFÜ-Verbindungen werden auch diejenigen über Mobilfunk immer wichtiger. Über die seit den 1990er-Jahren errichteten GSM-Netze (in Deutschland beispielsweise D1, D2, E-Plus etc.) kam ursprünglich vor allem ein Verfahren namens *GPRS* (*General Packet Radio Service*) zum Einsatz; es wurde mehrfach in Details verbessert, bietet aber noch immer keine allzu hohen Datentransferraten – sie liegen bei 53,6 KBit/s im Download und 26,8 KBit/s im Upload. Neuere Verfahren sind zum Teil erheblich schneller:

- *EDGE*: Download 217,6 KBit/s, Upload 108,8 KBit/s.
- *UMTS*: Download und Upload 384 KBit/s.
- *HSPA*: Download (HSDPA) 7,2 MBit/s, Upload (HSUPA) 1,4 MBit/s.
- *LTE* (*Long-Term Evolution*) ist der zurzeit noch aktuelle Standard. In der ersten Ausbaustufe wurden Übertragungsraten bis 100 MBit/s realisiert, die später auf 150 MBit/s erhöht wurden. Seit 2014 sind im Rahmen der nächsten Ausbaustufe (LTE-Advanced) auch Übertragungen im niedrigen Gigabit-Bereich möglich.
- Der neueste Standard *5G* bietet Übertragungsraten von mindestens 2,8 GBit/s im Download. Noch schnellere Übertragungsraten in diesem Rahmen werden allmählich hinzukommen. In Deutschland wurden im Jahr 2019 die 5G-Frequenzen an interessierte Firmen versteigert; in den Jahren 2000, 2010 und 2015 fanden ähnliche Versteigerungen für die Vorgängernetze statt.

Auch für die älteren Datenübertragungsstandards gab es alternativ Generationsnummern: 1G bezeichnet historische Mobilfunknetze vor der Handy-Revolution der 1990er-Jahre, 2G ist der GSM-Mobilfunk, 2.5G ist GPRS, und 3G steht für EDGE, UMTS und HSPA. LTE wurde im Marketing zwar mit dem Begriff 4G beworben, streng genommen handelte es sich jedoch um 3.9G als Weiterentwicklung von 3G. Erst LTE-Advanced erfüllte die volle 4G-Spezifikationen.

Mobilfunkzugänge können entweder für Web-, E-Mail- und andere Netzwerksoftware auf dem Mobiltelefon selbst verwendet werden, oder aber das Handy dient – beispielsweise über Bluetooth – als Mobilfunkmodem für einen Laptop oder ein Nur-WLAN-Tablet (der Fachbegriff dafür lautet *Tethering*). Speziell für UMTS oder HSPA gibt es auch eigenständige Netzwerkzugangsgeräte, die per USB an den Rechner angeschlossen werden und beinahe überall einen Internetzugang mit annehmbarer Geschwindigkeit bieten.

Um die Vorteile eines solchen Anschlusses wirklich zu nutzen, sollte aus Kostengründen ein Datenflatrate-Vertrag mit dem Mobilfunkanbieter abgeschlossen werden. Beachten Sie aber, dass diese Zugänge sehr oft eine Transfervolumenbeschränkung enthalten – Flatrates mit beliebigem Volumen sind nach wie vor sehr teuer. Sobald das Volumen für den entsprechenden Monat aufgebraucht ist, steht in der Regel nur noch eine langsamere Verbindung zur Verfügung.[9] Glücklicherweise verfügen praktisch alle modernen Smartphones (und Tablets sowieso) auch über WLAN, sodass zu Hause oder am Arbeitsplatz meist keine Datenübertragung über Mobilfunk erforderlich ist.

Für den Urlaub oder ähnliche Gelegenheiten werden auch Prepaid-SIM-Karten oder USB-Sticks angeboten. Auf keinen Fall sollten Sie den Fehler machen, im Nicht-EU-Ausland ungeprüft das sogenannte *Daten-Roaming* Ihres Mobiltelefons zu aktivieren. Innerhalb der EU wurden Zusatzkosten dafür im Juli 2017 nach langwierigen Verhandlungen und halbherzigen Preissenkungen endgültig abgeschafft, aber für das sonstige Ausland gilt das keineswegs.[10]

5.6 Die TCP/IP-Protokollfamilie

Nach einigen lahmen Versuchen, das OSI-Referenzmodell durch konkrete Protokolle tatsächlich zu implementieren, bemerkte man letzten Endes, dass die bereits Jahre zuvor entwickelten Internetprotokolle hervorragend als flexible, skalierbare und universelle Netzwerkprotokollfamilie einsetzbar sind. Die rasante Ausbreitung des Internets und die freie

9 Die Telekom kündigte im Frühjahr 2013 an, solche Regelungen künftig auch für DSL-Anschlüsse einführen zu wollen. Für Empörung sorgte dabei insbesondere, dass sie ihre eigenen Web-TV- und Unterhaltungsangebote nicht in dieses Volumen einzurechnen gedachte (mögliche Verletzung der Netzneutralität). Nach sehr viel Kritik und Spott gab die Telekom die Pläne bereits im Dezember 2013 wieder auf und strich die Drossel-Klausel aus allen Festnetzverträgen. Auch andere Provider haben dergleichen inzwischen aufgegeben.

10 Ich selbst kam im Herbst 2017 das erste Mal in den Genuss dieser Mobilfunkreisefreiheit – ironischerweise im Vereinigten Königreich, wo sie aufgrund des EU-Austritts inzwischen nicht mehr gilt.

Verfügbarkeit sorgten dafür, dass diese Protokolle heute häufiger als jeder andere Protokollstapel eingesetzt werden.

Abbildung 5.3 zeigt eine konkrete Version des zuvor bereits vorgestellten TCP/IP-Protokollstapels: Auf jeder Ebene sind einige der Protokolle zu erkennen, die dort arbeiten können. Die meisten davon werden in den folgenden Abschnitten genau erläutert; die Netzzugangsprotokolle der untersten Schicht wurden ebenfalls bereits vorgestellt. Ganz unten habe ich zusätzlich einige Beispiele für die Hardware angegeben, auch wenn sie kein Teil des eigentlichen TCP/IP-Stapels ist.

Zwischen der Hardware und dem Netzzugang auf der einen und den anwendungsorientierten Protokollen auf der anderen Seite befinden sich die Protokolle der Vermittlungs- und der Transportschicht. Insgesamt werden alle Protokolle, die auf den verschiedenen Ebenen eines Schichtenmodells zusammenarbeiten, als *Protokollstapel* oder auch *Protokollfamilie* bezeichnet. Allerdings konzentriert sich der Schwerpunkt von TCP/IP auf die beiden mittleren Ebenen des Internetprotokollstapels. Sie können zum einen auf fast jeden beliebigen Netzzugang aufsetzen, zum anderen wurde beinahe jede ernst zu nehmende Netzwerkanwendung inzwischen für diesen Protokollstapel umgesetzt – abgesehen von den klassischen Internetanwendungen, die ohnehin dafür geschrieben wurden.

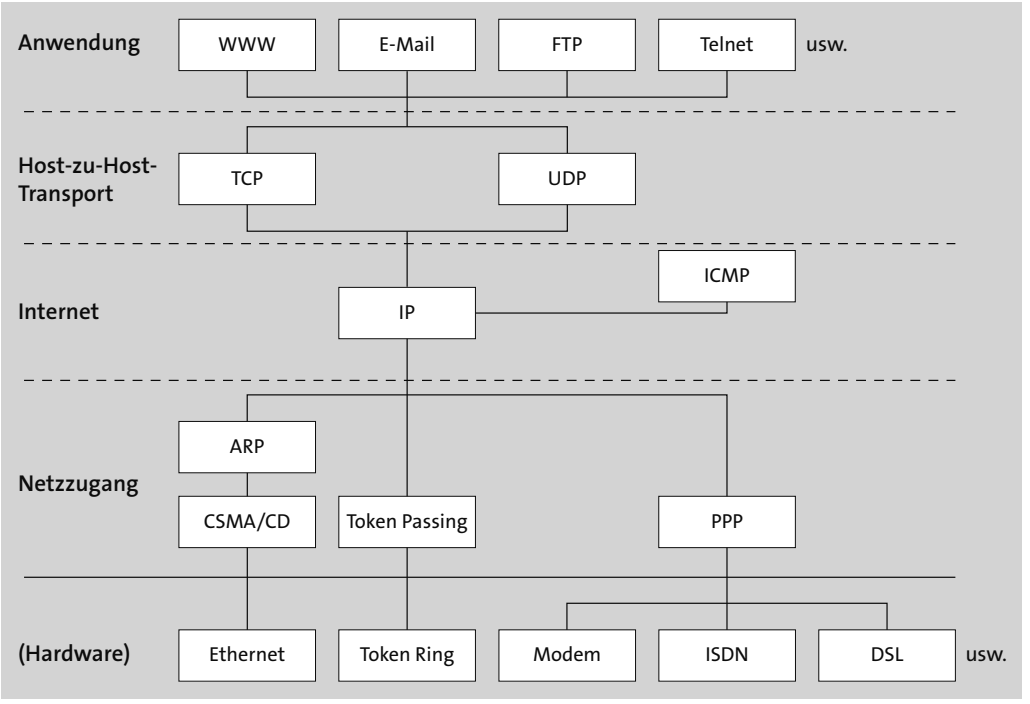


Abbildung 5.3 Der TCP/IP-Protokollstapel

Die Protokolle der mittleren Schichten sind dafür verantwortlich, dass Daten zuverlässig über verschiedene Teilnetze oder Netzwerksegmente hinweg übertragen werden können oder auch über Netze, die verschiedene Hardware oder Netzzugangsverfahren verwenden:

- Die Protokolle der Internetschicht regeln die Adressierung der Rechner und die Übertragung der Daten an den korrekten Rechner im Netzwerk. Darüber hinaus kümmern sie sich darum, dass Daten bei Bedarf in andere Teilnetze weitergeleitet werden, übernehmen also das sogenannte *Routing*.
- Auf der Host-zu-Host-Transportschicht werden die Daten in Pakete unterteilt und mit der Information versehen, welche Anwendung auf dem einen Host diese Daten an welche Anwendung auf dem anderen sendet.

Die Bezeichnung *TCP/IP* kombiniert die Namen der beiden wichtigsten Bestandteile des Protokollstapels: das *Internet Protocol* (IP) auf der Internetschicht und das *Transmission Control Protocol* (TCP), das am häufigsten verwendete Protokoll der Transportebene. In den folgenden Abschnitten werden diese Protokolle näher vorgestellt, anschließend wird die technische Seite einiger wichtiger Internetanwendungsprotokolle beleuchtet.

5.6.1 Netzzugang in TCP/IP-Netzwerken

Die unterste Ebene des Internetschichtenmodells ist der Netzzugang, nicht die Netzwerkhardware. Dies garantiert, dass sich die Internetprotokolle auf fast jeder beliebigen Hardware implementieren lassen, und in der Tat ist das geschehen: In allen Formen von LANs wie Ethernet oder Token Ring, in WANs über Wähl- und Standleitungen wie auch über die meisten Formen drahtloser Netze – überall laufen diese Protokolle. Das ist ein weiterer guter Grund dafür, dass sich die Protokolle des Internets als Standard für die Netzwerkkommunikation durchsetzen konnten.

Im Grunde wird innerhalb der Spezifikation der TCP/IP-Protokolle nicht einmal der Netzzugang im OSI-Sinn beschrieben, sondern lediglich die Zusammenarbeit des IP-Protokolls, das sich innerhalb des Internetprotokollstapels um Adressierung und Routing kümmert, mit verschiedenen Netzzugangsverfahren.

An dieser Stelle sollen nur zwei der wichtigsten Internetnetzzugangsverfahren genannt werden: das für den Zugriff auf Ethernet verwendete *Address Resolution Protocol* (ARP) und das *Point-to-Point Protocol* (PPP), das für serielle Verbindungen über Modem, ISDN oder DSL eingesetzt wird. PPP wurde in diesem Kapitel bereits beschrieben, daher folgt das Wichtigste zu ARP.

Das Address Resolution Protocol, erstmals beschrieben in RFC 826, übernimmt – kurz gesagt – die Umsetzung der vom Netzwerkadministrator vergebenen IP-Adressen in die vorgegebenen Hardwareadressen der Netzwerkschnittstellen.

Da die IP-Adresse den einzelnen Hosts willkürlich zugeteilt wird, kann sie auf der Netzzugangsschicht nicht bekannt sein: Auf einer bestimmten Schicht eines Protokollstapels werden die Steuerdaten der höher gelegenen Ebenen nicht ausgewertet, sondern als gewöhnliche Nutzdaten betrachtet. Deshalb kann beispielsweise eine Netzwerkkarte oder ein Hub nicht anhand der IP-Adresse entscheiden, für welche Station ein Datenpaket bestimmt ist; die Netzwerkhardware nimmt diese Adresse nicht einmal wahr.

Nachdem die IP-Software auf einem Host oder Router anhand der Empfänger-IP-Adresse festgestellt hat, dass die Daten überhaupt für das eigene Netz bestimmt sind, wird der ARP-Prozess gestartet, um diese IP-Adresse in die MAC-Adresse der Empfängerschnittstelle umzusetzen. Zu diesem Zweck sendet ein Rechner, der das ARP-Protokoll ausführt (beinahe jeder Rechner, der TCP/IP über Ethernet betreibt), ein sogenanntes *Broadcast-Datenpaket* in das Netzwerk. Es handelt sich um ein Datenpaket mit einer speziellen Empfängeradresse, das an alle Rechner im Netzwerk übertragen wird. Der Rechner, der seine eigene IP-Adresse im Inhalt dieses Pakets erkennt, antwortet als Einziger auf diese Anfrage und versendet seine eigene MAC-Adresse. Auf diese Weise wird ermittelt, für welchen Rechner das Datenpaket bestimmt ist. In Ausnahmefällen kann ein Rechner auch die MAC-Adressen anderer Stationen zwischenspeichern und in Vertretung antworten.

5.6.2 IP-Adressen, Datagramme und Routing

Auf der Internet- oder Vermittlungsschicht des Internetprotokollstapels arbeitet das Internet Protocol (IP). Als *Internet* wird in diesem Zusammenhang jedes Netzwerk bezeichnet, das diese Protokollfamilie verwendet. Das verdeutlicht den Umstand, dass die Internetprotokolle dem Datenaustausch über mehrere physikalische Netzwerke hinweg dienen können. Spezielle Rechner, die mindestens zwei Netzwerkschnittstellen besitzen, leiten die Daten zwischen diesen Netzen weiter. Sie werden als *IP-Router* oder *IP-Gateways* bezeichnet. Im engeren Sinn ist ein Router ein Rechner, der Daten zwischen zwei Netzen des gleichen physikalischen Typs weiterleitet; ein Gateway verbindet dagegen zwei physikalisch verschiedene Netze oder arbeitet gar auf Anwendungsebene (*Application Level Gateway*). Die beiden Begriffe werden jedoch oft synonym verwendet.

Eine IP-Adresse des klassischen Typs – der Version IPv4 gemäß RFC 791 – ist eine 32 Bit lange Zahl. Sie wird üblicherweise als vier durch Punkte getrennte Dezimalzahlen zwischen 0 und 255 geschrieben. Allerdings ist die Logik, die einer solchen Adresse zugrunde liegt, besser verständlich, wenn diese binär notiert wird:

Eine typische IP-Adresse wäre etwa `11000010000100010101000111000001`, in 8-Bit-Gruppen getrennt, ergibt sich daraus `11000010 00010001 01010001 11000001`; dies lautet in der gängigen Schreibweise dann `194.17.81.193`.

IP-Adressklassen

IP-Adressen bestehen aus zwei Komponenten: dem Netzwerkteil und dem Hostteil. Der Netzwerkteil gibt an, in welchem Netz sich der entsprechende Rechner befindet, während der Hostteil den einzelnen Rechner innerhalb dieses Netzes identifiziert.

Es gibt verschiedene Arten von IP-Adressen, die sich bezüglich der Länge des Netzwerk- beziehungsweise Hostteils voneinander unterscheiden. Traditionell wurden die verfügbaren Adressen in feste Klassen unterteilt. Bereits 1993 wurde die Klasseneinteilung aufgegeben und durch CIDR ersetzt (RFC 1518 und 1519), da sie für die rasant steigende Anzahl von Hosts viel zu unflexibel war. Aus diesem Grund wurde das CIDR-Verfahren entwickelt, das dynamisch zwischen dem Netzwerk- und dem Hostteil einer Adresse trennt. Dennoch sollen an dieser Stelle zuerst die ursprünglichen Klassen vorgestellt werden, denn auf diese Weise wird vieles leichter verständlich. Beachten Sie aber, dass IP-Adressen heutzutage immer über CIDR vergeben werden. Dabei wird neben der IP-Adresse auch die im Folgenden definierte Teilnetzmaske mitgeliefert, die in der historischen Klassenlogik nicht benötigt wurde.

Zu welcher Klasse eine IP-Adresse gehörte, zeigte sich an den Bits, die am weitesten links standen:

- Klasse A: Das erste Bit ist 0, folglich liegt die erste 8-Bit-Gruppe zwischen 0 und 127.
- Klasse B: Die ersten beiden Bits lauten 10, die erste Gruppe liegt im Bereich 128 bis 191.
- Klasse C: Die ersten drei Bits sind 110, sodass die erste Gruppe zwischen 192 bis 223 liegt.
- Klasse D: Die ersten vier Bits sind 1110, die Adressen reichen von 224 bis 239.

Die restlichen Adressen, die mit 240 bis 255 anfangen, wurden weder im Klassenbereich noch über CIDR vergeben und sind für zukünftige Anwendungszwecke reserviert. Diejenigen mit der Anfangssequenz 11110 (Startbyte 240 bis 247) wurden manchmal trotzdem als *Klasse E* bezeichnet.

Je nach Klasse ist der Teil, der das Netzwerk kennzeichnet, unterschiedlich lang, entsprechend existieren unterschiedlich viele Netze der verschiedenen Klassen. Die Bits, die ganz rechts in der Adresse stehen und nicht zum Netzwerkteil gehören, sind die Host-Bits. Je nach Länge des Netzwerkteils bleiben unterschiedlich viele Bits für den Hostteil übrig, sodass die Höchstzahl der Rechner in einem Netz variiert.

Tabelle 5.5 zeigt die wichtigsten Informationen zu den einzelnen Klassen im Überblick. In der Spalte »Netzwerk-Bits« stehen jeweils zwei Werte. Der erste stellt die Anzahl der Bits dar, die insgesamt den Netzwerkteil bilden. Da die Grenzen zwischen Netzwerk- und Hostteil an den Byte-Grenzen verlaufen, handelt es sich je nach Klasse um 1 bis 3 Byte. Da jedoch die Bits am Anfang der Adresse – wie zuvor gezeigt – die Klasse angeben, besteht die praktisch nutzbare Netzwerkangabe nur aus 7, 14 beziehungsweise 21 Bit. Der Rest der Adresse bildet den Hostteil, der je nach Klasse unterschiedlich groß ausfällt.

Klasse	Adressbereich	Netzwerk-Bits	Host-Bits	Anzahl Netze	Adressen pro Netz
A	0.0.0.0 bis 127.255.255.255	8 (7)	24	128	16,7 Mio.
B	128.0.0.0 bis 191.255.255.255	16 (14)	16	16.384	65.536
C	192.0.0.0 bis 223.255.255.255	24 (21)	8	2.097.152	256
D	224.0.0.0 bis 239.255.255.255	spezieller Bereich der Multicast-Adressen			

Tabelle 5.5 Die IP-Adressklassen

Innerhalb eines einzelnen Netzes – egal welcher Klasse – stehen die erste und die letzte mögliche Adresse nicht als Hostadressen zur Verfügung: Die niedrigste Adresse identifiziert das gesamte Netz als solches nach außen hin, aber keinen speziellen Host, die höchste ist die sogenannte *Broadcast-Adresse*: Werden Datenpakete innerhalb des Netzes an diese Adresse gesendet, werden sie von jedem Host empfangen.

Zum Beispiel bilden die Adressen, die mit 18.x.x.x beginnen, das Klasse-A-Netzwerk 18.0.0.0 mit der Broadcast-Adresse 18.255.255.255 und Hostadressen von 18.0.0.1 bis 18.255.255.254. Dieses Netz kann theoretisch bis zu 16.777.214 Hosts beherbergen (2^{24}–2).

Die Adressen, die mit 162.21.x.x anfangen, befinden sich in dem Klasse-B-Netzwerk 162.21.0.0, dessen Broadcast-Adresse 162.21.255.255 lautet. Es kann bis zu 65.534 Hosts (2^{16}–2) mit den Adressen 162.21.0.1 bis 162.21.255.254 enthalten.

Ein letztes Beispiel: Adressen, die mit 201.30.9.x beginnen, liegen in dem Klasse-C-Netz 201.30.9.0 mit der Broadcast-Adresse 201.30.9.255; die 254 möglichen Hostadressen (2^8–2) sind 201.30.9.1 bis 201.30.9.254.

Die sogenannten *Multicast-Adressen* der Pseudoklasse D nehmen eine Sonderstellung ein: Eine Multicast-Gruppe ist eine auf beliebige Netze verteilte Gruppe von Hosts, die sich dieselbe Multicast-IP-Adresse teilen. Dies ermöglicht einen erheblich ökonomischeren Versand von Daten, da sie nicht mehr je einmal pro empfangenden Host versendet werden, sondern nur noch kopiert werden müssen, wenn Empfängerrechner in unterschiedlichen Teilnetzen liegen. Aus diesem Grund ist Multicasting eine zukunftsträchtige Technologie für datenintensive Anwendungen wie etwa Videokonferenzen. Im Gegensatz dazu werden die individuellen Hostadressen als *Unicast-Adressen* bezeichnet.

Die Verteilung der IP-Adressen

Alle Adressen des IPv4-Adressraums werden von der *Internet Assigned Numbers Association* (*IANA*) verwaltet. Falls Sie jedoch für bestimmte Anwendungen in Ihrem Unternehmen eine oder mehrere feste IP-Adressen benötigen, sollten Sie sich üblicherweise an einen Internetprovider und nicht an die IANA selbst wenden.

Die 128 Netze der Klasse A sind bereits alle vergeben, in der Regel an große internationale Unternehmen aus dem Elektronik- und Computerbereich sowie an US-amerikanische Staats-, Militär- und Bildungsinstitutionen. Beispielsweise gehört das Netz 17.0.0.0 der Firma Apple, 18.0.0.0 dem Massachusetts Institute of Technology (MIT) und 19.0.0.0 der Ford Motor Company.

Die 16.384 Klasse-B-Netze sind ebenfalls weitgehend vergeben, insbesondere an US-amerikanische Unternehmen und Internetprovider.

Die mehr als zwei Millionen Netze der Klasse C schließlich sind inzwischen ebenfalls überwiegend belegt. Die meisten von ihnen gehören Unternehmen und Internetprovidern, die nicht in den USA ansässig sind, sondern etwa in Europa oder Asien. Da solche Institutionen oft mehr als 254 Hosts in ihrem Netz betreiben, wird ihnen häufig ein größerer Block aufeinanderfolgender Klasse-C-Netze zugewiesen.

Die aktuelle Verteilung der IPv4-Adressen können Sie auf der Website der IANA unter *https://www.iana.org/assignments/ipv4-address-space* einsehen.

Als das Konzept der IP-Adressen entstand, konnte niemand auch nur ansatzweise erahnen, welche Dimensionen das Internet einmal annehmen würde. Deshalb glaubte man ursprünglich, man könne es sich leisten, den Adressraum relativ großzügig aufzuteilen – bedenken Sie etwa, dass die Hälfte des Adressraums für die überaus ineffektiven Klasse-A-Adressen vergeudet wird. Um die drohende Verknappung der IP-Adressen zu verhindern oder zumindest hinauszuzögern, bis eine Alternative gefunden würde, wurden einige Adressbereiche zur Verwendung in privaten Netzwerken freigegeben, die nicht (oder nicht direkt) mit dem Internet verbunden sind. Es handelt sich um die folgenden Blöcke:

- das Klasse-A-Netz 10.0.0.0
- die 16 Klasse-B-Netze 172.16.0.0 bis 172.31.0.0
- die 256 Klasse-C-Netze 192.168.0.0 bis 192.168.255.0

Ein weiterer Block, der erst später freigegeben wurde, ist das Klasse-B-Netz 169.254.0.0, das einem besonderen Verwendungszweck vorbehalten ist: Moderne TCP/IP-Implementierungen in fast allen Betriebssystemen verwenden dieses Netz für *link local* – eine Möglichkeit, sich automatisch selbst IP-Adressen zuzuweisen, falls wider Erwarten keine Verbindung zu einem DHCP-Server hergestellt werden kann, der eigentlich für die automatische Zuweisung von Adressen zuständig wäre.

Zu guter Letzt existieren noch einige Netze mit anderen speziellen Bedeutungen:

- Die Adresse 0.0.0.0 kann innerhalb eines Netzes verwendet werden, um sich auf das aktuelle Netz selbst zu beziehen.
- Das Klasse-A-Netz 127.0.0.0 beherbergt den sogenannten *Loopback-Bereich*: Über das Loopback-Interface, eine virtuelle Netzwerkschnittstelle mit der Adresse 127.0.0.1, kann ein Host mit sich selbst Netzwerkkommunikation betreiben. Dies ist zum Beispiel nütz-

lich, um während der Programmierung von Client-Server-Anwendungen sowohl das Client- als auch das Serverprogramm auf dem lokalen Host laufen zu lassen.

- Schließlich wird die Adresse 255.255.255.255 als universelle Broadcast-Adresse verwendet: Ein Datenpaket, das an diese Adresse gesendet wird, wird wie beim normalen Broadcast von allen Hosts im Netzwerk empfangen. Nützlich ist diese Einrichtung für Schnittstellen, die ihre IP-Adresse dynamisch beziehen, da sie bei Inbetriebnahme in der Regel noch nicht einmal wissen, in welchem Netz sie sich eigentlich befinden. Auf diese Weise erhalten sie überhaupt erst die Möglichkeit, die Zuteilung einer Adresse anzufordern.

Die Vergabe der privaten Adressbereiche ist in RFC 1918 geregelt; die Festlegung der anderen speziellen Adressbereiche findet sich in RFC 3330.

Supernetting, Subnetting und CIDR

In der neueren Entwicklungsgeschichte des Internets hat sich herausgestellt, dass die traditionellen Adressklassen nicht für alle Anwendungsbereiche flexibel genug sind. Deshalb wurde ein neues Schema entwickelt, das die Trennlinie zwischen Netz- und Hostteil der Adressen an einer beliebigen Bit-Grenze ermöglicht. Das in RFC 1519 beschriebene Verfahren heißt *Classless Inter-Domain Routing* (CIDR).

Die folgenden beiden Anwendungsbeispiele verdeutlichen typische Probleme mit der alten Klassenlogik, die mithilfe von CIDR gelöst werden können:

- Ein Unternehmen besitzt das Klasse-B-Netzwerk 139.17.0.0. Es wäre jedoch wünschenswert, dass die vier Filialen des Unternehmens jeweils unabhängige Netze betreiben könnten. Dazu soll das vorhandene Netz in vier Teile untergliedert werden – ein Fall für das sogenannte *Subnetting*.
- Ein vor Kurzem neu gegründeter europäischer Internetprovider hat die 1.024 Klasse-C-Netze 203.16.0.0 bis 203.19.255.0 erhalten. Das Unternehmen möchte diese Netze als ein großes Netz verwalten, da das die dynamische Zuteilung an Hosts bei der Einwahl erheblich vereinfacht. Eine solche Zusammenfassung von Netzen wird *Supernetting* genannt.

Das Prinzip von CIDR basiert darauf, dass die traditionellen Byte-Grenzen zwischen Netz- und Hostteil völlig aufgehoben werden. Deshalb ist die Größe des Netzes bei einem CIDR nicht mehr am Beginn der Adresse zu erkennen. Stattdessen wird die Anzahl der Bits, die den Netzwerkteil der Adresse bilden, durch einen Slash getrennt hinter der Netzwerkadresse notiert. Zum Beispiel wird das Klasse-A-Netz 14.0.0.0 zu 14.0.0.0/8.

Eine alternative Darstellungsform für die Grenze zwischen Netz- und Hostteil bei CIDR-Adressen – insbesondere in der IP-Konfiguration der meisten Betriebssysteme – stellt die Teilnetzmaske (*Subnet Mask*) dar. In dieser Maske werden für die Bits des Netzwerkteils am Anfang der Adresse Einsen notiert, für die Bits des Hostteils am Ende der Adresse dagegen

Nullen. Genau wie die IP-Adresse selbst wird auch die Teilnetzmaske in vier dezimalen 8-Bit-Blöcken geschrieben.

Wie bereits erwähnt, wird die Klasseneinteilung bereits seit Mitte der 1990er-Jahre nicht mehr verwendet. Deshalb sind alle modernen Netzwerke auf die Teilnetzmaske angewiesen, da sie allein den Netzwerkteil einer Adresse kennzeichnet. Alle modernen Betriebssysteme verwenden Protokollaufrufe mit Teilnetzmaske. Die Netzwerkadresse wird dabei mithilfe einer Bitweise-Und-Operation aus IP-Adresse und Teilnetzmaske berechnet. Hier ein Beispiel:

IP-Adresse: 192.168.0.37
Teilnetzmaske: 255.255.255.0
Berechnete Netzwerkadresse: 192.168.0.0

Hier zum Vergleich die Binärdarstellung:

```
  11000000 10101000 00000000 00100101
& 11111111 11111111 11111111 00000000
-------------------------------------
  11000000 10101000 00000000 00000000
```

Tabelle 5.6 zeigt Beispiele für die Schreibweise der ursprünglichen klassenbasierten Adressen nach CIDR-Logik sowie ihre Teilnetzmasken.

Klasse	Beispielnetz	CIDR-Adresse	Teilnetzmaske
A	17.0.0.0	17.0.0.0/8	255.0.0.0
B	167.18.0.0	167.18.0.0/16	255.255.0.0
C	195.21.92.0	195.21.92.0/24	255.255.255.0

Tabelle 5.6 Die traditionellen IP-Adressklassen in CIDR-Darstellung

Das Subnetting aus dem ersten Beispiel, die Unterteilung des Netzes 139.17.0.0/16 in vier gleich große Teilnetze, kann folgendermaßen durchgeführt werden:

- Da die 65.536 rechnerischen Adressen in vier Teile unterteilt werden sollen, sind zwei weitere Bits für den Netzwerkteil der Adresse erforderlich ($4 = 2^2$).
- Da das ursprüngliche Klasse-B-Netz einen 16 Bit (zwei Byte) langen Netzwerkteil besitzt, erfolgt die Unterteilung der vier Adressbereiche nach Bit 18, also nach dem zweiten Bit des dritten Bytes; die vier neuen Netze sind demnach 139.17.0.0/18, 139.17.64.0/18, 139.17.128.0/18 sowie 139.17.192.0/18.

Tabelle 5.7 zeigt die Eigenschaften der vier neuen Netze.

Netzwerk	Erste Hostadresse	Letzte Hostadresse	Broadcast-Adresse	Teilnetzmaske
139.17.0.0/18	139.17.0.1	139.17.63.254	139.17.63.255	255.255.192.0
139.17.64.0/18	139.17.64.1	139.17.127.254	139.17.127.255	255.255.192.0
139.17.128.0/18	139.17.128.1	139.17.191.254	139.17.191.255	255.255.192.0
139.17.192.0/18	139.17.192.1	139.17.255.254	139.17.255.255	255.255.192.0

Tabelle 5.7 Subnetting – Unterteilung des Netzes 139.17.0.0/16 in vier gleich große Teilnetze

Im zweiten Beispiel geht es um Supernetting, also um die Zusammenfassung einzelner Netze zu einem größeren Gesamtnetz. Die Netze 203.16.0.0/24 bis 203.19.255.0/24 sollen zu einem einzigen Netz verbunden werden. Diese Aufgabe lässt sich auf folgende Weise lösen:

- Es werden 1.024 Klasse-C-Netze miteinander verbunden. 256 Netze der Klasse C ergäben einfach ein Gesamtnetz von der Größe eines Klasse-B-Netzwerks. Beispielsweise würde die Vereinigung der Netze 203.16.0.0/24 bis 203.16.255.0/24 das neue Netz 203.16.0.0/16 erzeugen. Um das gewünschte Netz der vierfachen Größe zu erhalten, muss die Grenze zwischen Netz- und Hostteil noch um zwei Bits weiter nach links verschoben werden.
- Die Adresse wird zwei Bits links von der Klasse-B-Grenze unterteilt, also vor dem vorletzten Bit des zweiten Bytes. Daraus ergibt sich die Netzwerkadresse 203.16.0.0/14 mit der Teilnetzmaske 255.252.0.0. Die Broadcast-Adresse des neuen Netzes ist 203.19.255.255; die möglichen Hostadressen reichen von 203.16.0.1 bis 203.19.255.255.

Im Allgemeinen bietet es sich an, die Teilnetzmaske des ursprünglichen Netzes, das aufgeteilt oder mit mehreren verbunden werden soll, zunächst in die Binärdarstellung umzurechnen. In dieser Schreibweise fällt es am leichtesten, die Grenze zwischen Netz- und Hostteil um die gewünschte Anzahl von Bits nach links oder nach rechts zu verschieben. Anschließend können Sie die Maske wieder in die vier üblichen 8-Bit-Gruppen unterteilen und in Dezimalzahlen umrechnen.

Diese Vorgehensweise soll im Folgenden an zwei weiteren Beispielen demonstriert werden. Das Klasse-B-Netzwerk 146.20.0.0/16 soll in acht Teilnetze unterteilt werden:

- Die ursprüngliche Netzmaske ist 255.255.0.0.
- In binärer Darstellung entspricht dies 11111111 11111111 00000000 00000000.
- Eine Aufteilung in acht Netze erfolgt durch eine Verschiebung der Grenze zwischen den beiden Adressteilen um drei Stellen ($8 = 2^3$) nach rechts.
- Die neue Netzmaske in binärer Schreibweise ist 11111111 11111111 11100000 00000000.
- Nach der erneuten Umrechnung in die dezimale Vierergruppendarstellung ergibt sich 255.255.224.0.

- Entsprechend ergeben sich die folgenden acht Netze:
  - 146.20.0.0/19
  - 146.20.32.0/19
  - 146.20.64.0/19
  - 146.20.96.0/19
  - 146.20.128.0/19
  - 146.20.160.0/19
  - 146.20.192.0/19
  - 146.20.224.0/19

Die vier Klasse-C-Netzwerke 190.16.0.0/24 bis 190.16.3.0/24 sollen zu einem gemeinsamen Netz verbunden werden:

- Die Teilnetzmaske der vier Netze lautet jeweils 255.255.255.0.
- Binär geschrieben, ergibt sich daraus 11111111 11111111 11111111 00000000.
- Die Zusammenfassung vier solcher Netze erfordert eine Verschiebung der Adressgrenze um zwei Bits ($4 = 2^2$) nach links.
- In Binärdarstellung lautet die neue Maske 11111111 11111111 11111100 00000000.
- Wird diese Maske wieder in Dezimalschreibweise umgerechnet, resultiert daraus 255.255.252.0.
- Das neue Netz besitzt die CIDR-Adresse 190.16.0.0/22.

Die folgenden Tabellen zeigen in übersichtlicher Form, wie die Aufteilung der alten IP-Adressklassen in verschiedene Anzahlen von Teilnetzen funktioniert. In Tabelle 5.8 wird Klasse A behandelt. Die – rein rechnerisch mögliche – Zusammenfassung mehrerer Klasse-A-Netze durch Supernetting wird in der Praxis nicht durchgeführt, weil erstens wohl niemand mehr als 16,7 Millionen Hosts in einem Teilnetz betreiben möchte und zweitens bereits alle Klasse-A-Netze an einzelne Betreiber vergeben wurden.

Netzwerk-Bits	Host-Bits	Anzahl Teilnetze	Anzahl Hosts	Teilnetzmaske
8	24	1	16.777.214	255.0.0.0
9	23	2	8.388.606	255.128.0.0
10	22	4	4.194.302	255.192.0.0
11	21	8	2.097.150	255.224.0.0
12	20	16	1.048.574	255.240.0.0

Tabelle 5.8 Bildung von CIDR-Teilnetzen aus einem Klasse-A-Netz

Netzwerk-Bits	Host-Bits	Anzahl Teilnetze	Anzahl Hosts	Teilnetzmaske
13	19	32	524.286	255.248.0.0
14	18	64	262.142	255.252.0.0
15	17	128	131.070	255.254.0.0
16	16	256	65.534	255.255.0.0
17	15	512	32.766	255.255.128.0
18	14	1.024	16.382	255.255.192.0
19	13	2.048	8.190	255.255.224.0
20	12	4.096	4.094	255.255.240.0
21	11	8.192	2.046	255.255.248.0
22	10	16.384	1.022	255.255.252.0
23	9	32.768	510	255.255.254.0
24	8	65.536	254	255.255.255.0
25	7	131.072	126	255.255.255.128
26	6	262.144	62	255.255.255.192
27	5	524.288	30	255.255.255.224
28	4	1.048.576	14	255.255.255.240
29	3	2.097.152	6	255.255.255.248
30	2	4.194.302	2	255.255.255.252

Tabelle 5.8 Bildung von CIDR-Teilnetzen aus einem Klasse-A-Netz (Forts.)

Tabelle 5.9 zeigt die Aufteilung eines Klasse-B-Netzes in beliebig kleine Teilnetze.

Netzwerk-Bits	Host-Bits	Anzahl Teilnetze	Anzahl Hosts	Teilnetzmaske
16	16	1	65.534	255.255.0.0
17	15	2	32.766	255.255.128.0
18	14	4	16.382	255.255.192.0

Tabelle 5.9 Bildung von CIDR-Teilnetzen aus einem Klasse-B-Netz

Netzwerk-Bits	Host-Bits	Anzahl Teilnetze	Anzahl Hosts	Teilnetzmaske
19	13	8	8.190	255.255.224.0
20	12	16	4.094	255.255.240.0
21	11	32	2.046	255.255.248.0
22	10	64	1.022	255.255.252.0
23	9	128	510	255.255.254.0
24	8	256	254	255.255.255.0
25	7	512	126	255.255.255.128
26	6	1.024	62	255.255.255.192
27	5	2.048	30	255.255.255.224
28	4	4.096	14	255.255.255.240
29	3	8.192	6	255.255.255.248
30	2	16.384	2	255.255.255.252

Tabelle 5.9 Bildung von CIDR-Teilnetzen aus einem Klasse-B-Netz (Forts.)

