Computernetze

Eine systemorientierte Einführung

Bearbeitet von
Larry L Peterson, Bruce S Davie, Angelika Shafir

3., Aufl. 2004. Buch. XVI, 802 S. Hardcover
ISBN 978 3 89864 242 2
Format (B x L): 16,5 x 24 cm
Gewicht: 1466 g

1 Grundlagen

Ein Netzwerk aufbauen

Nehmen wir an, wir wollen ein Rechnernetz entwickeln, das Potenzial für künftiges Wachstum auf globale Dimensionen hat und vielfältige Anwendungen unterstützt, z.B. Telekonferenzen, Video-on-Demand, elektronischen Handel, verteiltes Rechnen und digitale Bibliotheken. Welche verfügbaren Technologien können als zu Grunde liegende Bausteine dienen, und welche Art der Softwarearchitektur würden wir entwickeln, um diese Bausteine zu einem effektiven Kommunikationsdienst zu integrieren? Die Beantwortung dieser Frage ist das vorrangige Ziel dieses Buchs – also die Beschreibung der verfügbaren Elemente und wie sie dazu verwendet werden können, ein Netzwerk von Grund auf zu konstruieren.

I must Create
a System, or be
enslav'd by another
Man's; I will
not Reason and
Compare: my
business is to Create.

– William Blake

Bevor wir uns mit dem Design eines Rechnernetzes beschäftigen, müssen wir uns zuerst darauf einigen, was genau ein Rechnernetz ist. Früher bedeutete der Begriff *Netzwerk* eine Reihe serieller Leitungen, mit denen dumme Terminals an einen Großrechner angeschlossen wurden. Für einige impliziert der Begriff das Telefonnetz. Für andere ist das einzig interessante Netzwerk das für die Verbreitung von Videosignalen benutzte Kabelnetz. Diese Netzwerke haben das Hauptmerkmal gemeinsam, dass sie jeweils für eine bestimmte Datenart spezialisiert sind (Tastenanschläge, Sprache oder Video) und normalerweise ebenso spezialisierte Geräte (Terminals, Telefonapparate und Fernseher) verbinden.

Wodurch unterscheidet sich ein Rechnernetz von diesen anderen Netzwerktypen? Das wichtigste Merkmal eines Rechnernetzes ist seine Generalität. Rechnernetze werden vorwiegend aus universeller, programmierbarer Hardware aufgebaut und sind nicht für eine bestimmte Anwendung, etwa Telefongespräche oder Fernsehsignale, optimiert. Sie sind vielmehr in der Lage, viele verschiedene Datentypen zu befördern, und sie unterstützen eine breite, ständig wachsende Palette von Applikationen. In diesem Kapitel werden einige typische Anwendungen von Rechnernetzen beschrieben sowie die Anforderungen, die ein Netzwerkentwickler für die Unterstützung solcher Anwendungen berücksichtigen muss.

Wie fahren wir fort, wenn wir diese Anforderungen verstanden haben? Zum Glück müssen wir nicht das erste Netzwerk entwickeln. Das haben andere, vor allem die für das Internet verantwortlichen Wissenschaftler, bereits für uns getan. Wir schöpfen aus der Fülle an Erfahrungen mit dem Internet. Diese Erfahrungen sind in einer *Netzwerkarchitektur* verkörpert, die die verfügbaren Hard- und Softwarekomponenten identifiziert und zeigt, wie sie angeordnet werden können, um ein komplettes Netzwerksystem zu bilden.

Dieses Kapitel führt anhand von drei Aspekten in die Entwicklung von Rechnernetzen ein. Erstens werden die Anforderungen untersucht, die verschiedene Anwendungen und Nutzergruppen (z.B. Netzwerkbenutzer und -betreiber) an das Netzwerk stellen. Zweitens wird das Konzept einer Netzwerkarchitektur vorgestellt, die den Grundstein für den Rest dieses Buchs legt. Drittens werden einige der Schlüsselelemente für die Implementierung von Rechnernetzen vorgestellt. Schließlich werden wichtige Metriken identifiziert, die man verwendet, um die Leistungsfähigkeit von Computernetzen zu bestimmen.

1.1 Anwendungen

Die meisten Leute kennen das Internet durch seine Anwendungen: World Wide Web, E-Mail, Streaming-Audio und -Video, Chat-Rooms oder die gemeinsame Nutzung von Musik (Dateien). Das Web stellt beispielhaft eine intuitiv einfache Nutzeroberfläche dar. Die Anwender sehen sich Seiten voller Text- und Grafikobjekte an, klicken auf Objekte, über die sie mehr erfahren wollen, und schon erscheint eine entsprechende neue Seite. Den meisten ist auch bekannt, dass jedes auf einer Webseite wählbare Objekt unmittelbar unter der Oberfläche über einen Identifizierer mit der nächsten Seite, die man sich ansehen will, verbunden ist. Dieser

Identifizierer heißt Uniform Resource Locator (URL) und benennt eindeutig jede mögliche Page, die man im Webbrowser ansehen kann. Die URL

http://www.mkp.com/pd3e

führt beispielsweise zu einer Seite, die dieses Buch bei Morgan Kaufmann darstellt: Die Zeichenkette http bedeutet, dass zum Laden der Webseite das HyperText Transfer Protocol (HTTP) benutzt werden muss, www.mkp.com ist der Name des Rechners, der die Page ausgibt, und pd3e ist die eindeutige Bezeichnung der Seite auf der Homepage des Verlags.

Was die meisten Web-Benutzer allerdings nicht wissen, ist, dass durch Anklicken einer einzigen solchen URL nicht weniger als 17 Nachrichten über das Internet ausgetauscht werden, wobei bei dieser Zahl davon ausgegangen wird, dass die Page selbst so klein ist, dass sie in eine einzige Nachricht passt. Darin sind bis zu sechs Nachrichten enthalten, um den Server-Namen (www.mkp.com) in seine Internet-Adresse (213.38.165.180) zu übersetzen, drei Nachrichten, um eine TCP-Verbindung (Transmission Control Protocol) zwischen dem Browser und diesem Server aufzubauen, vier Nachrichten, damit der Benutzer-Browser seine HTTP-spezifische »get«-Anfrage senden und der Server ihm mit der angeforderten Page antworten kann (und beide Seiten den Empfang dieser Nachricht bestätigen können), und vier Nachrichten, um die TCP-Verbindung wieder abzubauen. Selbstverständlich beinhaltet das nicht die Millionen von Nachrichten, die Internet-Knoten im Lauf eines Tages untereinander austauschen, nur um sich gegenseitig darüber zu informieren, dass sie existieren und bereit sind, Webpages auszugeben, Namen in Adressen zu übersetzen und Nachrichten an ihr endgültiges Ziel weiterzuleiten.

Eine noch nicht so bekannte Internet-Anwendung wie das Surfen im Web, die sich aber stetig ausbreitet, ist Streaming-Audio und -Video. Obwohl eine komplette Videodatei zuerst von einem entfernten Rechner über das Internet heruntergeladen und dann auf dem lokalen Rechner abgespielt werden könnte, ähnlich wie man eine Webpage lädt und darstellt, müsste man in diesem Falle warten, bis die letzte Sekunde der Videodatei angekommen ist, bevor man sie sich ansehen kann. Streaming-Video bedeutet, dass der Sender die Quelle und der Empfänger die Senke des Videostroms (Stream) ist. Das heißt, die Quelle erzeugt einen Videostrom (eventuell unter Verwendung einer Videoerfassungskarte), sendet ihn in Form von Nachrichten über das Internet, und die Senke beginnt schon mit dem Abspielen, während dieser ankommt.

Um genau zu sein, ist Video keine Anwendung, sondern ein Datentyp. Ein gutes Beispiel einer Videoanwendung ist Video-on-Demand. Dabei wird ein bereitstehender Spielfilm von Platte gelesen und über das Netzwerk übertragen. Eine weitere – interessantere – Anwendung dieser Art ist die Videokonferenz, weil hierbei sehr straffe Zeiteinschränkungen gelten. Wie bei der Benutzung des Telefons muss die Interaktion zwischen den Teilnehmern hier zeitlich genau abgestimmt sein. Wenn der Teilnehmer an einem Ende gestikuliert, muss der entsprechende Video-Frame am anderen Ende so schnell wie möglich dargestellt werden. Durch zu große Verzöge-

Abb. 1.1: Das Videotool vic

rungen wird das System unbrauchbar. Video-on-Demand wird dagegen noch als zufriedenstellend betrachtet, wenn es ab dem Zeitpunkt, an dem der Benutzer das Video startet, 10 Sekunden dauert, bis der erste Frame wiedergegeben wird. Außerdem bedeutet interaktives Video, dass Frames in beiden Richtungen fließen, während eine Video-on-Demand-Anwendung in der Regel Frames nur in eine Richtung sendet.

Die Unix-Anwendung vic ist ein Beispiel eines beliebten Videokonferenzwerkzeugs. Abb. 1.1 zeigt das Control-Panel einer vic-Sitzung. Man beachte, dass vic nur ein Teil einer Reihe von Konferenzwerkzeugen ist, die am Lawrence Berkeley Laboratory und an der UC Berkeley entwickelt wurden. Die übrigen beinhalten eine Whiteboard-Anwendung (wb), mit denen die Benutzer einander Zeichnungen und Vortragsunterlagen senden können, ein audiovisuelles Werkzeug namens vat und ein Sitzungsverzeichnis (sdr), mit dem Videokonferenzen erstellt und angekündigt werden. Alle diese Werkzeuge laufen unter Unix (deshalb werden sie klein geschrieben) und sind kostenlos über das Internet erhältlich. Vergleichbare Werkzeuge sind für andere Betriebssysteme verfügbar.

Obwohl dies nur zwei Beispiele sind, so zeigen das Betrachten von Webseiten und die Möglichkeit einer Videokonferenz doch die Vielseitigkeit von Anwendungen, die auf der Basis des Internet möglich sind. Anderseits weisen sie darauf hin, wie komplex die Struktur des Internet ist. Dieses Buch fängt bei den Grundlagen an und beschäftigt sich dann jeweils mit einem Thema. Dabei wird der Leser erfahren, wie man ein Netzwerk konstruiert, das eine so große Breite von Anwendungen ermöglicht. Kapitel 9 schließt das Buch ab, indem diese beiden speziellen Anwendungen noch einmal genau betrachtet werden, ebenso wie mehrere weitere, die inzwischen im Internet häufig genutzt werden.