Tabelle 5.10 demonstriert schließlich, wie die Unterteilung eines Klasse-C-Netzes erfolgt. In kleineren Unternehmen könnte es durchaus praktisch sein, ein solches – ohnehin kleines – Netzwerk weiter zu unterteilen.

Netzwerk-Bits	Host-Bits	Anzahl Teilnetze	Anzahl Hosts	Teilnetzmaske
24	8	1	254	255.255.255.0
25	7	2	126	255.255.255.128
26	6	4	62	255.255.255.192
27	5	8	30	255.255.255.224
28	4	16	14	255.255.255.240
29	3	32	6	255.255.255.248
30	2	64	2	255.255.255.252

Tabelle 5.10 Bildung von CIDR-Teilnetzen aus einem Klasse-C-Netz

In der Praxis ermöglicht CIDR bereits einen erheblich flexibleren Netzwerkaufbau als die Verwendung der alten Klassen. Doch auch diese Verfahrensweise kann immer noch ungünstige Ergebnisse zur Folge haben, wenn Teilnetze mit erheblich unterschiedlichen Größen benötigt werden: Das größte benötigte Teilnetz bestimmt die Größe aller anderen; selbst das kleinste belegt eine Menge von Adressen, die es womöglich niemals benötigen wird.

Aus diesem Grund wurde das VLSM-Konzept (*Variable Length Subnet Mask*) eingeführt. Es handelt sich um ein spezielles Subnetting-Verfahren, bei dem ein vorhandenes Netz nicht mehr in gleich große, sondern in verschieden große Teilnetze unterteilt wird. Jedem dieser Teilnetze wird eine individuelle Teilnetzmaske zugewiesen.

Das grundlegende Prinzip von VLSM besteht darin, vom kleinsten benötigten Teilnetz auszugehen und die entsprechenden größeren Netze aus Blöcken solcher kleinsten Teilnetze zu bilden, denen dann größere Teilnetzmasken zugewiesen werden. Angenommen etwa, bei der Aufteilung eines Klasse-B-Netzes mit seinen 65.534 Hostadressen besäße das kleinste gewünschte Teilnetz zwölf Hosts, das größte etwa 500. Für die zwölf Hosts ist mindestens ein Netz mit der Teilnetzmaske 255.255.255.240 erforderlich, das 14 Hostadressen bietet. Aus diesen kleinen Teilnetzen können dann entsprechend größere aufgebaut werden, wobei die Grenzen zwischen den Netzen der Logik der jeweiligen Netzmaske entsprechen müssen.

An dieser Stelle soll ein einfaches Beispiel genügen: Ein Unternehmen betreibt das öffentliche Klasse-C-Netz 196.17.41.0/24. Dieses Netz soll auf die drei Abteilungen der Firma aufgeteilt werden; die beiden Router und die drei Server sollen ein viertes separates Teilnetz bilden. Tabelle 5.11 zeigt die klassische Aufteilung des Netzes in vier gleich große Teile nach CIDR-Logik.

Bereich	Anzahl Hosts	Teilnetz	Maximale Anzahl Hosts	Freie Adressen
Server/Router	5	196.17.41.0/26	62	57
Verwaltung	20	196.17.41.64/26	62	42
Programmierung	61	196.17.41.128/26	62	1
Design	30	196.17.41.192/26	62	32

Tabelle 5.11 Aufteilung des Netzes 196.17.41.0/24 in vier Teile nach dem CIDR-Schema

Es ist leicht zu erkennen, dass zwei der Teilnetze – Server/Router und Verwaltung – vollkommen überdimensioniert sind, während zumindest das Teilnetz der Programmierabteilung beinahe seine Belastungsgrenze erreicht hat. Stellen Sie sich vor, es würden noch zwei weitere Hosts in diese Abteilung aufgenommen: Schon wäre das Teilnetz zu klein, und es müsste über eine andere Verteilung nachgedacht werden. In diesem Beispiel könnte sie nur noch

darin bestehen, zwei der anderen Bereiche zusammenzulegen, um den Programmierbereich zu vergrößern.

Eine komplexere, aber für den konkreten Anwendungsfall sinnvollere Aufteilung des Netzes mithilfe der VLSM-Technik zeigt Tabelle 5.12.

Bereich	Anzahl Hosts	Teilnetz	Maximale Anzahl Hosts	Freie Adressen
Server/Router	5	196.17.41.0/27	30	25
Verwaltung	20	196.17.41.32/27	30	10
Design	30	196.17.41.64/26	62	32
Programmierung	61	196.17.41.128/25	126	65

Tabelle 5.12 Flexible Aufteilung des Netzes 196.17.41.0/24 in vier Teile nach dem VLSM-Schema

Für die IP-Konfiguration eines einzelnen Hosts macht es keinen Unterschied, ob das Teilnetz, in dem er sich befindet, nach der alten Klassenlogik, nach dem CIDR-Verfahren oder nach der VLSM-Methode konfiguriert wurde: In jedem Fall wird im Konfigurationsdialog des jeweiligen Betriebssystems die korrekte Teilnetzmaske eingestellt. Spezielle Unterstützung für VLSM benötigen lediglich Router, die in dem betroffenen Netz eingesetzt werden. Die meisten neueren Routing-Protokolle bieten diese Unterstützung.

Die Übertragung von IP-Datagrammen

Auf der Internetschicht des TCP/IP-Protokollstapels, auf der das IP-Protokoll arbeitet, werden die Datenpakete, wie bereits erwähnt, als *Datagramme* bezeichnet. Um die Datenübertragung mithilfe des IP-Protokolls genau zu erläutern, soll an dieser Stelle zunächst der IP-Header vorgestellt werden. Er enthält die Steuerdaten, die das IP-Protokoll zu einem Datenpaket hinzufügt, das ihm vom übergeordneten Transportprotokoll übergeben wird.

Der IPv4-Protokoll-Header wird wie das gesamte Protokoll in RFC 791 definiert. Seine Länge beträgt mindestens 20 Byte, dazu können bis zu 40 Byte Optionen kommen. Tabelle 5.13 zeigt den genauen Aufbau.

Byte	0		1	2	3
0	Version	IHL	Type of Service	Paketgesamtlänge	
4	Identifikation			Flags	Fragment-Offset
8	Time to Live		Protokoll	Header-Prüfsumme	

Tabelle 5.13 Aufbau des IPv4-Datagramm-Headers

Byte	0	1	2	3
12	Quelladresse			
16	Zieladresse			
20	Optionen			Padding
...	eventuell weitere Optionen			

Tabelle 5.13 Aufbau des IPv4-Datagramm-Headers (Forts.)

Die einzelnen Daten des IP-Headers sind folgende:

- *Version* (4 Bit): Die Versionsnummer des IP-Protokolls, die das Paket verwendet – bei IPv4, wie der Name schon sagt, die Version 4.
- *IHL* (4 Bit): Internet Header Length, die Länge des Internet-Headers in 32-Bit-Wörtern (entsprechen den Zeilen in Tabelle 5.13). Der kleinste mögliche Wert beträgt 5.
- *Type of Service* (8 Bit): Ein Code, der die Art des Datenpakets festlegt. Bestimmte Sorten von Paketen, etwa für den Austausch von Routing- oder Statusinformationen, werden von bestimmten Netzen bevorzugt weitergeleitet.
- *Paketgesamtlänge* (16 Bit): Die Gesamtlänge des Datagramms in Bytes, Header und Nutzdaten.
- *Identifikation* (16 Bit): Ein durch den Absender frei definierbarer Identifikationswert, der beispielsweise das Zusammensetzen fragmentierter Datagramme ermöglicht.
- *Flags* (3 Bit): Kontroll-Flags, die die Paketfragmentierung regeln. Das erste Bit ist reserviert und muss immer 0 sein, das zweite (DF) bestimmt, ob das Paket fragmentiert werden darf (Wert 0) oder nicht (1), das dritte (MF) regelt, ob dieses Paket das letzte Fragment (0) ist oder ob weitere Fragmente folgen (1).
- *Fragment-Offset* (13 Bit): Dieser Wert (angegeben in 64-Bit-Blöcken) legt fest, an welcher Stelle in einem Gesamtpaket dieses Paket steht, falls es sich um ein Fragment handelt. Das erste Fragment oder ein nicht fragmentiertes Paket erhält den Wert 0.
- *Time to Live* (8 Bit): Der TTL-Mechanismus sorgt dafür, dass Datagramme nicht endlos im Internet weitergeleitet werden, falls die Empfängerstation nicht gefunden wird. Jeder Router, der ein Datagramm weiterleitet, zieht von diesem Wert 1 ab; wird der Wert 0 erreicht, leitet der betreffende Router das Paket nicht mehr weiter, sondern verwirft es.
- *Protokoll* (8 Bit): Die hier gespeicherte Nummer legt fest, für welches Transportprotokoll der Inhalt des Datagramms bestimmt ist, zum Beispiel 6 für TCP oder 17 für UDP. Diese beiden wichtigsten Transportprotokolle werden im nächsten Abschnitt beschrieben.

- *Header-Prüfsumme* (16 Bit): Die Prüfsumme stellt eine einfache Plausibilitätskontrolle für den Datagramm-Header zur Verfügung. Ein Paket, dessen Header-Prüfsumme nicht korrekt ist, wird nicht akzeptiert und muss erneut versendet werden.
- *Quelladresse* und *Zieladresse* (je 32 Bit): Die IP-Adressen von Absender und Empfänger. IP-Adressen wurden zuvor bereits ausführlich behandelt.
- *Optionen* (variable Länge): Die meisten IP-Datagramme werden ohne zusätzliche Optionen versandt, da Absender- und Empfängerhost sowie alle auf dem Weg befindlichen Router die jeweils verwendeten Optionen unterstützen müssen. Zu den verfügbaren Optionen gehören unter anderem Sicherheitsfeatures und spezielle Streaming-Funktionen.

Das Problem der Paketfragmentierung entsteht dadurch, dass verschiedene physikalische Netzarten unterschiedliche Maximallängen für Datenpakete erlauben. Dieser Wert, der als *Maximum Transmission Unit* (MTU) bezeichnet wird, kann bei einigen Netzwerkschnittstellen per Software konfiguriert werden, bei anderen ist er vom Hersteller vorgegeben. Werden nun Datagramme aus einem Netz mit einer bestimmten MTU in ein anderes Netz mit einer kleineren MTU weitergeleitet, müssen die Daten in kleinere Pakete »umgepackt« werden. Wie bereits beschrieben, werden sie dazu mit Fragmentierungsinformationen versehen, damit sie später wieder richtig zusammengesetzt werden können.

Solange Quell- und Zieladresse im gleichen Netzwerk liegen, ist die Übertragung der Datagramme sehr einfach: Je nach Netzwerkart wird auf die passende Art (bei Ethernet zum Beispiel über ARP) diejenige Schnittstelle ermittelt, für die die Daten bestimmt sind. Anschließend wird das Datagramm an den korrekten Empfänger übermittelt. Dieser liest den IP-Header des Pakets, setzt eventuelle Fragmente wieder richtig zusammen und übermittelt das Paket an das Transportprotokoll, dessen Nummer im Header angegeben ist. Wie der Transportdienst mit den Daten umgeht, erfahren Sie im nächsten Abschnitt.

IPv6

Bereits in der ersten Hälfte der 1990er-Jahre wurde damit gerechnet, dass sehr bald keine weiteren IPv4-Adressen mehr verfügbar sein würden. Dass dies viel länger als gedacht noch nicht der Fall war, lag an der Einführung von CIDR, VLSM und NAT (Letzteres wird im übernächsten Unterabschnitt, »Weitere IP-Dienste«, beschrieben). Da das Internet aber weiterhin wächst, ist es nur noch eine Frage der Zeit, bis die Anzahl der Adressen endgültig erschöpft ist; was die Zuteilung an Dienstleister angeht, ist dies sogar jetzt schon der Fall.

Deshalb wurde schon vor einigen Jahren mit der Arbeit an einem Nachfolger für das IPv4-Protokoll begonnen, der vor allem einen größeren Adressraum durch längere IP-Adressen besitzen sollte. Letzten Endes fiel die Entscheidung auf Adressen von 128 Bit Länge. Dies ergibt theoretisch mehr als $3{,}4 \cdot 10^{38}$ verschiedene Adressen! Damit erscheint der Adressraum mehr als überdimensioniert; offensichtlich kann man damit jedem einzelnen Sandkorn auf

unserem Planeten mehrere eigene IP-Adressen zuweisen. Letzten Endes geht es allerdings eher darum, beinahe beliebig viele Netze von sehr unterschiedlicher Größe einrichten zu können. Abgesehen davon werden immer mehr tragbare Geräte entwickelt, die mit Netzwerken verbunden werden – etwa dynamisch über öffentliche WLAN-Access-Points.

Die aktuelle Version des neuen IP-Protokolls wird in RFC 2460 beschrieben. Da die Version 5 für Experimente mit Multicasting verwendet wurde, lautet die Versionsnummer des Protokolls IPv6; während seiner Entwicklung wurde es auch manchmal als *IPng* (für *next generation*) bezeichnet, zum Beispiel in RFC 1752, das den ersten Arbeitsentwurf beschreibt. Die IPv6-Adresse wird nicht in 8-Bit-Dezimalgruppen geschrieben wie bei IPv4, aber mit 16 Gruppen wäre sie ein wenig unhandlich. Stattdessen schreibt man acht vierstellige Hexadezimalgruppen, die durch Doppelpunkte getrennt werden. Eine IPv6-Adresse sieht zum Beispiel folgendermaßen aus:

```
4A29:30B4:0031:0000:0000:0092:1A3B:3394
```

Eine zulässige Verkürzung besteht darin, führende Nullen in einem Block wegzulassen sowie Blöcke, die nur aus Nullen bestehen, durch zwei aufeinanderfolgende Doppelpunkte zu ersetzen. Kurz gefasst, lautet die Beispieladresse also `4A29:30B4:31::92:1A3B:3394`. Um die Adresse eindeutig zu halten, darf eine solche Verkürzung innerhalb einer Adresse nur einmal vorgenommen werden.

Genau wie IPv4-Adressen werden auch die neuen IPv6-Adressen in zwei Teile unterteilt: Links steht ein Präfix, dahinter ein Individualteil, der dem Hostteil der IPv4-Adresse entspricht. Das Präfix gibt allerdings nicht das einzelne Netz an, zu dem die Adresse gehört, sondern informiert über den Adresstyp. Da die Präfixe wie bei IPv4 unterschiedliche Längen aufweisen können, wird das Präfix zusammen mit seiner Bit-Anzahl angegeben. Tabelle 5.14 gibt Ihnen einen Überblick über die verschiedenen Adressblöcke und ihre Verwendung.

Präfix	Verwendung
0::0/8	reserviert für spezielle Anwendungen
100::0/8	noch nicht zugeordnet
200::0/7	Abbildung von NSAP-Adressen
400::0/7	Abbildung von IPX-Adressen
600::0/7	noch nicht zugeordnet
800::0/5	noch nicht zugeordnet
1000::0/4	noch nicht zugeordnet

Tabelle 5.14 IPv6-Adressbereiche und -präfixe

Präfix	Verwendung
2000::0/3	global eindeutige Adressen
6000::0/3	noch nicht zugeordnet
8000::0/3	noch nicht zugeordnet
A000::0/3	noch nicht zugeordnet
C000::0/3	noch nicht zugeordnet
E000::0/4	noch nicht zugeordnet
F000::0/5	noch nicht zugeordnet
F800::0/6	noch nicht zugeordnet
FE00::0/7	noch nicht zugeordnet
FE00::0/9	noch nicht zugeordnet
FE80::0/10	auf eine Verbindung begrenzte Adressen
FEC0::0/10	auf eine Einrichtung begrenzte Adressen
FF00::0/8	Multicast-Adressen

Tabelle 5.14 IPv6-Adressbereiche und -präfixe (Forts.)

Die typischste Form von IPv6-Adressen, deren Stil am ehesten den öffentlich gerouteten IPv4-Adressen entspricht, ist die globale Unicast-Adresse. Ihre Struktur ist in RFC 2374 festgelegt und sieht folgendermaßen aus:

- externes Routing-Präfix (48 Bit)
- Site-Topologie (üblicherweise 16 Bit)
- Schnittstellen-Identifikationsnummer (normalerweise 64 Bit); wird in der Regel automatisch generiert, oft aus der MAC-Adresse der Schnittstelle oder aus der bisherigen IPv4-Adresse

Der IPv6-Datagramm-Header wurde gegenüber dem IPv4-Header erheblich vereinfacht. Durch die Auslagerung eventueller Optionen in sogenannte *Erweiterungs-Header* wird die Länge des Basis-Headers auf genau 320 Bit (40 Byte) festgelegt; einige Felder des IPv4-Headers wurden entfernt, weil sie keine Bedeutung mehr haben. Tabelle 5.15 zeigt den genauen Aufbau des IPv6-Headers.

Byte	0		1	2	3
0	Version	Klasse	Flow Label		
4	Payload Length			Next Header	Hop Limit
8 12 16 20	Quelladresse				
24 28 32 36	Zieladresse				

Tabelle 5.15 Aufbau des IPv6-Datagramm-Headers

Hier die Bedeutung der einzelnen Felder des Headers:

- *Version* (4 Bit): Die Versionsnummer des IP-Protokolls, hier natürlich 6.
- *Klasse* (8 Bit): Dieses Feld gibt die Priorität an, mit der das Datagramm übertragen werden soll. Es ist noch nicht abschließend geklärt, wie die entsprechenden Werte aussehen sollen.
- *Flow Label* (20 Bit): Ein Erweiterungsfeld, in das ein von 0 verschiedener Wert eingetragen wird, wenn IPv6-Router das Datagramm auf besondere Weise behandeln sollen. Es dient vor allem der Implementierung der Quality-of-Service-Funktionalität, mit deren Hilfe Paketsorten voneinander unterschieden werden, um beispielsweise Echtzeitanwendungen wie Streaming, Multimedia oder Videokonferenzen zu unterstützen.
- *Payload Length* (16 Bit): Die Länge der Nutzdaten, die auf den Header folgen.
- *Next Header* (8 Bit): Der Wert in diesem Feld gibt den Typ des ersten Erweiterungs-Headers an, falls einer vorhanden ist. Es gibt bisher sechs Arten von Erweiterungs-Headern; eine Übersicht finden Sie in Tabelle 5.16.
- *Hop Limit* (8 Bit): Ein neuer Name für die Time-to-Live-Funktion: Jeder Router zieht von dem ursprünglichen Wert 1 ab; bei Erreichen des Werts 0 wird das Paket verworfen.
- *Quell-* und *Zieladresse* (je 128 Bit): Die Adressen des Absenders und des Empfängers; genau wie bei IPv4, nur entsprechend der Protokollspezifikation 128 statt 32 Bit lang

Header	Next-Header- Code	Beschreibung
Hop-by-Hop Options Header	0	Optionen, die bei jedem Routing-Schritt ausgeführt werden müssen.
Routing Header	43	Festlegung der Router, über die das Paket geleitet werden soll.
Fragment Header	44	Der Absender muss bei IPv6 die MTU herausfinden und Fragmente selbst bilden; die Fragmentinformationen befinden sich hier.
Authentication Header	51	Authentifizierung des Absenders gegenüber dem Empfänger.
Encapsulating Security Payload Header	50	Dient der Verschlüsselung des Datagramms (IPv6).
Destination Options Header	60	Optionen, die nur für den Zielhost bestimmt sind.
Upper-Layer Header	59	Header einer höheren Schicht; aus IPv6-Sicht also Nutzdaten.

Tabelle 5.16 Die verschiedenen Typen von IPv6-Erweiterungs-Headern

Das größte Problem, das der sofortigen Einführung von IPv6 noch im Wege steht, ist ein organisatorisches: Zum einen kann man nicht einfach über Nacht flächendeckend umsteigen, da in diesem Fall die IP-Treiber aller Hosts und Router weltweit gewechselt werden müssten, was vollkommen illusorisch ist – zumal viele ältere Hardwarekomponenten, Betriebssysteme und Programme IPv6 gar nicht unterstützen und ihre Hersteller auch nicht vorhaben, diese Unterstützung nachträglich zu implementieren. Zum anderen ist es aber auch nicht möglich, gleichzeitig einen Teil des Internets mit IPv4 und einen anderen mit IPv6 zu betreiben und auf diese Weise allmählich auf die neue Version umzusteigen, da die beiden Adressierungsschemata miteinander inkompatibel sind.

Die Lösung, die letzten Endes gefunden wurde, besteht in der *Tunnelung* von IPv6-Paketen durch das klassische IPv4-Netzwerk. Tunnelung bedeutet nichts anderes, als dass jedes IPv6-Datagramm in ein IPv4-Datagramm verpackt wird. Das heißt, das IPv6-Paket bildet aus der Sicht des IPv4-Pakets die Nutzdaten, die mit einem v4-Header versehen werden. Am jeweiligen Zielpunkt, an dem wiederum IPv6 verfügbar ist, wird das Version-4-Datagramm »ausgepackt« und gemäß den Header-Daten weiterverarbeitet. IPv6-Tunnel-Dienste werden mittlerweile auch von mehreren kommerziellen und verschiedenen freien Anbietern, den Tunnel-Brokern, zur Verfügung gestellt.

IP-Routing

Komplizierter, aber auch interessanter wird es, wenn IP-Datagramme nicht für einen Host im lokalen Netz bestimmt sind, sondern für ein anderes Netzwerk. In diesem Fall muss das Paket an einen Router übergeben werden, der es weiterleitet. Die meisten Daten, die im Internet übertragen werden, passieren eine Vielzahl solcher Router, bis sie schließlich ihr Ziel erreichen. Um das Konzept des IP-Routings verstehen zu können, müssen Sie verschiedene Aspekte betrachten. Insbesondere ist die Frage von Bedeutung, auf welche Art und Weise überhaupt das korrekte Empfängernetzwerk gefunden wird.

Wichtig ist, dass man zwei Arten der Paketweiterleitung unterscheiden muss. Die reine Weiterleitung wird als *IP-Forwarding* bezeichnet; dabei sind nur zwei mögliche Netzwerkschnittstellen betroffen, sodass Quelle und Ziel jeweils feststehen. *Routing* im engeren Sinn beschreibt dagegen Verfahren, bei denen Entscheidungen zur Weiterleitung über verschiedene Wege an ein bestimmtes Ziel getroffen werden. Ein Router muss beide Verfahren beherrschen, sodass im Alltag oft nicht zwischen ihnen unterschieden wird. In LANs findet jedoch oft nur Forwarding, aber kein echtes Routing statt, da meist ohnehin nicht mehrere Router zur Auswahl stehen. Sowohl Forwarding als auch Routing lassen sich übrigens entweder statisch über festgelegte Tabellen oder dynamisch mithilfe von Protokollen erledigen.

Bei einem einzelnen Host können üblicherweise zwei Arten von Routern angegeben werden: zum einen die Router, die Daten in ein bestimmtes Fremdnetzwerk weiterleiten, und zum anderen das Default-Gateway (der Standard-Router), das alle Daten entgegennimmt, die weder für das lokale Netz noch für ein Netz mit einem speziellen Router bestimmt sind.

Beachten Sie übrigens, dass der Begriff *Gateway* zweideutig ist: Das Wort *Default-Gateway* beim IP-Forwarding oder Routing bezeichnet wie erwähnt den Standard-Router. Im Allgemeinen steht Gateway dagegen für einen Verbindungsrechner, der über sämtliche OSI-Schichten arbeitet und deshalb genauer als *Application Level Gateway* bezeichnet wird.

Bei einem privaten PC oder DSL-Router, der über eine Wählleitung mit dem Internet verbunden ist, besteht in der Regel nur eine Verbindung zu einem einzelnen Router des Providers. Welcher das ist, wird jedoch bei der Einwahl in das Netzwerk des Providers bestimmt, da auch die IP-Adresse bei jeder Einwahl dynamisch zugeteilt wird. Je nachdem, welche Adresse dem Host zugeteilt wird, ist möglicherweise ein anderer Router zuständig. Deshalb wird der Router bei der IP-Konfiguration des DFÜ-Netzwerkzugangs nicht fest angegeben, sondern durch das Einwahlprotokoll (üblicherweise PPP) mitgeteilt.

Anders sieht es dagegen oft bei Workstations in Unternehmen aus, die an ein lokales Netzwerk angeschlossen sind: Sämtliche Netzwerkkommunikation, sowohl mit dem lokalen Netz als auch mit dem Internet, findet über ein und dieselbe LAN-Schnittstelle statt, meistens über Ethernet. Innerhalb des LAN besitzt der Router für die Verknüpfung zum Internet eine bekannte IP-Adresse, die bei der IP-Konfiguration des Hosts angegeben wird. Mitunter besteht die Netzwerkinfrastruktur eines größeren Unternehmens auch aus mehreren Einzel-

netzen, die über interne Router miteinander vernetzt werden. In einem solchen Fall wird häufig der Router, der zu dem anderen lokalen Netz führt, als Router für dieses konkrete Netz angegeben, während der Internetrouter (dessen Zielnetz »alle anderen Netze« sind) als Standard-Router eingerichtet wird. Für den letzteren – routingtechnisch relativ interessanten – Fall sehen Sie hier ein Beispiel:

In einem Unternehmen bestehen die beiden lokalen Netze 196.87.98.0/24 und 196.87.99.0/24. Das erste Netz wird von der Grafikabteilung verwendet, das zweite von der Softwareentwicklung. In Abbildung 5.4 wird der Aufbau dieses Netzes dargestellt.

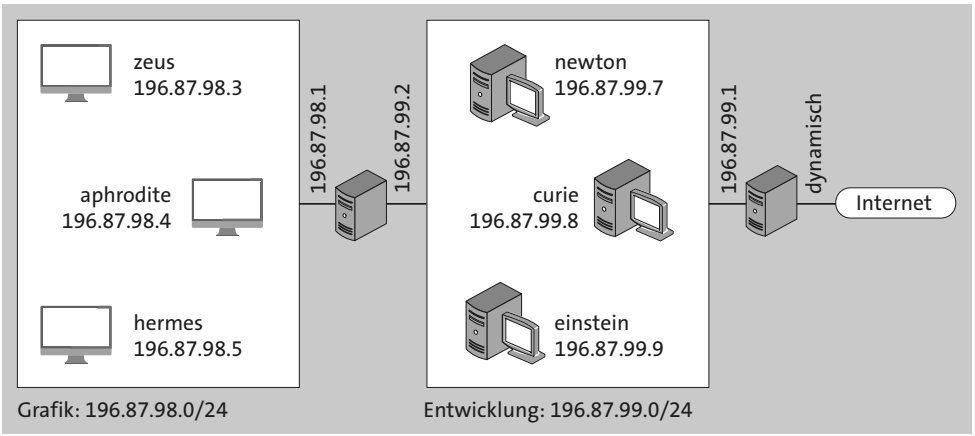


Abbildung 5.4 Verbindung zwischen zwei verschiedenen lokalen Netzen und dem Internet über zwei Router

Das Netzwerk der Grafikabteilung enthält die folgenden drei Rechner:

- *zeus* (196.87.98.3)
- *aphrodite* (196.87.98.4)
- *hermes* (196.87.98.5)

Zum Netzwerk der Entwicklungsabteilung gehören die drei folgenden Hosts:

- *newton* (196.87.99.7)
- *curie* (196.87.99.8)
- *einstein* (196.87.99.9)

Zwischen den beiden lokalen Netzen befindet sich ein Router, dessen Schnittstelle im Netz der Grafikabteilung die IP-Adresse 196.87.98.1 besitzt. Seiner anderen Schnittstelle für die Entwicklungsabteilung wurde die Adresse 196.87.99.2 zugewiesen. Ein zweiter Router verbindet die Entwicklungsabteilung mit dem Internet. Seine lokale Schnittstelle wurde mit der IP-Adresse 196.87.99.1 konfiguriert, die Adresse für die Internetschnittstelle wird vom Internetprovider dynamisch zugewiesen.

Interessant ist nun die Routing-Konfiguration der einzelnen Hosts. Die drei Rechner im Entwicklungsnetzwerk kennen zwei verschiedene Router: Der Standard-Router ist 196.87.99.1, als spezieller Router für Datenpakete an das Netz 196.87.98.0 wird 196.87.99.2 angegeben. Dagegen kennen die drei Hosts im Grafiknetzwerk nur einen einzigen Router, nämlich 196.87.98.1, der als Standard-Router eingerichtet wird. Ob Datenpakete jenseits dieses Routers für das Netz 196.87.99.0 oder für das Internet bestimmt sind, muss der Router selbst entscheiden; die Rechner schicken ihm einfach alle Datagramme, die nicht für das lokale Netz verwendet werden sollen.

Angenommen, *aphrodite* möchte auf Daten zugreifen, die *newton* bereitstellt. Die Daten sind offensichtlich nicht für das Netz 196.87.98.0 bestimmt, deshalb werden sie dem Router übergeben. Dieser erkennt, dass sie für das Netz 196.87.99.0 bestimmt sind, an das er unmittelbar angeschlossen ist. Er kann die Daten direkt an den Zielhost ausliefern.

Will dagegen *zeus* auf Daten aus dem Internet zugreifen, muss der Standard-Router des Grafiknetzes erkennen, dass die Daten nicht für das andere Netz bestimmt sind, an das er selbst angeschlossen ist, und sie an den nächsten Router weiterreichen.

Ein wenig anders verhält es sich, wenn ein Rechner aus dem Entwicklungsnetz, etwa *curie*, auf *zeus* zugreifen möchte. Es ist bereits in der Routing-Konfiguration von *curie* bekannt, dass ein bestimmter Router, nämlich 196.87.99.2, verwendet werden soll. Ebenso weiß beispielsweise *einstein*, dass Zugriffe auf das Internet über den Router 196.87.99.1 erfolgen müssen.

Damit ein Host weiß, wohin er Datenpakete eigentlich schicken muss, um ein bestimmtes Netz zu erreichen, müssen die einzelnen Router in seiner Netzwerkkonfiguration angegeben werden – dies funktioniert je nach Betriebssystem unterschiedlich.

Das Ergebnis dieser Konfiguration ist eine Routing-Tabelle, die ebenfalls je nach System unterschiedlich aussieht. Angenommen, alle Rechner im zuvor gezeigten Beispielnetzwerk liefen unter Unix-Varianten (die Grafikrechner unter macOS, die Entwicklungscomputer unter Linux). Dann sähe die Routing-Tabelle von *curie*, die durch den Unix-Befehl `netstat rn` angezeigt werden kann, so aus:

```
$ netstat -rn
Routing Tables
Destination Gateway FlagsRefcntUseInterface
127.0.0.1 127.0.0.1 UH 1 132lo0
196.87.99.0 196.87.99.8 U2649041le0
196.87.98.0 196.87.99.2 UG 0 0le0
default 196.87.99.1 UG 0 0le0
```

Die erste Zeile (Zieladresse 127.0.0.1) beschreibt das Erreichen der Loopback-Adresse: Das Interface (die Netzwerkschnittstelle) ist `lo0` (*local loopback*). Das Flag `H` zeigt an, dass es sich

um eine Route zum Erreichen eines einzelnen Hosts handelt. Das Flag `U` dagegen steht für *up* und bedeutet, dass die Route zurzeit intakt ist.

In der nächsten Zeile wird das lokale Netzwerk angegeben, in dem sich *curie* selbst befindet. Deshalb wird als Gateway einfach die IP-Adresse von *curie* ausgegeben. Das Interface `le0` ist die erste (und in diesem Fall einzige) Ethernet-Schnittstelle des Rechners.

Die dritte Zeile beschreibt die Route in das Grafiknetzwerk über den Router, dessen Adresse im Entwicklungsnetz 196.87.99.2 lautet. Das Flag `G` steht für *Gateway*, also für die Tatsache, dass für diese Route die Dienste eines Routers in Anspruch genommen werden.

In der letzten Zeile wird schließlich 196.87.99.1 als Default-Gateway angegeben, also als Router für alle Ziele, die nicht explizit in der Routing-Tabelle auftauchen.

Die Routing-Tabelle von *hermes* sieht einfacher aus:

```
Routing Tables
Destination GatewayFlagsRefcnt UseInterface
127.0.0.1 127.0.0.1UH 1 132lo0
196.87.98.0 196.87.98.5U26490411e0
default 196.87.98.1UG 0 0le0
```

Da das Grafiknetz nur einen Router kennt, gibt es lediglich den Loopback-Eintrag, die Information für das lokale Netz und schließlich den Default-Eintrag für alle anderen Netze.

Auf diese Weise werden Daten durch das gesamte Internet geroutet. Jedes Mal, wenn ein Router passiert wird, erfolgt ein sogenannter *Hop* der Daten. Wegen des TTL-Feldes von 8 Bit Größe, das im IP-Header enthalten ist und bereits beschrieben wurde, erreicht ein Datagramm sein Ziel stets mit höchstens 255 Hops – oder eben gar nicht.

Damit IP-Datenpakete ihr Ziel überhaupt erreichen können, muss im Prinzip jeder einzelne Router im gesamten Internet darüber Bescheid wissen, wie er jedes beliebige Netz erreichen kann. Zu diesem Zweck unterhält auch jeder Router Routing-Tabellen, die den bereits für die einzelnen Hosts gezeigten ähnlich sehen. Da das Internet ein Zusammenschluss von vielen einzelnen Netzwerken ist, müssen diese Tabellen jedoch ständig aktualisiert werden, denn es ergeben sich häufig Konfigurationsänderungen, weil neue Netze hinzukommen oder vorhandene geändert oder aufgegeben werden. Es wäre absolut unzumutbar, diese Konfigurationsänderungen fortlaufend manuell auf dem aktuellen Stand zu halten, was deshalb auch seit vielen Jahren nicht mehr üblich ist (außer innerhalb sehr kleiner Netze wie in dem Beispiel zuvor, in denen sich die Routing-Einstellungen selten ändern müssen).

Die Router im Internet müssen deshalb ständig Informationen darüber austauschen, an welche anderen Netzwerke sie jeweils Daten vermitteln. Sie müssen komplexe Routing-Entscheidungen treffen, indem sie den Aufwand und die Kosten verschiedener Routen vergleichen und die Pakete eben nicht direkt ans Ziel, sondern auf dem derzeit günstigsten Weg weiterleiten, damit diese nicht nur sicher, sondern auch möglichst schnell ihr Ziel erreichen. Dieses eigentliche Routing ist erheblich dynamischer als das einfache Forwarding, sodass die

Routing-Informationen ständig aktualisiert werden müssen. Auf diese Weise kann ein Paket bei Ausfall oder auch nur starker Belastung einer bestimmten Route über eine andere Route umgeleitet werden. Zu diesem Zweck wurde eine Reihe verschiedener Routing-Protokolle entwickelt, mit deren Hilfe das möglich wird. Jedes dieser Routing-Protokolle besitzt andere Eigenschaften, außerdem wird nicht jedes dieser Protokolle von jedem Hersteller unterstützt.

Zunächst muss zwischen zwei Arten von Routing unterschieden werden: dem Routing innerhalb zusammenhängender Netze eines einzelnen Betreibers (*Interior Routing*), der innerhalb dieses Bereichs frei über die Konfiguration entscheiden kann, und dem Routing zwischen voneinander unabhängigen derartigen Bereichen (*Exterior Routing*). Alle zusammenhängenden Netze eines Betreibers werden als *autonome Systeme* (*Autonomous Systems*, abgekürzt AS) bezeichnet. Einige Routing-Protokolle, etwa das veraltete RIP oder das aktuellere OSPF, dienen dem Routing innerhalb von autonomen Systemen, während andere, vor allem BGP, für das Routing zwischen den Grenzen autonomer Systeme zuständig sind. Diese drei genannten Routing-Protokolle werden im weiteren Verlauf des Kapitels kurz vorgestellt.

Wenn ein Router ein Routing-Protokoll ausführt, teilt er den benachbarten Routern mit, an welche Netze er Daten weiterleitet. Die meisten Routing-Protokolle machen außerdem Angaben über die »Kosten«, die für das Erreichen eines bestimmten Netzes kalkuliert werden müssen. Der Begriff *Kosten* hat nichts mit dem Preis zu tun, sondern bestimmt vor allem, über wie viele Hops ein bestimmtes Netzwerk durch den jeweiligen Router erreicht werden kann. Allerdings gibt es auch die Möglichkeit, die Kostenangaben willkürlich zu manipulieren – je nachdem, wie »gern« ein Router Daten an ein bestimmtes Netzwerk übermitteln soll. Wenn ein Router bestimmen muss, an welchen benachbarten Router er die Daten für ein bestimmtes Netz übergeben soll, sucht er sich denjenigen aus, der für dieses Netz geringere Kosten angibt. Diese Kostendaten werden auch als die *Metrik* des Routings bezeichnet.

Auf diese Weise wird versucht, die Datenströme zwischen den verschiedenen Backbone-Netzwerken möglichst gleichmäßig zu verteilen, außerdem bestehen verschiedene Arten von Verträgen oder Vereinbarungen zwischen den Netzbetreibern, was die Weiterleitung von Daten bestimmter anderer Netzwerke betrifft. Beispielsweise gab es in Deutschland in den 1990er-Jahren einen mehrjährigen Streit zwischen dem Deutschen Forschungsnetz (DFN), dem Betreiber der deutschen Universitätsnetze, und den kommerziellen Internetprovidern. Es ging um die Frage, wer wem mehr Datenverkehr aus dem jeweils anderen Netz zumutete. Erst durch die Einführung neuer zentraler Datenaustauschpunkte wie dem DE-CIX konnte der Konflikt beigelegt werden.

Hier einige wichtige Routing-Protokolle im Überblick:

- **Routing Information Protocol (RIP)**
  Das Routing Information Protocol (RIP) wird auf Unix-Routern durch den *Routing Daemon* (`routed`) ausgeführt. Beim Start von `routed` wird eine Anfrage ausgesendet. Alle anderen Router, die innerhalb desselben autonomen Systems ebenfalls `routed` ausführen, be-

antworten diese Anfrage durch Update-Pakete. Darin sind die Zieladressen aus den Routing-Tabellen der anderen Router und deren jeweilige Metrik enthalten.

Enthält ein Update-Paket die Routen zu Netzen, die noch gar nicht bekannt sind, fügt der Router sie seiner Routing-Tabelle hinzu. Außerdem werden Routen ersetzt, falls ein Update-Paket die Information enthält, dass ein bestimmtes Netzwerk über einen anderen Router mit geringeren Kosten zu erreichen ist.

Ein Router, auf dem `routed` läuft, sendet ebenfalls Update-Pakete, und zwar in der Regel alle 30 Sekunden. Erhält ein Router von einem anderen Router mehrere Male keine Update-Pakete mehr (häufig beträgt die Wartezeit 180 Sekunden), löscht er alle Einträge aus seiner Routing-Tabelle, die diesen Router verwenden. Außerdem werden diejenigen Einträge gelöscht, deren Kosten mehr als 15 Hops betragen. Letzteres beschränkt RIP auf kleinere autonome Systeme.

RIP interpretiert IP-Adressen streng nach der alten Klassenlogik und beherrscht weder CIDR noch VLSM. Dies ist der Hauptgrund dafür, dass es immer seltener verwendet wird.

Außerdem besteht das Problem, dass durch den plötzlichen Ausfall von Routern Konfigurationsfehler entstehen können: Alle Netze, die ursprünglich nur durch den ausgefallenen Router erreicht werden konnten, sind nun gar nicht mehr erreichbar. Das spricht sich jedoch nur allmählich herum, da ein Router zwar zunächst alle Routen entfernt, die durch den ausgefallenen Router führten, von den anderen jedoch wieder die Route zu dem Netz lernt, das nun nicht mehr erreichbar ist. Bei einem Update-Intervall von 30 Sekunden kann es recht lange dauern, bis die Router die Entfernung zu dem nicht mehr verfügbaren Netz auf die nicht mehr relevanten 16 Hops »hochgeschaukelt« haben.

Um dieses Szenario zu verhindern, wird eine Technik namens *Split Horizon* verwendet: Ein Router bietet Routing-Informationen nicht über die Verbindung an, über die er sie gelernt hat. Eine Erweiterung dieses Verfahrens ist *Poison Reverse*; hier wird den Routern, von denen eine bestimmte Verbindung gelernt wurde, aktiv die *Unendlich-Metrik* 16 angegeben.

Einige Probleme von RIP werden in der neueren Version RIP-2, die in RFC 1723 beschrieben wird, beseitigt; vor allem arbeitet diese Version mit CIDR-Adressierung.

- **Open Shortest Path First (OSPF)**
  Das in RFC 2178 beschriebene Open-Shortest-Path-First-Protokoll (OSPF) ist ein sogenanntes *Link-State-Protokoll*: Der einzelne Router speichert einen gerichteten Graphen des Netzwerks aus seiner jeweiligen Sicht. Ein gerichteter Graph ist eine Art Baumdiagramm mit dem lokalen Router als Wurzel; sein Aufbau erfolgt nach dem Shortest-Path-First-Algorithmus von Dijkstra: Die Kosten des lokalen Routers selbst werden mit 0 angegeben; von diesem zweigen die Routen zu den Nachbarn baumförmig ab, dann wiederum zu deren Nachbarn etc. In einem zweiten Schritt wird der Link-State-Graph optimiert. Falls mehrere Routen zu einem Ziel vorhanden sind, beispielsweise eine direkte und eine indirekte, wird jeweils die weniger kostengünstige Route entfernt.

Um die Link-State-Datenbank klein zu halten, werden größere autonome Systeme in kleinere Einheiten unterteilt, die *Areas*. Nur vereinzelte Router, die sogenannten *Bereichsgrenzrouter*, werden von den Routern innerhalb einer Area als Verbindung in andere Areas betrachtet.

Ein OSPF-Router gewinnt seine Erkenntnisse über die benachbarten Router, indem er sogenannte *Hello-Pakete* aussendet. Diese enthalten seine eigene Adresse und die Information, von welchen benachbarten Routern er bereits Routing-Daten erhalten hat. Ein Router, der ein Hello-Paket erhält, trägt den Absender dieses Pakets als Nachbarn in seinen eigenen Link-State-Graphen ein. Die Hello-Pakete werden in regelmäßigen Abständen ausgesandt, um den Nachbarn mitzuteilen, dass der Router noch bereit ist. Erhält ein Router keine weiteren Pakete von einem bestimmten Nachbarn, geht er davon aus, dass dieser nicht mehr zur Verfügung steht, entfernt ihn aus seiner Link-State-Datenbank und informiert das Netzwerk darüber.

OSPF-Router geben Daten über ihre Nachbarn an das gesamte Netzwerk weiter, indem sie *Link State Advertisements* (LSA) über alle ihre Netzwerkschnittstellen versenden. Der Empfänger eines LSA-Pakets leitet es weiter, indem er es ebenfalls über alle seine Schnittstellen versendet – mit Ausnahme derjenigen, über die er es empfangen hat. Dieses Verfahren der schnellen Verbreitung von Informationen über ein Netzwerk wird als *Flooding* bezeichnet.

- **Border Gateway Protocol (BGP)**
  Anders als bei den beiden zuvor behandelten Routing-Protokollen handelt es sich beim Border Gateway Protocol (BGP) um ein externes Routing-Protokoll, das Verbindungen zwischen verschiedenen autonomen Systemen regelt. Vom Standpunkt des externen Routings aus erscheinen die autonomen Systeme selbst als in sich geschlossene Gebilde, die nicht näher differenziert werden. BGP wird nur von den Bereichsgrenzroutern der autonomen Systeme ausgeführt, also in der Regel lediglich bei Internetprovidern oder großen Backbone-Netzbetreibern. Die meisten Firmennetze sind dagegen Teil eines autonomen Systems, das von einem Provider betrieben wird, führen also lediglich interne Routing-Protokolle wie OSPF aus.

  Die benachbarten BGP-Router, *Peers* genannt, kommunizieren über eine zuverlässige TCP-Verbindung, die über den dafür vorgesehenen TCP-Port 179 abgewickelt wird. Es wird stets eine vollständige Route mit allen ihren Knotenpunkten angegeben. Dies unterscheidet BGP von den meisten internen Routing-Protokollen, die nur die Verbindungen zu ihren unmittelbaren Nachbarn angeben. Aus diesem Grund wird BGP als *Pfadvektor-Protokoll* bezeichnet.

  Wird das erste Mal eine Verbindung zu einem Peer hergestellt, werden über sogenannte *Update-Pakete* die vollständigen Routing-Tabellen ausgetauscht, danach werden nur noch Änderungen mitgeteilt. Außerdem werden in regelmäßigen Abständen *KEEPALIVE*-Pakete versandt, falls keine Änderungen vorliegen, um den Peers mitzuteilen, dass der Router noch einsatzbereit ist.

Weitere IP-Dienste

In fast allen modernen TCP/IP-Netzwerken – insbesondere in lokalen Firmennetzen, die mit dem Internet verbunden sind – spielen zwei weitere Protokolle eine wichtige Rolle: DHCP dient dazu, den Rechnern im Netzwerk automatisch IP-Adressen zuzuweisen, während das NAT-Protokoll meist vom Standard-Router ausgeführt wird und die im Internet unbrauchbaren privaten IP-Adressen mit öffentlichen überschreibt und umgekehrt. Diese beiden Protokolle sollen hier näher vorgestellt werden.

Das in RFC 2131 und 2132 definierte *Dynamic Host Configuration Protocol* (DHCP) dient dazu, einem Host automatisch TCP/IP-Konfigurationsdaten zuzuweisen. Es ist eine Erweiterung des älteren *Bootstrap Protocol* (BOOTP). Ein Host, der seine Netzwerkparameter über DHCP beziehen möchte, sendet bei Inbetriebnahme eine Broadcast-Anfrage namens *BOOTREQUEST* an die allgemeine Broadcast-Adresse 255.255.255.255. Der Rechner muss also noch nicht einmal wissen, in welchem Netzwerk er sich befindet – das ist beispielsweise ideal für ein Notebook, das manchmal an ein Heim- und manchmal an ein Büronetzwerk angeschlossen wird. Läuft in dem Netz ein DHCP-Server, antwortet er mit einem Satz von Konfigurationsparametern, mit denen der Host seine TCP/IP-Konfiguration vornimmt.

Das wichtigste Merkmal von DHCP besteht in der dynamischen Vergabe von IP-Adressen, die der Netzwerkadministrator*innen das Leben erheblich erleichtert – insbesondere in solchen Netzwerken, in denen häufig Änderungen auftreten. Diese automatische Vergabe erfolgt in Form einer *Lease* (Pacht) mit beschränkter Gültigkeit. Ein Host, der ordnungsgemäß vom Netz abgemeldet wird (ein normaler Vorgang beim Herunterfahren moderner Betriebssysteme), gibt seine IP-Adresse selbst an den DHCP-Server zurück. Das Lease-Verfahren sorgt dagegen dafür, dass IP-Adressen auch dann wieder für den Server verfügbar werden, wenn ein Host unerwartet vom Netz getrennt oder unsachgemäß abgeschaltet wird. Bleibt ein Rechner über den Lease-Zeitraum hinaus im Netz aktiv, erfolgt in der Regel eine Verlängerung der Lease.

Auf dem DHCP-Server muss ein Teil der Adressen des Netzwerks, in dem er sich befindet, als DHCP-Pool konfiguriert werden, aus dem die Adressen automatisch an die anfragenden DHCP-Clients vergeben werden. Es muss darauf geachtet werden, genügend Adressen aus diesem Pool auszuschließen, weil eine Reihe von Internetdiensten eine feste IP-Adresse benötigt oder zumindest besser damit funktioniert.

Network Address Translation (NAT) ist eine relativ neue Entwicklung und löst dementsprechend ein modernes Problem: Immer mehr Netzwerke benötigen permanenten oder auch nur temporären Zugang zum Internet, obwohl sie mit den zuvor vorgestellten privaten IP-Adressen konfiguriert wurden. Es wäre bei der heutigen Anzahl von Internethosts und angeschlossenen Netzen auch gar nicht mehr möglich, allen angeschlossenen Netzwerken öffentliche IP-Adressen zuzuweisen. Da die privaten IP-Adressen jedoch nicht eindeutig sind, müssen sie beim Übergang ins Internet mit einer öffentlichen Adresse überschrieben werden und umgekehrt.

Eine aktuelle Form von NAT, die im Kernel moderner Unix-Systeme konfiguriert werden kann, wird auch als *IP-Masquerading* bezeichnet und geht noch einen Schritt weiter als NAT: Es ist nur eine externe IP-Adresse erforderlich, alle lokalen Adressen werden auf diese eine Adresse abgebildet. Unterschieden werden die Rechner in diesem Fall anhand der Clientportnummer der Datenpakete, die zur Transportebene gehört und im nächsten Abschnitt näher beschrieben wird. Aus diesem Grund wird das echte Masquerading manchmal auch als *PAT* (*Port Address Translation*) bezeichnet. Diese spezielle Form von NAT verbirgt die Details des internen Netzwerks vor dem Internet, die einzelnen Rechner sind von außen nicht erreichbar. Das ist ein angenehmer Nebeneffekt dieses Verfahrens, der zusätzlich der Sicherheit im Netzwerk dient.

Tabelle 5.17 zeigt ein Beispiel für klassisches NAT in einem privaten Netzwerk mit der Adresse 192.168.1.0/24. Jede interne IP-Adresse wird auf eine individuelle externe Adresse abgebildet.

Hostname	Interne IP-Adresse	Externe IP-Adresse
`gandalf`	192.168.1.4	204.81.92.6
`frodo`	192.168.1.5	204.81.92.3
`bilbo`	192.168.1.6	204.81.92.5

Tabelle 5.17 Beispiel für klassisches NAT

In Tabelle 5.18 wird dagegen für dasselbe Netzwerk ein Beispiel für IP-Masquerading (PAT) gezeigt. Das Konzept der Portnummern wird im weiteren Verlauf des Kapitels noch genauer beschrieben.

Hostname	Interne IP-Adresse	Externe IP-Adresse	Externe Portnummer
`gandalf`	192.168.1.4	204.81.92.4	22.191
`frodo`	192.168.1.5		22.192
`bilbo`	192.168.1.6		22.193

Tabelle 5.18 Beispiel für IP-Masquerading

Der Rechner, der NAT ausführt, ist üblicherweise derjenige Router, der das lokale Netz mit dem Netzwerk eines Providers und demzufolge mit dem Internet verbindet. NAT wird von allen gängigen Unix-Versionen sowie von Windows NT und seinen Nachfolgern unterstützt. Außerdem können die meisten ISDN- und DSL-Kompaktrouter NAT ausführen. Eine nähere Beschreibung vieler Aspekte von NAT findet sich in RFC 3022.

Auf einem Linux-System kann NAT beispielsweise durch die Kernel-Firewall *Netfilter/iptables* bereitgestellt werden. Dieser Aspekt wird auch in Kapitel 21, »Computer- und Netzwerksicherheit«, erwähnt.

Die Windows-Desktopsysteme enthalten einen eigenen NAT-Dienst namens *Internet Connection Sharing* (ICS). Dadurch fungiert eine Workstation als NAT-Router und ermöglicht so die Nutzung ihres Internetzugangs durch andere Maschinen. Diese NAT-Variante gilt allerdings als unsicher und ist zudem erheblich unflexibler als IP-Masquerading, da sie automatisch erfolgt und keinerlei Einstellungen zulässt.

In manchen Netzwerken geschieht der Zugriff auf Internetdienste nicht über Router, sondern über Gateways auf Anwendungsebene, die in der Öffentlichkeit als Stellvertreter (Proxys) des eigenen Rechners arbeiten. Solche Proxyserver gibt es für fast alle Dienste. Am bekanntesten sind Webproxys, die oft auch als Cache (Zwischenspeicher) für häufig aufgerufene Websites dienen.

Die bekannteste Proxy- und Webcache-Software ist der Open-Source-Proxy *squid*; Microsoft bietet ebenfalls ein entsprechendes Produkt namens *ISA Server* an. Auch der Webserver Apache (siehe Kapitel 14, »Server für Webanwendungen«) kann optional als Caching-Proxy für verschiedene Protokolle konfiguriert werden. Eine Anleitung dazu finden Sie in der Onlinedokumentation des Webservers Apache, konkret unter *http://httpd.apache.org/docs/2.4/en/mod/mod_proxy.html*.

5.6.3 Transportprotokolle

Eine Anwendung, die Daten über ein TCP/IP-Netzwerk wie das Internet übertragen möchte, beauftragt zu diesem Zweck ein Transportprotokoll, also ein Protokoll der Host-zu-Host-Transportschicht des Internetschichtenmodells. Nachdem im letzten Abschnitt das IP-Routing erläutert wurde, sollten Sie auch verstehen, warum der Vorgang als *Host-zu-Host-Transport* bezeichnet wird: Router betrachten von den Datenpaketen, die sie weiterleiten sollen, immer nur den IP-Header und werten dessen Informationen aus. Aus der Sicht des IP-Protokolls existieren die Daten der Transportschicht nicht. Umgekehrt ist also das Routing ein Implementierungsdetail, das für die Protokolle der Transportschicht nicht sichtbar ist. Aus ihrer Sicht kann der Zielhost immer unmittelbar erreicht werden.

Um den Bedürfnissen verschiedener Anwendungen gerecht zu werden, wurden zwei verschiedene wichtige Transportprotokolle definiert. Das häufiger verwendete *TCP-Protokoll* (definiert in RFC 793), das einen Teil des Namens der Protokollfamilie ausmacht, stellt den zuverlässigen Transport von Datenpaketen in einer definierten Reihenfolge zur Verfügung. Dagegen bietet das *UDP-Protokoll* (RFC 768) die Möglichkeit, Daten auf Kosten der Zuverlässigkeit möglichst schnell zu transportieren.

Ein wenig zwischen Vermittlungs- und Transportschicht liegt das ICMP-Protokoll (*Internet Control Message Protocol*), das für den Versand spezieller Datagramme verwendet wird, mit

deren Hilfe überprüft werden kann, ob ein entfernter Host im Netzwerk aktiv ist. Das entsprechende Dienstprogramm heißt *ping*. Es ist unter Windows und allen Unix-Varianten verfügbar. Um es zu benutzen, geben Sie auf der Konsole einfach Folgendes ein:

```
$ ping Hostname_oder_IP-Adresse
```

Die beiden Protokolle werden in den folgenden Abschnitten genauer beschrieben.

Das Transmission Control Protocol (TCP)

Wie Sie im letzten Abschnitt erfahren haben, werden IP-Datagramme jeweils individuell durch das Netzwerk geleitet. Deshalb kann auf der Basis von Datagrammen kein zuverlässiger Transport kontinuierlicher Datenströme erfolgen, weil es vollkommen normal ist, dass Datagramme nicht in der Reihenfolge ankommen, in der sie abgeschickt wurden. Darüber hinaus ist es möglich, dass sie auch gar nicht ankommen, weil auf der Ebene des IP-Protokolls keine entsprechende Kontrolle durchgeführt wird.

Um nun über den potenziell unsicheren Weg der IP-Datagramme Daten zuverlässig durch das Netzwerk zu transportieren, wird auf dieser höher gelegenen Ebene eine Flusskontrolle implementiert: Im Wesentlichen werden die Datenpakete durch das TCP-Protokoll durchnummeriert, um die korrekte Reihenfolge aufrechtzuerhalten. Im Übrigen erwartet der ursprüngliche Absender für jedes einzelne Datenpaket eine Bestätigung; bleibt sie zu lange aus, versendet der Absender das entsprechende Paket einfach erneut.

Als Erstes sollten Sie sich den TCP-Paket-Header ansehen, der in Tabelle 5.19 gezeigt wird.

Byte	0	1		2	3
0	Quellport			Zielport	
4	Sequenznummer				
8	Bestätigungsnummer				
12	Offset	reserviert	Flags	Fenster	
16	Prüfsumme			Urgent-Zeiger	
20	Optionen				Padding

Tabelle 5.19 Aufbau des TCP-Datenpaket-Headers

Ein TCP-Datenpaket-Header besteht aus den folgenden Bestandteilen:

- *Quellport* (16 Bit): Ports stellen eine Methode zur Identifikation der konkreten Anwendungen zur Verfügung, die auf den beteiligten Hosts miteinander kommunizieren. Der Quellport ist die Portnummer des Absenders.

- *Zielport* (16 Bit): Dies ist entsprechend die Portnummer des Empfängers.
- *Sequenznummer* (32 Bit): Normalerweise gibt diese Nummer an, dem wievielten Byte der zu übertragenden Sequenz das erste Nutzdatenbyte des Pakets entspricht. Ausnahme: Ist das SYN-Flag gesetzt, wird die Anfangssequenznummer (*Initial Sequence Number*, ISN) angegeben.
- *Bestätigungsnummer* (32 Bit): Das Startbyte der Sequenz, deren Übertragung als Nächstes erwartet wird; ist nur bei gesetztem ACK-Bit von Bedeutung.
- *Offset* (4 Bit): Anzahl der 32-Bit-Wörter, aus denen der Header besteht; gibt entsprechend den Beginn der Nutzdaten im Paket an.
- *reserviert* (6 Bit): Reserviert für zukünftige Anwendungen; muss 0 sein.
- *Flags* (6 Bit): Verschiedene Status-Bits, im Einzelnen:
  - URG: Urgent Data wird versandt; der Inhalt des Urgent-Zeigers muss beachtet werden.
  - ACK: Acknowledgement – das Bestätigungsfeld muss berücksichtigt werden.
  - PSH: Push-Funktion – ist dieses Bit gesetzt, wird die Pufferung des Pakets verhindert; es wird unmittelbar gesendet.
  - RST: Reset – Verbindung zurücksetzen.
  - SYN: Sequenznummern synchronisieren.
  - FIN: Ende der Sequenz, keine weiteren Daten vom Absender.
- *Fenster* (16 Bit): Die Anzahl von Datenbytes, die der Absender des Pakets zu empfangen bereit ist; basiert unter anderem auf der IP-MTU der verwendeten Schnittstelle.
- *Prüfsumme* (16 Bit): Anhand dieser einfacheren Plausibilitätskontrolle kann die Korrektheit der übertragenen Daten überprüft werden.
- *Urgent-Zeiger* (16 Bit): Ein Zeiger auf das Byte der aktuellen Sequenz, das Urgent Data enthält. Wird nur ausgewertet, wenn das URG-Flag gesetzt ist.
- *Optionen* (variable Länge): Enthält verschiedene hersteller- und implementierungsabhängige Zusatzinformationen; stets ein Vielfaches von 8 Bit lang.

Zwischen den beiden Hosts, die über TCP kommunizieren, wird eine virtuelle Punkt-zu-Punkt-Verbindung hergestellt; aus diesem Grund wird TCP auch als *verbindungsorientiertes Protokoll* bezeichnet. Dies ermöglicht den Transport eines kontinuierlichen Datenstroms über die potenziell unzuverlässigen IP-Datagramme, in die die TCP-Pakete verpackt werden. Um die Datenübertragung einzuleiten, findet zunächst ein sogenannter *Drei-Wege-Handshake* statt: Drei spezielle Datenpakete ohne Nutzdateninhalt werden ausgetauscht. Der Host, der die Verbindung initiiert, sendet ein Paket mit gesetztem SYN-Bit an den Empfänger. Dieser schickt ein Paket zurück, bei dem SYN und ACK gesetzt sind, und erwartet wiederum eine Antwort, bei der nur das ACK-Flag gesetzt ist. Erst nachdem das geschehen ist, beginnt die eigentliche Übertragung von Nutzdaten. Dieses Vorgehen garantiert, dass beide Hosts bereit sind, miteinander zu kommunizieren.

Anschließend sendet der Absender ein Paket nach dem anderen an den Empfängerhost, wobei die Sequenznummer stets um die im vorangegangenen Paket versandte Nutzdatenmenge erhöht wird. Der Empfänger beantwortet jedes empfangene Paket, dessen Prüfsumme mit dem Inhalt übereinstimmt, mit einem Bestätigungspaket, dessen ACK-Flag also gesetzt ist. Der Wert des Bestätigungsfelds ist der Byte-Offset der nächsten Datensequenz, die der Empfänger erwartet, ist also die Summe aus Sequenznummer und Nutzdatenlänge des soeben empfangenen Pakets.

Erhält der Absender die Bestätigung nicht innerhalb einer definierten Zeit (Time-out), sendet er das entsprechende Paket unaufgefordert erneut. Dieses Verfahren wird *positive Bestätigung* (*Positive Acknowledgement*) genannt, da lediglich der Erfolg gemeldet wird; von einem Misserfolg wird automatisch ausgegangen, wenn keine Meldung erfolgt. Dieses Verfahren ist zuverlässiger als das Arbeiten mit Misserfolgsmeldungen: Kommt die Erfolgsmeldung nicht an, wird das Paket einfach erneut versandt, ansonsten gibt es aber keine schädlichen Folgen (abgesehen von dem geringen Mehraufwand für ein überflüssig versandtes Paket, falls einmal lediglich die Bestätigung verloren gegangen ist). Käme dagegen eine Misserfolgsmeldung nicht an, würde das betreffende Paket nicht erneut versandt und den Empfänger niemals erreichen.

Ein weiterer wichtiger Bestandteil von TCP-Paketen sind die beiden 16 Bit langen Portnummern. Jedes Paket kann anhand des Portnummernpaars als zu einer bestimmten Sequenz und Anwendung gehörig identifiziert werden. Das ist auch absolut notwendig: Stellen Sie sich vor, Sie hätten zwei Browserfenster geöffnet, und in beiden würden gleichzeitig verschiedene Seiten von *www.rheinwerk-verlag.de* geladen. Anhand der IP-Adressen können die beiden Datenübertragungen nicht voneinander unterschieden werden, da die beiden Hosts, die hier miteinander kommunizieren, identisch sind. Es könnte also sehr leicht passieren, dass die Daten fehlerhaft zugeordnet werden und Sie zwei seltsame Mischungen der beiden Dokumente erhalten – ein Effekt wie in dem Horror-Klassiker »Die Fliege«!

Dieses Szenario kann deshalb nicht eintreten, weil die beiden Datenübertragungen nicht über dasselbe Paar von Portnummern erfolgen. In der Regel ist die Portnummer des Servers festgelegt, während der Client irgendeinen Port wählt, der gerade frei ist. Die untersten 1.024 Portnummern sind als sogenannte *Well-known Ports* für Standard-Serverdienste fest vergeben; für Clients wird eine zufällige Nummer (ein sogenannter *Ephemeral Port*) zwischen 1.024 und 65.535 verwendet. Beispielsweise benutzen Webserver, also HTTP-Server, üblicherweise den TCP-Port 80, FTP-Server den Port 21 und Telnet-Server den Port 23. Eine kleine Liste, die auch UDP betrifft, finden Sie in Tabelle 5.21.

In dem Beispiel mit den beiden Browserfenstern ist der Serverport jeweils 80; die Clientports sind dagegen unterschiedlich, beispielsweise 16832 und 16723. Dies verdeutlicht die Formulierung, dass nur die Portpaare und nicht die beiden einzelnen Ports unterschiedlich sein müssen, um Sequenzen voneinander abzugrenzen.

Gewöhnlich »lauscht« ein TCP-Serverdienst an seinem speziellen Port auf ankommende Verbindungsversuche. Unternimmt ein Client den Versuch, eine TCP-Verbindung mit dieser speziellen Portnummer als Empfängerport und einer zufälligen Nummer als Absender herzustellen, akzeptiert der Server dies nach den Regeln des Drei-Wege-Handshakes; eine Verbindung für den gegenseitigen Datenaustausch ist hergestellt.

Interessant ist schließlich das Thema Urgent Data: Manchmal muss ein Host einen anderen über einen besonderen Zustand informieren, beispielsweise eine Konfigurationsänderung oder einen lokal initiierten Abbruch mitteilen. Zu diesem Zweck wird das URG-Flag gesetzt; der Empfänger ermittelt daraufhin aus dem Urgent-Zeiger-Feld des Paket-Headers die Byte-Nummer innerhalb der Sequenz, in der sich diese dringlichen Daten befinden. Es handelt sich stets nur um ein einziges Byte, das auch als *Out-of-Bound-Byte* bezeichnet wird, weil es nicht zum gewöhnlichen Datenstrom gehört. Es ist also unmöglich, auf diesem Weg eine längere dringende Mitteilung zu versenden, aber immerhin besteht die Möglichkeit, bestimmte zwischen den Anwendungen vereinbarte Signale auszutauschen.

Das User Datagram Protocol (UDP)

Manche Anwendungen möchten auf den Komfort und die Sicherheit von TCP getrost verzichten, wenn sie die Daten dafür schneller ans Ziel befördern können. Die Möglichkeit eines solchen möglichst schnellen Versands bietet das UDP-Protokoll. Ob eine Anwendung für ihre Datenübertragung nun TCP, UDP oder beide verwenden möchte, entscheidet sie selbst.

Stellen Sie sich als Beispiel ein Netzwerkspiel vor, eine virtuelle 3D-Umgebung, in der Sie gegen Ihre Mitspieler »kämpfen« können. Ein solches Spiel ist ideal für die Erklärung des Nutzens beider Übertragungsarten geeignet: Bestimmte grundlegende Konfigurationsdaten (Lebt der Gegner überhaupt noch? Hat er auf mich geschossen?) sind entscheidend für den eigentlichen Spielverlauf und sollten deshalb zuverlässig über TCP übertragen werden. Dagegen sind andere Details (Pose und Gesichtsausdruck der gegnerischen Spielfigur, die Position von Gegnern außerhalb des »Gesichtsfelds« etc.) nicht so wichtig. Wenn überhaupt, sollten sie möglichst schnell übertragen werden. Fallen sie vorübergehend aus, schadet das auch nichts – ideale Kandidaten für die Übertragung mithilfe des schnelleren, aber weniger zuverlässigen UDP-Protokolls.

Der Hauptgrund dafür, dass sich Daten über UDP schneller übertragen lassen als über TCP, ist der erheblich kleinere und weniger komplexe Paket-Header. Der Aufbau dieses Headers, der gerade einmal (unveränderlich) 64 Bit groß ist, wird in Tabelle 5.20 dargestellt.

Byte	0	1	2	3
0	Quellport		Zielport	
4	Länge		Prüfsumme	

Tabelle 5.20 Aufbau des UDP-Headers

Die einzelnen Header-Felder haben dieselbe Bedeutung wie die gleichnamigen Felder beim TCP-Protokoll. Mit der Länge ist hier die Länge des gesamten Pakets inklusive dieses Headers gemeint. Der Quellport wird häufig einfach auf 0 gesetzt: Da UDP dem schnellen Versand einer einzelnen Nachricht dient, auf die in der Regel keine Antwort erwartet wird, ist es nicht nötig, diese Information festzulegen. Der Zielport ist dagegen meist der festgelegte Port des UDP-Servers, an den das Paket verschickt wird. UDP wird für viele einfache Internetdienste verwendet: die Uhrzeitsynchronisation über ein Netzwerk, den Echo-Dienst zur Kontrolle der Funktionstüchtigkeit von Verbindungen oder entfernten Hosts etc.

Im Gegensatz zum verbindungsorientierten TCP wird UDP als *nachrichtenorientiertes Protokoll* bezeichnet, da es dem schnellen, verbindungslosen Versand einzelner Pakete in Form kurzer Meldungen dient. Dies erklärt auch den Namen des Protokolls: Einer Anwendung, die von diesem Transportdienst Gebrauch macht, wird der unmittelbare und leichtgewichtige Zugriff auf IP-Datagramme ermöglicht.

Die Portnummern für gängige Serverdienste (bei UDP spricht man häufig auch von *Servicenummern*) liegen wie bei TCP zwischen 0 und 1.023. Sie werden genau wie öffentliche IP-Adressen von der IANA vergeben. In der Regel wird dieselbe Portnummer für beide Transportprotokolle verwendet, obwohl die meisten Anwendungen nur auf jeweils einem der beiden Protokolle laufen. Tabelle 5.21 zeigt einige oft verwendete Beispiele mit ihrem offiziellen Namen und dem am häufigsten verwendeten Transportprotokoll. Die vollständige Liste aller öffentlichen Serverdienste finden Sie unter *http://www.iana.org/assignments/port-numbers*. Falls Sie ein Unix-System nutzen, steht eine ähnliche, möglicherweise weniger vollständige Liste in der Konfigurationsdatei */etc/services* zur Verfügung.

Nummer	Transportprotokoll	Name	Beschreibung
7	TCP, UDP	`echo`	genaue Rückgabe der übermittelten Daten zur Kontrolle
13	TCP, UDP	`daytime`	Datum und Uhrzeit (RFC 867)
20	TCP	`ftp`	FTP-Datenstrom
21	TCP	`ftp`	FTP-Steuerung
22	TCP	`ssh`	Secure Shell – Telnet-Alternative mit Verschlüsselung
23	TCP	`telnet`	Terminal-Emulation
25	TCP	`smtp`	E-Mail-Versand
53	TCP, UDP	`domain`	Nameserver-Abfragen

Tabelle 5.21 Einige TCP/UDP-Portnummern für gängige Dienste

Nummer	Transportprotokoll	Name	Beschreibung
80	TCP	`http`	Webserver
110	TCP	`pop3`	E-Mail-Postfach-Server (klassisch)
143	TCP	`imap`	E-Mail-Postfach-Server (modern)
443	TCP, UDP	`https`	SSL-verschlüsselte Webserverkommunikation

Tabelle 5.21 Einige TCP/UDP-Portnummern für gängige Dienste (Forts.)

5.6.4 Das Domain Name System (DNS)

Der Einsatz von IP-Adressen zum Erreichen entfernter Rechner ist ideal, solange in die Datenübertragung nur Computer involviert sind. Für die Verwendung durch Menschen sind sie weniger gut geeignet (es gibt zum Beispiel nur wenige Menschen, die sich Telefonnummern auf Anhieb besser merken können als die zugehörigen Namen). Aus diesem Grund ist es seit den Anfängen des Internets und seiner Vorläufer üblich, einen Mechanismus einzurichten, der den beteiligten Menschen das Nutzen lesefreundlicher Namen anstelle der unhandlichen IP-Adressen ermöglicht.

Als das ARPANET entwickelt wurde, behalf man sich mit einer einfachen Textdatei, die pro Zeile einen Hostnamen und eine IP-Adresse nebeneinander auflistete. Noch heute verwenden Unix-Rechner eine ähnliche Datei namens */etc/hosts*. Auch unter Windows ist das Verfahren bekannt. Hier befindet sich die Datei in *<Windows-Verzeichnis>\system32\drivers\etc* und heißt – untypisch für Windows ohne Dateiendung – ebenfalls nur *hosts*. Allerdings wird dieses Verfahren heute immer seltener für die Namenszuordnung in lokalen Netzen eingesetzt, weil in immer mehr Firmennetzen DHCP verwendet wird.

Findet der Rechner einen Namenseintrag in seiner */etc/hosts*-Datei, wird er die entsprechende Adresse nicht mehr bei einem Nameserver nachfragen, sodass sich die IP-Adressen zu unerwünschten Hosts (zum Beispiel von bekannten Betreibern von Phishing und anderen Betrügereien oder hartnäckigen Werbetreibenden) durch harmlose lokale ersetzen lassen.

Früher wurde die Datei namens *hosts.txt* zentral verwaltet und regelmäßig unter den teilnehmenden Hosts im ARPANET ausgetauscht, um die Namensdaten aktuell zu halten. Als das Netz jedoch immer größer wurde, funktionierte dieses System nicht mehr, weil man mit den häufigen Änderungen nicht mehr nachkam und weil die gesamte Datei außerdem sehr umfangreich war und ihr Versand eine erhebliche Menge an Netzwerkverkehr erzeugte.

Schließlich wurde anstelle der einfachen Textdatei eine hierarchische, vernetzte Datenbank eingeführt, die bis heute ein verteiltes Netz von Nameservern bildet. Diese Server geben auf Anfrage Auskunft über die zu einem Hostnamen gehörende IP-Adresse oder umgekehrt. Außerdem leiten sie die Anfrage automatisch weiter, wenn sie selbst keine Antwort wissen. Das

System wird als *Domain Name System* (DNS) bezeichnet und ist Thema einer ganzen Reihe von RFCs. Die wesentlichen Grundlagen werden in RFC 1034 und 1035 beschrieben.

Damit Hostnamen im gesamten Internet eindeutig sind, werden sie hierarchisch als sogenannte *Domainnamen* vergeben. Zu diesem Zweck wird ein Name aus immer spezialisierteren Bestandteilen zusammengesetzt; das System lässt sich mit einem Pfad in einem Dateisystem vergleichen. Allerdings besteht ein wesentlicher Unterschied: Beim Dateisystempfad steht der allgemeinste Name vorn und der speziellste hinten, während es beim Domainnamen genau umgekehrt ist.

Beispielsweise bedeutet der Unix-Pfad */home/sascha/hb_fachinfo/netzwerk/protokolle.txt*, dass sich die Datei *protokolle.txt* im Verzeichnis *netzwerk* befindet, einem Unterverzeichnis von *hb_fachinfo*, das wiederum dem Verzeichnis *sascha* untergeordnet ist. *sascha* ist seinerseits ein Unterverzeichnis von *home*, das schließlich direkt unter der Wurzel des Dateisystems (*/*) liegt.

Dagegen ist der Domainname *www.buecher.lingoworld.de* genau andersherum aufgebaut: Der Host/Dienst *www* liegt in der Domain *buecher*, einer Subdomain von *lingoworld* in der Top-Level-Domain *.de*. Die Wurzel des DNS-Systems selbst ist nicht sichtbar, weil ihr Name der leere String ist.

Auf der jeweiligen Ebene des DNS-Systems muss ein bestimmter Name einmalig sein. Beispielsweise kann es *buecher.lingoworld.de* nur einmal geben. Unterhalb dieser Domain können untergeordnete Domains (Subdomains) oder die Namen einzelner Hosts oder Serverdienste bestehen, beispielsweise *www.buecher.lingoworld.de*, *ftp.buecher.lingoworld.de* oder *neuheiten.buecher.lingoworld.de*. Im Übrigen dürfen dieselben Namen natürlich auf über- oder untergeordneten oder auch auf »Geschwister-Ebenen« existieren: Es kann die Website *www.lingoworld.de* ebenso geben wie etwa *www.download.lingoworld.de*. Selbstverständlich ist auch *www.buecher.de* kein Problem – es handelt sich um eine andere Domain unterhalb der Top-Level-Domain *de*.

Aus Sicht der DNS-Administration wird jede Ebene eines solchen Namens auch als *Zone* bezeichnet, weil eine solche Ebene jeweils unabhängig von den übergeordneten Ebenen verwaltet wird. Beispielsweise kann der Administrator der Domain *lingoworld.de* Subdomains wie *buecher.lingoworld.de* oder *download.lingoworld.de* einrichten. Er kann die Verantwortung für eine Subdomain auch an jemand anderen delegieren, für den dann beispielsweise *buecher.lingoworld.de* wieder eine unabhängige Zone darstellt. Andererseits kann der Zonenverantwortliche für *lingoworld.de* nicht auf andere Zonen in der Domain *.de* zugreifen; beispielsweise geht ihn die Konfiguration der Zone *google.de* nichts an.

Die Infrastruktur der Domainnamen wird von den über das gesamte Internet verbreiteten Nameservern verwaltet. Alle führen ein Programm aus, das Anfragen nach Name-Adresse-Zuordnungen beantwortet, unbekannte Zuordnungen bei anderen Nameservern erfragt und dann meistens dauerhaft speichert. Die am häufigsten verwendete derartige Software heißt

BIND (*Berkeley Internet Name Domain*) und läuft unter allen Unix-Varianten; sie wird in Kapitel 15, »Weitere Internet-Serverdienste«, vorgestellt.