1.2 Anforderungen

Wir haben uns ein ehrgeiziges Ziel gesetzt: Verstehen, wie ein Rechnernetz von Grund auf entwickelt wird. Um dieses Ziel zu erreichen, beginnen wir mit den grundlegenden Prinzipien. Dann stellen wir die Fragen, die man sich ganz natürlich stellen würde, wenn man wirklich ein Netzwerk bauen wollte. In jedem Schritt verwenden wir die heute üblichen Protokolle, um die verschiedenen verfügbaren Designoptionen zu erläutern. Wir akzeptieren diese vorhandenen Konstruktionen aber nicht als Evangelium. Vielmehr werden wir die Frage stellen (und beantworten), *warum* Netzwerke auf diese und keine andere Art entwickelt werden. Man darf sich aber nicht nur mit dem Verständnis der heute üblichen Methoden zufriedengeben. Vielmehr ist es wichtig, die zu Grunde liegenden Konzepte zu kennen, weil sich Netzwerke mit technologischen Weiterentwicklungen und neuen Anwendungen ständig ändern. Unsere Erfahrungen haben gezeigt, dass man jedes neue Protokoll leicht verstehen kann, wenn man einmal das Grundkonzept verstanden hat.

Im ersten Schritt werden die Einschränkungen und Anforderungen identifiziert, die das Netzdesign beeinflussen. Bevor wir beginnen, müssen wir allerdings bedenken, dass die Erwartungen, die man an ein Netzwerk stellt, vom jeweiligen Blickwinkel abhängen:

- *Anwendungsprogrammierer* würden die Dienste aufzählen, die ihre Anwendung benötigt, beispielsweise die Gewähr, dass alle von der Anwendung gesendeten Nachrichten fehlerfrei innerhalb eines bestimmten Zeitraums zugestellt werden.
- *Netzwerkdesigner* würden die Eigenschaften eines kostengünstigen Designs aufzählen, beispielsweise, dass Netzressourcen effizient genutzt und gerecht unter mehreren Benutzern aufgeteilt werden.
- *Netzbetreiber* würden die Merkmale eines Systems aufführen, das sich leicht verwalten lässt, beispielsweise, dass Fehler leicht abgegrenzt und Nutzungsgebühren leicht berechnet werden können.

In diesem Abschnitt wird der Versuch unternommen, die verschiedenen Blickwinkel zu einer übergeordneten Einführung in die wichtigen Überlegungen identifiziert, die beim Netzwerkdesign entscheidend sind, zusammenzustellen. Dabei werden die Herausforderungen, die im gesamten Buch immer wieder aufgegriffen werden.

1.2.1 Konnektivität

Wir beginnen mit dem Offensichtlichen: Ein Netzwerk muss Konnektivität zwischen einer Reihe von Computern bieten. Zuweilen reicht es aus, ein begrenztes Netzwerk zu entwickeln, das nur ein paar ausgewählte Maschinen verbindet. Aus Gründen des Datenschutzes und der Sicherheit wird bei vielen privaten (unternehmenseigenen) Netzwerken das ausdrückliche Ziel verfolgt, die Zahl der angeschlossenen Rechner zu begrenzen. Demgegenüber wurden andere Netzwerke (man betrachte das Internet als bestes Beispiel) eigens dafür entwickelt, potenziell alle Rechner der Welt zu verbinden. Ein System, das dafür ausgelegt wurde, Wachstum in einem beliebigen Umfang zu unterstützen, gilt als *skalierbar*. Skalierbarkeit wird im weiteren Verlauf des Buchs anhand des Internet als Modell behandelt.

Links, Knoten und Wolken

Die Konnektivität eines Netzwerks greift auf mehreren Ebenen. Auf der niedrigsten Ebene kann sich ein Netzwerk aus zwei oder mehr Rechnern zusammensetzen, die direkt über ein physisches Medium, z.B. Koaxialkabel oder Lichtwellenleiter, aneinander angeschlossen sind. Wir nennen ein solches physisches Medium *Verbindungsleitung* und die daran angeschlossenen Rechner *Knoten*. (In einem anderen Zusammenhang wird mit dem Begriff »Knoten« kein vernetzter Rechner, sondern ein spezielles Hardwareteil bezeichnet; wir ignorieren dies aber für den Zweck unseres Themas.) Aus Abb. 1.2 wird ersichtlich, dass bestimmte Verbindungsleitungen nur zwei Knoten miteinander verbinden. In diesem Fall handelt es sich um eine *Punkt-zu-Punkt-Verbindung*. Demgegenüber sind bei der so genannten *Mehrfachzugriffsverbindung* mehr als zwei Knoten miteinander verbunden. Ob eine bestimmte Verbindungsleitung Punkt-zu-Punkt- oder Mehrfachzugriffsanschluss unterstützt, hängt davon ab, wie der Knoten an die Verbindungsleitung angeschlossen wird. Häufig sind auch Mehrfachzugriffsverbindungen eingeschränkt. Diese Begrenzung kann die von der Verbindungsleitung abgedeckte geografische Entfernung und/oder die Anzahl der Knoten, die daran angeschlossen werden können, betreffen. Eine Ausnahme hiervon bilden Satellitenverbindungen, die große geografische Bereiche abdecken können.

Würde man Rechnernetze auf Situationen einschränken, in denen alle Knoten über ein gemeinsames physisches Medium direkt miteinander verbunden sind, wären Netzwerke entweder hinsichtlich der unterstützten Rechneranzahl begrenzt,

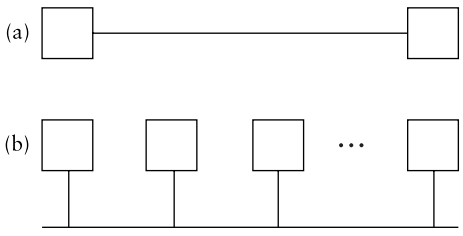

Abb. 1.2: Direkte Verbindungen:
(a) Punkt-zu-Punkt- und
(b) Mehrfachzugriffsverbindung

oder die Kabelmenge zur Verbindung der Knoten wäre bald unhandlich und sehr teuer. Zum Glück bedeutet Konnektivität zwischen zwei Knoten nicht unbedingt, dass sie physisch direkt miteinander verbunden sein müssen. Indirekte Konnektivität lässt sich zwischen einer Reihe zusammenarbeitender Knoten erzielen. Die beiden folgenden Beispiele zeigen, wie mehrere Rechner indirekt miteinander verbunden werden können.

Abb. 1.3 zeigt eine Reihe von Knoten, die jeweils an eine oder mehrere Punkt-zu-Punkt-Verbindungen angeschlossen sind. Auf den Knoten, die an mindestens zwei Verbindungsleitungen angeschlossen sind, läuft Software, welche die über eine Verbindungsleitung empfangenen Daten auf einer anderen weiterleitet. In einer systematischen Anordnung bilden diese weiterleitenden Knoten ein *vermitteltes Netzwerk*. Es gibt mehrere Arten vermittelter Netzwerke, von denen die bekanntesten das *leitungsvermittelte* und das *paketvermittelte* sind. Ersteres wird im Telefonnetz benutzt, während letzteres in der überwältigenden Mehrheit von Rechnernetzen benutzt wird und Kernpunkt dieses Buchs bildet. Ein wichtiges Merkmal von paketvermittelten Netzwerken ist, dass die Knoten untereinander diskrete Datenblöcke austauschen. Man kann sich diese Datenblöcke als Anwendungsdaten, z.B. eine Datei, eine E-Mail oder eine Grafik, vorstellen. Wir nennen jeden Datenblock entweder *Paket* oder *Nachricht* und unterscheiden vorläufig nicht zwischen den beiden Begriffen. Warum sie aber nicht immer das gleiche bedeuten, wird in Abschnitt 1.2.2 erläutert.

Paketvermittelte Netzwerke basieren normalerweise auf einer Strategie, die man *Speichervermittlung* nennt. Wie der Name andeutet, empfängt jeder Knoten in einem speichervermittelten Netzwerk zuerst ein Paket über eine Verbindungsleitung vollständig. Dabei wird das Paket im internen Speicher des Knotens gespeichert; anschließend wird das komplette Paket an den nächsten Knoten gesendet. Im Gegensatz dazu wird bei einem leitungsvermittelten Netzwerk zuerst eine dedizierte Verbindung über eine Folge von Verbindungsleitungen aufgebaut. Dann kann der Quellknoten über

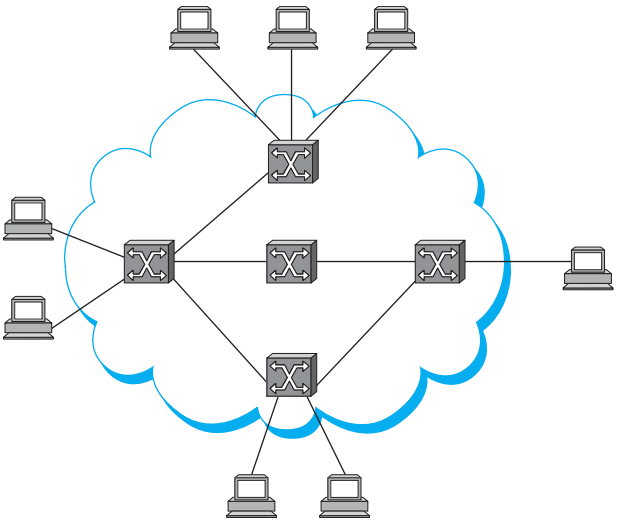

Abb. 1.3:
Vermitteltes Netzwerk

diese Verbindung einen Bitstrom an einen Zielknoten senden. Die Paketvermittlung wird an Stelle der Leitungsvermittlung in einem Rechnernetz hauptsächlich aus Effizienzgründen genutzt, wie im nächsten Abschnitt erläutert wird.

Die Wolke in Abb. 1.3 dient der Unterscheidung zwischen den Knoten innerhalb der Wolke, die das Netzwerk *implementieren* (das sind die *Switche*, deren einzige Aufgabe es ist, Pakete zwischenzuspeichern und weiterzuleiten), und denen außerhalb der Wolke, die das Netzwerk *nutzen* (und die man *Hosts* oder Endsysteme nennt; sie unterstützen Benutzer und führen Anwendungsprogramme aus). Nebenbei bemerkt, ist eine solche Wolke eines der wichtigsten Symbole im Vernetzungsbereich. Im allgemeinen wird mit einer Wolke ein beliebiger Netzwerktyp bezeichnet, egal ob es sich um eine einzelne Punkt-zu-Punkt-Verbindung, eine Mehrfachzugriffsverbindung oder ein vermitteltes Netzwerk handelt. Man kann sich eine Wolke in einer Zeichnung dieses Buchs als Platzhalter für eine der beschriebenen Vernetzungstechnologien vorstellen.

Eine zweite Methode, mit der mehrere Rechner indirekt verbunden werden können, ist in Abb. 1.4 dargestellt. In diesem Fall sind mehrere unabhängige Netzwerke (Wolken) zu einem *Internetwork* zusammengeschlossen. Einen Knoten, der an zwei oder mehr Netzwerke angeschlossen ist, nennt man *Router* oder *Gateway*. Er spielt in etwa die gleiche Rolle wie ein Vermittler, d.h. er leitet Nachrichten von einem Netzwerk zu einem anderen weiter. Ein Internetwork kann man sich ebenfalls als weitere Netzwerkart vorstellen, was bedeutet, dass es sich aus mehreren zusammengeschlossenen Internetworks zusammensetzen kann. Wir können also rekursiv beliebig große Netzwerke bauen, wenn wir mehrere Wolken zu größeren Wolken zusammenfassen.

Wenn eine Reihe von Hosts direkt oder indirekt miteinander verbunden sind, heißt das aber noch nicht, dass es uns gelungen ist, Konnektivität zwischen den Hosts herzustellen. Als letzte Anforderung muss jeder Knoten wissen, mit welchem

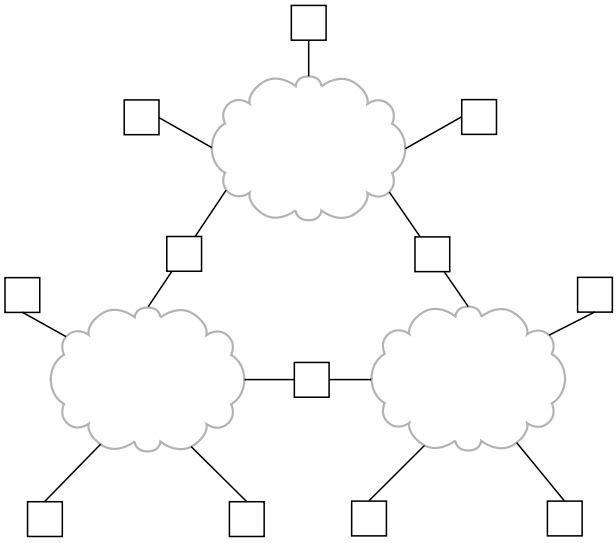

Abb. 1.4: Zusammenschluss mehrerer Netzwerke

der übrigen Knoten im Netzwerk er kommunizieren will. Man erreicht dies dadurch, dass jedem Knoten eine *Adresse* zugewiesen wird. Eine Adresse ist eine Byte-Kette, die einen Knoten identifiziert. Das heißt, das Netzwerk kann die Adresse eines Knotens benutzen, um ihn von den übrigen angeschlossenen Knoten zu unterscheiden. Möchte ein Quellknoten das Netzwerk veranlassen, eine Nachricht an einen bestimmten Zielknoten zu senden, gibt er dessen Adresse an. Sind der sendende und der empfangende Knoten nicht direkt verbunden, verwenden die Switche und Router des Netzwerks diese Adresse, um festzustellen, wie die Nachricht an das Ziel befördert werden kann. Die systematische Ermittlung, wie Nachrichten anhand der Adresse in Richtung Zielknoten zu befördern sind, nennt man *Routing*.

Bei dieser kurzen Einführung in Adressierung und Routing wurde davon ausgegangen, dass der Quellknoten eine Nachricht an einen einzelnen Zielknoten (*Unicast*) senden möchte. Dies ist das übliche, aber nicht das einzige Szenario. Der Quellknoten kann eine Nachricht an alle im Netz angeschlossenen Knoten (*Broadcast*) oder an eine bestimmte Gruppe von Knoten (*Multicast*) senden. Zusätzlich zu den knotenspezifischen Adressen besteht also eine weitere Anforderung an ein Netzwerk darin, dass es Multicast- und Broadcast-Adressen unterstützen muss.

1 **Als wichtigstes Konzept leiten wir daraus ab, dass wir ein Netzwerk rekursiv so definieren können, dass es sich aus zwei oder mehr Knoten zusammensetzt, die über eine Verbindungsleitung verbunden sind, oder dass zwei oder mehr Netzwerke über einen oder mehrere Knoten verbunden sind. Anders ausgedrückt: Ein Netzwerk kann sich aus mehreren Netzwerken zusammensetzen, wobei das Netzwerk auf der untersten Ebene mit einem physischen Medium implementiert wird. Eine der entscheidenden Herausforderungen bei der Bereitstellung von Konnektivität ist es, eine Adresse für jeden Knoten zu definieren, der im Netzwerk erreichbar ist (einschließlich der Unterstützung von Broadcast- und Multicast-Konnektivität) und diese Adresse zu verwenden, um Nachrichten zu dem bzw. den entsprechenden Zielknoten zu übertragen.**

1.2.2 Kostengünstige gemeinsame Nutzung von Ressourcen

Wie bereits erwähnt, bilden paketvermittelte Netzwerke den Themenschwerpunkt dieses Buchs. In diesem Abschnitt wird die wichtigste Anforderung an Rechnernetze – Effizienz – beschrieben, die uns zur Paketvermittlung als der bevorzugten Strategie führt.

Angesichts einer Sammlung von Knoten, die indirekt durch Verschachtelung von Netzwerken miteinander verbunden sind, können zwei beliebige Hosts über eine Reihe von Verbindungsleitungen und Knoten Nachrichten miteinander austauschen. Selbstverständlich möchten wir mehr als die Unterstützung von nur zwei kommunizierenden Hosts erreichen. Wir möchten es allen Hosts paarweise gestatten, Nachrichten miteinander auszutauschen. Die Frage lautet dann, wie alle zur Kommunikation bereiten Hosts sich das Netzwerk teilen, insbesondere, wenn sie es gleichzeitig nutzen wollen. Darüber hinaus muss ein weiteres Problem gelöst werden: Wie kön-

nen sich mehrere Hosts die gleiche *Verbindungsleitung* für den gleichzeitigen Zugriff teilen?

Um zu verstehen, wie sich Hosts ein Netzwerk teilen, müssen wir ein grundlegendes Konzept, das *Multiplexen*, einführen. Dieses bedeutet, dass eine Systemressource gemeinsam von mehreren Benutzern genutzt wird. Auf einer hohen Abstraktionsebene kann man Multiplexen etwa mit einem Timesharing-Computer vergleichen, bei dem eine einzige physische CPU gleichzeitig von mehreren (gemultiplexten) Prozessen genutzt wird. Dabei hat jeder Prozess den Eindruck, er sei der alleinige Nutzer. Vergleichbar dazu werden die von mehreren Benutzern gesendeten Daten über die physischen Verbindungen des Netzwerks gemultiplext.

Abb. 1.5 verdeutlicht, wie dies funktioniert. Hier senden die drei Hosts auf der linken Seite des Netzwerks (L1–L3) Daten an die drei Hosts rechts (R1–R3), indem sie sich ein vermitteltes Netzwerk teilen, das nur eine Verbindungsleitung umfasst. (Der Einfachheit halber gehen wir davon aus, dass Host L1 mit Host R1 usw. kommuniziert). In dieser Situation werden die drei Datenströme der drei Host-Paare auf einer einzigen Verbindungsleitung von Switch 1 gemultiplext. Das *Demultiplexen* zurück in getrennte Ströme wird dann von Switch 2 übernommen. Man beachte, dass wir absichtlich vage darüber bleiben, was genau ein »Datenstrom« bedeutet. In diesem Zusammenhang gehen wir einfach davon aus, dass jeder Host auf der linken Seite eine große Datenmenge an sein jeweiliges Gegenstück rechts senden möchte.

Für das Multiplexen mehrere Datenströme in einer Verbindungsleitung stehen verschiedene Methoden zur Verfügung. Eine weit verbreitete Methode ist das *synchrone Zeitmultiplexverfahren* (Synchronous Time-Division Multiplexing, *STDM*). STDM basiert auf dem Konzept, die Zeit in gleich große Anteile aufzuteilen und jedem Host im Rundumverfahren Gelegenheit zu geben, seine Daten über die gemeinsame Verbindungsleitung zu senden. Das heißt, in Zeitanteil 1 werden Daten des ersten Host-Paars übertragen, in Zeitanteil 2 werden Daten des zweiten Host-Paars übertragen usw. Dieses Verfahren wird solange fortgesetzt, bis alle Paare nacheinander übertragen konnten. Dann erhält wieder das erste Paar Gelegenheit, Daten zu übertragen, und das Verfahren wird wiederholt. Eine weitere Methode ist das *Frequenzmultiplexverfahren* (Frequency-Division Multiplexing, *FDM*). Bei dieser Methode wird jeder Datenstrom über die gemeinsame Verbindungsleitung in einer anderen Frequenz übertragen. Das ist etwa vergleichbar mit den Signalen unterschiedlicher Fernsehstationen, die im gleichen Fernsehkabel in je einer anderen Frequenz übertragen werden.

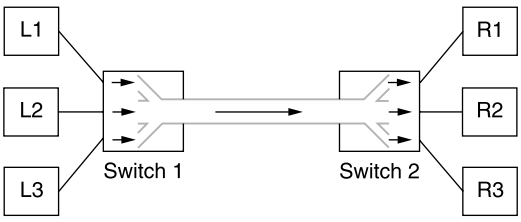

Abb. 1.5: Multiplexen mehrerer logischer Datenströme auf einer physischen Verbindung

Die beiden Konzepte sind zwar leicht zu verstehen, jedoch ist sowohl STDM als auch FDM auf unterschiedliche Art begrenzt. Erstens: Wenn eines der Host-Paare keine Daten zu senden hat, bleibt sein Anteil an der Verbindungsleitung, d.h. sein Zeitanteil bzw. seine Frequenz, ungenutzt, während ein anderes Host-Paar Daten zu übertragen hat, aber warten muss, bis es an die Reihe kommt. Bei der rechnergestützten Kommunikation kann die Zeit, die eine Verbindungsleitung ungenutzt bleibt, sehr lang sein. Man betrachte beispielsweise die Zeit, die man verbringt, um eine Webseite zu lesen (während die Verbindung im Leerlauf ist), im Vergleich zu der Zeit, die man aufwendet, um eine Seite abzurufen. Zweitens: STDM und FDM sind auf Fälle begrenzt, in denen die maximale Anzahl der Datenströme im Voraus feststeht und bekannt ist. Bei STDM ist es nicht möglich, die Menge zu ändern oder zu erhöhen, und bei FDM können keine neuen Frequenzen hinzugefügt werden.