Auf der obersten Ebene des DNS existiert die spezielle Zone, deren Name der leere String ist. Diese Zone wird durch die Root-Nameserver der ICANN verwaltet und enthält Verweise auf alle Top-Level-Domains. Davon gibt es, organisatorisch gesehen, zwei Sorten (auch wenn es keinen technischen Unterschied gibt):

- Die *Generic TLDs* (allgemeine Top-Level-Domains) wie *.com* oder *.org* unterteilen die jeweiligen Domains, die unter ihnen liegen, nach der Funktion ihrer Betreiber.
- Die *Country TLDs* oder *ccTLDs* (Länder-TLDs) sind dagegen für eine geografische Einteilung vorgesehen.

In der Praxis kommt es ohnehin zu einer Vermischung: Zum einen sind viele Generic TLDs mittlerweile für beliebige Betreiber nutzbar, zum anderen gibt es einige Länder-TLDs, die wegen ihrer spezifischen Abkürzung für bestimmte Branchen interessant sind – am bekanntesten ist in diesem Zusammenhang wohl der Südsee-Inselstaat Tuvalu mit seiner bei Fernsehsendern und Web-Videodiensten beliebten TLD *.tv*.

Tabelle 5.22 listet einige häufig verwendete Top-Level-Domains auf. Die mit Abstand meisten Betreiberdomains enthält die Generic TLD *.com*, gefolgt von der länderspezifischen Domain *.de* (Deutschland).

Top-Level-Domain	Bedeutung
Generic Top-Level-Domains	
.com	*commercial* (Firmen)
.org	*organization* (Organisationen und Vereine)
.net	*network* (Netzwerkbetreiber, Internetinfrastruktur)
.edu	*educational* (US-Schulen und -Universitäten)
.gov	*government* (US-Regierung, US-Behörden, öffentlicher Dienst)
.mil	*military* (US-Militär)
.info	*information* (allgemeine Informationsdienste)
.aero	*aeronautics* (Luftfahrtindustrie, Fluggesellschaften)
Länder-Top-Level-Domains	
.at	Österreich
.ch	Schweiz

Tabelle 5.22 Übersicht über einige wichtige Top-Level-Domains

Top-Level-Domain	Bedeutung
.cn	Volksrepublik China
.de	Deutschland
.es	Spanien
.fr	Frankreich
.it	Italien
.jp	Japan
.ru	Russland
.tr	Türkei
.uk	Vereinigtes Königreich
.us	USA
.va	Vatikanstadt

Tabelle 5.22 Übersicht über einige wichtige Top-Level-Domains (Forts.)

Der jeweilige Haupt-Nameserver einer Top-Level-Domain enthält Verweise auf sämtliche unterhalb dieser Domain befindlichen Second-Level-Domains. Je nach konkreter TLD handelt es sich dabei entweder unmittelbar um die einzelnen Domains, die von Betreibern angemeldet werden können, oder eine Domain ist in sich noch einmal in Organisationsstrukturen unterteilt. Bei allen Generic TLDs und den meisten Länder-TLDs ist Ersteres der Fall. Nur einige Länder-TLDs werden noch einmal organisatorisch unterteilt: Zum Beispiel verwendet das Vereinigte Königreich Unterteilungen wie *.co.uk* für Firmen, *.ac.uk* für Universitäten und *.org.uk* für Vereine und Organisationen; auch in der Türkei findet man entsprechende Unterteilungen wie *.com.tr*, *.edu.tr* und *.org.tr*.

In den letzten Jahren wurden zahllose neue Top-Level-Domains eingeführt, beispielsweise für spezifische Arten von Firmen oder Vereinen wie *.tel* für Telekommunikationsunternehmen oder *.museum* für Museen, aber auch für Städte oder Regionen (in Deutschland wurde die Einführung solcher TLDs oft mit dem Beispiel *.berlin* beworben; meine Heimatstadt hat sogar zwei eigene TLDs: den deutschen Namen *.koeln* und das internationale *.cologne*).

Aus Sicherheitsgründen sollten die Zonendaten für die Domain eines einzelnen Betreibers auf mindestens zwei voneinander unabhängigen (das heißt in verschiedenen autonomen Systemen befindlichen) Nameservern vorliegen. Die Daten müssen dafür nur auf einem der beiden Server erstellt werden, der als *primärer Master-Nameserver* bezeichnet wird; der andere – *Slave-Nameserver* genannt – repliziert sie automatisch. Größere Unternehmen und Institutionen verwalten in der Regel ihre eigenen Zonen. Der externe Slave-Nameserver mit

denselben Zonendaten befindet sich in diesem Fall meist beim zuständigen Backbone-Provider, über den diese Betreiber mit dem Internet verbunden sind. Privatpersonen oder kleinere Firmen besitzen dagegen zwar häufig eine eigene Domain (*www.meinname.de* ist werbewirksamer als so etwas wie *home.t-online.de/users/meinname*), unter dieser Domain laufen allerdings oft lediglich eine beim Provider gehostete Website und einige E-Mail-Adressen. In diesem Fall werden die Zonendaten meist beim Hosting-Provider und einem anderen Provider verwaltet; die beiden Provider stellen sich den Slave-Nameservice dann gegenseitig zur Verfügung.

Bei den meisten Einzelrechnern oder kleineren Firmennetzwerken besteht die ganze DNS-Konfiguration oft lediglich aus der Eingabe der IP-Adresse eines Nameservers des eigenen Providers; in vielen Fällen ist sogar dies unnötig, weil die Standard-Nameserver beim Verbindungsaufbau bekannt gegeben werden. Die Nameserver werden stets befragt, wenn Namensdaten erforderlich sind.

Gebe ich zum Beispiel in meinem Webbrowser »www.google.de« ein, überprüft dieser zunächst, ob er die IP-Adresse vielleicht bereits kennt. Andernfalls fragt er den Standard-Nameserver des Providers. Dieser weiß die Antwort entweder selbst und liefert sie unmittelbar zurück oder wendet sich an den übergeordneten Nameserver – in diesem Fall den für die Top-Level-Domain *.de* zuständigen Server. Der wiederum kennt die für die Domain *google.de* zuständigen Nameserver und leitet die Anfrage an den ersten von ihnen weiter. Dieser ermittelt die IP-Adresse des Dienstes *www.google.de* und gibt sie zurück. Nun weiß der Browser, welche IP-Adresse er verwenden muss. Außerdem speichert der Nameserver des Providers die gefundene Adresse ebenfalls in seinem Cache ab, um die nächste entsprechende Anfrage schneller beantworten zu können.

Insgesamt stellt das DNS ein leistungsfähiges, flexibles und effizientes System zur Verwaltung benutzerfreundlicher Hostnamen zur Verfügung. Es wird im gesamten Internet eingesetzt und zumindest clientseitig von jedem beliebigen Betriebssystem unterstützt. Allerdings handelt es sich nicht um die einzige Art und Weise der Namensverwaltung. Gerade in herstellerabhängigen lokalen Netzen werden Dienste wie der Windows-Namensdienst WINS oder das *Network Information System* (NIS) von Sun eingesetzt. Letzteres ist nicht nur ein Namens-, sondern auch ein einfacher Verzeichnisdienst.

5.6.5 Verschiedene Internetanwendungsprotokolle

Genau wie beinahe jede Hardware die TCP/IP-Protokolle unterstützt, laufen auf der Anwendungsschicht des Internetprotokollstapels auch fast alle Arten von Netzwerkanwendungen. Dazu gehören unter anderem Anwendungsprotokolle, die ursprünglich für bestimmte herstellerabhängige Netzwerke konzipiert wurden, beispielsweise die Standard-Fileserver-Protokolle der diversen Betriebssysteme: Das unter Windows verwendete SMB-Protokoll (*Server Message Blocks*) wurde zunächst auf Microsofts eigenes NetBEUI-Netzwerk aufgesetzt, das

von Apple konzipierte AppleShare lief ursprünglich nur unter AppleTalk. Mittlerweile werden diese speziellen Anwendungsprotokolle praktisch exklusiv auf TCP/IP aufgesetzt.

An dieser Stelle geht es lediglich darum, die grundlegende Funktionsweise einiger typischer Internetdienste auf der Ebene ihrer Protokolle zu beschreiben. Falls Sie also Details über die Verwendung von Internet-Client-Server-Diensten oder deren Konfiguration unter einem bestimmten Betriebssystem suchen, sollten Sie Kapitel 14, »Server für Webanwendungen«, und Kapitel 15, »Weitere Internet-Serverdienste«, lesen. Hier erfahren Sie dagegen eher, was hinter den Kulissen wirklich passiert, wenn Sie eine E-Mail versenden oder eine Webseite anfordern.

Das (für Administrator*innen und Programmierer*innen) Angenehme an den meisten Internetanwendungsprotokollen ist, dass die Protokollbefehle in Form von Klartextwörtern in Englisch verschickt werden. Wenn Sie mit einem Packet Sniffer oder einfach mit `telnet` die Inhalte der über das Netzwerk übertragenen Datenpakete kontrollieren, können Sie deshalb unmittelbar verstehen, worüber die verschiedenen Hosts »reden«. Auf diese Weise ist es verhältnismäßig einfach, Konfigurations- oder Programmierfehler auf der Ebene der Anwendungsprotokolle zu entdecken und zu beseitigen.

In der Regel bestehen die Anforderungen eines Internetanwendungsclients aus einzeiligen Befehlen, die vom Server mit einer Statusmeldung und manchmal auch mit der Lieferung konkreter Daten beantwortet werden. Sie können das Verhalten eines Clients simulieren, indem Sie mit einem Terminalprogramm wie `telnet` eine Verbindung zu dem passenden Host und Port aufbauen und die entsprechenden Befehle von Hand eintippen.

Telnet und SSH

Eine der ältesten Anwendungen des Internets ist die Terminal-Emulation: Ein Programm ermöglicht Ihnen über ein Terminal, das an Ihren Computer angeschlossen ist, die Arbeit an einem anderen Computer, zu dem eine Netzwerkverbindung besteht. Telnet ist eines der wichtigsten Werkzeuge für Systemadministrator*innen, die auf diese Weise entfernte Rechner verwalten, ohne sich physisch dorthin zu begeben (insbesondere an Wochenenden oder nach Feierabend schätzen Admins diese Möglichkeit, weil sie eventuelle Pannenhilfe von zu Hause aus erledigen können). Die Telnet-Spezifikation ist in RFC 854 festgelegt.

Fast alles, was an dieser Stelle über Telnet gesagt wird, gilt sinngemäß auch für SSH, die *Secure Shell*. Im Grunde genommen handelt es sich dabei um eine sichere Variante von Telnet, die mit Verschlüsselung arbeitet. Der gravierendste Schönheitsfehler des klassischen Telnet besteht nämlich darin, dass es Daten im Klartext überträgt – und das gilt unter anderem auch für Passwörter und ähnlich sensible Daten. SSH ist nicht in einem RFC spezifiziert, denn obwohl es sich aus frei verfügbaren Komponenten zusammensetzt, ist die ursprüngliche Implementierung kommerziell. Nähere Informationen dazu erhalten Sie auf der Website *http://www.ssh.com*. Eine freie Implementierung, die in den meisten Linux- und anderen Unix-Systemen zum Einsatz kommt, ist OpenSSH (*http://www.openssh.com*).

Der Telnet-Server lauscht auf dem TCP-Port 23 auf eingehende Verbindungen (SSH verwendet Port 22). Wenn ein TCP-Verbindungsversuch erfolgt, wird auf dem entfernten Host zunächst nach Usernamen und Passwort gefragt, bevor die Verbindung tatsächlich genutzt werden kann. Im Grunde genommen wird eine Standard-Unix-Shell zur Verfügung gestellt, in der man auch lokal auf dem entsprechenden Rechner arbeiten würde.

Telnet und SSH sind daher beliebig flexibel, was den Inhalt der in beide Richtungen übermittelten Daten angeht. Wenn Sie erst einmal mit dem Telnet-Server verbunden sind, können Sie jedes beliebige Programm auf dem entfernten Host ausführen, für das Sie Benutzerrechte besitzen. Dazu gehören auch solche Programme, die nicht zeilenorientiert, sondern mit einer Vollbildmaske arbeiten – zum Beispiel die klassischen Unix-Texteditoren *vi* und *Emacs*. Deshalb genügt die zeilenorientierte Kommunikation zwischen Client und Server bei Telnet nicht: In einem Vollbildprogramm kann jeder einzelne Tastendruck eine Bedeutung haben, die unmittelbar umgesetzt werden muss. Falls Sie Beispiele dafür sehen möchten, was Sie in einer SSH- oder Telnet-Sitzung eingeben können, lesen Sie einfach die Abschnitte über die Linux- oder Unix-Shell in Kapitel 6, »Betriebssysteme«. Alles, was dort steht, gilt auch für den Fernzugriff.

Die einzigen Programme, die nicht über Telnet ausgeführt werden können, sind solche, die auf einer grafischen Benutzeroberfläche laufen. Allerdings bietet die Unix-Welt auch für dieses Problem die passende Lösung: Sie benötigen auf Ihrem lokalen Rechner zusätzlich zu dem Telnet-Client einen X-Window-Server, der die grundlegenden Zeichenfunktionen für das GUI zur Verfügung stellt. Sobald dieser X-Server läuft, können Sie ein X-basiertes Anwendungsprogramm auf dem entfernten Server starten und Ihren eigenen Rechner als Ziel der grafischen Darstellung angeben (in der Regel mit dem Parameter `display`). Angenommen, Ihr eigener Rechner besäße im lokalen Netz die IP-Adresse 192.168.0.9. Dann könnten Sie in das Telnet-Programm, in dem eine Sitzung auf einem anderen Rechner läuft, Folgendes eingeben, um in Ihrem X-Server ein *xterm* (ein X-basiertes Terminal) zu starten:

```
# xterm -display 192.168.0.9:0.0
```

Der Zusatz `0.0` hinter der IP-Adresse bedeutet sinngemäß »erster X-Server, erster Bildschirm«. Wichtig ist, dass Sie den Begriff *X-Server* nicht falsch verstehen: Hier läuft der Server, der die grafische Oberfläche als Dienstleistung zur Verfügung stellt, auf Ihrem eigenen Rechner, während der Client das auf dem entfernten Rechner laufende Programm ist, dessen Ausgabe in Ihrem lokalen X-Window-System erfolgt. Es gibt auch X-Server für andere Betriebssysteme als Unix, beispielsweise für Windows. Sie sind natürlich nicht für die grafische Darstellung lokaler Programme gedacht (das können die eingebauten GUIs von macOS oder Windows selbst gut genug), sondern für grafisch orientierte Programme, die auf entfernten Unix-Rechnern laufen.

Im Übrigen sollten Sie das Telnet-Protokoll, das die Terminal-Emulation bereitstellt, nicht mit dem Unix- und Windows-Dienstprogramm `telnet` verwechseln. Letzteres kann nämlich

– wie bereits erwähnt – mit fast jedem Internetserver kommunizieren, wenn Sie die passenden Parameter eingeben (IP-Adresse beziehungsweise Hostname und Portnummer beziehungsweise Standarddienstname).

FTP

Das *File Transfer Protocol* (FTP) ist beinahe das genaue Gegenteil von Telnet: ein aus ganz wenigen Befehlen bestehendes, klartextbasiertes, zeilenorientiertes Protokoll. Es gehört zu den frühesten Internetanwendungen überhaupt. Seine erste Definition steht in RFC 172 von 1971, die aktuelle Spezifikation befindet sich in RFC 959. Den reinen Datei-Download über FTP beherrscht heutzutage fast jeder Webbrowser; die meisten stellen auch die Verzeichnisansicht des entfernten Rechners übersichtlich und angenehm dar. Genau wie Telnet weitgehend von SSH abgelöst wurde, werden inzwischen meist verschlüsselte FTP-Varianten wie SFTP oder FTPS eingesetzt, für die jedoch die meisten im Folgenden beschriebenen Verfahren und Befehle ebenfalls gelten.

In der Praxis wird jedoch überwiegend ein grafisch orientierter FTP-Client verwendet, der dem lokalen Dateinavigator Ihres Betriebssystems idealerweise möglichst ähnlich sieht. Die häufigste Anwendung für ein solches Programm dürfte die Pflege einer eigenen Website sein. Dabei bearbeiten Sie die Daten in der Regel auf Ihrem eigenen Rechner und laden sie anschließend mithilfe eines solchen FTP-Programms auf den Server Ihres Hosting-Providers hoch, um sie zu veröffentlichen. Bekannte FTP-Clients sind beispielsweise FileZilla für verschiedene Betriebssysteme, WS_FTP für Windows oder Fetch für macOS. Auch in gängige Website-Editoren wie Adobe Dreamweaver sind FTP-Module eingebaut.

Falls Sie jedoch genau sehen möchten, wie FTP-Client (auf Ihrem eigenen Rechner) und -Server (auf dem entfernten Rechner) miteinander kommunizieren, können Sie das in Unix und Windows eingebaute Konsolenprogramm `ftp` verwenden. Die Befehle, die Sie auf der Clientseite eingeben, sind jeweils einzeilig und bestehen aus einem Schlüsselwort mit eventuellen Parametern, gefolgt von einem Zeilenumbruch. Die Antwort des Servers ist zunächst eine Statusmeldung, die aus einer dreistelligen dezimalen Codenummer und einem Meldungstext besteht; häufig folgen auf die Statusmeldung zusätzliche Datenzeilen. Um dem Client das Ende einer solchen Datensequenz zu signalisieren, beginnt die letzte Zeile der Antwort des Servers wieder mit derselben Codenummer wie die erste.

Die folgenden Zeilen zeigen den Mitschnitt einer FTP-Sitzung mit dem Host *www.lingoworld.de*. Der Name `www` besagt natürlich, dass es sich eigentlich um einen Webserver handelt. Es ist durchaus üblich, dass Hosting-Provider den Webserver unmittelbar per FTP zugänglich machen, um die eigene Website hochzuladen:

```
> ftp www.lingoworld.de
Verbunden zu www.lingoworld.de.
220 FTP Server ready.
Benutzer (www.lingoworld.de:(none)): XXXXX
```

```
331 Password required for XXXXX.
Kennwort: [Eingabe wird nicht angezeigt]
230 User XXXXX logged in.
Ftp> pwd
257 "/" is current directory.
Ftp> cd extra
250 CWD command successful.
Ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
test.txt
info.txt
226-Transfer complete.
226 Quotas off
21 Bytes empfangen in 0,01 Sekunden (2,10 KB/s)
Ftp> get test.txt
200 PORT command successful.
150 Opening ASCII mode data connection for test.txt (2589 bytes).
226 Transfer complete.
2718 Bytes empfangen in 0,04 Sekunden (67,95 KB/s)
Ftp> quit
221 Goodbye.
```

In dieser Sitzung wird zunächst die Anmeldung durchgeführt (der Username wird angezeigt, allerdings habe ich ihn hier geändert; die Passworteingabe hat kein grafisches Feedback), anschließend werden die folgenden Befehle eingesetzt (die ersten drei entsprechen gleichnamigen Unix-Shell-Befehlen, siehe Kapitel 6, »Betriebssysteme«):

- `pwd`: *print working directory* – aktuelles Arbeitsverzeichnis ausgeben
- `cd`: *change directory* – Verzeichnis auf dem entfernten Server wechseln
- `ls`: *list* – Verzeichnisinhalt anzeigen
- `get`: die angegebene Datei in das aktuelle lokale Verzeichnis herunterladen
- `quit`: die Sitzung und das FTP-Programm beenden

Weitere wichtige Befehle sind folgende:

- `put`: die angegebene Datei in das aktuelle entfernte Verzeichnis hochladen
- `binary`: umschalten in den Binärmodus
- `ascii`: umschalten in den ASCII-Modus
- `help`: eine Liste aller verfügbaren Befehle anzeigen

Es ist wichtig, dass Sie den Unterschied zwischen dem ASCII- und dem Binärmodus verstehen. Das ganze Problem hat damit zu tun, dass die verschiedenen Betriebssysteme sich nicht

auf einen gemeinsamen Standard für Zeilenumbrüche in Textdateien einigen konnten. Wie in Kapitel 17, »Weitere Datei- und Datenformate«, genau erläutert wird, verwendet Unix das ASCII-Zeichen mit dem Code 10 (LF, *Line Feed* oder Zeilenvorschub), klassisches Mac OS setzt das ASCII-Zeichen 13 ein (CR, *Carriage Return* oder Wagenrücklauf), und Windows sowie die meisten Netzwerkanwendungsprotokolle nutzen beide Zeichen hintereinander.

Im ASCII-Modus werden die Zeilenumbrüche innerhalb einer Datei bei der Übertragung jeweils umgewandelt, sodass beispielsweise die auf Ihrem Windows-Rechner gespeicherten Textdateien mit CR/LF auf dem entfernten Unix-Server mit dem für dessen Verhältnisse korrekten (Nur-)LF ankommen und umgekehrt. Sie sollten jedoch begreifen, dass dieses bei Textdateien recht segensreiche Feature bei Binärdateien wie Bildern oder Programmen den sicheren Tod zur Folge hat. Wird jedes Vorkommen des Byte-Werts 10 durch die beiden Bytes 13 und 10 ersetzt oder umgekehrt, werden die Bytes in einer solchen Datei verändert und planlos verschoben! Natürlich ist eine auf diese Weise behandelte Bild-, Audio- oder Programmdatei unbrauchbar.

Die meisten grafisch orientierten FTP-Programme entscheiden je nach Dateityp passend selbst, ob sie ASCII- oder Binärübertragung verwenden sollen. Bei dem Konsolen-FTP-Programm müssen Sie für jede einzelne Datei selbst in den richtigen Modus umschalten. Das ist ein – aber nicht der einzige – Grund dafür, dass die Arbeit mit der Konsolenversion von FTP in der Praxis fast unzumutbar ist.

E-Mail

Die E-Mail, die sich unter dem Dach eines Mailclients wie Thunderbird, Google Mail oder Apple Mail so einheitlich präsentiert, bedarf in Wirklichkeit der Zusammenarbeit mit mindestens zwei verschiedenen Servern. Der eine ist für den Versand von E-Mails zuständig und führt zu diesem Zweck das Protokoll SMTP (*Simple Mail Transport Protocol*) aus. Ein anderer enthält das E-Mail-Postfach, in dem an Sie adressierte Nachrichten ankommen; dieser Dienst wird entweder von dem weitverbreiteten POP3-Protokoll (*Post Office Protocol Version 3*) oder dem komfortableren IMAP (*Internet Message Access Protocol*) versehen.

Wenn Sie eine E-Mail versenden möchten, wird diese an einen SMTP-Server übermittelt, der sich um die Weiterleitung der Nachricht an den Empfänger kümmert. SMTP, definiert in RFC 2821 (Neufassung von RFC 821), ist ähnlich wie FTP ein einfaches textbasiertes Protokoll aus wenigen Befehlen; der zuständige Server wartet am TCP-Port 25 auf Verbindungen.

Einige SMTP-Server von Internetprovidern kontrollieren bis heute nicht die Identität des Absenders. Das Problem dabei ist, dass solche Server dadurch leicht für das Versenden von Spam verwendet werden können oder dass sogar jemand eine falsche Identität vortäuschen kann. Dabei sieht die SMTP-Spezifikation durchaus mehrere mögliche Authentifizierungsverfahren vor:

- Manche SMTP-Server überprüfen die IP-Adresse des Hosts, von dem die Verbindung initiiert wurde – ein ideales Verfahren für normale Internetprovider, die nur ihrer eigenen Kundschaft Zugriff auf ihre SMTP-Server gewähren möchten.
- Eine andere Möglichkeit besteht darin, die Anmeldung am E-Mail-Empfangsserver desselben Providers als Voraussetzung für den E-Mail-Versand zu verlangen. Dieses Verfahren wird *SMTP after POP* genannt. Nachteil: Manche E-Mail-Clients können nicht damit umgehen, sodass man jedes Mal vor dem E-Mail-Versand auf Mail empfangen klicken muss.
- Das sicherste Verfahren wurde SMTP erst nachträglich hinzugefügt (inzwischen ist es aber glücklicherweise flächendeckend verbreitet): die persönliche Anmeldung beim SMTP-Server mit Username und Passwort.

Sie können die Kommunikation mit einem SMTP-Server über das Programm `telnet` abwickeln. Eine solche Sitzung sieht beispielsweise folgendermaßen aus (die konkreten Namens- und Adressdaten habe ich anonymisiert):

```
> telnet smtp.myprovider.de smtp
220 smtp.myprovider.de ESMTP Thu, 13 Apr 2023 12:37:21 +0100
HELO
250 smtp.myprovider.de Hello[203.51.81.17]
MAIL From: absender@myprovider.de
250 <absender@myprovider.de> is syntactically correct
RCPT To: empfaenger@elsewhere.com
250 <empfaenger@elsewhere.com> verified
DATA
354 Enter message, ending with "." on a line by itself
FROM: Sascha <absender@myprovider.de>
To: Jack <empfaenger@elsewhere.com>
Subject: Gruesse

Hallo Jack,
hier ist wieder einmal Post für dich.
Viel Spaß damit!
Gruss, Sascha
.
250 OK id=18QdIY-00048Y-00
QUIT
221 smtp.myprovider.de closing connection.
```

In dieser kurzen Konversation werden die folgenden SMTP-Befehle verwendet:

- `HELO`: Mit diesem Befehl meldet sich der Client beim Server an; eventuell findet in diesem Zusammenhang die bereits beschriebene Überprüfung der Client-IP-Adresse statt. Manche SMTP-Server verlangen auch die Angabe eines Domainnamens hinter dem Befehl.

- `MAIL`: Dieser Befehl leitet die Erzeugung einer neuen Nachricht ein; der Absender muss im Format `From: E-Mail-Adresse` oder `From: Name <E-Mail-Adresse>` angegeben werden.
- `RCPT`: Gibt einen Empfänger im Format `To: E-Mail-Adresse` oder `To: Name <E-Mail-Adresse>` an.
- `DATA`: Alle folgenden Zeilen des Clients werden als Teil der eigentlichen E-Mail-Nachricht aufgefasst, bis eine Zeile folgt, die nur einen Punkt (`.`) enthält.
- `QUIT`: Die Sitzung wird hiermit beendet; alle bis zu diesem Zeitpunkt erzeugten E-Mail-Nachrichten werden versandt.

Die E-Mail-Nachricht selbst (zwischen `DATA` und der Abschlusszeile mit dem Punkt) ist eine klassische Textnachricht, deren Aufbau in RFC 822 (aktualisiert in RFC 2822) beschrieben wird. Prinzipiell besteht sie aus mehreren Header-Zeilen im Format `Feldname: Wert`, gefolgt von einer Leerzeile und dem eigentlichen Text. Der minimale Header enthält den Absender (`From`), den Empfänger (`To`) und einen Betreff (`Subject`). Absender und Empfänger dürfen wie bei den SMTP-Befehlen `MAIL` und `RCPT` diverse Formate besitzen. Weitere häufige Header-Felder sind die Kopienempfänger (`Cc` für *Carbon Copy*) sowie die unsichtbaren Kopienempfänger (`Bcc` für *Blind Carbon Copy*). Die normalen Empfänger der Kopien werden in der Nachricht angezeigt, die unsichtbaren nicht.

Ein alternatives Format für E-Mails, das heutzutage bereits häufiger verwendet wird als RFC 822, ist das MIME-Format. Die verschiedenen Aspekte von MIME werden in RFC 2045 bis 2049 dargelegt. Die Abkürzung MIME steht für *Multipurpose Internet Mail Extensions*. Es handelt sich um ein Format, das für den Versand beliebiger Text- und Binärdaten geeignet ist, sogar von verschiedenen Datentypen innerhalb derselben Nachricht.

Der MIME-Header ist eine Erweiterung des RFC-822-Headers. Die wichtigsten neuen Felder sind `Content-type`, das den Datentyp angibt, und `Content-Transfer-Encoding`, mit dessen Hilfe das Datenformat festgelegt wird. Ersteres beschreibt also den Inhalt der Nachricht, Letzteres die Form, in der sie versandt wird. Der Inhaltstyp (`Content-Type`), meist *MIME-Type* genannt, besteht aus zwei Bestandteilen, die durch einen Slash (/) voneinander getrennt werden: dem Haupttyp und dem genaueren Untertyp. Haupttypen sind beispielsweise `text` (ASCII-Text), `image` (Bilddaten), `audio` (Audiodaten), `video` (Digitalvideo) oder `application` (proprietäres Datenformat eines bestimmten Anwendungsprogramms). Tabelle 5.23 listet einige gängige MIME-Types auf. Die vollständige Liste aller registrierten Typen finden Sie online unter *http://www.iana.org/assignments/media-types/index.html*.

Typ	Beschreibung
`text/plain`	reiner Text ohne Formatierungsbefehle
`text/html`	HTML-Code

Tabelle 5.23 Einige gängige MIME-Datentypen

Typ	Beschreibung
`text/xml`	XML-Code
`image/gif`	Bild vom Dateityp GIF
`image/jpeg`	Bild vom Dateityp JPEG
`image/png`	Bild vom Dateityp PNG
`audio/wav`	Sounddatei vom Typ Microsoft Wave
`audio/aiff`	Sounddatei vom Typ Apple AIFF
`audio/mpeg`	komprimierte Sounddatei vom Typ MP3
`video/avi`	Digitalvideo vom Typ Microsoft Video for Windows
`video/mov`	Digitalvideo vom Typ Apple QuickTime
`video/mpeg`	Digitalvideo vom Typ MPEG
`application/x-shockwave-flash`	komprimierter Adobe-Flash-Film (Dateiendung *.swf*)
`application/x-director`	komprimierter Adobe-Director-Film (Dateiendung *.dcr*)
`application/x-www-form-urlencoded`	POST-Formulardaten bei HTTP-Anfragen an Webserver (Näheres dazu erfahren Sie in Kapitel 13, »Datenbanken«, Kapitel 17, »Weitere Datei- und Datenformate«, und Kapitel 18, »Webseitenerstellung mit HTML und CSS«.)
`multipart/mixed`	»Umschlag« für mehrere MIME-Unterabschnitte
`multipart/alternative`	»Umschlag« für denselben Inhalt in mehreren Alternativformaten
`multipart/form-data`	»Umschlag« für POST-Formulardaten einschließlich Datei-Uploads (siehe Kapitel 18, »Webseitenerstellung mit HTML und CSS«)

Tabelle 5.23 Einige gängige MIME-Datentypen (Forts.)

Die drei letzten Typen in der Tabelle machen MIME besonders interessant: Ein MIME-Dokument vom Typ `multipart/mixed` kann beliebig viele Teile enthalten, die jeweils einen vollständigen MIME-Header besitzen und wiederum beliebige MIME-Types aufweisen können. Mithilfe dieser Technik werden in modernen E-Mail-Programmen Attachments (Dateianhänge) der Mail hinzugefügt. Dagegen wird ein Abschnitt vom Typ `multipart/alternative` eingesetzt, um denselben Inhalt in verschiedenen alternativen Darstellungsformen zu umschließen, beispielsweise ein Bild im GIF- und im PNG-Format oder (wahrscheinlich die häufigste

Anwendung) den Text einer E-Mail-Nachricht einmal im einfachen Text- und einmal im HTML-Format. Abbildung 5.5 zeigt beispielhaft, wie eine MIME-Nachricht mit zwei Dateianhängen aufgebaut sein könnte.

Dasselbe Verfahren – in diesem Fall mit dem MIME-Type `multipart/form-data` – kommt bei Webformularen zum Einsatz, wenn diese neben gewöhnlichen Auswahl- oder Eingabefeldern auch Datei-Uploads unterstützen.

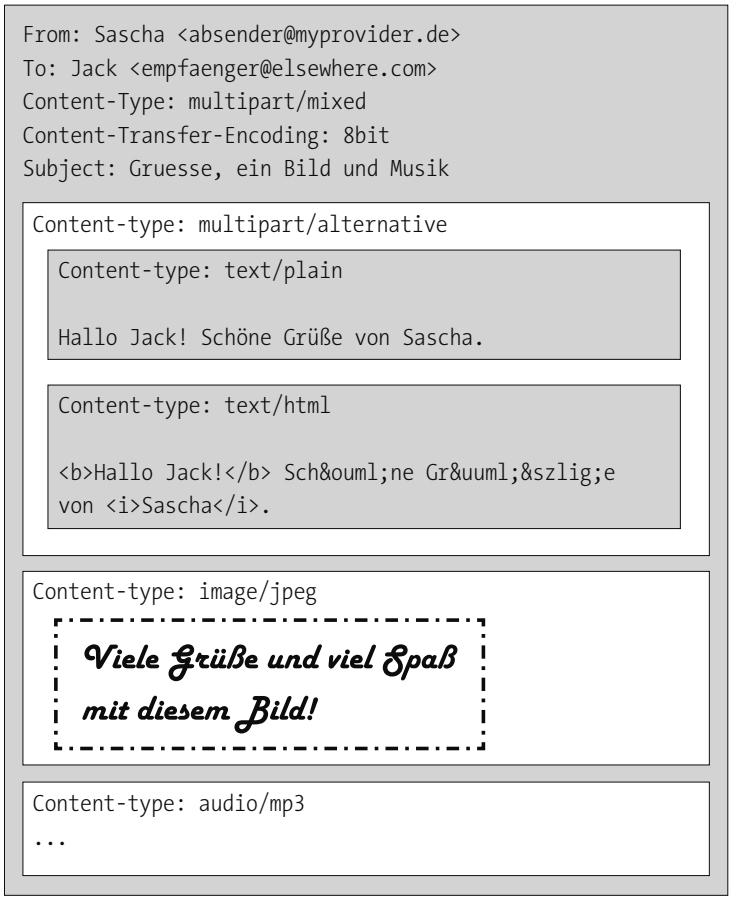


Abbildung 5.5 Beispiel für eine E-Mail im MIME-Multipart-Format

Der `Content-Transfer-Encoding`-Header gibt dem Empfängerclient einen Hinweis darauf, auf welche Weise die ankommenden Daten zu interpretieren sind. Häufig verwendete Werte sind etwa folgende:

- `7bit`: Keine Codierung; eignet sich für 7-Bit-ASCII (englischer Text). Automatischer Zeilenumbruch nach spätestens 1.000 Zeichen.
- `8bit`: Keine Codierung; eignet sich für 8-Bit-Text (internationaler Text). Ebenfalls automatischer Zeilenumbruch nach spätestens 1.000 Zeichen.

- `binary`: Keine Codierung, es erfolgt kein automatischer Zeilenumbruch.
- `quoted-printable`: Spezielle Codierung von Sonderzeichen, die über 7-Bit-ASCII hinausgehen. Beispiel: »größer« wird zu »gr=FC=DFer«. (Die Codierung besteht aus einem Gleichheitszeichen, gefolgt von hexadezimalem Zeichencode.)
- `base64`: Bevorzugte Codierung für Binärdateien. Ein spezieller Algorithmus packt die Daten 7-Bit-kompatibel um. Dieses Format ist auch dann nicht von Menschen lesbar, wenn Klartext codiert wird – aus »Hallo Welt!« wird beispielsweise `SGFsbG8gV2VsdCE=`.

Die Codierungsformen `quoted-printable` und `base64` besitzen den Vorteil, dass die Mailnachricht formal kompatibel mit RFC 822 bleibt und entsprechend auch über alte Mailserver versandt und empfangen werden kann.

Der E-Mail-Empfang über einen POP3-Server erfolgt auf textbasierte Art, ähnlich wie bei SMTP. Der Server kommuniziert über den TCP-Port 110. Die Beschreibung von POP3 steht in RFC 1939. Zur Verdeutlichung hier wiederum eine `telnet`-basierte Konversation mit einem (unkenntlich gemachten) POP3-Server:

```
# telnet pop.myprovider.de pop3
+OK POP3 server ready
USER absender
+OK
PASS XXXXX
+OK
LIST
1898
.
RETR 1
+OK 953 octets
Return-path: <empfanger@elsewhere.com>
Envelope-to: absender@myprovider.de
Delivery-date: Mon, 24 Apr 2023 03:08:24 +0100
Received: from [207.18.31.76] (helo=smtp.elsewhere.com)
 by mxng13.myprovider.de with esmtp (Exim 3.35 #1)
 id 18QeUU-000270-00
 for absender@myprovider.de; Mon, 24 Apr 2023
 03:08:18 +0100
Received: from box (xdsl-202-21-109-17.elsewhere.com
[202.21.109.17])
by smtp.elsewhere.com (Postfix) with SMTP id
CA500866C1
for <absender@myprovider.de>; Mon, 24 Apr 2023
03:08:14 +0100 (MET)
Message-ID: <001901c2aaf2$31ce81e0$0200a8c0@box>
```

```
From: "Jack" <empfaenger@elsewhere.com>
To: "Sascha" <absender@provider.de>
Subject: Gruesse
Date: Mon, 24 Apr 2023 03:14:30 +0100
MIME-Version: 1.0
Content-Type: text/plain;
charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable

Hi!
Wie geht's?
Alles klar?
Ciao.
.
DELE 1
+OK
QUIT
+OK
```

In dieser Sitzung kommen die folgenden POP3-Befehle zum Einsatz:

- `USER`: Angabe des Usernamens für die Anmeldung
- `PASS`: Angabe des Passworts für die Anmeldung
- `LIST`: nummerierte Liste der verfügbaren E-Mails mit der jeweiligen Länge in Bytes
- `RETR`: E-Mail mit der angegebenen Nummer empfangen
- `DELE`: E-Mail mit der angegebenen Nummer vom Server löschen
- `QUIT`: Sitzung beenden

Die meisten E-Mail-Programme führen `RETR` und `DELE` standardmäßig unmittelbar nacheinander durch, die Nachrichten verbleiben also in der Regel nicht auf dem Server. Bei IMAP-Servern ist es dagegen meist anders: Der besondere Vorteil des IMAP-Protokolls besteht darin, dass auf dem Mailserver selbst verschiedene Ordner eingerichtet werden können, um Mails dort zu verwalten. Auf diese Weise erleichtert IMAP die E-Mail-Verwaltung für den mobilen Einsatz. Die aktuelle Version von IMAP ist das in RFC 2060 dargestellte IMAP4. Ein IMAP-Server funktioniert ähnlich wie ein POP3-Server, verwendet allerdings den TCP-Port 142.