SANs, LANs, MANs und WANs

Größe ist eine Möglichkeit, Netzwerke zu charakterisieren. Zwei bekannte Beispiele sind LANs (Local Area Networks) und WANs (Wide Area Networks). LANs erstrecken sich normalerweise über weniger als 1 km, während WANs weltweit sein können. MANs (Metropolitan Area Networks) decken normalerweise einige zig Kilometer ab. Der Grund dafür, dass diese Klassifizierungen interessant sind, ist dass die Größe eines Netzwerks oft Auswirkungen auf die zu Grunde liegende Technologie hat, die verwendet werden kann. Ein zentraler Faktor ist dabei die Zeit, die es dauert, bis sich Daten von einem Ende des Netzwerks zum anderen ausbreiten. Dieser Faktor wird in späteren Kapiteln behandelt.

Als historischer Hinweis ist zu bemerken, dass der Begriff »Wide Area Network« für die ersten WANs nicht verwendet wurde, weil es keine andere Netzwerkart gab, von der man sie unterscheiden konnte. Als Computer noch selten und teuer waren, machte man sich nicht viel Gedanken über die Verbindung aller Computer in einem lokalen Bereich. Es gab nur einen Computer in jedem Bereich. Erst als sich Computer verbreiteten, wurden LANs erforderlich. Ferner wurde der Begriff »WAN« eingeführt, um die größeren Netzwerke zu beschreiben, die geographisch weit entfernte Computer miteinander verbanden.

Eine neue Netzwerkart ist SAN (System Area Network). Das Anwendungsgebiet von SANs liegt im Bereich des Cluster-Computing, d.h. der Kopplung von PCs oder Arbeitsplatzrechnern zu einem Funktions-, Daten- und Lastverbund mit sehr hoher Performanz. Beispielsweise sind HiPPI (High Performance Parallel Interface) und Fiber Channel zwei bekannte SAN-Technologien, mit denen massiv parallele Prozessoren zu skalierbaren Speicher-Servern verknüpft werden. (Da SANs meist Computer mit Speicher-Servern verbinden, werden sie auch als *Storage* Area Networks bezeichnet). Diese Netzwerke werden in diesem Buch nicht ausführlich beschrieben, verdienen aber Beachtung, weil sie hinsichtlich Leistung den anderen Netzarten oft überlegen sind und vermehrt für die Verbindung solcher Netzwerke in LANs und WANs benutzt werden.

Die Form des Multiplexens, die wir in diesem Buch vorwiegend betrachten, heißt *statistisches Multiplexen*. Die Bezeichnung ist nicht besonders hilfreich für das Verständnis des zu Grunde liegenden Konzepts, das an sich sehr einfach ist und auf zwei Hauptfaktoren gründet. Erstens ist es mit STDM dahingehend vergleichbar, dass die Verbindungsleitung zu einem Zeitpunkt nur von einem Datenstrom benutzt wird. Über die Verbindungsleitung fließen erst die Daten eines Host-Paars, dann die eines anderen usw. Im Gegensatz zu STDM werden die Daten von jedem Host-Paar aber nicht innerhalb eines im Voraus festgesetzten Zeitschlitzes, sondern auf Anfrage übertragen. Das heißt, nur wenn ein Host-Paar Daten zu übertragen hat, erhält es Zugriff auf die Verbindungsleitung, statt darauf warten zu müssen, bis es an die Reihe käme, während die Zeitanteile bzw. Frequenzen der Host-Paare, die gerade keine Daten zu übertragen haben, ungenutzt blieben. Durch diese Vermeidung von Leerlaufzeiten erreicht die Paketvermittlung ihre Effizienz.

In der bisherigen Definition verfügt statistisches Multiplexen aber über keinen Mechanismus, der sicherstellt, dass alle Datenströme schließlich komplett über die gemeinsame Verbindungsleitung übertragen werden. Das heißt, nachdem ein Host-Paar mit dem Senden der Daten beginnt, brauchen wir eine Methode, um die Übertragung zu begrenzen, damit das nächste Host-Paar übertragen kann. Hierfür setzt das statistische Multiplexen eine obere Grenze der Größe des Datenblocks, der in einem Datenstrom zu einem bestimmten Zeitpunkt übertragen werden darf. Dieser in der Größe begrenzte Datenblock wird als *Paket* bezeichnet, im Unterschied zu einer *Nachricht*, die ein Anwendungsprogramm möglicherweise übertragen möchte, weil ein paketvermitteltes Netzwerk also die maximale Paketgröße einschränkt, gelingt es einem Host vielleicht nicht, eine Nachricht vollständig in einem Paket zu senden. Das kann bedeuten, dass die Quelle die Nachricht auf mehrere Pakete aufteilen muss, die der Empfänger dann wieder zusammensetzen muss.

Anders ausgedrückt: Jeder Host sendet eine Reihe von Paketen über die gemeinsame Verbindungsleitung, wobei paketweise die Entscheidung getroffen wird, welches Paket von welchem Host als Nächstes übertragen wird. Hat nur ein Host Daten zu übertragen, kann er eine Reihe von Paketen direkt nacheinander senden. Möchten mehrere Hosts gleichzeitig Daten schicken, werden ihre Pakete abwechselnd übertragen. Abb. 1.6 zeigt einen Switch, der Pakete von mehreren Quellen in einer einzigen gemeinsamen Verbindungsleitung multiplext.

Die Entscheidung, welches Paket als Nächstes über eine gemeinsame Verbindungsleitung gesendet wird, kann auf unterschiedliche Art getroffen werden. Bei einem Netzwerk mit mehreren Switchen, die über Verbindungsleitungen miteinander verbunden sind, etwa wie im Beispiel in Abb. 1.5, würde der Switch, der Pakete auf der gemeinsamen Verbindungsleitung überträgt, die Entscheidung treffen. (Wir werden später noch sehen, dass nicht alle paketvermittelten Netzwerke Switche enthalten und daher andere Mechanismen anwenden, um Entscheidungen über die Übertragung von Paketen zu treffen.) Jeder Switch in einem paketvermittelten Netzwerk trifft dabei seine Entscheidung unabhängig, auf der Grundlage einzelner Pakete. Ein Faktor, mit dem sich Netzwerkdesigner befassen müssen, ist der, diese Entscheidung fair zu treffen. Ein Switch kann beispielsweise so ausgelegt werden,

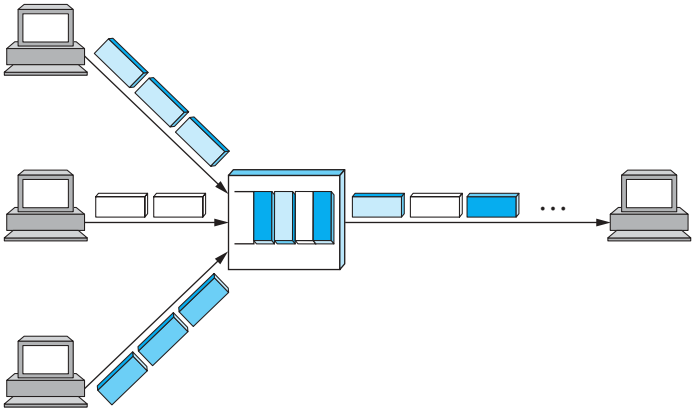

Abb. 1.6: Ein Switch multiplext Pakete von mehreren Quellen auf eine gemeinsame Leitung.

dass er Pakete auf FIFO-Grundlage (First-In-First-Out oder »wer zuerst kommt, malt zuerst«) überträgt. Bei einer anderen Methode werden die übertragungswilligen Hosts im Rundumverfahren – ähnlich wie beim STDM – bedient. Dabei kann beispielsweise gewährleistet werden, dass bestimmte Hosts einen bestimmten Anteil an der Bandbreite der Verbindungsleitung erhalten. Oder es kann gewährleistet werden, dass die Pakete bestimmter Hosts im Switch nie um mehr als eine bestimmte Zeitdauer verzögert werden. Ein Netzwerk, das Hosts die Inanspruchnahme einer solchen Sonderbehandlung gestattet, unterstützt *Dienstgüte* (Quality of Service, *QoS*).

Da der Switch in Abb. 1.6 drei ankommende Paketströme in eine abgehende Verbindungsleitung multiplexen muss, kann es bei dieser Konstellation auch sein, dass der Switch Pakete schneller empfängt als die gemeinsame Verbindungsleitung bewältigen kann. In diesem Fall ist der Switch gezwungen, Pakete in seinem Speicher zwischenzuspeichern. Wenn ein Switch über eine längere Zeitspanne Pakete schneller empfängt als er senden kann, geht ihm früher oder später der Speicherplatz aus, und einige Pakete müssen verworfen werden. Gerät ein Switch in diesen Betriebszustand, sagt man, er ist *überlastet*.

1 **Fazit: Statistisches Multiplexen ist eine kostengünstige Methode für mehrere Benutzer (z.B. Datenströme von Host zu Host), um Netzwerkressourcen (Verbindungsleitungen und Knoten) gemeinsam zu nutzen. Die Methode definiert das Paket als kleinste Einheit, mit der die Verbindungsleitungen des Netzwerks für verschiedene Datenströme aufgeteilt werden, sodass jeder Switch über die Nutzung der Verbindungsleitungen auf Paketbasis entscheiden kann. Eine faire Zuweisung der Leitungskapazität auf verschiedene Datenströme und die Handhabung einer möglichen Überlastung sind die größten Herausforderungen beim statistischen Multiplexen.**

1.2.3 Unterstützung gemeinsamer Dienste

Während im vorherigen Abschnitt die Herausforderungen bei der Bereitstellung kostengünstiger Konnektivität für eine Gruppe von Hosts behandelt wurde, wäre es grob vereinfacht, ein Rechnernetz so zu betrachten, als würde es lediglich Pakete zwischen mehreren Rechnern übertragen. Man muss sich ein Netzwerk vielmehr so vorstellen, dass es die Mittel für eine Reihe von Anwendungsprozessen bereitstellt, die auf diesen Rechnern laufen und miteinander kommunizieren wollen. Anders ausgedrückt: Die nächste Anforderung an ein Rechnernetz lautet, dass die Anwendungsprogramme, die auf den an das Netz angeschlossenen Hosts laufen, in der Lage sein müssen, auf sinnvolle Weise miteinander zu kommunizieren.