Eine weitere beliebte Form der E-Mail-Nutzung sind webbasierte Freemail-Dienste wie Google Mail, GMX oder Hotmail. Dabei handelt es sich um gewöhnliche IMAP-SMTP-Kombinationen, die über eine Website mit persönlicher Anmeldung zugänglich gemacht werden. Das Programm, das mit den E-Mail-Servern kommuniziert, läuft auf dem Webserver und wird per Browser zugänglich gemacht.

Das World Wide Web

Das Web ist heute die dominierende Internetanwendung überhaupt, und zwar in einem solchen Maße, dass viele Menschen das WWW mit dem gesamten Internet gleichsetzen. Wer auf das Web zugreifen möchte, verwendet dazu eine spezielle Clientsoftware, den sogenannten *Webbrowser*. Nach der Eingabe einer Dokumentadresse stellt der Browser eine TCP-Verbindung zu dem gewünschten Webserver her und fordert über das HTTP-Protokoll das gewünschte Dokument an. Das Dokument ist üblicherweise in der Seitenbeschreibungssprache HTML verfasst (unterstützt durch CSS und JavaScript), die der Browser interpretiert und in eine auf bestimmte Art und Weise formatierte Webseite umwandelt.

Eine solche Seite kann außerdem Verweise auf eingebettete Dateien wie Bilder oder Multimedia enthalten, die der Browser auf dieselbe Art anfordert wie das HTML-Dokument selbst und an der passenden Stelle auf der Seite platziert. Ein weiteres wichtiges Element von Webseiten sind die Hyperlinks, anklickbare Verknüpfungen zu anderen Dokumenten. Wenn Sie einen Hyperlink aktivieren, wird die entsprechende Datei angefordert und in den Browser geladen.

Damit das Web funktionieren kann, wirken einige wesentliche Konzepte zusammen:

- Das Anwendungsprotokoll HTTP, über das Dokumente beim Server angefordert und von diesem ausgeliefert werden. Die aktuelle Version des Protokolls, HTTP 1.1, wird in RFC 2616 beschrieben; in Kapitel 14, »Server für Webanwendungen«, erhalten Sie genauere Informationen darüber.
- Ein spezielles Format für Dokumentadressen, das als *Uniform Resource Locator* (URL) bezeichnet wird und dessen Definition sich in RFC 1738 befindet. Die URL wird beispielsweise in die Adresszeile des Browsers eingegeben; sie sieht zum Beispiel so aus: *https://www.rheinwerk-verlag.de/*.
- Die Seitenbeschreibungssprache HTML, in der Hypertext-Dokumente für das WWW geschrieben werden. Neuere HTML-Versionen werden nicht mehr in RFCs definiert; eine genaue Beschreibung von HTML finden Sie in Kapitel 18, »Webseitenerstellung mit HTML und CSS«.

5.7 Übungsaufgaben

Im Folgenden ist jeweils genau eine Antwort richtig.

1. Was ist keine Aufgabe eines Netzwerks?
   - Kommunikation im Team
   - gemeinsame Stromversorgung mehrerer Rechner
   - Austausch von Daten
   - verteilte Anwendungen

2. Welches der folgenden Merkmale gehört nicht zwangsläufig zur paketvermittelten Datenübertragung?
   - Absender- und Empfängeradresse in jedem Paket
   - die Unterteilung der Daten in kleinere Einheiten
   - ein Bestätigungsverfahren, das die Datenauslieferung garantiert
   - die Fähigkeit zur Weiterleitung der Datenpakete über verschiedene Wege
3. Welcher bis heute bedeutende Internetdienst wurde 1972 erfunden?
   - CGI
   - World Wide Web
   - Newsgroups
   - E-Mail
4. Wie werden die Standards des Internets dokumentiert?
   - in Patentschriften
   - in IEEE-Drafts
   - in Diplom- und Doktorarbeiten
   - in öffentlich verfügbaren RFC-Dokumenten
5. Welche Geräte wurden als erste zur Datenfernübertragung verwendet?
   - Funkgeräte
   - Akustikkoppler
   - Telegraphen
   - Modems
6. Was war die entscheidende Neuerung am World Wide Web?
   - die Einführung von Hypertext
   - die wissenschaftliche Internetanwendung
   - die Verwendung eines textbasierten Kommunikationsprotokolls
   - die Anwendung von Hypertext über ein Netzwerk
7. Wie heißt das Protokoll, das für die WWW-Kommunikation verwendet wird?
   - LWP
   - WWWP
   - HTTP
   - HTML
8. Welche OSI-Schicht ist Nummer 3?
   - Bit-Übertragungsschicht
   - Vermittlungsschicht

   - Sicherungsschicht
   - Transportschicht
9. Welche Nummer hat die OSI-Darstellungsschicht?
   - 6
   - 5
   - 7
   - 4
10. Was ist eine Aufgabe der OSI-Sicherungsschicht?
   - Datenstromverschlüsselung
   - Erzeugung von Datenpaketen
   - Steuerung des Zugriffs auf das Übertragungsmedium
   - Routing
11. Welche ist keine Schicht im Internetschichtenmodell (DoD- oder DDN-Modell)?
   - Netzzugangsschicht
   - Internetschicht
   - Sitzungsschicht
   - Anwendungsschicht
12. Welcher OSI-Schicht entspricht die Internetschicht des Internetschichtenmodells in etwa?
   - Sicherungsschicht
   - Vermittlungsschicht
   - Transportschicht
   - Sitzungsschicht
13. Welches der folgenden Protokolle arbeitet auf der Transportschicht des Internetschichtenmodells?
   - FTP
   - TCP
   - IP
   - ARP
14. Welche Netzwerkart hat die größte Reichweite?
   - MAN
   - WAN
   - GAN
   - LAN

15. Bei welcher Netzwerktopologie sind alle Stationen mit einem zentralen Verteiler verbunden?
    - Baum
    - Stern
    - Ring
    - Bus
16. Welche der folgenden Aussagen über Server ist zutreffend?
    - Ein Server ist ein spezieller, sehr teurer Computer.
    - Ein Server ist ein Programm, das eine bestimmte Dienstleistung bereitstellt.
    - Ein Server muss stets in einen 19-Zoll-Schrank montiert werden.
    - Ein Server ist ein Programm, das eine Benutzeroberfläche für Netzwerkdienste bereitstellt.
17. Welche der folgenden Aussagen über Clients ist zutreffend?
    - Ein Client ist ein Programm, das eine bestimmte Dienstleistung bereitstellt.
    - Client ist lediglich eine andere Bezeichnung für einen Desktop-PC.
    - Ein Client ist ein Programm, das eine Benutzeroberfläche für Netzwerkdienste bereitstellt.
    - Client ist lediglich eine andere Bezeichnung für einen Browser.
18. Wie nennt man es, wenn die Aufgaben eines Webservers auf mehrere physikalische Rechner verteilt werden?
    - Web Caching
    - Proxy Service
    - Load Balancing
    - Round Robin
19. Welcher der folgenden Server ist kein klassischer Netzwerkserver?
    - Mailserver
    - X-Window-Server
    - Fileserver
    - Printserver
20. Welcher IEEE-802-Standard beschreibt drahtlose Netzwerke?
    - IEEE 802.3
    - IEEE 802.5
    - IEEE 802.11
    - IEEE 802.15

21. Wodurch wird eine bestimmte Ethernet-Karte eindeutig gekennzeichnet?
    - durch die MAC-Adresse
    - durch die IP-Adresse
    - durch den Domainnamen
    - Die Ethernet-Karte selbst besitzt kein eindeutiges Identifikationsmerkmal.
22. Was bedeutet der *Promiscuous Mode* bei einer Ethernet-Karte?
    - Sie erhält mehrere IP-Adressen.
    - Sie dient als DHCP-Provider.
    - Sie empfängt alle Datenpakete aus ihrem Netzsegment.
    - Sie wird IP-Multicast-fähig.
23. Was geschieht beim CSMA/CD-Verfahren, wenn eine Datenkollision auftritt?
    - Aufgrund der Funktionsweise kann es in CSMA/CD-Netzen nicht zu Kollisionen kommen.
    - Der Verteiler (Hub oder Switch) regelt, welches Gerät als Erstes erneut senden darf.
    - Der Verteiler (Hub oder Switch) versendet nacheinander Kopien der kollidierten Pakete.
    - Jedes an der Kollision beteiligte Gerät sendet nach einer individuellen Zufallswartezeit erneut.
24. Wie setzt sich die Bezeichnung 10 Base 2 zusammen?
    - 10 mm dickes Kabel mit maximal 2 MBit/s
    - maximal 10 MBit/s, maximal 200 m langes Netzsegment
    - maximal 10 MBit/s, mindestens 2 m Abstand zwischen zwei Stationen
    - höchstens zehn Stationen pro maximal 200 m langem Netzsegment
25. Welcher Steckertyp wird für Twisted-Pair-Ethernet, aber auch etwa für ISDN verwendet?
    - RG-58
    - RJ-45
    - RJ-11
    - Cinch
26. Welche UTP-Kabel-Kategorie wird mindestens für Fast Ethernet benötigt?
    - 3
    - 4
    - 5
    - 6

27. Was ist der entscheidende Unterschied zwischen einem Hub und einem Switch?
    - Für Fast Ethernet ist auf jeden Fall ein Switch notwendig.
    - Nur ein Switch kann Ethernet-Schnittstellen unterschiedlicher Geschwindigkeiten miteinander verbinden.
    - Ein Switch besitzt mehr Ports als ein Hub.
    - Ein Switch verhindert Datenkollisionen durch direkte Verbindungen zwischen den Stationen.
28. Welche maximale Datenübertragungsrate unterstützt Fast Ethernet?
    - 10 MBit/s
    - 52 MBit/s
    - 100 MBit/s
    - 480 MBit/s
29. Zu welchem Zweck wurde das im WLAN-Bereich übliche Frequency-Hopping-Verfahren ursprünglich entwickelt?
    - Mobiltelefonie
    - Satellitenkommunikation
    - Flugnavigation
    - Torpedofernsteuerung
30. Welches Netzzugangsverfahren wird für WLAN verwendet?
    - CSMA/CD
    - Token Passing
    - CSMA/CA
    - PPP
31. Welchen Sicherheitsstandard für WLAN sollte man mindestens verwenden?
    - WEP
    - WPA2
    - WPA1
    - SSL
32. Welche technische Besonderheit sorgt für die hohen DSL-Übertragungsraten über normale Telefonleitungen?
    - Datenkomprimierung
    - Kanalbündelung
    - hochfrequente Signale
    - DSL verwendet nie normale Telefonleitungen, sondern Lichtwellenleiter.

33. Worin besteht der entscheidende Unterschied zwischen ADSL und SDSL?
    - SDSL bietet ein anderes Abrechnungsmodell, das für geschäftliche Anschlüsse interessanter ist.
    - SDSL ist schneller als ADSL.
    - ADSL kann gewöhnliche Telefonleitungen verwenden, SDSL nicht.
    - SDSL bietet identische Datenübertragungsraten in beide Richtungen, ADSL ist im Upload langsamer als im Download.
34. Welche der folgenden Angaben ist keine gültige IPv4-Adresse?
    - 197.17.8.21
    - 212.211.210.209
    - 31.310.30.13
    - 0.7.8.9
35. Mit welchem Bit-Muster beginnt eine IP-Adresse der historischen Klasse C?
    - 0
    - 10
    - 110
    - 1110
36. In welchem Bereich liegt das erste Byte einer IP-Adresse der Klasse B?
    - 64 bis 127
    - 128 bis 191
    - 192 bis 223
    - 128 bis 159
37. Welche der folgenden IP-Adressen ist keine private, frei verfügbare Adresse gemäß RFC 1918?
    - 192.168.27.11
    - 172.21.47.11
    - 10.0.8.15
    - 172.47.11.12
38. Zu welchem der folgenden Netze gehört die IP-Adresse 196.17.8.92 nicht?
    - 196.17.0.0/16
    - 196.17.0.0/17
    - 196.17.8.0/26
    - 196.17.8.0/25

39. Das Netz 156.19.0.0/16 wird per CIDR in vier gleich große Netze unterteilt. Welches der folgenden ist eines der neuen Teilnetze?
    - 156.19.0.0/20
    - 156.19.16.0/18
    - 156.19.64.0/18
    - 156.19.128.0/19
40. Wie viele Hosts kann jedes Teilnetz maximal enthalten, wenn ein IPv4-Netz mit der Teilnetzmaske 255.255.255.0 in vier Teile unterteilt wird?
    - 62
    - 128
    - 127
    - 64
41. Was ist der Vorteil von VLSM gegenüber CIDR?
    - Die Teilnetzmaske kann auch zwischen ganzen Bytes der Adressen verlaufen.
    - Mehrere Netze lassen sich zu einem größeren Netz zusammenfassen (Supernetting).
    - Ein Netz lässt sich in Teilnetze unterschiedlicher Größe unterteilen.
    - Es gibt keinen Vorteil; CIDR ist flexibler als VLSM.
42. Wie lang ist der IPv4-Header mindestens?
    - 16 Byte
    - 20 Byte
    - 24 Byte
    - 32 Byte
43. Welche Bedeutung besitzt der TTL-Wert (*Time To Live*) im IP-Datagramm?
    - Er gibt die Uhrzeit an, zu der das Datagramm erzeugt wurde.
    - Er gibt die Uhrzeit an, bis zu der das Datagramm bestehen wird.
    - Er zählt die Sekunden, die das Datagramm bereits existiert.
    - Er zählt die Hops, die das Datagramm erlebt, bis 0 herunter.
44. Zu welchem Problem können unterschiedliche MTUs verschiedener Netzwerkschnittstellen bei IP-Datagrammen führen?
    - Die Datagramme können nicht weitergeleitet werden.
    - Die Datagramme werden fragmentiert.
    - Die Datagramme müssen erneut gesendet werden.
    - Die Datagramme werden langsamer transportiert.

45. Was ist die genaue Definition eines Default-Gateways?
    - der Router ins nächstgelegene Netz
    - der Router ins Internet
    - der Router für Verbindungen in alle Netze, für die kein separater Router existiert
    - der Router in einem Netz, in dem kein weiterer Router existiert
46. Welche Adresse ist vom Netzwerk 156.81.0.0/19 aus nur über einen Router zu erreichen?
    - 156.81.9.18
    - 156.81.18.9
    - 156.81.81.9
    - 156.81.0.9
47. Das Netzwerk 152.17.0.0/17 ist über drei Router mit anderen Netzen verbunden: r1 für das Netzwerk 152.17.128.0/17, r2 für 152.18.0.0/16 und r3 für alle anderen Netze. Welcher Router wird für eine Verbindung zur Adresse 152.18.210.22 verwendet?
    - r1
    - r2
    - r3
    - Keiner; die Adresse befindet sich im aktuellen Teilnetz.
48. Was wird im Zusammenhang mit IP-Routing als autonomes System bezeichnet?
    - ein Rechner, der nicht an ein Netzwerk angeschlossen ist
    - ein lokales Netzwerk ohne Verbindung zu anderen Teilnetzen
    - die Gesamtheit aller Netzwerke eines Betreibers
    - ein allein stehender Router
49. Zu welchem Zweck wird ein internes Routing-Protokoll eingesetzt?
    - für das Routing innerhalb eines Teilnetzes
    - für das Routing innerhalb eines autonomen Systems
    - für das Routing innerhalb eines Netzes ohne Außenverbindung
    - für das Routing zwischen mehreren virtuellen Netzwerkschnittstellen auf einem einzelnen Rechner
50. Welches der folgenden Routing-Protokolle ist ein externes Routing-Protokoll?
    - RIP
    - OSPF
    - BGP
    - keins der genannten

51. Welcher TCP/IP-Dienst ermöglicht die automatische Vergabe von IP-Adressen?
    - BOOTP
    - NAT
    - DHCP
    - ARP
52. Wie lang ist eine IPv6-Adresse?
    - 64 Bit
    - 128 Bit
    - 32 Bit
    - variabel
53. Welches Transportprotokoll verwendet das Verbindungstestprogramm ping?
    - UDP
    - ICMP
    - TCP
    - keins, stattdessen RawIP
54. Welche Flags sind (nacheinander) bei den drei Datenpaketen gesetzt, die den Drei-Wege-Handshake bei TCP bilden?
    - SYN, ACK, SYN
    - SYN, SYN, ACK
    - SYN, SYN/ACK, ACK
    - SYN, ACK, SYN/ACK
55. Welche TCP-Portnummern sind Well-known Ports – festgelegte Portnummern für Serverdienste?
    - 0 bis 255
    - 0 bis 1023
    - variiert je nach Betriebssystem
    - 0 bis 32767
56. Was ist der Vorteil von UDP gegenüber TCP als Transportprotokoll?
    - schnellere Übertragung auf Kosten der Zuverlässigkeit
    - schnellere Übertragung durch höhere Übertragungsraten
    - sichere Übertragung durch Verschlüsselung
    - Möglichkeit der Übertragung in Nicht-IP-Netze

57. Welchen TCP-Port verwendet ein Webserver standardmäßig?
    - 21
    - 53
    - 80
    - 110
58. Welcher Serverdienst verwendet standardmäßig den TCP-Port 23?
    - echo
    - ftp
    - smtp
    - telnet
59. Aus welcher Datei können auf einem Unix-System Adressauflösungen von Hostnamen gelesen werden?
    - /var/hosts.txt
    - /etc/services.txt
    - /etc/hosts
    - /var/addresses
60. Welche der folgenden Aussagen über DNS ist zutreffend?
    - Das System besteht aus einer Textdatei, die bei Änderungen auf jeden Nameserver kopiert wird.
    - Ein DNS-Administrator kann Subdomains seiner Zone entweder selbst verwalten oder als untergeordnete Zonen delegieren.
    - Ein Webserver muss den Hostnamen *www* erhalten, damit er funktioniert.
    - Jeder Internetteilnehmer benötigt einen eigenen Nameserver.
61. Welche der folgenden Generic Top-Level-Domains gibt es nicht?
    - .com
    - .net
    - .doc
    - .info
62. Welche Länder-Top-Level-Domain verwendet Großbritannien?
    - .bn
    - .uk
    - mehrere: .en für England, .sc für Schottland, .wa für Wales und .ni für Nordirland
    - .bi für die gesamten Britischen Inseln

63. Welchen Vorteil besitzt SSH gegenüber Telnet?
    - sichere Datenübertragung durch Verschlüsselung
    - SSH verwendet TCP, Telnet dagegen UDP.
    - Telnet-Server funktionieren nur unter Unix, SSH-Server auch unter Windows.
    - Die Texteingabe ist bei SSH komfortabler.
64. Welchen der folgenden FTP-Befehle gibt es nicht?
    - `binary`
    - `get`
    - `decimal`
    - `put`
65. Für welche Dateien braucht bei der FTP-Übertragung nicht der Binärmodus verwendet zu werden?
    - Bilddateien
    - ausführbare Programme
    - HTML-Dokumente
    - Videodateien
66. Wie wird einem SMTP-Server mitgeteilt, dass der eigentliche E-Mail-Text beendet ist?
    - durch eine Leerzeile
    - durch den Befehl `END`
    - durch ein EOF (entspricht der Tastenkombination Strg + D)
    - durch einen einzelnen Punkt in einer Zeile
67. Welcher MIME-Type wird für ein JPEG-Bild verwendet?
    - application/x-jpeg
    - image/jpeg
    - image/jpg
    - application/image-jpg
68. Welcher RFC-822-Header gibt den MIME-Type an?
    - Content-type
    - Content-transfer-encoding
    - Mime-type
    - File-type

69. In welcher E-Mail-Codierung wird »Köln« als »K=FCln« wiedergegeben etc.?
    - 7bit
    - quoted-printable
    - base64
    - binary
70. Was ist der Vorteil von IMAP gegenüber POP3?
    - IMAP-Server bieten mehr Speicherplatz pro Postfach.
    - Auf einem IMAP-Server kann eine Person mehrere E-Mail-Adressen haben.
    - Auf IMAP-Servern werden die Daten länger aufbewahrt.
    - Auf einem IMAP-Server können Ordner zur E-Mail-Verwaltung angelegt werden.

Kapitel 7
Grundlagen der Programmierung

There are 10 kinds of people: Those who understand binary notation and those who do not.
– Anonym

Ein Computer ist immer nur so nützlich wie die verfügbare Software. Heutzutage gibt es vorgefertigte Programme für beinahe jeden Verwendungszweck; viele von ihnen lernen Sie in verschiedenen Kapiteln dieses Buchs kennen. Dennoch gibt es – neben der Tatsache, dass Ihre Ausbildung es vielleicht verlangt – eine Reihe guter Gründe, Software selbst zu entwickeln:

- Trotz der immensen Fülle an Standardsoftware ist manchmal nicht genau das passende Programm verfügbar.
- Manche Software ist so teuer, dass sie das Budget von Privatpersonen oder auch kleineren Unternehmen bei Weitem übersteigt.
- Open-Source-Software löst das Preisproblem in vielen Fällen, aber gerade diese lässt sich mithilfe eingebauter Programmierschnittstellen oft noch besser an die eigenen Bedürfnisse anpassen.
- Es gibt kaum eine verlässlichere Möglichkeit, die Funktionsweise eines Computers zu verstehen, als ihn zu programmieren.

Um Software entwickeln zu können, benötigen Sie Programmierkenntnisse. Je nach Art und Einsatzgebiet von Programmen sind die verschiedenen Programmiersprachen unterschiedlich gut geeignet. Beispielsweise sind einige Sprachen besonders auf Geschwindigkeit optimiert, andere dagegen sind benutzerfreundlicher und leichter zu erlernen, wieder andere sind lediglich für spezielle Arten von Programmen geeignet oder funktionieren nur innerhalb eines bestimmten Anwendungsprogramms. In diesem Buch beschäftigen sich mehrere Kapitel oder Teile von ihnen mit verschiedenen Aspekten der Programmierung:

- In Kapitel 1, »Einführung«, finden Sie einen kurzen Abriss zur Geschichte der Programmiersprachen.
- In Kapitel 3, »Elektronische und technische Grundlagen«, wird ein virtueller Prozessor erläutert, der über eine einfache Maschinensprache (genauer gesagt, eine Assembler-Sprache) mit wenigen Instruktionen verfügt.

- Kapitel 6, »Betriebssysteme«, ermöglicht einen Einstieg in die Programmierung mit der Windows PowerShell und enthält einen kurzen Abschnitt über Linux/Unix-Shell-Skripte.
- Das vorliegende Kapitel bietet einen allgemeinen Einstieg in die Programmierung. Anhand zweier verschiedener gängiger Sprachen werden die wichtigsten Komponenten von Computerprogrammen vorgestellt.
- Kapitel 8, »Algorithmen und Datenstrukturen«, erläutert gängige Algorithmen, also Problemlösungsstrategien, die beispielsweise dem Suchen nach und dem Sortieren von Daten dienen.
- In Kapitel 9, »Weitere Konzepte der Programmierung«, werden zahlreiche spezielle Programmiertechniken behandelt. Unter anderem werden Sie in die Mustererkennung mit regulären Ausdrücken eingeführt und lernen einige Grundlagen der System- und Netzwerkprogrammierung kennen.
- Kapitel 10, »Datenanalyse, Machine Learning, künstliche Intelligenz«, stellt die Grundlagen diverser Algorithmen zur automatisierten Auswertung von Datenreihen vor.
- Kapitel 11, »Software-Engineering«, geht den wichtigen Schritt von der einzelnen Programmdatei zum Softwareprojekt. Sie lernen verschiedene nützliche Techniken kennen, um größere Programme zu planen und den Überblick in ihnen zu behalten.
- In Kapitel 13, »Datenbanken«, wird auf die Verwendung von Schnittstellen zur Programmierung datenbankgestützter Anwendungen eingegangen.
- In Kapitel 16, »XML«, erhalten Sie einen kurzen Überblick über wichtige Schnittstellen zur XML-Programmierung.
- In Kapitel 19, »Webserveranwendungen«, wird die Programmiersprache PHP vorgestellt.
- Kapitel 20, »JavaScript und Ajax«, führt in die Skriptsprache JavaScript ein, mit der sich Webseiten im Browser »zum Leben erwecken« lassen. Sie lernen die Grundlagen der Sprache, das dynamische Nachladen von Inhalten mit Ajax, das JavaScript-Anwendungs-Framework React.js sowie die serverseitige JavaScript-Engine Node.js kennen.

Das vorliegende Kapitel dient als grundlegendes Tutorial für fortgeschrittene Computeranwender*innen, die bisher noch nicht programmiert haben. Es ist ebenfalls nützlich, wenn Sie grundsätzlich Programmierkenntnisse haben, aber eine oder mehrere der vorgestellten Sprachen noch nicht kennen. Im Einzelnen lernen Sie in diesem Kapitel eine Compiler- und eine Skriptsprache kennen. Eine der Sprachen ist überwiegend objektorientiert, die andere ist eine Multiparadigmensprache, das heißt, sie bietet Aspekte verschiedener Programmiersprachenarten (siehe Abschnitt 1.2.3, »Die Entwicklung der Programmiersprachen«, in Kapitel 1, »Einführung«). Im Einzelnen handelt es sich um folgende Sprachen:

- Python – Multiparadigmen-Skriptsprache
- Java – objektorientierte Compilersprache

7.1 Python

Es gibt unterschiedliche Meinungen darüber, welche Programmiersprache am besten geeignet ist, um das Programmieren zu erlernen. Bis zur 9. Auflage dieses Buchs machte die über 50 Jahre alte, aber noch immer weitverbreitete Sprache C den Anfang – unter anderem weil ihre grundlegende Syntax von vielen verschiedenen Sprachen übernommen wurde (Aspekte davon werden Sie bei den weiterhin in diesem Buch behandelten Sprachen Java, PHP und JavaScript kennenlernen). Seit der vorigen Auflage habe ich mich nicht nur aus Platzgründen dafür entschieden, stattdessen mit Python zu beginnen, weil sich die knappe, präzise Syntax aufs Wesentliche konzentriert.

Zudem ist Python eine sogenannte *Multiparadigmensprache* – also keine rein imperative, objektorientierte oder funktionale Sprache. Stattdessen bietet Python viele Aspekte dieser und anderer Programmiersprachen, sodass Sie die Chance haben, sie alle kennenzulernen.

Bei Python handelt es sich um eine *interpretierte Sprache*, das heißt, Programme werden im Quellcode ausgeliefert und zur Laufzeit von einem speziellen Programm, dem *Python-Interpreter*, übersetzt. Hinter den Kulissen findet, wie bei den meisten modernen Skriptsprachen, eine sogenannte *Just-in-Time-Kompilierung* statt – der Code wird keineswegs Zeile für Zeile während der Ausführung übersetzt, sondern in der Regel viel schneller, und die übersetzte Form wird im Arbeitsspeicher oder sogar auf einem Datenträger zwischengespeichert.

Die erste Version der Sprache wurde ab 1989 von dem niederländischen Programmierer *Guido van Rossum* entwickelt; die fertige Version 1.0 erschien 1994. Der Name bezieht sich nicht auf das gleichnamige Tier, sondern auf die britische Comedy-Gruppe *Monty Python*, von der van Rossum ein großer Fan ist. In der offiziellen Python-Dokumentation wimmelt es daher von Anspielungen auf Monty-Python-Filme und -Sketche (der vorliegende Abschnitt hält an dieser Tradition fest). Inzwischen arbeiten viele Menschen an der Weiterentwicklung von Python, aber bis 2018 hatte Guido van Rossum als *Benevolent Dictator for Life* (BDFL) das letzte Wort. Nachdem er 2019 vorübergehend seinen weitgehenden Ruhestand geplant hatte, begann er im November 2020, bei Microsoft zu arbeiten, wo er sich um eine engere Integration von Python in die Windows-Betriebssysteme kümmert. Außerdem wird Microsofts Cloud-Computing-Service Azure verbreitet für – oft Python-basierte – KI-Anwendungen eingesetzt.

Im Jahr 2000 wurde Python 2.0 vorgestellt; die wichtigsten neuen Features waren Unicode-Unterstützung und eine voll ausgestattete Garbage Collection (die automatische Bereinigung des Speichers von nicht mehr benötigten Daten). Python 3.0, veröffentlicht im Jahr 2008, ist eine ausführliche Neuimplementierung der Sprache und nicht zu Version 2 abwärtskompatibel. Allerdings wurden zahlreiche Features seitdem für die Verwendung mit 2.x rückportiert. Da es noch immer diverse weitverbreitete Bibliotheken gibt, die nicht mit Python 3 kompatibel sind, und da Python in vielen Softwareprojekten eine wichtige Rolle als

eingebettete Skriptsprache spielt, wurde auch Python 2 bis 2020 mit Sicherheitsupdates und Bugfixes weitergepflegt. Dieser Abschnitt behandelt jedoch ausschließlich Python 3 in dem aktuellen Release 3.11.

Auf den meisten Linux-Distributionen ist Python bereits vorinstalliert oder kann mithilfe des Paketmanagers nachinstalliert werden. Manchmal verbirgt sich hinter dem Kommando `python` immer noch Version 2.x, während Python 3 als `python3` aufgerufen wird. macOS enthält ebenfalls Python 2 ab Werk, während Python 3 nachinstalliert werden kann. Installer mit ausführlicher Anleitung für macOS, Windows und andere Betriebssysteme finden Sie auf der Python-Website *www.python.org*.

Besonders für Machine-Learning-Algorithmen ist es nützlich, statt der offiziellen Python-Distribution die *Anaconda*-Distribution zu installieren, da sie bereits ab Werk die wichtigsten Module für diesen Aufgabenbereich enthält und mit nützlichen Tools ausgestattet ist. Eine Anaconda-Standardinstallation enthält nicht das allerneueste Python-Release, sondern zurzeit zum Beispiel 3.9, was aber für die meisten Zwecke genügt. Downloads und Installationsanleitungen finden Sie unter *www.anaconda.com/products/individual*; Näheres zu spezifischen Anaconda-Tools erfahren Sie in Kapitel 10, »Datenanalyse, Machine Learning, künstliche Intelligenz«.

Im Folgenden wird der Interpreteraufruf stets als `python3` geschrieben, um klarzustellen, dass Sie diese Version benötigen, um alle Beispiele ausprobieren zu können. Um ein Python-Skript auszuführen, geben Sie dieses ein:

```
$ python3 Dateiname
```

Die Dateiendung für Python-Skripte ist üblicherweise *.py*.

Wenn Sie den Interpreter ohne Angabe eines Dateinamens starten, erhalten Sie eine interaktive Shell, hier zum Beispiel die Ausgabe der Anaconda-Distribution unter macOS:

```
$ python3
Python 3.9.7 (default, Sep 16 2021, 08:50:36)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Hier können Sie Python-Ausdrücke und Kommandos eingeben, die sofort ausgewertet werden, was eine sehr praktische Methode ist, Python auszuprobieren. In den folgenden Unterabschnitten wird die Python-Shell oft zum Einsatz kommen; die entsprechenden Beispiele werden durch ihren Prompt gekennzeichnet: `>>>`.

Sie können die Shell unter anderem als einfachen Taschenrechner verwenden:

```
>>> 3+4
7
```

Wenn Sie die Shell wieder beenden möchten, geben Sie

```
>>> exit()
```

ein. Auf Unix-Systemen genügt es auch, Strg + D (End of File) zu drücken.

In der Anaconda-Distribution steht Ihnen `ipython` (*interactive Python*) als alternative Shell zur Verfügung. Sie hat gegenüber der Standard-Shell einige Vorteile – beispielsweise können Sie in der History auch mehrzeilige Eingaben zurückholen und editieren. In `ipython` sieht die Eingabeaufforderung so aus:

```
In [1]:
```

Entsprechend wird die zugehörige Ausgabe – wenn es sich um einen Ausdruck und nicht um einen expliziten Ausgabebefehl handelt – so gekennzeichnet:

```
Out [1]:
```

Die einzelnen Ein- und Ausgabeblöcke werden mit 1 beginnend durchnummeriert.

7.1.1 Das erste Beispiel

Bevor die einzelnen Bestandteile der Programmiersprache Python genauer erläutert werden, sehen Sie hier zunächst ein einfaches Beispielprogramm. Dabei handelt es sich um eine erweiterte Fassung des klassischen »Hello World«-Beispiels. Es ist Tradition, das Erlernen einer Programmiersprache mit einem Programm zu beginnen, das diese Begrüßung ausgibt. Unter *http://helloworldcollection.de* finden Sie eine Website mit »Hello World«-Programmen in gut 600 Programmiersprachen.[1]

Sie werden zuerst mit »Hallo Welt!« begrüßt, können anschließend Ihren Namen eingeben und erhalten daraufhin noch einen persönlicheren Gruß.

Geben Sie in einem Editor Ihrer Wahl das folgende Skript ein und speichern Sie es unter *hello.py*:

```
print("Hallo Welt!")
name = input("Ihr Name bitte: ")
print("Hallo " + name + "!")
```

Listing 7.1 hello.py ist das erste Python-Beispiel – eine erweiterte Fassung des berühmten »Hello World«-Programms.

1 Noch beeindruckender ist die Website *http://99-bottles-of-beer.net*, die zurzeit 1.500 verschiedene Implementierungen zur Ausgabe des Sauflieds »99 Bottles of Beer« enthält.

Anschließend können Sie es wie folgt ausführen:

```
$ python3 hello.py
Hallo Welt!
Ihr Name bitte: Sascha
Hallo Sascha!
```

Im Einzelnen bedeuten die drei Zeilen Folgendes:

- `print("Hallo Welt!")`

  Die Funktion `print()` gibt einen String aus, gefolgt von einem Zeilenumbruch. Die Funktion hat zusätzliche Parameter, die beispielsweise die Ersetzung des Zeilenumbruchs durch ein anderes Trennzeichen erlauben, wobei auch ein leerer String möglich ist, um mehrere Strings ohne Unterbrechung auszugeben.

- `name = input("Ihr Name bitte: ")`

  Diese Anweisung definiert eine Variable mit der Bezeichnung `name`. Der Funktionsaufruf `input()` ermöglicht die Eingabe einer Textzeile zur Laufzeit. Durch die Wertzuweisung wird die Eingabe, ohne den abschließenden Zeilenumbruch, in der Variablen `name` gespeichert.

- `print("Hallo " + name + "!")`

  Auch dieser Aufruf von `print()` dient der Ausgabe; der einzige Unterschied zur ersten Zeile des Skripts besteht darin, dass die Ausgabe hier mithilfe von `+` aus zwei festen Textbestandteilen und dem Wert einer Variablen zusammengesetzt wird.

7.1.2 Grundelemente von Python

Dieser Unterabschnitt betrachtet die wesentlichen Bestandteile von Python. In späteren Unterabschnitten werden die Objektorientierung, Aspekte der funktionalen Programmierung sowie ausgewählte Elemente der Python-Standardbibliothek behandelt.

Besonderheiten der Syntax

In vielen Programmiersprachen enden Anweisungen mit einem Semikolon, in Python dagegen einfach mit einem Zeilenumbruch:

```
>>> print("Hallo")
Hallo
>>> print("Welt")
Welt
```

Allerdings können Sie das Semikolon optional verwenden, um mehrere Anweisungen in dieselbe Zeile zu schreiben:

```
>>> print("Hallo"); print("Welt")
Hallo
Welt
```

Dies ist jedoch nur in Ausnahmefällen empfehlenswert, da es nicht zu einer besseren Lesbarkeit der Programme beiträgt.

Umgekehrt können Sie Anweisungen über mehrere Zeilen verteilen, wenn Sie vor dem jeweiligen Zeilenumbruch einen Backslash setzen:

```
>>> 2 * \
... 3 \
... * 4
24
```

Auch das ist nur selten eine empfehlenswerte Notlösung. Besser ist es gegebenenfalls, einen Ausdruck in Klammern zu setzen, denn auch dieser darf sich dann über mehrere Zeilen erstrecken:

```
>>> (2 +
... 3 +
... 4)
9
```

Die Einrückung macht den Unterschied

Die wichtigste Besonderheit von Python ist, dass *allein die Einrückung* darüber bestimmt, in welchen Zusammenhang eine Anweisung gehört. Dies wurde von einer ebenfalls in den Niederlanden entwickelten Sprache namens ABC übernommen, an der Guido van Rossum mitarbeitete, bevor er Python entwickelte.

Betrachten Sie dazu das folgende Beispiel:

```
print("Allgemeine Anweisung") # Wird immer ausgeführt.
if a > 0:
    print("a ist positiv.")   # Nur, wenn a > 0.
    print("Noch eine Zeile.") # Nur, wenn a > 0.
print("Ende.")               # Wird immer ausgeführt.
```

Dieses Konzept ist gewöhnungsbedürftig und zwingt zu äußerster Disziplin beim Einrücken, aber es folgt dem allgemeinen Konzept von Python, möglichst sparsam und pragmatisch zu sein. Außerdem schadet Disziplin beim Einrücken ohnehin nicht, denn auch in Sprachen, in denen die Einrückung keine syntaktische Bedeutung hat, macht sie Programme lesbarer.

Gängige Python-Konvention ist das Einrücken um vier Leerzeichen pro Ebene. Es ist empfehlenswert, sich daran zu halten – auch wenn andere Varianten zulässig sind, solange sie konsistent verwendet werden.

Bestandteile von Python-Programmen

Computerprogramme bestehen im Wesentlichen aus *Anweisungen* (englisch: *Instructions*), die bestimmen, was in welcher Reihenfolge geschehen soll. Diese Anweisungen lassen sich in verschiedene Gruppen unterteilen:

- *Funktionsaufrufe* bestehen aus dem Namen der gewünschten Funktion und Argumenten, also Objekten oder Werten, mit denen die Funktion arbeiten soll. Manche Funktionen erledigen eine Aufgabe – `print()` kümmert sich beispielsweise um die Ausgabe der angegebenen Werte –, während andere mithilfe irgendeines Verfahrens einen Wert ermitteln und zurückgeben, der seinerseits entweder ausgegeben oder in komplexeren Ausdrücken weiterverwendet werden kann – die Funktion `max()` liefert etwa den größten Wert der übergebenen Liste von Werten zurück. Es gibt einen später in diesem Kapitel genauer erläuterten technischen Unterschied zwischen allein stehenden Funktionen und Methoden; Letztere sind Funktionen, die in bestimmte Objekte eingebaut sind.
- *Wertzuweisungen* bestehen aus drei Komponenten:
  - einer Variablen (nichts anderes als ein benannter Speicherplatz), der ein Wert zugewiesen werden soll,
  - dem Gleichheitszeichen `=` als Wertzuweisungsoperator und
  - dem Wert, der in der Variablen gespeichert werden soll.

  Da die Variable, in der der Wert abgelegt werden soll, auf der linken Seite der Wertzuweisung steht, wird sie auch allgemeiner als *LVALUE* (kurz für *Left Value*) bezeichnet, denn wie noch zu klären sein wird, kann es sich auch um andere Elemente als um einzelne Variablen handeln. Zusätzlich zum einfachen Gleichheitszeichen gibt es zahlreiche weitere Modifikationsoperatoren, die den bereits vorhandenen Wert des LVALUE ändern. Der zugewiesene Wert schließlich kann ein beliebig komplexer Ausdruck sein, der zunächst ausgewertet wird, bevor er im LVALUE gespeichert wird. Nach der Wertzuweisung kann der LVALUE wiederum in andere Ausdrücke eingesetzt werden, wo er dann jeweils für seinen aktuellen Wert steht und in die Ermittlung neuer Werte einfließen kann.
- *Kontrollstrukturen* sind alle Elemente, die den geradlinigen Programmablauf aus einzelnen Anweisungen unterbrechen, indem sie beispielsweise vor der Ausführung von Code eine Bedingung prüfen (Fallentscheidung) oder Anweisungen für verschiedene Fälle mehrfach ausführen (Schleife). Erst das Vorhandensein von Kontrollstrukturen macht Programmierung flexibel genug, um dadurch beliebige Aufgaben zu lösen.
- Schließlich ermöglicht die *Deklaration* eigener Funktionen und Klassen (Vorlagen für Objekte eines bestimmten Typs) die Wiederverwendbarkeit von Programmteilen und ist so ein wichtiger Bestandteil guten Softwaredesigns.

Nicht zu unterschätzende Bestandteile von Computerprogrammen sind obendrein die *Kommentare*. Sie sind kein ausführbarer Programmcode, sondern Erläuterungen für diejenigen, die den Code schreiben, lesen und später überarbeiten. Das Kommentieren wesentlicher Pro-

grammbestandteile ist ein wichtiger Aspekt lesbaren Codes, denn es ermöglicht einen wesentlich schnelleren Zugriff auf Informationen über ein Programm als das Lesen mehrerer Codezeilen. Zudem ist es einfacher, Programmfehler zu finden, wenn dokumentiert wird, was die Aufgabe eines Anweisungsblocks sein soll.

Python bietet zwei verschiedene Arten von Kommentaren. Die Raute (#) leitet einen einzeiligen Kommentar ein, der bis zum nächsten Zeilenumbruch reicht. Beispiel:

```
print("Test")  # Anweisung und Kommentar
# Nur Kommentar
```

Ein mehrzeiliger Kommentar beginnt und endet jeweils mit drei doppelten Anführungszeichen. Dieser Kommentarstil wird vor allem zur Generierung von Programmdokumentationen aus dem Quellcode verwendet. Hier ein Beispiel:

```
"""Mehrzeiliger Kommentar
Wird üblicherweise zur Programmdokumentation verwendet.
"""
```

Ausdrücke

Wie bereits erwähnt, können Werte aus beliebig komplexen *Ausdrücken* ermittelt werden. Die Ausdrücke wiederum können aus folgenden verschiedenen Bestandteilen zusammengesetzt sein:

- Die einfachsten Bestandteile sind *Literale*, also Werte, die wörtlich so gemeint sind, wie sie geschrieben wurden. Es gibt beispielsweise numerische Literale wie `42` oder `3.1415926` (beachten Sie den Dezimalpunkt statt des Kommas) oder Strings, das heißt Textliterale wie `"Hallo Welt"`, die Sie an den Anführungszeichen erkennen.
- *Operatoren* verknüpfen Werte auf unterschiedliche Weise miteinander. Dazu gehören beispielsweise die mathematischen Grundrechenarten. Beachten Sie, dass für die Multiplikation das Zeichen * und für die Division / zum Einsatz kommen.
- Weitere Bestandteile sind *Variablen* oder allgemein *LVALUEs*, denen zuvor ein Wert zugewiesen wurde und deren aktueller Wert jeweils im Ausdruck verwendet wird.
- Dabei sind auch *Funktions-* oder *Methodenaufrufe*, sofern sie einen Wert zurückgeben, mit dem dann weiter operiert wird.
- *Klammern* ermöglichen es schließlich, die Rangfolge der Operatoren (etwa die aus der Arithmetik bekannte und auch in praktisch allen Programmiersprachen geltende Regel »Punkt- vor Strichrechnung«) zu übergehen: `3 + 5 * 4` bedeutet etwas anderes als `(3 + 5) * 4`, nämlich 23 beziehungsweise 32. Wie in der Mathematik können Sie Klammern beliebig tief ineinander verschachteln – zu diesem spezifischen Zweck allerdings ausschließlich runde, da eckige Klammern in Python und vielen anderen Programmiersprachen eine spezielle Bedeutung haben.

Literale

Literale als einfachste Formen – und auch als Bestandteile – von Ausdrücken brauchen nicht mehr auf irgendeine Weise ausgewertet zu werden, sondern sind wie gesagt wörtlich (englisch: *literally*) so gemeint, wie sie im Programm stehen.

Numerische Literale können als ganze Zahlen (*Integer*) oder als Fließkommazahlen vorkommen. *Ganze Zahlen* werden üblicherweise dezimal angeben, beispielsweise `-23` oder `42`. Optional können Sie den Unterstrich _ als Trennzeichen verwenden, typischerweise für Tausenderstellen:

```
>>> 1_000_000
1000000
```

Eine Besonderheit von Ganzzahlen in Python ist ihre beliebige Größe (soweit sie in den Speicher passen). In praktisch allen anderen Programmiersprachen haben sie eine bestimmte Speichergröße, zum Beispiel 32 oder 64 Bit, und können daher nur Werte in den in Kapitel 2, »Mathematische Grundlagen«, beschriebenen Bereichen enthalten. In Python können Sie dagegen beispielsweise die Operation 12345^{12345} ausführen:

```
>>> 12345**12345
286786522500366944...
```

Das Ergebnis ist eine Zahl mit 50.510 Stellen, was Sie durch folgenden Python-Ausdruck ermitteln können:

```
>>> len(str(12345**12345))
50510
```

Die fertig berechnete Zahl wird hier zunächst mit `str()` in einen String umgewandelt, und anschließend wird mit `len()` dessen Länge gemessen.

Neben Dezimalzahlen können Sie auch Hexadezimalzahlen schreiben. Diese werden durch ein vorangestelltes `0x` gekennzeichnet:

```
>>> 0xABCD
43981
```

Oktalzahlen werden mit vorangestelltem `0o` (Ziffer Null, kleiner Buchstabe o) geschrieben:

```
>>> 0o27
23
```

Auch Dualzahlen sind als Literale zu schreiben, indem Sie ihnen die Zeichenfolge `0b` voranstellen:

```
>>> 0b101010
42
```

Sie können Ganzzahlen übrigens ebenfalls ins Dual-, Oktal- oder Hexadezimalsystem umwandeln – allerdings sind die Ergebnisse formal keine Zahlen mehr, sondern Strings. Beispiele:

```
>>> bin(23) # Dezimal -> Dual
'0b10111'
>>> oct(23) # Dezimal -> Oktal
'0o27'
>>> hex(23) # Dezimal -> Hexadezimal
'0x17'
>>> oct(0xff) # Hexadezimal -> Oktal
'0o377'
>>> bin(0o42) # Oktal -> Dual
'0b100010'
```

Eine grundsätzlich andere Art von Zahlen sind *Fließkommazahlen* (englisch: *Floating Point Numbers*). Diese Literale werden mit Dezimalpunkt geschrieben, beispielsweise `0.23` oder `-1.03`, oder mithilfe der Exponentialschreibweise, etwa `4e4` für $4 * 10^4$ (40.000) oder `3.1e-3` für $3 * 10^{-3}$ (0,0031). Im Gegensatz zu Ganzzahlen haben sie eine fest definierte Speichergröße und damit eine maximale Genauigkeit, nämlich 64 Bit.

Eine Besonderheit von Python ist, dass bereits in der Grundausstattung *komplexe Zahlen* enthalten sind. Eine komplexe Zahl besteht aus einem *Realteil*, also einer beliebigen reellen Zahl, und einem hinzuaddierten *Imaginärteil* (ein Vielfaches von *i*, definiert als eine Zahl, deren Quadrat –1 ist, also $i := i^2 = -1$). Da das Quadrat sowohl positiver als auch negativer Zahlen für reelle Zahlen positiv ist, werden Zahlen mit negativem Quadrat, die beispielsweise in der Physik eine Rolle spielen, als imaginär bezeichnet. Komplexe Zahlen werden wie folgt geschrieben:

```
>>> complex(3, 4)
(3+4j)
```

Innerhalb der Klammern von `complex()` wird also zuerst der Real- und dann der Imaginärteil angegeben. Die interne Darstellung besteht, wie Sie sehen, aus Klammern, dem Realteil, einem Plus- oder Minuszeichen (je nachdem, ob der Imaginär-Multiplikator positiv oder negativ ist), dem Multiplikator selbst und dem Buchstaben `j`. Die eingebauten Attribute `real` und `imag` liefern den Imaginär- beziehungsweise Realteil zurück:

```
>>> complex(3, 5).real
3.0
>>> complex(4, -4).imag
-4.0
```

Ansonsten können Sie mit komplexen Zahlen gemäß den für sie geltenden mathematischen Regeln rechnen, auch gemischt mit anderen Zahlentypen. Beispiele:

```
>>> complex(1, 2) + complex(3, 4)
(4+6j)
>>> complex(3, 5) + 7
(10+5j)
```

Strings, auf Deutsch manchmal *Zeichenketten* genannt, werden zur Speicherung und Verarbeitung von Text verwendet. Auch für sie gibt es verschiedene Schreibweisen:

- doppelte Anführungszeichen, zum Beispiel `"The Holy Grail"`
- einfache Anführungszeichen, etwa `'Dead Parrot'`
- Sogenannte *Here Documents* (auf Deutsch auch Hier-Dokumente) beginnen mit drei Anführungszeichen (einfachen oder doppelten), können sich über beliebig viele Zeilen erstrecken und enden wieder mit drei Anführungszeichen (derselben Sorte, mit der sie begonnen wurden). Der Name bezieht sich darauf, dass diese Strings »bis hierhin« reichen. Dies ist ein Beispiel aus der interaktiven Shell, das zeigt, wie die Zeilenumbrüche Teil des Strings werden:

  ```
  >>> '''Arthur
  ... Galahad
  ... Lancelot
  ... Robin'''
  'Arthur\nGalahad\nLancelot\nRobin'
  ```

  Mehrzeilige Kommentare werden wie erwähnt genauso geschrieben wie Here Documents mit doppelten Anführungszeichen – im Prinzip sind sie auch identisch, werden aber nicht im Programm ausgewertet, da sie an den entsprechenden Stellen weder als Werte in Variablen gespeichert noch in Ausdrücken verwendet werden.

Anders als in manchen anderen Skriptsprachen gibt es keinen funktionalen Unterschied zwischen einfachen und doppelten Anführungszeichen (siehe etwa die Sprache PHP in Kapitel 19, »Webserveranwendungen«), und auch ein einzelnes Zeichen in einfachen Anführungszeichen ist ein String – anders als in der später in diesem Kapitel behandelten Sprache Java, in der ein solches einzelnes Zeichen ein eigener Datentyp ist. Sie können innerhalb eines Strings mit einer bestimmten Sorte von Anführungszeichen die jeweils andere Sorte ohne Escaping verwenden:

```
>>> "'We have Spam, bacon, sausage and Spam,' the waitress said."
"'We have Spam, bacon, sausage and Spam,' the waitress said."
>>> '"We have Spam, egg, Spam, Spam, bacon and Spam," she added.'
'"We have Spam, egg, Spam, Spam, bacon and Spam," she added.'
```

Möchten Sie dagegen dieselbe Sorte Anführungszeichen verschachteln, müssen Sie sie mithilfe eines Backslashs (\) escapen:

```
>>> '"\'Tis but a scratch," the Black Knight roared.'
'"\'Tis but a scratch," the Black Knight roared.'
>>> "\"It's just a flesh wound,\" he added."
'"It\'s just a flesh wound," he added.'
```

Neben den Anführungszeichen gibt es noch andere sogenannte *Escape-Sequenzen*. Wenn Sie den Backslash als Zeichen benötigen, müssen Sie ihn aus naheliegenden Gründen ebenfalls escapen, und zwar als `\\`. Speziellere Escape-Sequenzen sind beispielsweise `\n` für den Zeilenumbruch und `\t` für den Tabulator.

Besondere Literale sind `True` und `False`, die zur Boolean-Klasse (`bool`) gehören, sowie `None`, die einzige Instanz der Klasse `NoneType`. Beachten Sie, dass alle diese Werte mit großen Anfangsbuchstaben geschrieben werden müssen. `True` und `False` sind boolesche Wahrheitswerte, also zum Beispiel die Ergebnisse von Vergleichsoperationen:

```
>>> 2 < 3
True
>>> 2 > 3
False
```

`None` ist eine sogenannte leere Referenz, also ein Verweis auf nichts. Dieser spezielle Wert wird oft verwendet, um anzuzeigen, dass eine Wertermittlung noch nicht stattgefunden hat oder nicht möglich ist.

Variablen, die noch nicht definiert wurden, haben nicht den Wert `None`, stattdessen führt der Zugriff auf sie zu einer Fehlermeldung, wie das folgende Beispiel zeigt:

```
>>> print(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Neben den bisher besprochenen skalaren (einwertigen) Literalen gibt es noch verschiedene Arten von Listen und weitere zusammengesetzte Literale, etwa eine einfache Liste wie `[1, 2, 3]`. Diese werden später in einem eigenen Unterabschnitt behandelt.

Jedes Literal, und natürlich auch der Wert jedes Ausdrucks, gehört in Python zu einer bestimmten Klasse; Sie können diese mithilfe der Funktion `type(`*`ausdruck`*`)` ermitteln. Beispiele:

```
>>> type(42)
<class 'int'>
```

```
>>> type(3.1415926)
<class 'float'>
>>> type("Camelot")
<class 'str'>
>>> type([1,2,3])
<class 'list'>
>>> type(True)
<class 'bool'>
```

Variablen verwenden

Eine Variable ist ein benannter Speicherplatz. Sie können darin einen bestimmten Wert oder ein bestimmtes Objekt ablegen, und wenn Sie den Namen in einem Ausdruck benutzen, wird der jeweils aktuelle Wert eingesetzt. Variablen werden in Python durch einfache Wertzuweisungen definiert. Beispiele:

```
>>> a = 1
>>> b = "Hallo"
>>> a
1
>>> b
'Hallo'
```

Der Typ ist dabei ausschließlich an den enthaltenen Wert gebunden, nicht an die Variable selbst. Sie können derselben Variablen daher nacheinander Werte verschiedener Typen zuweisen:

```
>>> x = 1
>>> type(x)
<class 'int'>
>>> x = "Test"
>>> type(x)
<class 'str'>
```

Aus diesem Grund wird Python als *dynamisch typisierte* Sprache bezeichnet.

In Bezeichnern sind Buchstaben, Ziffern und Unterstriche erlaubt, wobei das erste Zeichen keine Ziffer sein darf. Es wird zwischen Groß- und Kleinschreibung unterschieden, wie die Fehlermeldung im folgenden Versuch zeigt:

```
>>> test = 3
>>> test
3
>>> Test
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Test' is not defined
```

Übrigens sind nicht nur die 26 Buchstaben des lateinischen Alphabets erlaubt, sondern praktisch alle Buchstaben, Silbenzeichen und Ideogramme, die im Unicode-Zeichensatz vorkommen:

```
>>> schöne_variable = 1
>>> schöne_variable
1
>>> こんにちは = "Hello"
>>> こんにちは
'Hello'
>>> πύθων = "My favourite programming language"
>>> πύθων
'My favourite programming language'
```

Ob Sie diese nutzen möchten, sollten Sie sorgfältig abwägen. Bedenken Sie, dass verschiedene Betriebssysteme standardmäßig unterschiedliche Zeichensätze verwenden (Details dazu erfahren Sie in Kapitel 17, »Weitere Datei- und Datenformate«), sodass solche spezielleren Schriftzeichen möglicherweise nicht gut auf diese übertragbar sind. Mit lateinischen Buchstaben ohne Umlaute sind Sie auf der sicheren Seite, und das vorliegende Buch beschränkt sich abgesehen von den obigen Beispielen auf diese.

Eine weitere Konvention in diesem Kapitel und in diesem Buch ist, dass weitgehend englische Variablen-, Funktions- und Klassennamen verwendet werden. Teams, die zusammen an Software arbeiten, werden zunehmend internationaler, und Englisch ist die Standardsprache für die Softwareentwicklung, zumal das Grundvokabular der allermeisten Programmiersprachen ebenfalls auf englischen Wörtern basiert.

Regeln für Bezeichner in Python

Es gibt einige Konventionen für Bezeichner in Python:

- Variablen und Funktionen beginnen mit einem Kleinbuchstaben: Wenn sie aus mehreren Wörtern bestehen, werden diese durch Unterstriche getrennt (*snake_case*). Beispiele: `value`, `number_of_elements`
- Klassennamen werden großgeschrieben (mit Ausnahme von Datentypen und vielen Basisklassen der Standardbibliothek), und mehrere Wörter werden durch Binnenmajuskeln voneinander getrennt (*CamelCase*). Beispiele: `Book`, `BookPublisher`
- Elemente, die als privat gelten, also nicht Teil der öffentlichen Schnittstelle eines Programms sind, sollten mit einem einzelnen Unterstrich beginnen. Beispiele: `_internal_value`, `_InternalClass`

- Viele interne Elemente von Python selbst beginnen und enden mit je zwei Unterstrichen, etwa die Methode `__init__()` – der Konstruktor einer Klasse – oder die Konstante `__name__` (Name des aktuellen Moduls oder Ausführungskontexts).
- In Python gibt es formal keine allein stehenden *Konstanten* (Platzhalter mit permanent festgelegtem Wert). Trotzdem ist es üblich, Variablen, die in diesem Kontext verwendet werden, wie die echten Konstanten anderer Programmiersprachen komplett in Großbuchstaben zu schreiben. Beispiele:

  ```
  MILE = 1.609
  INCH = 2.54
  ```

Eine praktische Eigenschaft von Python ist, dass Sie als LVALUE nicht nur eine einzelne Variable, sondern auch eine durch Kommata getrennte Aufzählung von Variablen angeben können. Wichtig ist allerdings, dass Sie hinter dem Gleichheitszeichen die entsprechende Anzahl von Werten – ebenfalls durch Kommata getrennt – angeben müssen, da es sonst Fehlermeldungen gibt. Beispiel:

```
>>> a, b = 1, 2
>>> a
1
>>> b
2
```

Dieses Idiom wird besonders häufig eingesetzt, um die Werte zweier Variablen zu vertauschen:

```
>>> a, b = b, a
>>> a
2
>>> b
1
```

In den meisten Programmiersprachen müssen Sie dafür den – in Python natürlich ebenfalls möglichen – Umweg über eine Hilfsvariable gehen:

```
>>> a = "one"
>>> b = "two"
>>> a
'one'
>>> b
'two'
>>> helper = a
>>> a = b
```

```
>>> b = helper
>>> a
'two'
>>> b
'one'
```

Variablen sind grundsätzlich Referenzen, das heißt, sie verweisen auf Objekte, die sich irgendwo im Speicher befinden. Wo sich das Objekt im Speicher befindet (beziehungsweise ob zwei identisch aussehende Objekte dasselbe Objekt sind), können Sie mithilfe der Identitätsfunktion `id()` herausfinden (bei Ihnen werden die konkreten Werte höchstwahrscheinlich andere sein als hier):

```
>>> id(42)
4297372000
>>> id("Hallo")
4338199216
```

Es wird zwischen *unveränderlichen Objekten* (*Immutable Objects*) und *veränderlichen Objekten* (*Mutable Objects*) unterschieden. Veränderliche Objekte behalten ihren Speicherort auch dann, wenn ihr Wert verändert wird, während eine Wertänderung einer Variablen mit einem unveränderlichen Objekt nach der Wertänderung auf eine andere Speicherstelle verweist.

Skalare Datentypen sind in der Regel unveränderlich, wie das folgende Beispiel zeigt – der Identitätswert verändert sich:

```
>>> a = 23
>>> id(a)
4297371392
>>> a += 1   # a um 1 erhöhen
>>> id(a)
4297371424
```

Zusammengesetzte Datentypen wie Listen oder Objekte komplexer Klassen sind dagegen meist veränderlich; sie behalten ihre Identität auch über eine Wertänderung hinaus:

```
>>> a = [1, 2, 3]  # einfache Liste
>>> id(a)
4338176968
>>> a.append(4)    # weiteres Element hinzufügen
>>> a
[1, 2, 3, 4]
>>> id(a)
4338176968
```

Wenn eine Variable auf ein veränderliches Objekt verweist und Sie einer weiteren Variablen den Wert der ersten zuweisen, dann verweisen beide auf dasselbe Objekt, und ihr Wert verändert sich entsprechend. Hier ein Beispiel mit zwei Variablen, die auf dieselbe Liste verweisen:

```
>>> ref1 = ['ham', 'eggs', 'bacon']
>>> ref2 = ref1
>>> ref2.append('Spam')   # Element an ref2 anhängen
>>> ref1
['ham', 'eggs', 'bacon', 'Spam']
```

Wie Sie sehen, verändert das Hinzufügen des Elements zu `ref2` auch den Wert von `ref1`, da sie auf dasselbe Objekt verweisen (was Sie bei Bedarf wieder mithilfe von `id()` überprüfen können).

Bei unveränderlichen Objekten kann es durchaus vorkommen, dass verschiedene Variablen auf dasselbe interne Objekt verweisen, solange derselbe Wert gespeichert wird, da dieses Vorgehen speicherschonend ist (es wird als *Copy-on-Write* bezeichnet und ist weit verbreitet). Das muss jedoch nicht der Fall sein. In jedem Fall modifiziert die Änderung der einen Variablen aber nicht den Wert der anderen. Sehen Sie sich dazu folgendes Beispiel an:

```
>>> var1 = 23
>>> var2 = var1
>>> id(var1)
4297371392
>>> id(var2)
4297371392
>>> var2 += 1    # um 1 erhöhen
>>> var2
24
>>> var1
23
>>> id(var2)
4297371424
```

Wie Sie sehen, hatten `var1` und `var2` in meiner Sitzung dieselbe Identität, solange sie denselben Wert speicherten. Sobald `var2` verändert wird, hat diese eine neue Identität und einen neuen Wert, während `var1` den ursprünglichen Wert behält.

Wenn Sie statt eines zusätzlichen Verweises auf ein veränderliches Objekt eine Kopie dieses Objekts benötigen, die sich unabhängig vom Original modifizieren lässt, müssen Sie dies explizit angeben. Da die Vorgehensweise für verschiedene Datentypen unterschiedlich ist, wird das später im Buch für die jeweiligen Einzelfälle erläutert.

Operatoren einsetzen

Operatoren kennen Sie beispielsweise aus der Mathematik. Sie verknüpfen die Werte von (meist zwei) Werten oder Ausdrücken – den Operanden – nach einer bestimmten Vorschrift.

Die arithmetischen Operatoren für numerische Datentypen sind auch in Python die einfachsten und bekanntesten Operatoren. Neben + (Addition), - (Subtraktion), * (Multiplikation) und / (Division) gibt es noch den speziellen Operator % (Modulo), der den Rest einer ganzzahligen Division berechnet:

```
>>> 7 % 4  # 7 / 4 = 1, Rest 3
3

>>> 7 % 3  # 7 / 3 = 2, Rest 1
1
```

Der normale Divisionsoperator / gibt unabhängig vom Datentyp der Operanden immer eine Fließkommazahl (Klasse `float`) zurück, wie das folgende Beispiel zeigt:

```
>>> 6 / 3
2.0
>>> type(6 / 3)
<class 'float'>
```

Die spezielle Variante // gibt dagegen eine Ganzzahl (`int`) zurück, solange beide Operanden ebenfalls `int` sind:

```
>>> 6 // 3
2
>>> 7 // 3
2
>>> type(7 // 3)
<class 'int'>
```

Ist mindestens einer der Operanden ein `float`, gibt allerdings auch // ein Ergebnis vom Typ `float` zurück, aber mit dem Wert der ganzzahligen Division:

```
>>> 6.5 // 3
2.0
>>> type(6.5 // 3)
<class 'float'>
```

Denn das korrekte Fließkommaergebnis ist natürlich ein anderes:

```
>>> 6.5 / 3
2.1666666666666665
```

Die Ziffer 5 am Ende des Ergebnisses ist übrigens ein Rundungsfehler; diese entstehen aufgrund der endlichen Genauigkeit von Fließkommazahlen.

Beachten Sie, dass der zweite Operand bei keinem der drei Divisionsoperatoren 0 sein darf, ansonsten erhalten Sie eine Fehlermeldung.

Eine Besonderheit in Python gegenüber vielen anderen Sprachen ist der bereits nebenbei gezeigte Potenzoperator **, wobei `a ** b` für a^b steht. Hier einige Beispiele:

```
>>> 2 ** 10
1024
>>> 3 ** 3
27
>>> 9 ** 0.5
3.0
```

Es wurde bereits gezeigt, dass + nicht nur zur numerischen Addition verwendet werden kann, sondern auch zur Verkettung von Strings:

```
>>> planet = "Welt"
>>> "Hallo " + planet
'Hallo Welt'
```

Wenn Sie Strings mit Teilausdrücken verketten möchten, die selbst keine Strings sind, müssen Sie für Letztere explizit `str()` aufrufen, um sie in ihre String-Darstellung umzuwandeln:

```
>>> "3 + 5 = " + str(3 + 5)
'3 + 5 = 8'
```

Auch den Operator * können Sie für Strings verwenden – genauer gesagt, für einen String und eine positive Ganzzahl, um einen String zu erzeugen, der den ursprünglichen String entsprechend oft wiederholt:

```
>>> "Ho! " * 3
'Ho! Ho! Ho! '
>>> 3 * "Kölle Alaaf! "
'Kölle Alaaf! Kölle Alaaf! Kölle Alaaf! '
```

Obwohl sich die *Bit-Operatoren*, von denen einige in Kapitel 2, »Mathematische Grundlagen« erläutert wurden, aus den logischen Operatoren entwickelt haben, aber auf die einzelnen Bits ganzer Zahlen angewendet werden, handelt es sich von ihrer Wirkung her doch eher um arithmetische Operatoren, sodass sie hier zuerst vorgestellt werden. Die logischen Operatoren kommen nach den Vergleichsoperatoren an die Reihe. Im Einzelnen gibt es folgende Bit-Operatoren, deren Wirkung hier jeweils an einem dual dargestellten 8-Bit-Beispiel gezeigt wird:

- Das *bitweise Und* (geschrieben `&`) setzt im Ergebnis diejenigen Bits auf den Wert 1, die in beiden Operanden 1 sind, alle anderen auf 0. Beispiel: `94 & 73` führt zu dem Ergebnis 72. Erläutern lässt sich dieses Ergebnis nur anhand der binären Darstellung:

```
  0101 1110
& 0100 1001
-----------
  0100 1000
```

- Das *bitweise Oder* (`|`) setzt alle Bits auf 1, die in mindestens einem der Operanden den Wert 1 haben. `94 | 73` ergibt demzufolge 95:

```
  0101 1110
| 0100 1001
-----------
  0101 1111
```

- Das *bitweise exklusive Oder* (`^`) setzt nur diejenigen Bits auf 1, die in genau einem Operanden den Wert 1 haben, alle anderen dagegen auf 0. `94 ^ 73` liefert das Ergebnis 23:

```
  0101 1110
^ 0100 1001
-----------
  0001 0111
```

- Die *Bit-Verschiebung nach links* (`<<`) verschiebt die Bits des ersten Operanden um die Anzahl von Stellen nach links, die der zweite Operand angibt. Beispiel: `73 << 2` ergibt 292, entspricht also einer Multiplikation mit 2^2 (4).
- Die *Bit-Verschiebung nach rechts* (`>>`) verschiebt die Bits des ersten Operanden dagegen um die angegebene Anzahl von Stellen nach rechts. Beispiel: `94 >> 3` führt zu dem Ergebnis 11, da die letzten drei Binärstellen wegfallen.
- Die *bitweise Negation* oder das *Bit-Komplement* (eine vorangestellte Tilde `~`) setzt alle Bits mit dem Wert 1 auf 0 und umgekehrt. Aufgrund der Art und Weise, wie negative Zahlen codiert werden, ergibt `~73` in Python den Wert –74.

Die Bit-Operatoren werden manchmal anstelle entsprechender arithmetischer Operationen verwendet, weil sie etwas schneller sind und auch weil sie manchmal verständlicher machen, welche Art von Berechnung gerade stattfindet. Beispielsweise wird eine Reihe von Ein/Aus-Einstellungen manchmal speicherplatzschonend als Abfolge einzelner Bits gespeichert, wobei für die einzelnen Einstellungen oft (formale) Konstanten mit den Werten dieser Bits definiert werden. Das folgende Beispiel definiert vier mögliche Features von Fenstern in einer grafischen Benutzeroberfläche, die jeweils aktiviert oder deaktiviert werden können, was durch die Bit-Werte 1 beziehungsweise 0 gekennzeichnet wird:

```
>>> TITLE_BAR = 1
>>> MENU = 2
>>> CLOSE_BUTTON = 4
>>> RESIZABLE = 8
```

Die Eigenschaften eines konkreten Fensters werden durch die Bitweises-Oder-Operation aus den Konstanten zusammengesetzt:

```
>>> window_features = TITLE_BAR | CLOSE_BUTTON
>>> window_features
5
```

Ob ein Fenster eines der Features aufweist oder nicht, können Sie anschließend mit dem bitweisen Und überprüfen. Das Ergebnis ist der jeweilige Bit-Wert, wenn das Feature vorhanden ist, ansonsten 0. Da Letzteres als `False` interpretiert wird, ist dies sehr praktisch:

```
>>> window_features & CLOSE_BUTTON
4
>>> window_features & RESIZABLE
0
```

Schließlich können Sie das bitweise exklusive Oder verwenden, um zwischen dem aktuellen Zustand eines Features und dem jeweils anderen hin- und herzuschalten (aus wird zu an und umgekehrt):

```
>>> window_features = window_features ^ CLOSE_BUTTON
>>> window_features
1
>>> window_features = window_features ^ CLOSE_BUTTON
>>> window_features
5
>>> window_features = window_features ^ RESIZABLE
>>> window_features
13
>>> window_features = window_features ^ RESIZABLE
>>> window_features
5
```

Alle arithmetischen und Bit-Operatoren können mit einem Gleichheitszeichen kombiniert werden, um als spezielle Wertzuweisungsoperatoren eine Variable zu modifizieren. Hier nur zwei Beispiele:

```
>>> a = 7
>>> a += 6          # Wert von a um 6 erhöhen
```

```
>>> a
13
>>> a <<= 2        # Wert von a um 2 Bit nach links verschieben
>>> a
52
```

Die nächste Gruppe von Operatoren sind die *Vergleichsoperatoren*. Im Einzelnen sind folgende definiert:

- `<` (kleiner als)
- `>` (größer als)
- `<=` (kleiner oder gleich)
- `>=` (größer oder gleich)
- `==` (gleich)
- `!=` (ungleich)

Der Rückgabewert jeder Vergleichsoperation ist `True`, wenn sie zutrifft, und andernfalls `False`.

Bei numerischen Werten verhalten sich diese Operatoren wie erwartet; dabei wird bei Bedarf automatisch zwischen Fließkomma- und Ganzzahlen konvertiert:

```
>>> 23 == 23
True
>>> 23 < 23
False
>>> 42 == 42.0
True
```

Eine weitergehende Konvertierung, etwa zwischen Strings und numerischen Werten, erfolgt nicht, wie das nächste Beispiel zeigt:

```
>>> 3 == "3"
False
```

Vergleiche, die eine Ordnung der Elemente erfordern, ergeben sogar eine Fehlermeldung, wenn die Typen inkompatibel sind:

```
>>> 3 < "3"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: int() < str()
```

Ansonsten können Vergleichsoperatoren auch für Strings, Listen und andere Typen verwendet werden, solange die verglichenen Objekte denselben Typ haben. Bei Strings ist die Ord-

nung die Position der enthaltenen Zeichen im Zeichensatz, während bei Listen sowohl die Anzahl der Elemente als auch deren interne Ordnung betrachtet werden. Hier einige Beispiele:

```
>>> "hallo" == "hello"
False
>>> "hallo" < "hello"
True
>>> "comp" < "computer"
True
>>> [1, 2, 3] == [1, 2, 3]
True
>>> [1, 2, 3] == [1, 2, 3, 4]
False
>>> [1, 2, 3] < [1, 2, 3, 4]
True
>>> ["a", "b", "c"] < ["b", "c", "d"]
True
```

Bei gerichteten Vergleichen von Listen müssen alle Elemente beider Listen denselben (oder einen konvertierbaren) Typ haben, ansonsten führt der Versuch wiederum zu Fehlermeldungen:

```
>>> [1, 2, 3] == [1.0, 2.0, 3.0]    # konvertierbar
True
>>> [1, 2, 3] < ["a", "b", "c"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: int() < str()
>>> [1, "a"] < ["a", 1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: int() < str()
```

Die *logischen Operatoren* verknüpfen Aussagen gemäß den in Kapitel 2, »Mathematische Grundlagen«, erarbeiteten Regeln: Der Operator `and` ist das logische Und. Das Ergebnis ist nur `True`, wenn alle Teilausdrücke als `True` interpretiert werden können. Beispiele:

```
>>> 1 == 1 and 1 < 2
True
>>> 1 == 1 and 1 == 2
False
```

Mit `or` führen Sie ein logisches Oder aus. Der Gesamtausdruck ist `True`, wenn mindestens ein Teilausdruck `True` ist:

```
>>> 1 == 1 or 1 < 2
True
>>> 1 == 1 or 1 == 2
True
>>> 1 > 2 or 1 == 2
False
```

Schließlich steht `not` für die logische Verneinung: Ausdrücke, die `True` sind, werden `False` und umgekehrt, wie die folgenden Beispiele zeigen:

```
>>> not 1 == 1
False
>>> not 1 > 2
True
```

Da die logischen Operatoren die niedrigste Priorität von allen haben, brauchen Sie die mit ihnen behandelten Ausdrücke für gewöhnlich nicht in Klammern zu setzen (außer um ihre Priorität untereinander zu regeln): `not 1 == 1` bedeutet stets `not(1 == 1)` statt `(not 1) == 1`, obwohl natürlich beides `False` ergibt.

Wie im Zusammenhang mit Fallentscheidungen noch ausführlicher geklärt wird, werden folgende Ausdrücke, die nicht bereits `bool` sind, als `False` interpretiert: `0`, leere Strings, leere Listen und ähnliche Konstrukte sowie `None`. Alle anderen Ausdrücke gelten als `True`, wie die Behandlung durch logische Operatoren zeigt:

```
>>> 23 and 42
42
>>> 23 and 0
0
>>> "Text 1" and "Text 2"
'Text 2'
>>> "Text 1" and ""
''
>>> [1, 2, 3] and [5, 6, 7]
[5, 6, 7]
>>> [1, 2, 3] and []
[]
>>> 23 or 42
23
>>> 23 or 0
23
```

```
>>> 0.0 or 0
0
>>> "Text 1" or "Text 2"
'Text 1'
>>> "Text 1" or ""
'Text 1'
>>> "" or ""
''
>>> [1, 2, 3] or [5, 6, 7]
[1, 2, 3]
>>> [1, 2, 3] or []
[1, 2, 3]
>>> [] or ()
()
```

Diese Reihe von Ausdrücken zeigt zwei Merkmale der logischen Operatoren:

- Das jeweilige Ergebnis ist nicht `bool` (`True` oder `False`), sondern einer der Teilausdrücke, wenn diese selbst nicht vom Typ `bool` sind, aber es handelt sich um ein Ergebnis, das gemäß den genannten Regeln als `True` beziehungsweise `False` interpretiert wird.
- Die Operatoren unterliegen der *Short-Circuit-Logik* (englisch für Kurzschluss): Sobald klar ist, dass der Gesamtausdruck wahr oder falsch ist, stoppt die Auswertung. Das ist bei `and` der Fall, sobald ein Operand als `False` interpretiert wird, und bei `or`, sobald einer als `True` interpretiert wird.

Die Short-Circuit-Logik können Sie sich sogar zunutze machen, um mit `and` eine Anweisung auszuführen, wenn ein Ausdruck `True` ist, oder mit `or`, wenn er `False` ist:

```
>>> 1 and print("Das ist wahr.")
Das ist wahr.
>>> 0 and print("Das ist falsch.") # keine Ausgabe
0
>>> 1 or print("Das ist wahr.") # keine Ausgabe
1
>>> 0 or print("Das ist falsch.")
Das ist falsch.
```

Beachten Sie, dass die mit dem Kommentar »keine Ausgabe« gekennzeichneten Fälle in der interaktiven Konsole die Auswertung des jeweiligen Ausdrucks anzeigen, in einem normalen Programm jedoch wie gewünscht einfach übergangen werden.