Wenn zwei Anwendungsprogramme miteinander kommunizieren möchten, müssen außer der einfachen Übertragung einer Nachricht von einem Host zu einem anderen viele komplexe Aktionen ablaufen. Eine Möglichkeit wäre es für Anwendungsprogrammierer, alle komplizierten Funktionen in jedes Anwendungsprogramm zu integrieren. Da viele Anwendungen aber gemeinsame Dienste benötigen, ist es viel sinnvoller, diese gemeinsamen Dienste lediglich einmal zu implementieren. Dann können die Anwendungsentwickler die Anwendungen so auslegen, dass sie diese Dienste nutzen. Den Netzwerkdesignern stellt sich dabei die Herausforderung, die richtige Zusammenstellung dieser Dienste zu identifizieren. Das Ziel ist nun, die Komplexität des Netzwerks vor der Anwendung zu verbergen, ohne die Möglichkeiten der Entwickler zu sehr einzuschränken.

Intuitiv betrachten wir das Netzwerk dahingehend, dass es logische *Kanäle* bereitstellt, über die Prozesse auf Anwendungsebene miteinander kommunizieren können. Jeder Kanal stellt die von der jeweiligen Anwendung benötigten Dienste bereit. Vergleichbar damit, wie wir eine Wolke benutzen, um Konnektivität zwischen einer

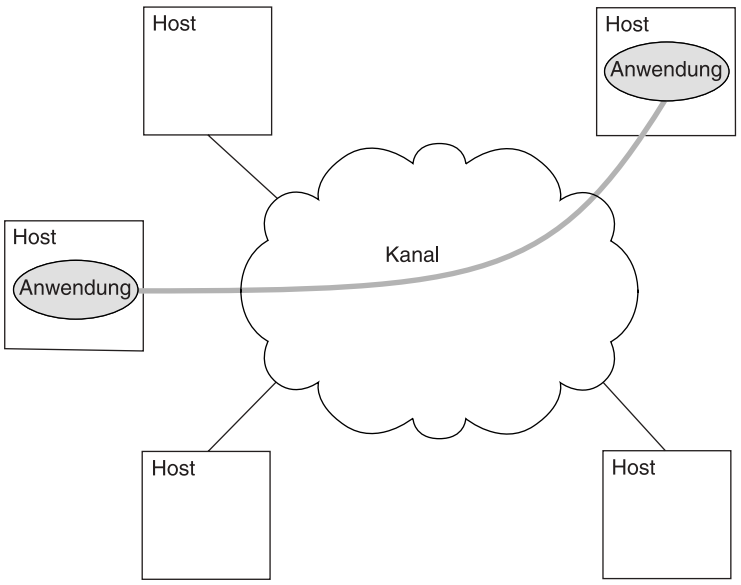

Abb. 1.7: Prozesse kommunizieren über einen abstrakten Kanal.

Reihe von Rechnern abstrakt darzustellen, betrachten wir jetzt einen Kanal als Verbindungsglied zwischen Prozessen. Abb. 1.7 zeigt zwei Prozesse auf Anwendungsebene, die über einen logischen Kanal miteinander kommunizieren, der seinerseits innerhalb einer Wolke implementiert ist, die eine Reihe von Hosts verbindet. Man kann sich den Kanal als »Pipeline« vorstellen, die zwei Anwendungen verbindet, sodass eine sendende Anwendung Daten an einem Ende einspeisen und erwarten kann, dass das Netzwerk die Daten der Anwendung am anderen Ende der Pipeline zustellt.

Die Herausforderung hierbei ist es, zu ermitteln, welche Funktionalität die Kanäle den Anwendungsprogrammen bereitstellen sollen. Benötigt die Anwendung beispielsweise eine Garantie, dass die über den Kanal gesendeten Nachrichten zugestellt werden, oder ist es akzeptabel, dass einige nicht ankommen? Müssen die Nachrichten beim empfangenden Prozess in der gleichen Reihenfolge ankommen, in der sie gesendet wurden, oder kümmert es den Empfänger nicht, in welcher Reihenfolge sie ankommen? Muss das Netzwerk sicherstellen, dass keine Dritten den Kanal belauschen können oder ist Datenschutz nicht von Belang? Im allgemeinen bietet ein Netzwerk viele verschiedene Kanaltypen. Jede Anwendung kann dann den Typ auswählen, der ihre Anforderungen am besten erfüllt. Im restlichen Teil dieses Abschnitts werden Faktoren für die Definition nützlicher Kanäle behandelt.

Identifizierung gemeinsamer Kommunikationsmuster

Um abstrakte Kanäle entwickeln zu können, muss man zuerst die Kommunikationsbedürfnisse einer repräsentativen Reihe von Anwendungen feststellen. Dann extrahiert man ihre gemeinsamen Kommunikationsanforderungen. Schließlich integriert man die Funktionalität, die diese Anforderungen erfüllt, in das Netzwerk.

Unter den ältesten von einem Netzwerk unterstützten Anwendungen sind Dateitransferprogramme wie FTP (File Transfer Protocol) oder NFS (Network File System). Die Einzelheiten sind zwar sehr unterschiedlich, beispielsweise, ob ganze Dateien übertragen oder nur einzelne Dateiblöcke zu einem bestimmten Zeitpunkt gelesen/geschrieben werden. Generell ist jedoch die Kommunikationskomponente des entfernten Dateizugriffs (Remote File Access) durch das Zusammenspiel zweier Prozesse charakterisiert: einer, der eine Datei zum Lesen oder Schreiben anfordert und ein zweiter, der diese Anfrage erfüllt. Der Prozess, der den Zugriff auf die Datei anfordert, heißt *Client*, und derjenige, der den Zugriff unterstützt, wird als *Server* bezeichnet.

Will der Client eine Datei lesen, muss er eine kurze Anfragenachricht an einen Server senden. Der Server antwortet mit einer langen Nachricht, in der sich die Daten aus der angeforderten Datei befinden. Das Schreiben läuft umgekehrt ab: Der Client sendet eine lange Nachricht mit den zum Server zu schickenden Daten, und der Server reagiert mit einer kurzen Nachricht, mit der er bestätigt, dass die Daten auf Platte geschrieben wurden. Eine digitale Bibliothek, wie sie zum Beispiel das World Wide Web verkörpert, stellt eine andere Anwendung dar, die sich ganz ähnlich verhält: Ein Client-Prozess schickt eine Anfrage (Request) und ein Server-Prozess antwortet, indem er die gewünschten Daten zurücksendet.

In Kenntnis von File Transfer, einer digitalen Bibliothek und zwei Video-Anwendungen, die in der Einleitung beschrieben sind (Videokonferenz und Video-on-

demand) als repräsentative Beispiele könnten wir uns dazu entschließen, die beiden folgenden Kanaltypen bereitzustellen: *Anfrage/Antwort*-Kanäle und *Nachrichtenstrom*-Kanäle. Der Anfrage/Antwort-Kanal würde sich für die Dateitransfer- und die digitale Bibliotheksanwendung eignen. Er würde gewährleisten, dass jede von einer Seite gesendete Nachricht am anderen Ende empfangen und nur jeweils eine Kopie einer Nachricht zugestellt wird. Der Anfrage/Antwort-Kanal könnte auch Datenschutz und -integrität bieten, sodass nicht autorisierte Benutzer die zwischen den Client- und Server-Prozessen ausgetauschten Daten nicht lesen oder ändern können.

Der Nachrichtenstrom-Kanal könnte von der Video-on-Demand- und Videokonferenzanwendung benutzt werden, falls er so parametrisiert wird, dass er sowohl Ein- als auch Zweiwegverkehr sowie verschiedene Verzögerungsmerkmale unterstützt. Der Nachrichtenstrom-Kanal müsste nicht unbedingt gewährleisten, dass alle Nachrichten zugestellt werden, da eine Videoanwendung angemessen funktioniert, auch wenn einige Frames nicht empfangen werden. Er müsste aber sicherstellen, dass die zugestellten Nachrichten in der gleichen Reihenfolge ankommen, in der sie gesendet wurden, damit alle Frames entsprechend der Bildfolge wiedergegeben werden. Wie der Anfrage/Antwort- müsste der Nachrichtenstrom-Kanal eventuell Datenschutz und -integrität der Videodaten sicherstellen. Schließlich sollte der Nachrichtenstrom-Kanal unter Umständen Multicast unterstützen, damit mehrere Parteien an der Telekonferenz teilnehmen bzw. das Video betrachten können.

In der Regel streben Netzwerkentwickler eine möglichst kleine Anzahl abstrakter Kanaltypen an, die eine möglichst große Anzahl von Anwendungen bedienen. Dabei besteht allerdings das Risiko, dass man versucht, mit zu wenigen Kanalabstraktionen auszukommen. Einfach ausgedrückt: Mit einem Hammer in der Hand kommt einem alles wie ein Nagel vor. Wenn beispielsweise nur Nachrichtenstrom- und Anfrage/Antwort-Kanäle zur Verfügung stehen, ist man versucht, sie für die nächste Anwendung zu benutzen, egal, um welche es sich handelt, auch wenn kein Kanaltyp eigentlich genau die von der Anwendung geforderte Semantik aufweist. Aus diesem Grund werden wahrscheinlich Netzwerkentwickler neue Kanaltypen erfinden und vorhandene Kanäle um zusätzliche Optionen erweitern – solange, wie Anwendungsprogrammierer immer wieder neue Anwendungen entwickeln.