Ein logisches Exklusiv-Oder gibt es nicht, aber es kann durch die folgende Formel simuliert werden:

```
(a and not b) or (not a and b)
```

Wenn Sie nur das reine bool-Ergebnis brauchen, können Sie auch die einzelnen Operanden explizit in bool-Werte umwandeln und dann != verwenden:

```
bool(a) != bool(b)
```

Ein Python-spezifischer Operator ist der *Identitätsoperator* is, der nur dann True ergibt, wenn dasselbe Objekt referenziert wird. Hier ein Beispiel dafür, wie er funktioniert:

```
>>> var1 = [1, 2, 3]
>>> var2 = var1
>>> var1 is var2
True
>>> [1, 2, 3] is [1, 2, 3]
False
```

Da var1 und var2 auf dasselbe veränderliche Objekt verweisen, sind sie identisch, sodass var1 is var2 den Wert True zurückliefert. Die beiden identisch aussehenden veränderlichen Literale mit dem Wert [1, 2, 3] sind dagegen nicht dasselbe Objekt, und is liefert False zurück. Aus Bequemlichkeitsgründen ist auch der Umkehroperator is not definiert; a is not b ist dabei die Umkehrung von a is b und eine alternative Schreibweise für not a is b.

Schließlich gibt es noch den Elementoperator in, der True zurückgibt, wenn der linke Operand im rechten enthalten ist. Er funktioniert sowohl zum Testen von Teil-Strings in anderen Strings als auch für skalare Werte als Elemente in Listen und anderen zusammengesetzten Typen. Hier sehen Sie ein paar Beispiele:

```
>>> "al" in "Hallo"
True
>>> "ha" in "Hallo"
False
>>> "Hallo" in "Hallo"
True
>>> 3 in [1, 2, 3, 4]
True
>>> 3 in ["1", "2", "3", "4"]
False
```

Wie Sie sehen, wird bei Strings zwischen Groß- und Kleinschreibung unterschieden: "ha" in "Hallo" ergibt False. Werden zwei identische Strings verglichen, gilt der linke als Teil-String des rechten; "Hallo" in "Hallo" ergibt True. Und wie üblich werden Ziffern nicht automatisch in Strings konvertiert oder umgekehrt, sodass der String "3" nicht in der Liste [1, 2, 3, 4] enthalten ist. Bei Listen kann der Operator übrigens nur zum Prüfen auf einzelne Elemente verwendet werden; zum Testen auf Teilmengen kommt < beziehungsweise <= zum Einsatz. Genau wie beim Identitätsoperator ist auch hier eine Umkehrung definiert; sie heißt not in und liefert jeweils das Gegenteil von in zurück.

Die Rangfolge der Operatoren zeigt sich in Python wie folgt (je weiter oben ein Operator steht, desto stärker bindet er, und desto früher wird er ausgewertet):

- Exponent (**)
- Bit-Komplement (~), Plus als Vorzeichen (+) und Minus als Vorzeichen (-)
- Multiplikation (*), Division (/), ganzzahlige Division (//) und Modulo (%)
- Addition (+) und Subtraktion (-)
- Bit-Verschiebung nach links (<<) und nach rechts (>>)
- bitweises Und (&)
- bitweises Oder (|) und Exklusiv-Oder (^)
- kleiner als (<), kleiner oder gleich (<=), größer als (>) und größer oder gleich (>=)
- gleich (==) und ungleich (!=)
- Zuweisungs- und Modifikationsoperatoren (=, +=, -=, *=, /= etc.)
- Identität (`is`) und Nichtidentität (`is not`)
- Element (`in`) und Nichtelement (`not in`)
- logisches Und (`and`), logisches Oder (`or`) und logische Verneinung (`not`)

Wie üblich kann die Rangfolge mithilfe von Klammern verändert werden. Beispiel:

```
>>> 23 + 42 * 2
107
>>> (23 + 42) * 2
130
```

Listen, Mengen und andere Typen mit mehreren Elementen nutzen

Neben den bisher hauptsächlich behandelten skalaren Datentypen bietet Python ein paar sehr praktische Typen mit mehreren Elementen. Einige der wichtigsten davon werden in diesem Unterabschnitt behandelt; im nächsten erfahren Sie dann, wie Sie diese Objekte elementweise durchlaufen (iterieren) können.

Die einfache *Liste* ist der gängigste dieser Datentypen. Es handelt sich um eine indexsortierte Sammlung beliebig vieler Elemente beliebigen Typs. *Indexsortiert* bedeutet, dass die Ordnung nicht in der Größe der Elemente selbst begründet ist, sondern in der Reihenfolge, in der sie sich in der Liste befinden. Eine Liste wird durch ein Paar eckiger Klammern gekennzeichnet, und die Elemente darin werden durch Kommata voneinander getrennt. Das folgende Beispiel erzeugt eine Liste verschiedener ganzer Zahlen:

```
>>> list1 = [7, 4, 9, 16, 1, 7]
>>> list1
[7, 4, 9, 16, 1, 7]
```

Der Zugriff auf einzelne Elemente erfolgt durch den *Indexoperator*, der ebenfalls durch eckige Klammern dargestellt wird. Der Index des ersten Elements ist dabei 0. Hier zwei Beispiele:

```
>>> list1[0]
7
>>> list1[4]
1
```

Mit negativen Indexwerten lässt sich übrigens vom Ende her gerechnet auf eine Liste zugreifen; der Indexwert -1 entspricht dabei dem letzten Element:

```
>>> satz = ["Das", "ist", "ja", "wohl", "das", "Letzte"]
>>> satz[-1]
'Letzte'
>>> satz[-3]
'wohl'
```

Sie können den Indexoperator auch benutzen, um dem entsprechenden Element einer Liste einen anderen Wert zuzuweisen. Beispiel:

```
>>> peoples_front_of_judea = ["Reg", "Stan", "Judith", "Brian"]
>>> peoples_front_of_judea
['Reg', 'Stan', 'Judith', 'Brian']
>>> peoples_front_of_judea[1] = "Loretta"
>>> peoples_front_of_judea
['Reg', 'Loretta', 'Judith', 'Brian']
```

Indizes können ebenfalls verwendet werden, um Teillisten zurückzuliefern; in diesem Fall werden sie als *Slice-Operator* (Scheibe oder Ausschnitt) bezeichnet. Die Schreibweise dafür ist [*Startindex*:*Endindex*], wobei der *Endindex* dem Index des ersten Elements entspricht, das nicht mehr in der Teilliste enthalten sein soll (sprachlich »bis ausschließlich«). Das folgende Beispiel beginnt bei Index 1, also dem zweiten Element der Liste, und schließt das fünfte Element aus, sodass die Indizes 1 bis 3 enthalten sind, die dem zweiten bis vierten Element entsprechen:

```
>>> list1
[7, 4, 9, 16, 1, 7]
>>> list1[1:4]
[4, 9, 16]
```

Wenn zwei aufeinanderfolgende Indizes angegeben werden, ist das Ergebnis nicht etwa ein einzelnes Element, sondern eine Liste mit einem Element – beachten Sie die eckigen Klammern um das erste Ergebnis in diesem vergleichenden Beispiel:

```
>>> list1[2:3]   # Teilliste mit einem Element
[9]
>>> list1[2]     # einzelnes Element
9
```

Sie können hinter einen weiteren Doppelpunkt noch ein drittes Argument setzen, das eine Schrittweite angibt. Sehen Sie sich dazu ein Beispiel mit einer Liste aufeinanderfolgender Elemente an:

```
>>> list2 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list2[2:8:2]
[2, 4, 6]
```

Wenn Sie den ersten oder den zweiten Wert weglassen, wird die Liste ab ihrem Anfang beziehungsweise bis zum Ende ausgewählt:

```
>>> list2[:5]
[0, 1, 2, 3, 4]
>>> list2[5:]
[5, 6, 7, 8, 9]
```

Eine negative Schrittweite wählt die entsprechenden Elemente in umgekehrter Reihenfolge aus. Natürlich müssen Anfangs- und Endelement entsprechend angepasst werden. Beispiele:

```
>>> list2[::-1]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
>>> list2[7:2:-2]
[7, 5, 3]
```

Teillisten sind Kopien der ursprünglichen Liste anstelle von Referenzen auf diese, das heißt, wenn sie Variablen zugewiesen werden, können sie unabhängig von der ursprünglichen Liste manipuliert werden. Die spezielle Schreibweise `[:]`, ganz ohne numerische Werte, erzeugt eine Kopie der gesamten Liste. Das folgende Beispiel erzeugt auf diese Weise eine Kopie von `list2`, hängt mithilfe der Methode `append()` ein weiteres Element an und zeigt anschließend, dass die beiden Listen verschieden sind:

```
>>> list3 = list2[:]
>>> list3.append(10)
>>> list3
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list2
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Auch der Slice-Operator kann zur Wertzuweisung an den entsprechenden Ausschnitt der Liste verwendet werden; der zugewiesene Wert muss in diesem Fall ebenfalls eine Liste sein.

Bei einem einfachen Slice ohne Schrittweite können Sie unabhängig von der Größe des Ausschnitts beliebig viele Elemente einfügen. Hier ein Beispiel, das ein einzelnes Element durch drei neue ersetzt:

```
>>> liste = [0, 1, 2, 3]
>>> liste[1:2] = ["Neues Element 1", "Neues Element 2", "Neues Element 3"]
>>> liste
[0, 'Neues Element 1', 'Neues Element 2', 'Neues Element 3', 2, 3]
```

Wenn Sie dem Ausschnitt eine leere Liste zuweisen, können Sie auf diese Weise Elemente aus der Liste löschen:

```
>>> liste = ["Bleibt da", "kann weg", "kann auch weg", "bleibt auch da"]
>>> liste[1:3] = []
>>> liste
['Bleibt da', 'bleibt auch da']
```

Umgekehrt können Sie Elemente einfügen, ohne eines zu löschen, indem Sie denselben Index für das Start- und das ausschließende Endelement angeben:

```
>>> liste = ["vor den neuen", "nach den neuen"]
>>> liste[1:1] = ["neues Element 1", "neues Element 2"]
>>> liste
['vor den neuen', 'neues Element 1', 'neues Element 2', 'nach den neuen']
```

Wenn Sie Slices mit Schrittweite zur Wertzuweisung verwenden, muss die Anzahl der Elemente in der ersetzenden Liste mit der Anzahl der ersetzten Elemente übereinstimmen. Dies kann zum Beispiel so aussehen:

```
>>> liste = [0, 9, 2, 9, 4, 9, 6, 9]
>>> liste[1:8:2] = [1, 3, 5, 7]
>>> liste
[0, 1, 2, 3, 4, 5, 6, 7]
```

Sofern Sie nicht auf die korrekte Anzahl von Elementen achten, erhalten Sie eine Fehlermeldung:

```
>>> liste = ["a", "b", "c", "d", "e", "f", "g"]
>>> liste[1:7:2] = ["x", "x"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: attempt to assign sequence of size 2 to extended slice of size 3
```

Die Operanden + und * haben besondere Bedeutungen für Listen: `Liste + Liste` hängt die beiden Listen aneinander, und `Liste * Integer` erzeugt eine neue Liste, die die enthaltenen Ele-

mente so oft wiederholt wie angegeben. In beiden Fällen werden Kopien der ursprünglichen Liste(n) erzeugt, falls Sie nicht die Modifikationsoperatoren += beziehungsweise *= verwenden. Beispiele:

```
>>> list_a = [1, 2, 3]
>>> list_b = [4, 5, 6]
>>> list_a + list_b
[1, 2, 3, 4, 5, 6]
>>> list_a * 2
[1, 2, 3, 1, 2, 3]
>>> list_a += list_b
>>> list_a
[1, 2, 3, 4, 5, 6]
>>> list_a *= 2
>>> list_a
[1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]
```

Ein anderer Datentyp mit mehreren Elementen ist das *Tupel* (englisch: *Tuple*). Es kann als unveränderliche Liste betrachtet werden, ist aber eher eine Art Datensatz mit einer festgelegten Anzahl von Elementen. Ein Tupel-Literal wird als kommaseparierte Liste in runde Klammern geschrieben, beispielsweise wie folgt:

```
>>> tuple1 = (1, 4, 9, 16, 25)
>>> tuple1
(1, 4, 9, 16, 25)
```

Soll ein Tupel nur ein Element haben, müssen Sie trotzdem ein Komma hinter dem Element einfügen, denn ein einzelner skalarer Wert in runden Klammern wird nicht als Tupel interpretiert:

```
>>> tuple2 = (1,)
>>> type(tuple2)
<class 'tuple'>
>>> test_tuple = (1)
>>> type(test_tuple)
<class 'int'>
```

Alternativ können Sie die eingebaute Funktion `tuple()` verwenden, um eine Standardliste in ein Tupel umzuwandeln:

```
>>> list1 = ["Brian", "Reg", "Loretta"]
>>> tuple(list1)
('Brian', 'Reg', 'Loretta')
>>> tuple([4,5,6])
(4, 5, 6)
```

Auf Elemente eines Tupels wird wie bei Listen mit dem Index- oder dem Slice-Operator zugegriffen. Hier einige Beispiele:

```
>>> pfoj = ("Reg", "Loretta", "Judith", "Brian")
>>> pfoj[3]
'Brian'
>>> pfoj[2:4]
('Judith', 'Brian')
>>> pfoj[0:4:2]
('Reg', 'Judith')
```

Da ein Tupel eine unveränderliche Liste ist, können diese Operatoren natürlich nicht zur Wertzuweisung verwendet werden; der Versuch erzeugt eine Fehlermeldung:

```
>>> pfoj[1] = "Stan"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Die Operatoren + und * beziehungsweise += und *= können allerdings genau wie bei Listen eingesetzt werden. Betrachten Sie dazu die folgenden Beispiele:

```
>>> round_table = ("Arthur", "Lancelot")
>>> round_table * 2                     # modifizierte Kopie
('Arthur', 'Lancelot', 'Arthur', 'Lancelot')
>>> round_table
('Arthur', 'Lancelot')
>>> round_table + ("Galahad", "Robin")   # modifizierte Kopie
('Arthur', 'Lancelot', 'Galahad', 'Robin')
>>> round_table
('Arthur', 'Lancelot')
>>> round_table += ("Galahad", "Robin")  # modifiziertes Original
>>> round_table
('Arthur', 'Lancelot', 'Galahad', 'Robin')
>>> round_table *= 2                    # modifiziertes Original
>>> round_table
('Arthur', 'Lancelot', 'Galahad', 'Robin', 'Arthur', 'Lancelot', 'Galahad', 'Robin')
```

Ein anderer Typ mit mehreren Elementen ist die Menge (Klassenname `set`). Sie enthält eine ungeordnete Sammlung unterschiedlicher Elemente, das heißt, der Zugriff kann nicht mithilfe von Indexoperationen erfolgen. Mengen werden hauptsächlich verwendet, um zu prüfen, ob bestimmte Werte darin enthalten sind oder nicht, sowie für Mengenoperationen wie Vereinigungs-, Schnitt- oder Teilmenge.

Ein Mengen-Literal wird als kommaseparierte Liste von Werten in geschweiften Klammern geschrieben. Hier ein Beispiel:

```
>>> primes = {2, 3, 5, 7, 11, 13, 17, 19}
```

Diese Menge enthält alle Primzahlen unter 20, sie kann also verwendet werden, um zu überprüfen, ob eine Zahl eine solche Primzahl ist oder nicht. Dazu kommt der bereits besprochene Operator `in` zum Einsatz (beziehungsweise `not in`, um zu prüfen, ob eine Zahl keine Primzahl kleiner als 20 ist):

```
>>> primes = {2, 3, 5, 7, 11, 13, 17, 19}
>>> 11 in primes
True
>>> 15 in primes
False
>>> 9 not in primes
True
>>> 7 not in primes
False
```

Falls Sie bei der Wertzuweisung dasselbe Element mehrfach aufführen, wird es nur einmal in die Menge übernommen:

```
>>> test_set = {1, 1, 2, 2, 3, 3}
>>> test_set
{1, 2, 3}
```

Es wurde bereits erwähnt, dass die Elemente einer Menge ungeordnet sind; das bedeutet, dass die folgende Prüfung für Mengen `True` ergibt:

```
>>> {1, 2} == {2, 1}
True
```

Für geordnete Sammlungen wie Listen oder Tupel gilt das selbstverständlich nicht, wie dieses Beispiel mit Listen zeigt:

```
>>> [1, 2] == [2, 1]
False
```

Beachten Sie, dass leere Mengen nicht als `{}` geschrieben werden dürfen; dies ist der Datentyp Dictionary, der im Anschluss behandelt wird. Die korrekte Schreibweise für leere Mengen ist `set()`. Die eingebaute Funktion `set()` kann auch verwendet werden, um die Elemente anderer Sammlungen in eine Menge zu übernehmen; betrachten Sie dazu das folgende Beispiel mit einem Tupel:

```
>>> tuple1 = (1, 2, 3, 4, 1)
>>> set1 = set(tuple1)
>>> set1
{1, 2, 3, 4}
```

Das im Tupel doppelt vorhandene Element `1` wird wie gehabt aus der Menge entfernt. Mithilfe der Operation `len(`*liste_oder_tupel*`) == len(set(`*liste_oder_tupel*`))` können Sie auf einfache Weise sicherstellen, dass *liste_oder_tupel* keine doppelten Elemente enthält.

Wenn Sie prüfen möchten, ob eine Menge Teilmenge einer anderen ist, wenden Sie den Operator < an; entsprechend wird > für eine Obermenge eingesetzt. Beispiele:

```
>>> set1 = {1, 2, 3, 4}
>>> {1, 2} < set1
True
>>> {5} < set1
False
>>> set1 > {3}
True
>>> set1 > {5}
False
```

Da < und > für echte Teil- beziehungsweise Obermengen verwendet werden, ist eine Menge nicht Teil- oder Obermenge ihrer selbst. Die Operatoren <= und >= ergeben jedoch `True`, denn sie stehen für »Teilmenge oder gleich« beziehungsweise »Obermenge oder gleich«.

Weitere interessante Mengenoperationen sind schließlich Vereinigungs-, Schnitt- und Differenzmenge. Die Vereinigungsmenge wird mithilfe des Operators | gebildet, der bei numerischen Werten für das bitweise Oder steht; sie enthält jedes Element beider ursprünglicher Mengen. Hier ein Beispiel:

```
>>> set1 = {1, 2, 3, 4}
>>> set2 = {4, 5, 6, 7}
>>> set1 | set2
{1, 2, 3, 4, 5, 6, 7}
```

Die Schnittmenge, dargestellt durch den Operator & (das bitweise Und), enthält nur diejenigen Elemente, die in beiden Mengen vorkommen:

```
>>> set_a = {1, 2, 3, 4, 5}
>>> set_b = {4, 5, 6, 7, 8}
>>> set_a & set_b
{4, 5}
```

Die Differenz- oder Restmengenoperation schließlich entfernt aus der linken Menge alle diejenigen Elemente, die auch in der rechten Menge vorkommen; der zuständige Operand ist das Minuszeichen (-):

```
>>> original_set = {1, 2, 3, 4, 5}
>>> removal_set = {3, 4, 6, 7}
>>> original_set - removal_set
{1, 2, 5}
```

Alle drei Operanden können auch in der Wertzuweisungsvariante mit nachgestelltem Gleichheitszeichen verwendet werden, um die linke Menge selbst gemäß der gewünschten Operation zu verändern. Beispiele:

```
>>> test_set = {1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> test_set
{1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> test_set |= {9, 10, 11}             # Vereinigungsmenge
>>> test_set
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
>>> test_set &= {1, 3, 5, 7, 8, 12, 14} # Schnittmenge
>>> test_set
{8, 1, 3, 5, 7}
>>> test_set -= {8, 1, 2}               # Differenzmenge
>>> test_set
{3, 5, 7}
```

Die unveränderliche Variante der Menge heißt `frozenset`. Für diese gibt es keine eigene Literal-Schreibweise; sie wird stets mithilfe des Funktionsaufrufs `frozenset()` gebildet, wobei Sie eine konventionelle Menge, eine Liste oder ein Tupel als Argument verwenden können. Der einzige Unterschied zur normalen Menge besteht darin, dass Sie bei einer unveränderlichen Menge keine Elemente hinzufügen oder entfernen können – dafür sind bei gewöhnlichen Mengen die Methoden `add()` und `remove()` zuständig. Die Modifikation per Vereinigungs-, Schnitt- oder Differenzmengenoperator (`|=`, `&=` beziehungsweise `-=`) funktioniert dagegen auch beim `frozenset`. Hier ein kurzes Beispiel, das zunächst eine Menge erstellt, dann ein `frozenset` mit deren Elementen erzeugt und anschließend versucht, beiden mittels `add()` ein Element hinzuzufügen:

```
>>> set1 = {1, 2, 3}
>>> set2 = frozenset(set1)
>>> set1.add(4)
>>> set1
{1, 2, 3, 4}
>>> set2.add(4)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'frozenset' object has no attribute 'add'
>>> set2
frozenset({1, 2, 3})
```

Für alle bisher behandelten Sammlungsdatentypen gibt es die prädikatenlogischen Funktionen `all()` und `any()`, die `True` sind, wenn alle Elemente der Sammlung `True` sind beziehungsweise wenn mindestens ein Element `True` ist:

```
>>> all(all_true)
True
>>> all(some_true)
False
>>> any(all_true)
True
>>> any(some_true)
True
>>> any(all_false)
False
```

Ein besonderer Sammlungsdatentyp ist das *Dictionary*. Es handelt sich um eine beliebig lange Liste von Schlüssel-Wert-Paaren, wobei die Schlüssel eindeutig sein müssen, die Werte jedoch nicht. In manchen anderen Programmiersprachen wird das Konzept als Hash oder als assoziatives Array bezeichnet.

Das Dictionary-Literal steht wie eine Menge in geschweiften Klammern; Schlüssel und Wert werden durch Doppelpunkte voneinander getrennt und die einzelnen Paare durch Kommata.

Anders als bei Listen muss ein Schlüssel kein numerischer Index sein, sondern kann jeder unveränderliche Datentyp sein. Am häufigsten werden Strings als Schlüssel verwendet. Das folgende Beispiel nutzt die Abkürzungen der Wochentage als Schlüssel und die ausgeschriebenen Varianten als Werte:

```
>>> {"Mo":"Montag", "Di":"Dienstag", "Mi":"Mittwoch",
... "Do":"Donnerstag", "Fr":"Freitag", "Sa":"Samstag",
... "So":"Sonntag"}
{'Mo': 'Montag', 'Di': 'Dienstag', 'Mi': 'Mittwoch', 'Do': 'Donnerstag', 'Fr':
'Freitag', 'Sa': 'Samstag', 'So': 'Sonntag'}
```

Es ist nicht garantiert, dass die Speicherung wie hier in der Reihenfolge stattfindet, die Sie ursprünglich angegeben haben. Bei einem Dictionary geht es eben nicht um eine bestimmte Anordnung, sondern um die Beziehung zwischen den Schlüsseln und Werten.

Um anhand des Schlüssels auf ein Element zuzugreifen, wird der Schlüssel als Index verwendet:

```
>>> weekdays = {"Mo": "Montag", "Di": "Dienstag",
... "Mi": "Mittwoch", "Do": "Donnerstag", "Fr": "Freitag",
... "Sa": "Samstag", "So": "Sonntag"}
>>> weekdays["Sa"]
'Samstag'
```

Wenn Sie versuchen, auf ein nicht vorhandenes Element zuzugreifen, erhalten Sie eine Fehlermeldung wie beispielsweise diese:

```
>>> weekdays["X"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'X'
```

Um den Fehler zu vermeiden, können Sie vor dem Zugriffsversuch über den Index den Operator `in` verwenden, um zu testen, ob ein bestimmter Schlüssel vorhanden ist. Beispiele:

```
>>> "Sa" in weekdays
True
>>> "X" in weekdays
False
```

Mit einem einfachen `in` können Sie allerdings nur das Vorhandensein von Schlüsseln überprüfen; die Suche nach einem bestimmten Wert hat nicht das gewünschte Ergebnis:

```
>>> "Samstag" in weekdays
False
```

Um nach Werten zu suchen, können Sie aber die Methode `values()` eines Dictionarys aufrufen; sie liefert eine spezielle Liste vom Typ `dict_values` zurück, in der Sie wiederum mittels `in` suchen können:

```
>>> "Samstag" in weekdays.values()
True
>>> "Pythontag" in weekdays.values()
False
>>> weekdays.values()
dict_values(['Freitag', 'Mittwoch', 'Sonntag',
'Dienstag', 'Donnerstag', 'Samstag', 'Montag'])
```

Anstatt zuerst mit `in` zu überprüfen, ob ein Schlüssel enthalten ist, können Sie auch die Methode `get()` verwenden, die `None` oder einen optionalen Vorgabewert zurückgibt, wenn der angeforderte Schlüssel nicht existiert:

```
>>> weekdays.get('Y')
>>> weekdays.get('Y') is None
True
>>> weekdays.get('Y', "Das ist kein Wochentag!")
'Das ist kein Wochentag!'
```

Genau wie bei Listen kann der Indexoperator übrigens auch beim Dictionary zum Hinzufügen oder Ändern von Werten verwendet werden. Beispiele:

```
>>> dict = {}                  # leeres Dictionary
>>> dict["banana"] = "yellow"  # Neu-Wertzuweisung
>>> dict
{'banana': 'yellow'}
>>> dict["apple"] = "red"
>>> dict
{'banana': 'yellow', 'apple': 'red'}
>>> dict["apple"] = "green"    # Wertänderung
>>> dict
{'banana': 'yellow', 'apple': 'green'}
```

Strings sind eine Art Mischtyp; ihr Inhalt kann als einzelner Wert oder als Abfolge einzelner Zeichen betrachtet werden. Deshalb können Sie auf Strings auch mit dem Index- oder dem Slice-Operator zugreifen, allerdings nur lesend (denn es handelt sich wie erwähnt um einen unveränderlichen Typ). Hier einige Beispiele:

```
>>> text = "Dies ist ein Test-String."
>>> text[5]
'i'
>>> text[5:8]
'ist'
>>> text[0:10:2]
'De s '
>>> text[-1]
'.'
>>> text[::-1]
'.gnirtS-tseT nie tsi seiD'
```

Die Funktion `range()` stellt keine richtige Liste bereit, sondern einen Wertebereich, auf den Sie jedoch ebenfalls mit dem Indexoperator zugreifen können. Es können ein bis drei Argumente übergeben werden:

- `range(n)` umfasst den Bereich von `0` bis `n-1`.
- `range(m, n)` umfasst den Bereich von `m` bis `n-1`.

- `range(m, n, step)` umfasst den Bereich von `m` bis `n-1` mit der Schrittweite `step` beziehungsweise von `m` bis `n+1` bei negativer Schrittweite.

Hier zwei Beispiele für Ranges mit verschiedenen Zugriffen durch den Index- beziehungsweise den Slice-Operator:

```
>>> range1 = range(1, 11)       # Werte 1-10
>>> range1[0]                   # erstes Element
1
>>> range1[-1]                  # letztes Element
10
>>> range1[2:4]                 # Slice
range(3, 5)
>>> range2 = range(2, 21, 2)    # Werte 2-20, Schrittweite 2
>>> range2[0]                   # erstes Element
2
>>> range2[-1]                  # letztes Element
20
>>> range2[10:2:-4]             # Slice
range(20, 6, -8)
```

Mit `list(range(...))` können Sie alle im Bereich enthaltenen Zahlen in eine Liste übernehmen, wodurch sich die Elemente am einfachsten untersuchen lassen. Wieder ein paar Beispiele:

```
>>> list(range(10))         # 0 bis 10-1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(5, 10))      # 5 bis 10-1
[5, 6, 7, 8, 9]
>>> list(range(2, 20, 2))   # 2 bis 20-1, Schrittweite 2
[2, 4, 6, 8, 10, 12, 14, 16, 18]
>>> list(range(20, 2, -2))  # 20 bis 2+1, Schrittweite -2
[20, 18, 16, 14, 12, 10, 8, 6, 4]
```

Ein praktisches Hilfsmittel für alle Aufzählungstypen ist die Funktion `len()`, die die Anzahl der Elemente zurückgibt. Hier zwei Beispiele, eines für eine Liste und eines für einen String, bei dem entsprechend die Anzahl der Zeichen ermittelt wird:

```
>>> len([1, 2, 3, 4])
4
>>> len("Hello world")
11
```

Mit Kontrollstrukturen arbeiten

Um den Programmablauf zu steuern, besitzt Python wie jede vollwertige Programmiersprache *Kontrollstrukturen*. Diese lassen sich in Fallentscheidungen und Schleifen unterteilen – eine Fallentscheidung verzweigt nur einmal aufgrund des Wahrheitswerts eines Ausdrucks, während eine Schleife unter Umständen mehrmals durchlaufen wird.

Eine *Fallentscheidung* prüft den Wahrheitswert eines Ausdrucks, bevor sie die davon abhängigen Anweisungen ausführt. Die einfachste Fallentscheidung hat folgende Struktur:

```
if Ausdruck:
    Anweisung
    ...
# Normaler Programmablauf geht hier weiter.
```

Wenn der überprüfte Ausdruck `True` ist oder als `True` interpretiert wird, werden die abhängigen Anweisungen ausgeführt, andernfalls geht der Programmablauf direkt in derjenigen Zeile weiter, die wieder die Einrückung der `if`-Anweisung aufweist. Die Zahl `0`, der leere String `""`, leere Aufzählungstypen wie die leere Liste `[]` oder das leere Dictionary `{}` sowie `None` werden wie gesagt als `False` interpretiert, alle anderen Nicht-Boolean-Ausdrücke als `True`.

Wie bereits erwähnt, bestimmt die Einrückung, welche Zeilen von der Fallentscheidung abhängen und welche nicht. Dabei ist es wichtig, dass die Einrückung aller betroffenen Zeilen identisch ist. Empfehlenswert sind vier Leerzeichen pro Stufe; die Hauptsache ist aber, dass Sie sich konsequent an die einmal gewählte Form halten.

Das folgende Beispiel überprüft, ob die Variable `favourite_colour` den Wert `"red"` hat, und gibt in diesem Fall die Meldung `"You may pass!"` aus:

```
>>> favourite_colour = "red"
>>> if favourite_colour == "red":
...     print("You may pass!")
...
You may pass!
```

Da `favourite_colour` zuvor der Wert `"red"` zugewiesen wurde, erfolgt die erwartete Ausgabe aus der abhängigen Codezeile. Wenn Sie dasselbe Beispiel erneut mit einem anderen Wert für `favourite_colour` ausführen, wird nichts ausgegeben:

```
>>> favourite_colour = "blue"
>>> if favourite_colour == "red":
...     print("You may pass!")
...
>>>
```

Besonderheit der interaktiven Shell

Beachten Sie, dass Sie den `if`-Block in der interaktiven Python-Shell mit einer Leerzeile abschließen müssen, sodass er sofort ausgeführt wird. In einem gespeicherten Skript können Sie dagegen mit einer nicht mehr eingerückten Zeile fortfahren, bei der der Programmablauf unabhängig von der Fallentscheidung in jedem Fall weitergeht.

Wie in den meisten Programmiersprachen kann die `if`-Fallentscheidung auch in Python einen `else`-Zweig haben, dessen abhängige Anweisungen ausgeführt werden, wenn der überprüfte Ausdruck nicht `True` ist. Sehen Sie sich dazu dieses erweiterte Beispiel an:

```
>>> favourite_colour = "red"
>>> if favourite_colour == "red":
...     print("You may pass!")
... else:
...     print("You shall not pass!")
...
You may pass!
```

Wie im ersten Beispiel erfolgt die Ausgabe des `if`-Zweigs, aber die Gegenprobe zeigt, dass für eine andere Lieblingsfarbe der `else`-Zweig ausgeführt wird:

```
>>> favourite_colour = "blue"
>>> if favourite_colour == "red":
...     print("You may pass!")
... else:
...     print("You shall not pass!")
...
You shall not pass!
```

Für den Fall, dass das erste `if` nicht zutrifft, können Sie einen weiteren Ausdruck überprüfen; das Schlüsselwort dafür heißt `elif` (Abkürzung für *else if*). Trifft der darin überprüfte Ausdruck zu, werden die zugehörigen Anweisungen ausgeführt, andernfalls geht es bei einem eventuell vorhandenen `else`-Zweig, beim nächsten `elif` oder im normalen Programmablauf weiter. Das folgende Beispiel überprüft eine weitere potenzielle Lieblingsfarbe und gibt einen eigenen Kommentar dazu aus:

```
>>> favourite_colour = "pink"
>>> if favourite_colour == "red":
...     print("You may pass!")
... elif favourite_colour == "pink":
...     print("Close, but not close enough -- you failed!")
... else:
...     print("You shall not pass!")
```

```
...
Close, but not close enough -- you failed!
```

Eine spezielle Schreibweise von `if` und `else` – ohne `elif` – ist der *konditionale Ausdruck* (englisch: *Conditional Expression*), der am ehesten dem aus der Tradition der Programmiersprache C stammenden ternären (dreigliedrigen) Fallentscheidungsoperator `Ausdruck ? Dann-Wert : Sonst-Wert` entspricht. Das Format sieht wie folgt aus:

```
Dann-Wert if Ausdruck else Sonst-Wert
```

Der `else`-Teil darf dabei nicht weggelassen werden. Hier zwei einfache Beispiele, je eines für den Dann-Fall und den Sonst-Fall:

```
>>> a = 5
>>> "Ja, 5" if a == 5 else "Nein, keine 5"
'Ja, 5'
>>> "Ja, 4" if a == 4 else "Nein, keine 4"
'Nein, keine 4'
```

Sie können diese Art von Ausdrücken einer Variablen zuweisen; beachten Sie aber, dass der konditionale Ausdruck in diesem Fall in Klammern stehen muss:

```
>>> monster = "Jabberwocky"
>>> what_to_do = ("Run!" if monster == "Jabberwocky" else "Never mind")
>>> what_to_do
'Run!'
>>> monster = "Killer Rabbit"
>>> what_to_do = ("Run!" if monster == "Jabberwocky" else "Never mind")
>>> what_to_do
'Never mind'
```

Auch in zusammengesetzten Ausdrücken können Sie dieses Format verwenden, und zwar ebenfalls in Klammern:

```
>>> holy_grail = True
>>> "You have found" + (" the Holy Grail" if holy_grail else " nothing special")
'You have found the Holy Grail'
>>> holy_grail = False
>>> "You have found" + (" the Holy Grail" if holy_grail else " nothing special")
'You have found nothing special'
```

Seit der Python-Version 3.10 (zurzeit im öffentlichen Betatest) kommt eine neue Form der Fallentscheidung hinzu: `match/case` ähnelt dem in vielen anderen Programmiersprachen verbreiteten `switch/case`, das später in diesem Kapitel für Java erläutert wird. Es geht darum,

eine Variable mit einer Reihe möglicher Einzelwerte zu vergleichen und je nach Wert unterschiedliche Anweisungen auszuführen.

Hier ein Beispiel, das alle wichtigen Eigenschaften der Anweisungen zeigt – es bestimmt, gemessen an der Note in einer Prüfung, ob diese bestanden wurde:

```
match grade:
    case 1, 2, 3:
        print("Bestanden.")
    case 4:
        print("Gerade noch Glück gehabt.")
    case 5, 6:
        print("Durchgefallen.")
    case _:
        print("Ungültige Note.")
```

Wie Sie sehen, kann ein Fall für einen oder mehrere mögliche Werte gelten. Der besondere Fall _ steht für alle nicht speziell aufgeführten Werte und dient, wie im Beispiel gezeigt, oft dem Abfangen ungültiger Werte.

Eine *Schleife* wiederholt die verschachtelten Anweisungen mehrfach. Die einfachste Schleife ist die `while`-Schleife, deren Syntax folgendermaßen aussieht:

```
while Ausdruck:
    Anweisung
    ...
# normaler Programmablauf
```

Die verschachtelten Anweisungen werden so lange immer wieder ausgeführt, wie der überprüfte Ausdruck `True` ergibt. Hier ein interaktives Beispiel, das mittels `input()` Benutzereingaben entgegennimmt und dies wiederholt, bis die gewünschte Eingabe erfolgt ist:

```
>>> passwort = "geheim"
>>> eingabe = ""
>>> while eingabe != passwort:
...     eingabe = input("Passwort bitte: ")
...
Passwort bitte: pass
Passwort bitte: test
Passwort bitte: geheim
>>>
```

Natürlich können Sie die `while`-Schleife beispielsweise auch zum Zählen verwenden, wenn Sie die Zählervariable innerhalb des Schleifenrumpfs, also in den abhängigen Anweisungen, manipulieren (allerdings gibt es wesentlich elegantere Konstrukte zum Zählen, wie Sie bald sehen werden). Das folgende Beispiel listet die Zahlen von 0 bis 4 auf:

```
>>> i = 0
>>> while i < 5:
...     print(i)
...     i += 1
...
0
1
2
3
4
```

»else«-Blöcke in »while«-Schleifen

Eine Python-Besonderheit ist die Tatsache, dass `while`-Schleifen einen `else`-Block haben können. Dieser wird ausgeführt, falls der geprüfte Ausdruck von Anfang an `False` war, sodass der Schleifenrumpf niemals ausgeführt wird. Beispiel:

```
>>> a = 10
>>> while a < 10:
...     print(a)
...     a += 1
... else:
...     print("a war nie kleiner als 10.")
...
a war nie kleiner als 10.
```

Die `for`-Schleife wird in Python verwendet, um über die Elemente eines Objekts zu *iterieren*, was bedeutet, diese Elemente der Reihe nach durchzugehen. Deshalb wird `for` auch als *Iterator* bezeichnet. Die Syntax sieht wie folgt aus:

```
for Variable in Aufzählung:
    Anweisung  # Variable hat nacheinander Werte der Elemente in Aufzählung
    ...
# normaler Programmablauf
```

Betrachten Sie als erstes Beispiel die Iteration über die folgende Liste (es handelt sich um die römisch-dekadenten Snacks, die Brian in Monty Pythons »Life of Brian« im Amphitheater verkauft):

```
>>> snacks = [
... "Wolf Nipple Chips",
... "Dromedary Pretzels",
... "Jaguar Ear Lobes",
... "Tuscany Fried Bat",
```

```
... "Otter Noses"]
>>> for snack in snacks:
...     print("Get your " + snack + " while supplies last!")
...
Get your Wolf Nipple Chips while supplies last!
Get your Dromedary Pretzels while supplies last!
Get your Jaguar Ear Lobes while supplies last!
Get your Tuscany Fried Bat while supplies last!
Get your Otter Noses while supplies last!
```

Mit einer `for`-Schleife können Sie über jeden Aufzählungstyp iterieren, außerdem über einige andere Typen – beispielsweise die einzelnen Zeichen eines Strings:

```
>>> for letter in "letters":
...     print("Das Wort enthält den Buchstaben " + letter)
...
Das Wort enthält den Buchstaben l
Das Wort enthält den Buchstaben e
Das Wort enthält den Buchstaben t
Das Wort enthält den Buchstaben t
Das Wort enthält den Buchstaben e
Das Wort enthält den Buchstaben r
Das Wort enthält den Buchstaben s
```

Bei einem Dictionary wird über die Schlüssel iteriert; der Zugriff auf die Werte erfolgt wie üblich durch den Indexoperator. Beispiel:

```
>>> countries = {"de": "Deutschland", "at": "Österreich", "fr": "Frankreich"}
>>> for tld in countries:
...     print(countries[tld] + " hat die Top-Level-Domain " + tld)
...
Frankreich hat die Top-Level-Domain fr
Österreich hat die Top-Level-Domain at
Deutschland hat die Top-Level-Domain de
```

Die `for`-Schleife kann auch innerhalb eines Ausdrucks in eckigen Klammern verwendet werden, um eine neue Liste zu erzeugen; ein solches Konstrukt wird als *List-Comprehension* bezeichnet. Dies ist die Syntax:

```
neue_liste = [ausdruck for element in original_liste]
```

Die Liste *neue_liste* wird mit den Elementen befüllt, die aus der Iteration über *original_liste* stammen. Hier ein Beispiel, das die Quadrate der Zahlen von 1 bis 10 berechnet und sie in einer neuen Liste speichert:

```
>>> numbers = list(range(1, 11))
>>> squares = [number ** 2 for number in numbers]
>>> squares
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Die Konstruktion kann auch zusammen mit `if` verwendet werden, um Listen nach bestimmten Kriterien zu filtern. Das Ganze sieht schematisch so aus:

```
gefilterte_liste = [ausdruck for element in original_liste if ausdruck]
```

Die neue Liste *gefilterte_liste* enthält danach den berechneten Ausdruck für alle diejenigen Elemente von *original_liste*, auf die der überprüfte Ausdruck zutrifft. Das folgende Beispiel filtert nur die geraden Zahlen aus einer fortlaufenden Liste natürlicher Zahlen heraus:

```
>>> numbers = list(range(1, 11))
>>> even = [number for number in numbers if number % 2 == 0]
>>> even
[2, 4, 6, 8, 10]
```

Für alle Schleifentypen gibt es übrigens die manchmal nützliche Möglichkeit, den aktuellen Durchlauf oder die gesamte Schleife vorzeitig zu beenden. Die Anweisung `continue` springt sofort zum (eventuellen) nächsten Durchlauf, wie das folgende Beispiel zeigt:

```
>>> for i in range(1, 11):
...     # Nicht durch 2 teilbar? Weiter
...     if i % 2 != 0:
...         continue
...     print(i)
...
2
4
6
8
10
```

Mit `break` wird dagegen die Ausführung der gesamten Schleife abgebrochen, was etwa die folgende (mathematisch primitive) Primzahlprüfung zeigt:

```
>>> test_number = 35
>>> for i in range(2, test_number):
...     if test_number % i == 0:
...         print("Keine Primzahl")
...         break
...
Keine Primzahl
```

Funktionen und Methoden verwenden

Ein weiteres Hilfsmittel zum Bilden von Ausdrücken und Anweisungen sind die vereinzelt bereits gezeigten aufrufbaren Funktionen und Methoden. Als Beispiel werden hier einige beschrieben, die der Ein- und Ausgabe dienen.

Eine *Funktion* steht allein, also nicht auf ein Objekt bezogen. Die Funktion `print()` dient beispielsweise dazu, Text auf der Konsole auszugeben:

```
>>> print("Hallo Python")
Hallo Python
```

Eine *Methode* ist dagegen eine Funktion innerhalb eines Objekts, die in dessen Klasse definiert ist. Sie wird durch einen Punkt getrennt hinter das Objekt geschrieben. Ein Beispiel ist die Methode `append()` einer Liste; sie dient dazu, Elemente anzuhängen, zum Beispiel:

```
>>> list1 = [1, 2, 3, 4, 5]
>>> list1
[1, 2, 3, 4, 5]
>>> list1.append(6)
>>> list1
[1, 2, 3, 4, 5, 6]
```

Text auf der Konsole ein- und ausgeben

Zum Schreiben von Programmen werden stets Funktionen zur Ein- und Ausgabe benötigt. Soweit die Konsole betroffen ist, sind diese im Sprachkern von Python enthalten (einige Dateizugriffsfunktionen sind ebenfalls eingebaute Funktionen, aber nicht alle). Es handelt sich im Wesentlichen um die Funktionen `print()` zur Ausgabe und `input()` zur Eingabe.

`print()` nimmt im einfachsten Fall einen String oder ein als String interpretierbares Objekt entgegen und gibt dessen Wert, gefolgt von einem Zeilenumbruch, auf der Konsole aus:

```
>>> print("Hello Python")
Hello Python
>>> print(4 + 3)
7
>>> print(1 == 2)
False
```

Sie können beliebig viele durch Kommata getrennte Ausdrücke ausgeben lassen, wobei diese standardmäßig durch ein Leerzeichen und nicht durch einen Zeilenumbruch voneinander getrennt werden. Beispiele:

```
>>> print("Hello", "World,", "hello", "Python!")
Hello World, hello Python!
>>> print(3, "+", 4, "=", 3 + 4)
3 + 4 = 7
```

Sie können das Leerzeichen zwischen den einzelnen durch Kommata getrennten Argumenten sowie den Zeilenumbruch am Ende durch andere Zeichen oder Zeichenfolgen ersetzen. Dazu werden *benannte Argumente* eingesetzt, eine Besonderheit von Python: Nach einer definierten oder auch beliebigen Anzahl nicht benannter Argumente können solche in der Form `Schlüsselwort = Wert` folgen, die meistens für Konfigurationsaufgaben verwendet werden.

Zwei der möglichen benannten Argumente für `print()` sind `sep` (kurz für Separator) als Trenn-String zwischen den einzelnen Standardargumenten und `end` für den String am Ende. Das folgende Beispiel verwendet ein Komma und ein Leerzeichen als Separator und einen Punkt, gefolgt von einem Zeilenumbruch, als Endmarkierung:

```
>>> print("Arthur", "Galahad", "Lancelot", "Robin", sep = ", ", end = ".\n")
Arthur, Galahad, Lancelot, Robin.
```

Für Benutzereingaben kommt die Funktion `input()` zum Einsatz. Sie wartet, bis zur Laufzeit des Programms eine Eingabe vorgenommen und diese mit ↵ abgeschlossen wurde. Anschließend gibt sie den Inhalt der Benutzereingabe (ohne den abschließenden Zeilenumbruch) als Wert zurück. Dadurch kann dieser Wert beispielsweise einer Variablen zugewiesen werden. Das folgende Beispiel nimmt eine Eingabe entgegen und speichert sie in der Variablen `eingabe`:

```
>>> eingabe = input()
hallo
>>> eingabe
'hallo'
```

Optional können Sie der Funktion ein Argument übergeben, dessen String-Wert als Eingabeaufforderung ausgegeben wird:

```
>>> name = input("Ihr Name? ")
Ihr Name? Sascha
>>> name
'Sascha'
```

Mit Dateien arbeiten

Für den Zugriff auf Dateien werden diese zunächst mit `open()` geöffnet; das Ergebnis ist im Erfolgsfall ein Objekt, dessen Methoden etwa zum Lesen und Schreiben von Daten in der

Datei verwendet werden. Die wichtigsten Parameter von `open()` sind der Dateiname als erstes, unbenanntes Argument sowie der Modus (Lesen, Schreiben etc.) als zweites Argument oder – empfehlenswerter – als benanntes Argument `mode`.

Das folgende Beispiel öffnet die Datei *test.txt* im aktuellen Verzeichnis zum Lesen (`mode = "r"`; dies ist eigentlich Standard und könnte daher weggelassen werden) und weist das erhaltene Objekt der Variablen `file` zu; *test.txt* muss dazu natürlich existieren:

```
>>> file = open("test.txt", mode = "r")
```

Um den gesamten Inhalt der Datei auszulesen, wird die Methode `read()` des Objekts `file` aufgerufen:

```
>>> content = file.read()
>>> content
'Dies ist eine Textdatei\nSie hat drei Zeilen\nDie letzte endet nicht mit
einem Zeilenumbruch\n'
```

Wie Sie sehen, fügt `read()` am Ende des ausgelesenen Textes einen Zeilenumbruch ein, obwohl die Originaldatei in der letzten Zeile keinen enthält, wie der Inhalt vermuten lässt.

Wenn Sie ein zweites Mal `read()` für dieselbe Datei aufrufen, erhalten Sie einen leeren String als Ergebnis:

```
>>> file.read()
''
```

Das liegt daran, dass intern ein *Dateizeiger* (englisch: *File Cursor*) verwendet wird, der die aktuelle Position markiert. Mithilfe der Methode `seek(Position)` können Sie den Zeiger auf eine andere Position setzen. Hier ein Beispiel, das zum dritten Zeichen der Datei (Position 2) springt und von dort erneut liest:

```
>>> file.seek(2)
2
>>> file.read()
'es ist eine Textdatei\nSie hat drei Zeilen\nDie letzte endet nicht mit
einem Zeilenumbruch\n'
```

Das File-Objekt stellt (zumindest für Textdateien) auch einen Iterator bereit, mit dem Sie die Datei zeilenweise auslesen können. Das folgende Beispiel kehrt zum Anfang der Datei zurück, liest sie in einer `for`-Schleife zeilenweise ein und gibt die Zeilen aus; die String-Methode `strip()` entfernt dabei den jeweiligen Zeilenumbruch am Ende:

```
>>> for line in file:
...     print(line.strip())
...
```

```
Dies ist eine Textdatei
Sie hat drei Zeilen
Die letzte endet nicht mit einem Zeilenumbruch
```

Neben `mode = "r"` gibt es diverse weitere Modi, um die Datei zum Lesen, zum Schreiben oder für diverse Mischungen daraus mit verschiedenen Optionen zu öffnen. Mit `mode = "w"` (*write*) öffnen Sie die Datei nur zum Schreiben; falls sie existiert, wird sie überschrieben, ansonsten neu angelegt. Auch `mode = "a"` (*append*) öffnet die Datei zum Schreiben und legt sie gegebenenfalls neu an; falls sie jedoch existiert, wird der Dateizeiger an ihr Ende gesetzt, sodass Sie weiteren Inhalt hinzufügen können.

Um in eine Datei zu schreiben, wird die Methode `write()` des File-Objekts verwendet. In diesem Fall müssen Sie sich selbst um die abschließenden Zeilenumbrüche kümmern, falls Sie welche benötigen. Alternativ können Sie die Funktion `print()` verwenden, wenn Sie darin den benannten Parameter `file` auf Ihr Dateiobjekt setzen.

Das folgende Beispiel öffnet die Datei *ausgabe.txt* zum Schreiben und schreibt mit beiden Verfahren je eine Zeile hinein. Anschließend wird die Datei mit `close()` geschlossen, dann erneut zum Lesen geöffnet, und ihr Inhalt wird eingelesen:

```
>>> file = open("ausgabe.txt", "w")
>>> file.write("Erste Zeile, mit write() geschrieben.\n")
38
>>> print("Zweite Zeile, mit print() geschrieben.", file = file)
>>> file.close()
>>> file = open("ausgabe.txt", "r")
>>> file.read()
'Erste Zeile, mit write() geschrieben.\nZweite Zeile, mit print() geschrieben.\n'
```

Die speziellen Modi `"r+"`, `"w+"` und `"a+"` öffnen eine Datei zum kombinierten Lesen und Schreiben. `"r+"` führt zu einer Fehlermeldung, falls die Datei nicht existiert, während `"w+"` und `"a+"` sie in diesem Fall neu anlegen. `"w+"` ersetzt eine eventuell vorhandene Datei; `"r+"` und `"a+"` lassen sie bestehen. `"a+"` schreibt stets ans Ende der vorhandenen Datei – unabhängig von der Position des Dateizeigers, an der gelesen wird. `"r+"` und `"w+"` setzen den Dateizeiger dagegen zunächst an den Anfang und beachten dessen Position sowohl beim Lesen als auch beim Schreiben.

Hier ein Beispiel, das `"w+"` verwendet und in einer Datei schreibt, liest und den Zeiger positioniert, um die (durch ihre Position gekennzeichneten) ungeraden Ziffern durch Leerzeichen zu ersetzen:

```
>>> file = open("even.txt", "w+")
>>> file.write("01234567890")
11
```

```
>>> file.seek(0)
0
>>> file.read()
'01234567890'
>>> for i in [1, 3, 5, 7, 9]:
...     file.seek(i)
...     file.write(" ")
...
1
1
3
1
5
1
7
1
9
1
>>> file.seek(0)
0
>>> file.read()
'0 2 4 6 8 0'
```

Die vielen Ausgabezeilen der Schleife sind die jeweiligen Positionen von `seek()` und die Anzahl der von jedem `write()` geschriebenen Zeichen (1). Es handelt sich nicht um Ausgaben, die in einem regulären Python-Skript auf der Konsole erscheinen würden, sondern um die übliche Anzeige von Funktionsrückgabewerten innerhalb der interaktiven Shell. In eigenen Skripten können Sie diese Rückgabewerte auslesen, um zu überprüfen, ob Ihre Dateioperationen ordnungsgemäß funktioniert haben.

Strings formatieren

Besonders interessant für die Ausgabe ist die String-Formatierungsmethode `format()` der String-Klasse (interner Name `str`). Die Methode ähnelt der C- und PHP-Funktion `sprintf()`, ist jedoch noch praktischer und vielseitiger als diese. Das allgemeine Format lautet:

```
Format-String.format(Wert1, Wert2, ...)
```

Der Format-String enthält Platzhalter, die durch die Werte ersetzt werden. Der einfachste Platzhalter ist {}; er wird gemäß seiner Position im String durch das entsprechende Argument in seiner Standard-String-Darstellung ersetzt. Hier ein einfaches Beispiel mit zwei derartigen Platzhaltern:

Auf einen Blick

1 Einführung 27
2 Mathematische Grundlagen 65
3 Elektronische und technische Grundlagen 141
4 Hardware 171
5 Netzwerkgrundlagen 235
6 Betriebssysteme 341
7 Grundlagen der Programmierung 433
8 Algorithmen und Datenstrukturen 573
9 Weitere Konzepte der Programmierung 645
10 Datenanalyse, Machine Learning, künstliche Intelligenz 693
11 Software-Engineering 733
12 Geschäftsprozessanalyse 787
13 Datenbanken 805
14 Server für Webanwendungen 865
15 Weitere Internet-Serverdienste 921
16 XML 945
17 Weitere Datei- und Datenformate 993
18 Webseitenerstellung mit HTML und CSS 1031
19 Webserveranwendungen 1103
20 JavaScript und Ajax 1203
21 Computer- und Netzwerksicherheit 1293

Inhalt

Materialien zum Buch 17
Vorwort 19

1 Einführung 27

1.1 Informationstechnik, Informatik und EDV 27
1.1.1 Fachrichtungen der Informatik 28
1.1.2 Überblick über die IT-Ausbildung 29

1.2 Die Geschichte der Rechenmaschinen und Computer 37
1.2.1 Die Vorgeschichte 39
1.2.2 Die Entwicklung der elektronischen Rechner 41
1.2.3 Die Entwicklung der Programmiersprachen 50

1.3 Digitale Speicherung und Verarbeitung von Informationen 57
1.3.1 Digitale Bilddaten 58
1.3.2 Digitale Audiodaten 59
1.3.3 Digitale Speicherung von Text 60

1.4 Übungsaufgaben 61

2 Mathematische Grundlagen 65

2.1 Einführung in die Logik 65
2.1.1 Aussagen 66
2.1.2 Aussageformen 67
2.1.3 Logische Verknüpfungen 67

2.2 Mengenlehre und diskrete Mathematik 78
2.2.1 Mengenoperationen 78
2.2.2 Abbildungen 84
2.2.3 Folgen und Reihen 88
2.2.4 Beweise 90

2.3 Mathematische Verfahren im Alltag 93
2.3.1 Dreisatz 93
2.3.2 Lösen von Gleichungen und Gleichungssystemen 94

2.4 Grundlagen der Stochastik ... 97
2.4.1 Wahrscheinlichkeitsrechnung ... 97
2.4.2 Statistik ... 100
2.5 Grundlagen der linearen Algebra ... 104
2.5.1 Vektoren ... 106
2.5.2 Matrizen ... 110
2.6 Grundlagen der Analysis ... 118
2.6.1 Arten von Funktionen ... 119
2.6.2 Nullstellen und Ableitungen ... 121
2.7 Informationsspeicherung im Computer ... 123
2.7.1 Zahlensysteme ... 124
2.7.2 Bits und Bytes ... 129
2.8 Übungsaufgaben ... 134
2.8.1 Praktische Übungen ... 134
2.8.2 Kontrollfragen ... 136

3 Elektronische und technische Grundlagen 141

3.1 Elektronische Grundlagen ... 141
3.1.1 Einfache Schaltungen ... 142
3.1.2 Zusammengesetzte Schaltungen ... 145
3.2 Automatentheorien und -simulationen ... 149
3.2.1 Algorithmen ... 149
3.2.2 Die Turingmaschine ... 157
3.2.3 Der virtuelle Prozessor ... 162
3.3 Übungsaufgaben ... 168
3.3.1 Praktische Übungen ... 168
3.3.2 Kontrollfragen ... 168

4 Hardware 171

4.1 Grundlagen ... 171
4.2 Die Zentraleinheit ... 175
4.2.1 Aufbau und Aufgaben des Prozessors ... 177
4.2.2 Der Arbeitsspeicher ... 187

4.2.3 Das BIOS ... 189
4.2.4 Bus- und Anschlusssysteme ... 193

4.3 Die Peripherie ... 203
4.3.1 Massenspeicher ... 204
4.3.2 Eingabegeräte ... 218
4.3.3 Ausgabegeräte ... 221
4.3.4 Soundhardware ... 227

4.4 Übungsaufgaben ... 228

5 Netzwerkgrundlagen 235

5.1 Einführung ... 235
5.1.1 Was ist ein Netzwerk? ... 235
5.1.2 Entstehung der Netzwerke ... 237

5.2 Funktionsebenen von Netzwerken ... 243
5.2.1 Das OSI-Referenzmodell ... 243
5.2.2 Das Schichtenmodell der Internetprotokolle ... 246
5.2.3 Netzwerkkommunikation über die Schichten eines Schichtenmodells ... 248

5.3 Klassifizierung von Netzwerken ... 252
5.3.1 Die Reichweite des Netzwerks ... 252
5.3.2 Die Netzwerktopologie ... 254
5.3.3 Der Zentralisierungsgrad des Netzwerks ... 255

5.4 Netzwerkkarten, Netzwerkkabel und Netzzugangsverfahren ... 261
5.4.1 Die verschiedenen Ethernet-Standards ... 262
5.4.2 Drahtlose Netze ... 267

5.5 Datenfernübertragung ... 271
5.5.1 DSL-Dienste ... 273
5.5.2 Internetzugänge über Mobilfunk ... 274

5.6 Die TCP/IP-Protokollfamilie ... 275
5.6.1 Netzzugang in TCP/IP-Netzwerken ... 277
5.6.2 IP-Adressen, Datagramme und Routing ... 278
5.6.3 Transportprotokolle ... 305
5.6.4 Das Domain Name System (DNS) ... 311
5.6.5 Verschiedene Internetanwendungsprotokolle ... 315

5.7 Übungsaufgaben ... 327

6 Betriebssysteme 341

6.1 Entwicklung der Betriebssysteme 342
6.1.1 Die Vorgeschichte 342
6.1.2 Die Geschichte von Unix 344
6.1.3 PC-Betriebssysteme 346
6.2 Aufgaben und Konzepte 350
6.2.1 Allgemeiner Aufbau von Betriebssystemen 351
6.2.2 Prozessverwaltung 358
6.2.3 Speicherverwaltung 363
6.2.4 Dateisysteme 365
6.3 Windows 371
6.3.1 Allgemeine Informationen 371
6.3.2 Windows im Einsatz 378
6.3.3 Die Windows-Eingabeaufforderung 378
6.3.4 Die Windows PowerShell 381
6.3.5 Windows-Server 392
6.4 Linux und Unix 394
6.4.1 Arbeiten mit der Shell 396
6.4.2 Die wichtigsten Systembefehle 408
6.4.3 Automatisierung 417
6.5 Übungsaufgaben 424

7 Grundlagen der Programmierung 433

7.1 Python 435
7.1.1 Das erste Beispiel 437
7.1.2 Grundelemente von Python 438
7.1.3 Objektorientierung in Python 486
7.1.4 Die Python-Standardbibliothek einsetzen 516
7.2 Java 524
7.2.1 Einführungsbeispiel 525
7.2.2 Wichtige Merkmale von Java 528
7.2.3 Objektorientierte Programmierung mit Java 545
7.2.4 Weitere Java-Elemente 556
7.3 Übungsaufgaben 569

8 Algorithmen und Datenstrukturen 573

8.1 Algorithmen erarbeiten und implementieren 574
8.1.1 Einen Algorithmus planen 574
8.1.2 Den Algorithmus implementieren 577
8.1.3 Ein effizienterer GGT-Algorithmus 579

8.2 Datensammlungen sortieren 581
8.2.1 Bubblesort implementieren 581
8.2.2 Quicksort einsetzen 586

8.3 Nach Daten suchen 588
8.3.1 In Listen suchen 588
8.3.2 Nicht sequenzielle Datenstrukturen durchsuchen 590

8.4 Bäume und Graphen 610
8.4.1 Bäume verwenden 610
8.4.2 Graphen verwenden 621

8.5 Bedingungserfüllungsprobleme 632
8.5.1 Den Algorithmus für Bedingungserfüllungsprobleme implementieren 633
8.5.2 Anwendungsbeispiel: Ein Sudoku lösen 636

8.6 Übungsaufgaben 643

9 Weitere Konzepte der Programmierung 645

9.1 Reguläre Ausdrücke 645
9.1.1 Muster für reguläre Ausdrücke schreiben 647
9.1.2 Programmierung mit regulären Ausdrücken 650

9.2 Systemnahe Programmierung 663
9.2.1 Prozesse und Pipes 663
9.2.2 Threads 667

9.3 Einführung in die Netzwerkprogrammierung 671
9.3.1 Die Berkeley Socket API 671
9.3.2 Ein praktisches Beispiel 677

9.4 Externe Module und Abhängigkeiten 680
9.4.1 Externe Python-Module installieren 680
9.4.2 NumPy verwenden 681
9.4.3 Das Java-Build-Tool Maven 687

9.5 Übungsaufgaben 690

10 Datenanalyse, Machine Learning, künstliche Intelligenz 693

10.1 Einführung 694
10.1.1 Was ist künstliche Intelligenz? 694
10.1.2 Machine Learning im Überblick 698

10.2 Daten auswählen und aufbereiten 699
10.2.1 Textdaten aufbereiten 700
10.2.2 Bilddaten vorbereiten 705
10.2.3 Numerische Daten visualisieren 708

10.3 Konkrete Machine-Learning-Verfahren 716
10.3.1 Lineare Regression 717
10.3.2 Logistische Regression 723
10.3.3 K-Means-Clustering 725
10.3.4 Künstliche neuronale Netzwerke 727

10.4 Übungsaufgaben 730

11 Software-Engineering 733

11.1 Überblick 734
11.1.1 Der Entwicklungszyklus 735
11.1.2 Planung und Analyse 736
11.1.3 Entwurf 742
11.1.4 Implementierung und Test 743
11.1.5 Dokumentation 745
11.1.6 Konkrete Entwicklungsverfahren 746

11.2 Werkzeuge 750
11.2.1 UML 750
11.2.2 Entwurfsmuster 757
11.2.3 Unit-Tests 772
11.2.4 Weitere nützliche Software 779

11.3 Übungsaufgaben 784

12 Geschäftsprozessanalyse 787

12.1 Überblick ... 787
12.1.1 Historische Entwicklung ... 788
12.1.2 Geschäftsprozesse ... 790
12.1.3 Einteilung der Aufgabenbereiche im Prozessmanagement ... 791
12.2 Prozesse modellieren mit BPMN ... 793
12.2.1 BPMN 2.0 im Überblick ... 794
12.2.2 Beispiele für BPMN-Diagramme ... 798
12.3 Übungsaufgaben ... 803

13 Datenbanken 805

13.1 Die verschiedenen Datenbanktypen ... 806
13.1.1 Einzeltabellendatenbanken ... 808
13.1.2 Relationale Datenbanken ... 809
13.1.3 Objektorientierte Datenbanken ... 817
13.2 MySQL – ein konkretes RDBMS ... 820
13.2.1 MySQL installieren und konfigurieren ... 821
13.2.2 Erste Schritte mit dem »mysql«-Client ... 824
13.3 SQL-Abfragen ... 825
13.3.1 Datenbanken und Tabellen erzeugen ... 826
13.3.2 Auswahlabfragen ... 831
13.3.3 Einfüge-, Lösch- und Änderungsabfragen ... 835
13.3.4 Transaktionen ... 837
13.4 MySQL-Administration ... 838
13.4.1 »mysqladmin« ... 838
13.4.2 Zugangsverwaltung ... 839
13.4.3 Import und Export von Daten, Backups ... 844
13.4.4 Konfigurationsdateien ... 847
13.4.5 Log-Dateien ... 848
13.4.6 Replikation ... 849
13.5 Grundlagen der Datenbankprogrammierung ... 851
13.6 CouchDB im Überblick ... 856
13.6.1 Das Konzept von CouchDB ... 857
13.6.2 Praktischer Einstieg in CouchDB ... 857

13.7 Übungsaufgaben ... 860
13.7.1 Praktische Übungen ... 860
13.7.2 Kontrollfragen ... 861

14 Server für Webanwendungen 865

14.1 HTTP im Überblick ... 865
14.1.1 Ablauf der HTTP-Kommunikation ... 866
14.1.2 HTTP-Statuscodes ... 870
14.1.3 HTTP-Header ... 874
14.2 Der Webserver Apache ... 879
14.2.1 Apache im Überblick ... 880
14.2.2 Apache-Module ... 881
14.2.3 Apache installieren ... 883
14.2.4 Apache konfigurieren ... 886
14.2.5 Andere Webserver im Überblick ... 898
14.3 PHP installieren und einrichten ... 899
14.3.1 PHP installieren ... 899
14.3.2 Die PHP-Konfigurationsdatei »php.ini« ... 903
14.4 Virtualisierung und Container ... 906
14.4.1 Virtualisierungslösungen im Überblick ... 907
14.4.2 VirtualBox als konkretes Beispiel ... 908
14.4.3 Container-Virtualisierung mit Docker ... 912
14.4.4 Cloud Computing ... 915
14.5 Übungsaufgaben ... 917
14.5.1 Praktische Übungen ... 917
14.5.2 Kontrollfragen ... 918

15 Weitere Internet-Serverdienste 921

15.1 Namens- und Verzeichnisdienste ... 921
15.1.1 Der DNS-Server BIND ... 921
15.1.2 Der Verzeichnisdienst OpenLDAP ... 928

15.2 Die »Meta-Server« inetd und xinetd ... 938
15.2.1 »inetd« ... 938
15.2.2 »xinetd« ... 939
15.3 Übungsaufgaben ... 941

16 XML 945

16.1 Der Aufbau von XML-Dokumenten ... 947
16.1.1 Die grundlegenden Bestandteile von XML-Dokumenten ... 948
16.1.2 Wohlgeformtheit ... 955
16.2 DTDs und XML Schema ... 958
16.2.1 Document Type Definitions (DTDs) ... 958
16.2.2 Namensräume ... 969
16.2.3 XML Schema ... 971
16.3 XSLT ... 974
16.3.1 Ein einfaches Beispiel ... 976
16.3.2 Wichtige XSLT- und XPath-Elemente ... 978
16.4 Grundlagen der XML-Programmierung ... 981
16.4.1 XML-Verarbeitungsmethoden im Überblick ... 982
16.4.2 Das Python-Modul »xml.etree« ... 984
16.5 Übungsaufgaben ... 987
16.5.1 Praktische Übungen ... 987
16.5.2 Kontrollfragen ... 988

17 Weitere Datei- und Datenformate 993

17.1 Textdateien und Zeichensätze ... 993
17.1.1 Das Problem des Zeilenumbruchs ... 994
17.1.2 Zeichensätze ... 996
17.1.3 Textbasierte Dateiformate ... 1003
17.2 Binäre Dateiformate ... 1015
17.2.1 Datenkomprimierung ... 1017
17.2.2 Bilddateiformate ... 1018
17.2.3 Multimedia-Dateiformate ... 1022
17.2.4 Archivdateien verwenden ... 1024
17.3 Übungsaufgaben ... 1027

18 Webseitenerstellung mit HTML und CSS 1031

18.1 HTML und XHTML 1032
18.1.1 Die Grundstruktur von HTML-Dokumenten 1033
18.1.2 Textstrukturierung und Textformatierung 1035
18.1.3 Listen und Aufzählungen 1043
18.1.4 Hyperlinks 1046
18.1.5 Bilder in Webseiten einbetten 1051
18.1.6 Tabellen 1055
18.1.7 Formulare 1062
18.1.8 Einbetten von Multimedia-Dateien 1070
18.1.9 Metatags und Suchmaschinen 1071
18.2 Cascading Style Sheets (CSS) 1074
18.2.1 Stylesheets platzieren 1075
18.2.2 Stylesheet-Wertangaben 1078
18.2.3 Stylesheet-Eigenschaften 1080
18.2.4 Layer erzeugen und positionieren 1085
18.3 Übungsaufgaben 1094

19 Webserveranwendungen 1103

19.1 PHP 1103
19.1.1 Sprachgrundlagen 1104
19.1.2 Klassen und Objekte 1122
19.1.3 Include-Dateien, Autoloader und Namespaces 1141
19.1.4 Webspezifische Funktionen 1143
19.1.5 Auf MySQL-Datenbanken zugreifen 1149
19.1.6 Unit-Tests mit PHPUnit 1161
19.1.7 PHP als Kommandozeilensprache verwenden 1170
19.2 Eine REST-API implementieren 1171
19.2.1 Die API im Überblick 1172
19.2.2 Die Grundarchitektur der API 1175
19.2.3 Der komplette Quellcode 1177
19.2.4 Die API testen 1199
19.3 Übungsaufgaben 1200

20 JavaScript und Ajax 1203

20.1 Grundlagen 1204
20.1.1 JavaScript im HTML-Dokument 1204
20.1.2 Fehler suchen mit der JavaScript-Konsole 1207
20.1.3 Ausdrücke und Operationen 1208
20.1.4 Funktionen 1213
20.1.5 Objektorientiertes JavaScript 1217
20.1.6 Formulare und Event-Handler 1220
20.1.7 Datum und Uhrzeit verwenden 1232
20.1.8 Bilder manipulieren 1235
20.1.9 Browser- und Fensteroptionen 1238

20.2 Das Document Object Model (DOM) 1244
20.2.1 W3C-DOM im Überblick 1245
20.2.2 Eine DOM-Baum-Anzeige 1248
20.2.3 DOM in der Praxis anwenden 1251
20.2.4 Dokumentinhalte verändern und austauschen 1253
20.2.5 »data«-Attribute verwenden 1256

20.3 Ajax 1257
20.3.1 Die erste Ajax-Anwendung 1258
20.3.2 Datenaustauschformate: XML und JSON 1264

20.4 Die JavaScript-Bibliothek React.js 1265
20.4.1 Einführungsbeispiel 1265
20.4.2 Eigene React-Child-Komponenten definieren 1271
20.4.3 Einen API-Client mit React schreiben 1275
20.4.4 Der REST-Client im Detail 1278

20.5 Übungsaufgaben 1290

21 Computer- und Netzwerksicherheit 1293

21.1 PC-Gefahren 1294
21.1.1 Viren und Würmer 1294
21.1.2 Trojaner und Backdoors 1300
21.1.3 Weitere Schädlinge 1301

21.2 Netzwerk- und Serversicherheit ... 1307
21.2.1 Servergefahren ... 1307
21.2.2 Wichtige Gegenmaßnahmen ... 1309
21.2.3 Kryptografie ... 1315
21.3 Übungsaufgaben ... 1318

Anhang 1321

A Glossar ... 1321
B Zweisprachige Wortliste ... 1333
B.1 Englisch – Deutsch ... 1333
B.2 Deutsch – Englisch ... 1338
C Kommentiertes Literatur- und Linkverzeichnis ... 1345
C.1 Allgemeine Einführungen und Überblicke ... 1345
C.2 Mathematische Grundlagen ... 1346
C.3 Elektronische und technische Grundlagen ... 1347
C.4 Hardware ... 1348
C.5 Netzwerkgrundlagen ... 1348
C.6 Betriebssysteme ... 1348
C.7 Grundlagen der Programmierung ... 1350
C.8 Algorithmen und Datenstrukturen ... 1350
C.9 Weitere Konzepte der Programmierung ... 1351
C.10 Datenanalyse, Machine Learning, künstliche Intelligenz ... 1351
C.11 Software-Engineering ... 1353
C.12 Geschäftsprozessanalyse ... 1354
C.13 Datenbanken ... 1354
C.14 Server für Webanwendungen ... 1355
C.15 XML ... 1355
C.16 Webseitenerstellung mit HTML und CSS ... 1355
C.17 Webserveranwendungen ... 1356
C.18 JavaScript und Ajax ... 1356
C.19 Computer- und Netzwerksicherheit ... 1357

Index ... 1359

Materialien zum Buch

Auf der Webseite zu diesem Buch stehen folgende Materialien für Sie zum Download bereit:

- **Lösungen zu den Aufgaben und Kontrollfragen**
- **Quellcode sämtlicher Programmierbeispiele aus dem Buch**

Gehen Sie auf *www.rheinwerk-verlag.de/5728*. Klicken Sie auf den Reiter MATERIALIEN. Sie sehen die herunterladbaren Dateien samt einer Kurzbeschreibung des Dateiinhalts. Klicken Sie auf den Button HERUNTERLADEN, um den Download zu starten. Je nach Größe der Datei (und Ihrer Internetverbindung) kann es einige Zeit dauern, bis der Download abgeschlossen ist.

IT-Handbuch für Fachinformatiker*innen

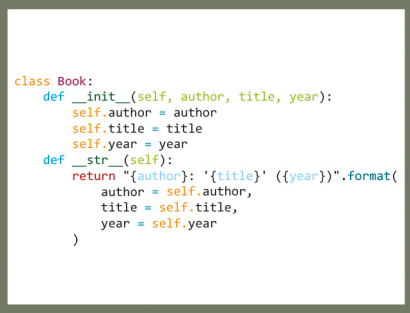
»Das Kompendium für angehende Fachinformatiker«
ADMIN Magazin

Das Standardwerk für Ausbildung und Studium!

Dieses Buch liefert Ihnen das komplette IT-Grundwissen für die Fachinformatik-Ausbildung und angrenzende Berufe. Ob Hardware, Betriebssysteme, Netzwerke, Server, Datenbanken, Programmierung, Webentwicklung – alles wird Ihnen leicht verständlich erklärt. Übungen und Kontrollfragen bereiten Sie optimal auf die IHK-Prüfung vor. Ideal auch zum Selbststudium.

Überblick IT-Ausbildung

IT-Grundlagen von A bis Z



```
class Book:
    def __init__(self, author, title, year):
        self.author = author
        self.title = title
        self.year = year
    def __str__(self):
        return "{author}: '{title}' ({year})".format(
            author = self.author,
            title = self.title,
            year = self.year
        )
```

Netzwerke und Programmierung

IT-Grundlagen und Betriebssysteme verstehen

Hier erfahren Sie alles über die mathematischen und technischen Voraussetzungen von Computern, über Dateiformate und Dateiverwaltung, die Hardware und den Aufbau von Betriebssystemen. Sascha Kersken geht dabei auch auf den Praxiseinsatz von Windows, Linux und macOS ein.

Netzwerktechnik in Theorie und Praxis

Profitieren Sie von einer gründlichen Einführung in Netzwerktechnik und TCP/IP und lernen Sie, wie Sie Server unter Windows und Linux einrichten und Serverdienste einsetzen. So rüsten Sie sich bestens für alle Anwendungsfälle im beruflichen Alltag.

Programmieren lernen und Websites entwickeln

Sie eignen sich die Grundlagen der Programmierung an und lernen Algorithmen, Datenstrukturen, Objektorientierung, Machine Learning, KI und Datenbanken kennen. Außerdem erfahren Sie, wie Sie Webanwendungen mit HTML, CSS, PHP und JavaScript entwickeln.

Sascha Kersken arbeitet seit vielen Jahren als Softwareentwickler sowie als Trainer für EDV-Schulungen in den Themengebieten Netzwerke und Internet, interaktive Medien und Programmierung. Sein IT-Handbuch für Fachinformatiker*innen ist seit vielen Jahren das Standardwerk in der IT-Ausbildung.

Aus dem Inhalt

- Überblick IT-Ausbildung
- Grundbegriffe der Informationstechnik
- Mathematische und technische Grundlagen
- Hardware und Betriebssysteme
- Windows, Linux, macOS – Grundlagen und Praxis
- Netzwerke: Technik und Praxis
- Programmierung in Java und Python
- Algorithmen und Datenstrukturen
- Datenanalyse, Machine Learning, KI
- Datenbanken
- Software-Engineering
- Internet- und Webtechnologien
- IT-Sicherheit
- Aktuelle Themen: PHP 8, Docker, Jenkins, React u. a.

Rheinwerk Computing

€ 39,90 [D] € 41,10 [A]

Gedruckt in Deutschland
Papier aus nachhaltiger Waldwirtschaft
Mineralölfreie Druckfarben

Für Windows, macOS und Linux
Ausbildung/Studium

ISBN 978-3-8362-9591-8