Unabhängig davon, *welche* Funktionalität genau ein bestimmter Kanal bereitstellt, gilt es immer die Frage zu beantworten, *wo* diese Funktionalität implementiert werden soll. In vielen Fällen ist es am einfachsten, die Host-zu-Host-Konnektivität des zu Grunde liegenden Netzwerks schlichtweg als *Bit-Pipeline* zu betrachten, wobei die jeweilige Kommunikationssemantik von den beteiligten Hosts bereitgestellt wird. Diese Vorgehensweise hat den Vorteil, dass die Switche im Netzwerk so einfach wie möglich sein können. Sie leiten einfach Pakete weiter. Das bedeutet andererseits aber, dass die Hosts einen Großteil der Unterstützung der semantisch reichhaltigen Prozess-zu-Prozess-Kanäle übernehmen müssen. Die alternative Methode wäre, diese Funktionalität auf die Switche zu verlagern, sodass die Hosts als »dumme« Endgeräte (z.B. Telefonapparate) fungieren. Im weiteren Verlauf wird die Frage, wie die Erbringung verschiedener Netzdienste zwischen Paket-Switchen und Hosts (Geräten) aufgeteilt werden kann, als wiederkehrendes Thema im Netzwerkdesign behandelt.

Zuverlässigkeit

Aus den soeben betrachteten Beispielen wird deutlich, dass zuverlässige Nachrichtenzustellung eine der wichtigsten Funktionen eines Netzwerks sein kann. Ohne zu verstehen, wie und warum Netzwerke versagen können, lässt sich aber nur schwerlich ermitteln, wie diese Zuverlässigkeit bereitzustellen ist. Zuerst gilt es zu erkennen, dass Rechnernetze nicht in einer perfekten Welt existieren. Rechner stürzen ab und werden neu gestartet, Lichtwellenleiter brechen, elektrische Störungen verändern Bits in den übertragenen Daten, Switchen geht der Pufferspeicher aus, und als ob diese physikalischen Probleme noch nicht genug wären, verfrachtet die Software, die die Hardware steuert, Pakete manchmal ins virtuelle Niemandsland. Eine wichtige Anforderung an ein Netzwerk besteht deshalb darin, bestimmte Ausfallarten zu maskieren (verbergen), damit das Netzwerk den nutzenden Anwendungen zuverlässiger scheint, als es tatsächlich ist.

Ausfälle, mit denen sich Netzwerkentwickler befassen müssen, werden allgemein in drei Klassen aufgeteilt. Die erste Klasse betrifft die Bitebene. Wenn ein Paket über eine Verbindungsleitung übertragen wird, können in den Daten *Bitfehler* auftreten. Das bedeutet, dass eine 1 zu einer 0, und umgekehrt, vertauscht wird. Manchmal werden nur einzelne Bits verändert. Noch häufiger treten aber *Burstfehler* auf. Das heißt, es sind mehrere aufeinander folgende Bits betroffen. Bitfehler werden normalerweise durch Einwirkungen von außen verursacht, z.B. Blitzschlag, Stromstöße oder Mikrowellenherde, die störend auf die Datenübertragung einwirken. Andererseits kommen solche Bitfehler eher selten vor und beeinträchtigen im Durchschnitt nur eines von 10^6 bis 10^7 Bits in einem herkömmlichen Kupferkabel und eines von 10^{12} bis 10^{14} Bits in einem typischen Glasfaserkabel. An späterer Stelle werden Techniken beschrieben, mit denen diese Bitfehler mit hoher Wahrscheinlichkeit erkannt werden können. Nach der Erkennung können solche Fehler teilweise korrigiert werden, sofern wir wissen, welche Bits verfälscht wurden. In manchen Fällen dreht man sie einfach wieder um. In anderen Fällen ist der Schaden so groß, dass man das ganze Paket verwerfen muss. Dann muss der Sender das Paket erneut übertragen.

Die zweite Klasse betrifft die Paketebene. Konkret heißt das, dass das Netzwerk ein komplettes Paket verliert. Ein Grund dafür kann sein, dass die im Paket enthaltenen Bitfehler nicht korrigiert werden konnten, sodass das Paket verworfen werden musste. Eher wahrscheinlich ist aber, dass einer der Knoten, der das Paket handhaben muss, z.B. ein Switch, der es von einer Leitung auf eine andere befördert, derart überlastet ist, dass er das Paket nicht zwischenspeichern kann und es folglich wegwerfen muss. Dies betrifft das in Abschnitt 1.2.2 angesprochene Überlastungsproblem. Weniger häufig, aber dennoch möglich ist, dass die Software, die auf einem der Knoten läuft, die das Paket handhaben, einen Fehler macht. Beispielsweise schickt sie ein Paket irrtümlich auf der falschen Verbindung los, sodass es seinen Weg zum Ziel nie findet. An späterer Stelle wird die Handhabung von verlorenen Paketen beschrieben. Dabei ist eine der größten Schwierigkeiten, dass zwischen einem Paket, das wirklich verlorengegangen ist, und einem, das lediglich zu spät am Ziel ankommt, unterschieden werden muss.

Die dritte Fehlerklasse betrifft die Knoten- und Verbindungsebene. Beispielsweise kann eine Verbindungsleitung brechen oder ein angeschlossener Rechner stürzt ab. Dies kann durch abgestürzte Software, Stromausfall oder einen unbedarften Bediener verursacht werden. Solche Ausfälle lassen sich zwar mehr oder weniger schnell beheben, können aber eine länger andauernde verheerende Wirkung auf das Netzwerk haben. Das Netzwerk fällt dabei nicht unbedingt ganz aus. Bei einem paketvermittelten Netzwerk ist es beispielsweise möglich, den Datenverkehr um einen ausgefallenen Knoten oder eine defekte Leitung herumzuleiten. Eine der Schwierigkeiten bei dieser Fehlerklasse ist es, zwischen einem ausgefallenen und einem langsamen Rechner bzw. einer gebrochenen und einer qualitativ schlechten Verbindungsleitung, die viele Bitfehler verursacht, zu unterscheiden.

1 **Die wichtigste Erkenntnis dieses Abschnitts ist, dass man die Anforderungen der Anwendungen und die Grenzen der zu Grunde liegenden Technologie verstehen muss, um nützliche Kanäle definieren zu können. Die Herausforderung liegt darin, die Lücke zu füllen zwischen dem, was die jeweilige Anwendung erwartet, und dem, was die zu Grunde liegende Technologie bereitstellen kann. Dies nennt man manchmal auch die** *semantische Lücke***.**

1.3 Netzwerkarchitektur

Der Leser hat zweifellos bemerkt, dass im vorherigen Abschnitt recht umfangreiche Anforderungen an das Netzwerkdesign gestellt wurden. Ein Rechnernetz muss allgemeine, kostengünstige, faire, robuste und leistungsstarke Konnektivität für eine große Anzahl an Computern bereitstellen. Als wäre dies noch nicht genug, verharren Netzwerke nicht fest an einem bestimmten Entwicklungsstand, sondern müssen laufend entsprechend den Änderungen der zu Grunde liegenden Technologien und den Ansprüchen durch Anwendungsprogramme weiterentwickelt werden. Die Entwicklung eines Netzwerks, das diese Anforderungen erfüllt, ist also kein leichtes Unterfangen.

Um diese Komplexität bewältigen zu können, wurden generelle Pläne entwickelt, die man insgesamt als *Netzwerkarchitektur* bezeichnet. Dies ist eine Art Leitfaden für die Entwicklung und Implementierung von Netzwerken. In diesem Abschnitt wird anhand der wesentlichen Konzepte ausführlich erklärt, was wir mit Netzwerkarchitektur meinen. Weiterhin werden zwei der bekanntesten Architekturen – die OSI- und die Internet-Architektur – beschrieben.

1.3.1 Schichten und Protokolle

Wenn ein System zu komplex wird, führt der Systementwickler eine weitere Abstraktionsebene ein. Eine Abstraktion dient der Definition eines vereinheitlichenden Modells, das die wichtigsten Aspekte des Systems erfasst. Dieses Modell kann in einem Objekt gekapselt werden, welches ein Interface zur Manipulation durch

andere Systemkomponenten bietet, und die Einzelheiten, wie das Objekt implementiert wird, vor den Nutzern des Objekts verbirgt. Die Herausforderung ist hier, Abstraktionen zu identifizieren, die einen Dienst bereitstellen, der sich in zahlreichen unterschiedlichen Fällen als nützlich erweist und im zu Grunde liegenden System effizient implementiert werden kann. Genau das haben wir mit der Einführung des Konzepts eines Kanals im vorherigen Abschnitt getan: Wir haben Anwendungen eine Abstraktion bereitgestellt, die die Komplexität des Netzwerks vor Anwendungsentwicklern verbirgt.

Abstraktionen führen oft zu Schichten, insbesondere in Netzwerksystemen. Die Grundidee ist dabei, dass man mit den Diensten beginnt, die von der zu Grunde liegenden Hardware geboten werden, und dann weitere Schichten hinzufügt, die jeweils eine höhere (abstraktere) Dienstebene bereitstellen. Die Dienste der hohen Schichten werden durch Verwendung der von den niedrigeren Schichten bereitgestellten Dienste implementiert. Greifen wir zur Veranschaulichung auf die im vorherigen Abschnitt beschriebenen Anforderungen zurück, können wir uns beispielsweise vorstellen, dass ein Netzwerk zwei Abstraktionsschichten umfasst, die zwischen dem Anwendungsprogramm und der zu Grunde liegenden Hardware liegen, wie aus Abb. 1.8 ersichtlich wird. Die Schicht unmittelbar oberhalb der Hardware stellt Host-zu-Host-Konnektivität bereit. Sie abstrahiert (verbirgt) dabei die Tatsache, dass zwischen zwei beliebigen Hosts eine komplexe Netztopologie liegen kann. Die nächsthöhere Schicht baut auf dem verfügbaren Host-zu-Host-Kommunikationsdienst auf und unterstützt Prozess-zu-Prozess-Kanäle. Sie verbirgt z.B. die Tatsache, dass das Netzwerk gelegentlich Nachrichten verliert.

Abb. 1.8:
Darstellung eines Netzwerksystems in Schichten

Die Auslegung eines Netzwerks in Schichten bietet zwei angenehme Merkmale: Erstens lässt sich ein Netzwerk in leichter handhabbare Komponenten zerlegen. Statt eine monolithische allumfassende Software zu implementieren, kann man mehrere Schichten implementieren, die jeweils einen ganz bestimmten Teil des Ganzen lösen. Zweitens erhält man ein modulares Design. Entscheidet man sich beispielsweise an irgendeinem Punkt, einen neuen Dienst hinzuzufügen, braucht man nur die

Funktionalität einer Schicht zu ändern und kann die auf allen übrigen Schichten vorhandenen Funktionen wiederverwenden.

Sich ein System als lineare Folge von Schichten vorzustellen, wäre nun aber eine allzu große Vereinfachung. Häufig gibt es mehrere Abstraktionen auf einer bestimmten Systemebene, die den höheren Schichten jeweils einen anderen Dienst bereitstellen, alle aber auf den gleichen Abstraktionen der unteren Ebenen aufbauen. Zur Verdeutlichung betrachten wir die beiden in Abschnitt 1.2.3 beschriebenen Kanaltypen: einer stellt einen Anfrage/Antwortdienst und der andere einen Nachrichtenstromdienst bereit. Diese beiden Kanäle können auf bestimmten Ebenen eines mehrschichtigen Netzwerksystems alternative Dienste bieten (siehe Abb. 1.9).

Abb. 1.9: Mehrschichtiges System mit alternativen Abstraktionen auf einer bestimmten Schicht

Mit diesem Grundlagenwissen über Schichtenmodelle können wir jetzt auf die Architektur eines Netzwerks näher eingehen. Für Neulinge gleich vorab: Die abstrakten Objekte, aus denen sich die Schichten eines Netzwerksystems zusammensetzen, nennt man *Protokolle*. Ein Protokoll bietet einen Kommunikationsdienst, den Objekte höherer Ebenen (z.B. Anwendungsprozesse oder vielleicht Protokolle höherer Ebenen) zum Austauschen von Nachrichten benutzen. Man kann sich beispielsweise ein Netzwerk vorstellen, das ein Anfrage/Antwortprotokoll und ein Nachrichtenstromprotokoll unterstützt, das den oben behandelten Anfrage/Antwort- und Nachrichtenstromkanälen entspricht.

Jedes Protokoll definiert zwei verschiedene Interfaces: Das erste ist ein *Dienst-Interface* zu den übrigen Objekten auf dem gleichen Computer, die seine Kommunikationsdienste nutzen wollen. Dieses Dienst-Interface definiert die Operationen, die lokale Objekte mit dem Protokoll ausführen können. Beispielsweise würde ein Anfrage/Antwortprotokoll Operationen unterstützen, durch die eine Anwendung Nachrichten senden und empfangen kann. Zweitens definiert ein Protokoll ein *Partner-Interface* zu seinem Gegenstück (*Peer* bzw. Partner) auf einem anderen Rechner. Dieses zweite Interface definiert Form und Bedeutung von Nachrichten, die zwischen Protokollpartnern ausgetauscht werden, um den Kommunikationsdienst zu implementieren. Dies würde die Art festlegen, in der ein Anfrage/

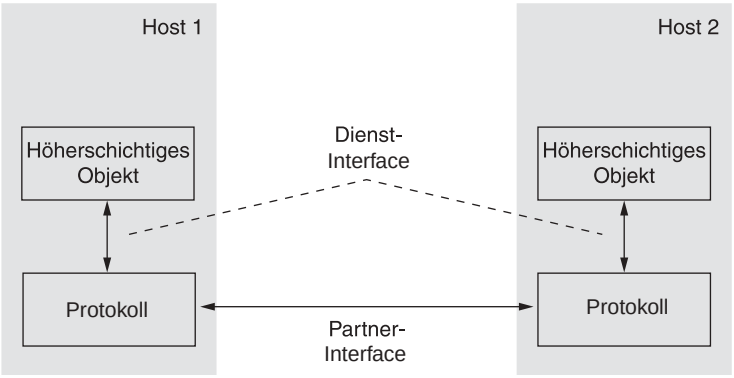

Abb. 1.10: Dienst- und Partner-Interface

Antwortprotokoll auf einem Rechner mit seinem Partner auf einem anderen kommuniziert. Mit anderen Worten: Ein Protokoll definiert einen Kommunikationsdienst, den es lokal anbietet, zusammen mit einer Reihe von Regeln für die Nachrichten, die das Protokoll mit seinem bzw. seinen Partner(n) austauscht, um diesen Dienst zu implementieren. Diese Situation ist in Abb. 1.10 dargestellt.

Abgesehen von der Hardwareebene, auf der die Partner direkt über eine Leitung miteinander kommunizieren, ist die Peer-zu-Peer-Kommunikation indirekt. Das heißt, jedes Protokoll kommuniziert mit *seinem* Partner durch Weitergabe von Nachrichten an das Protokoll einer niedrigeren Ebene, das seinerseits die Nachricht seinem Partner zustellt. Darüber hinaus gibt es potenziell mehrere Protokolle auf einer bestimmten Ebene, die jeweils einen anderen Kommunikationsdienst bieten. Wir stellen deshalb die Protokollfolge, aus der sich ein Netzwerksystem zusammensetzt, mit Hilfe eines *Protokollgraphen* dar. Die Knoten des Graphen entsprechen den Protokollen und die Verbindungslinien stellen eine *Abhängigkeitsbeziehung* dar. Abb. 1.11 zeigt z.B. einen Protokollgraphen für das oben beschriebene hypothetische Schichtensystem. Die Protokolle RRP (Request/Reply Protocol; Anfrage/Antwort-Protokoll) und MSP (Message Stream Protocol; Nachrichtenstromprotokoll) implementieren zwei unterschiedliche Arten von Prozess-zu-Prozess-Kanälen. Beide hängen vom HHP (Host-zu-Host-Protokoll) ab, das einen Verbindungsdienst zwischen zwei Hosts bereitstellt.

Bei diesem Beispiel wird davon ausgegangen, dass das Dateitransferprogramm auf Host 1 mit Hilfe des vom RRP-Protokoll gebotenen Kommunikationsdienstes eine Nachricht an seinen Partner auf Host 2 senden will. Die Anwendung der Datei fordert das RRP auf, die Nachricht zu senden. Um mit seinem Partner zu kommunizieren, ruft RRP dann die Dienste des HHP auf, das seinerseits die Nachricht an seinen Partner auf dem anderen Rechner überträgt. Nachdem die Nachricht beim HHP-Protokoll auf Host 2 angekommen ist, leitet HHP die Nachricht nach oben zum RRP weiter, das die Nachricht dem Dateitransferprogramm zustellt. Man sagt auch, die Anwendung benutzt einen so genannten *Protokoll-Stack* – in diesem Fall RRP/HHP.

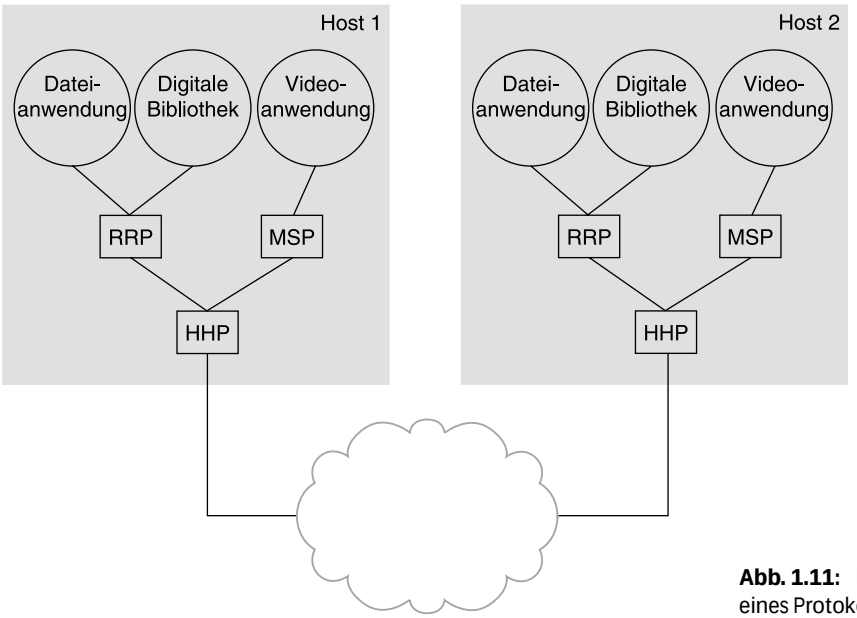

Abb. 1.11: Beispiel eines Protokollgraphen

Der Begriff *Protokoll* wird auf zweierlei Art verwendet. Zum einen bezieht er sich auf die abstrakten Interfaces, d.h. auf die vom Dienst-Interface definierten Operationen, sowie auf Form und Bedeutung der zwischen den Partnern ausgetauschten Nachrichten. Zum anderen bezieht er sich auf das Modul, das diese beiden Interfaces implementiert. Um zwischen den Interfaces und dem Modul, das diese Interfaces implementiert, zu unterscheiden, nennen wir erstere die *Protokollspezifikation*. Spezifikationen werden im Allgemeinen mit Hilfe einer Kombination aus Text, Pseudocode, Zustandsübergangsdiagrammen, Bildern von Paketformaten und anderen abstrakten Notationen ausgedrückt. Ein bestimmtes Protokoll kann von verschiedenen Programmierern auf unterschiedliche Art implementiert werden, solange sich jeder an die Spezifikation hält. Dabei muss sichergestellt werden, dass zwei verschiedene Implementierungen der gleichen Spezifikation erfolgreich Nachrichten austauschen können. Zwei oder mehr Protokollmodule, die eine Protokollspezifikation genau implementieren, können zusammenarbeiten und man sagt, sie *interoperieren*.

Man kann sich viele verschiedene Protokolle und Protokollgraphen vorstellen, die die Kommunikationsanforderungen einer Reihe von Anwendungen erfüllen. Glücklicherweise gibt es Standardisierungsgremien, z.B. die International Standards Organization (ISO) und die Internet Engineering Task Force (IETF), die Regeln und Richtlinien für bestimmte Protokollgraphen ausarbeiten. Insgesamt werden die Regeln, die Form und Inhalt eines Protokollgraphen vorgeben, *Netzwerkarchitektur* genannt. Standardisierungsaspekte gehen über Zweck und Umfang dieses Buchs hinaus. An dieser Stelle sei nur kurz erwähnt, dass Standardisierungsgremien wie ISO und IETF wohldefinierte Verfahren für die Einführung, Bewertung und Genehmi-

gung von Protokollen in der jeweiligen Architektur spezifiziert haben. Wir beschreiben die von ISO und IETF definierten Architekturen kurz. Zuerst müssen aber zwei weitere Aspekte über die Mechanismen eines Protokollgraphen erklärt werden.

Kapselung

Was passiert bei dem Beispiel in Abb. 1.11, wenn eines der Anwendungsprogramme eine Nachricht an das RRP-Protokoll abgibt, um sie an seinen Partner zu senden? Aus Sicht des RRP ist die Nachricht, die es von der Anwendung erhalten hat, eine nicht zu interpretierende Byte-Kette. Das RRP kümmert sich nicht darum, ob diese Bytes ein ganzzahliges Array, eine E-Mail, ein digitales Bild oder etwas anderes darstellen. Es hat schlichtweg den Auftrag, sie an seinen Partner zu senden. Das RRP muss seinem Partner aber Steuerinformationen übergeben, um ihm mitzuteilen, wie die Nachricht bei Empfang zu behandeln ist. Das RRP fügt also einen *Header* (Kopfteil) ein. Im allgemeinen ist ein Header eine kleine Datenstruktur von einigen wenigen bis zu ein paar Dutzend Byte, die von den Partnern benutzt werden, um miteinander zu kommunizieren. Wie der Name andeutet, wird der Header immer der eigentlichen Nachricht vorangestellt. Manchmal werden diese Steuerinformationen aber auch an das Ende der Nachricht angehängt. In diesem Fall handelt es sich um einen *Trailer* (Endteil). Das genaue Format für den vom RRP angehängten Header ist in seiner Protokollspezifikation definiert. Die eigentliche Nachricht, also die im Auftrag der Anwendung zu übertragenden Daten, nennt man *Nutzdaten*. Die Daten der Anwendung werden in der vom RRP-Protokoll für die Übertragung vorbereiteten Nachricht *gekapselt*.

Diese Kapselung von Daten wird dann auf jeder Ebene des Protokollgraphen wiederholt. Das HHP kapselt die vom RRP erhaltene Nachricht beispielsweise, indem es seinen eigenen Header anfügt. Wenn wir jetzt davon ausgehen, dass das HHP die Nachricht über ein Netzwerk an seinen Partner sendet, wird die Nachricht nach ihrer Ankunft beim Ziel-Host in der entgegengesetzten Reihenfolge verarbeitet: Das HHP entfernt zuerst seinen Header von der Nachricht, interpretiert ihn (unternimmt also eine entsprechend dem Inhalt des Headers geeignete Aktion) und leitet die Nutzdaten der Nachricht zum RRP weiter, das den von seinem Partner angehängten Header liest, die darin angegebenen Aktionen ausführt und die Nutzdaten der Nachricht an das Anwendungsprogramm weitergibt. Die vom RRP an die Anwendung auf Host 2 weitergegebene Nachricht ist absolut identisch mit derjenigen, die das Anwendungsprogramm nach unten zum RRP auf Host 1 abgegeben hat. Die Anwendung bekommt nie einen der Header zu sehen, die angehängt wurden, um die Kommunikationsdienste auf der niedrigeren Ebene zu realisieren. Dieser gesamte Prozess ist in Abb. 1.12 dargestellt. Man beachte, dass bei diesem Beispiel Knoten im Netzwerk (z.B. Switche und Router) den der Nachricht vorangestellten HHP-Header zur Erfüllung ihrer Aufgaben einsehen können.

Wenn das Protokoll einer niedrigeren Ebene die ihm vom Protokoll einer höheren Ebene abgegebene Nachricht nicht interpretiert, meinen wir damit, dass es nicht weiß, wie es aus den in der Nachricht enthaltenen Daten irgendeine Bedeutung

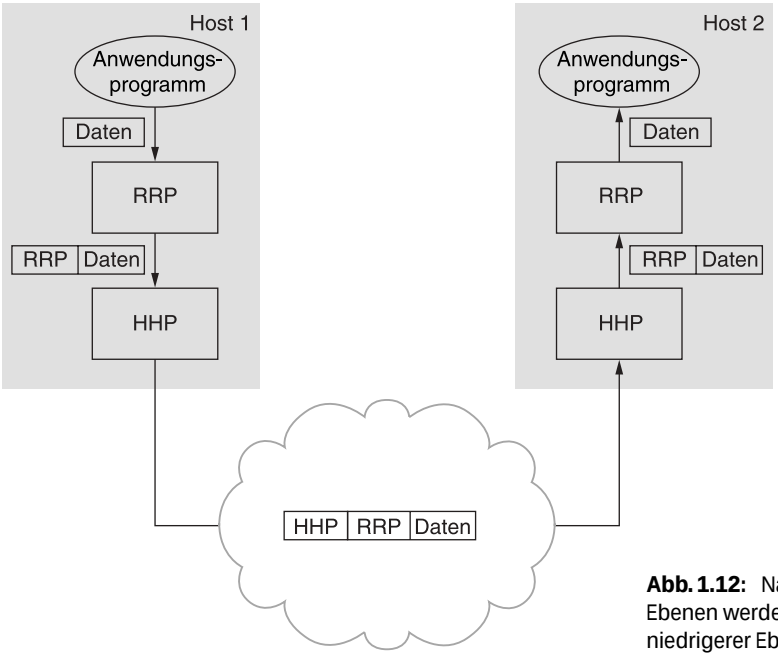

Abb. 1.12: Nachrichten höherer Ebenen werden in Nachrichten niedrigerer Ebenen gekapselt.

extrahieren kann. Manchmal jedoch wendet das Protokoll der niedrigeren Ebene eine einfache Operation auf die erhaltenen Daten an, um sie beispielsweise zu komprimieren oder zu verschlüsseln. In diesem Fall wandelt das Protokoll den gesamten Nutzdatenteil der Nachricht um, einschließlich der Daten der Ursprungsanwendung und aller Header, die von Protokollen höherer Ebenen an diese Daten angehängt wurden.

Multiplexen und Demultiplexen

Aus Abschnitt 1.2.2 ist bereits ein grundlegendes Konzept der Paketvermittlung bekannt: Das Multiplexen mehrerer Datenflüsse über eine einzige Verbindungsleitung. Das gleiche Konzept lässt sich nicht nur auf Vermittlungsknoten, sondern sowohl nach oben als auch nach unten im gesamten Protokollgraphen anwenden. Bei dem Beispiel in Abb. 1.11 könnte das RRP einen logischen Kommunikationskanal implementieren. Die Nachrichten würden von zwei unterschiedlichen Anwendungen über diesen Kanal am Quell-Host gemultiplext, um sie dann am Ziel-Host wieder auf die entsprechende Anwendung zu demultiplexen.

Aus praktischer Sicht bedeutet dies lediglich, dass der Header, den RRP an seine Nachrichten anhängt, einen Bezeichner enthält, der die Anwendung, an die die Nachricht gerichtet ist, identifiziert. Wir nennen diesen Bezeichner den *Demultiplexschlüssel* oder kurz *Demuxschlüssel* des RRP. Auf dem Quell-Host fügt das RRP den entsprechenden Demuxschlüssel in seinen Header ein. Wird die Nachricht dem RRP auf dem Ziel-Host zugestellt, entnimmt es seinen Header, prüft den Demuxschlüssel und demultiplext die Nachricht zur richtigen Anwendung.

Das RRP ist nicht das einzige Protokoll, das Multiplexen unterstützt. Fast jedes Protokoll benutzt diesen Mechanismus. Das HHP hat z.B. einen eigenen Demuxschlüssel, um festzustellen, welche Nachrichten zum RRP und welche zum MSP befördert werden sollen. Zwischen den Protokollen – auch nicht zwischen denen innerhalb einer einzelnen Netzarchitektur – besteht aber keine einheitliche Vereinbarung darüber, woraus genau ein Demuxschlüssel besteht. Einige Protokolle verwenden ein 8-Bit-Feld (was bedeutet, dass sie nur 256 Protokolle höherer Ebenen unterstützen können), während andere 16- oder 32-Bit-Felder verwenden. Ferner haben einige Protokolle ein einziges Demultiplexfeld in ihrem Header, während andere über mehrere solcher Felder verfügen. Ist nur ein Demultiplexfeld vorhanden, wird der gleiche Demuxschlüssel an beiden Enden der Kommunikation benutzt. Werden zwei Felder verwendet, benutzt jedes Ende einen anderen Schlüssel, um das Protokoll (bzw. Anwendungsprogramm) der höheren Ebene zu identifizieren, dem die Nachricht zuzustellen ist.

1.3.2 OSI-Architektur

Die ISO war eine der ersten Organisationen, die einen allgemeinen Weg, Computer zu verbinden, formal definiert hat. Die von der ISO definierte OSI-Architektur (*Open Systems Interconnection*) ist in Abb. 1.13 dargestellt. Sie spezifiziert die Aufteilung der Netzwerkfunktionalität in sieben Schichten, wobei eines oder mehrere Protokolle die Funktionalität einer bestimmten Schicht implementieren. In diesem Sinn ist die schematische Darstellung in Abb. 1.13 kein Protokollgraph an sich, sondern vielmehr ein *Referenzmodell* für einen Protokollgraphen. Die ISO hat, meist in Verbindung mit einer weiteren Standardisierungsorganisation, der International Telecommunications Union (ITU)[1], eine Reihe von Protokollspezifikationen auf der Grundlage der OSI-Architektur veröffentlicht. Diese Reihe nennt man auch »X Punkt«, weil Bezeichnungen wie X.25, X.400, X.500 usw. für die Protokolle vergeben wurden. Auf diesen Standards basieren verschiedene Netzwerke, darunter das öffentliche X.25-Netz und private Netzwerke wie Tymnet.

In der Betrachtung von unten nach oben handhabt die *Bitübertragungsschicht* (*Physical Layer*) die Übertragung eines Bitstroms über eine Verbindungsleitung. Die *Sicherungsschicht* (*Data Link Layer*) fasst Abschnitte dieses Bitstroms zu einem größeren Verbund zusammen, den man als *Frame* bezeichnet. Netzwerkadapter sowie Gerätetreiber, die Teil des Betriebssystem eines angeschlossenen Knotens sind, implementieren typischerweise die Sicherungsschicht. Dies bedeutet, dass den Hosts nicht der unstrukturierte Bitstrom, sondern Frames zugestellt werden. Die *Vermittlungsschicht* (*Network Layer*) handhabt das Routing zwischen den Knoten eines paketvermittelten Netzwerks. Auf dieser Schicht nennt man die zwischen Knoten ausgetauschte Dateneinheit nicht Frame, sondern *Paket*, obwohl kein fundamentaler Unterschied besteht. Die drei unteren Schichten werden in allen Netzknoten

1. Durch eine Untergruppe der ITU für Telekommunikation (ITU-T) wurde das ehemalige Comité Consultatif International de Télégraphique et Téléphonique (CCITT) abgelöst.

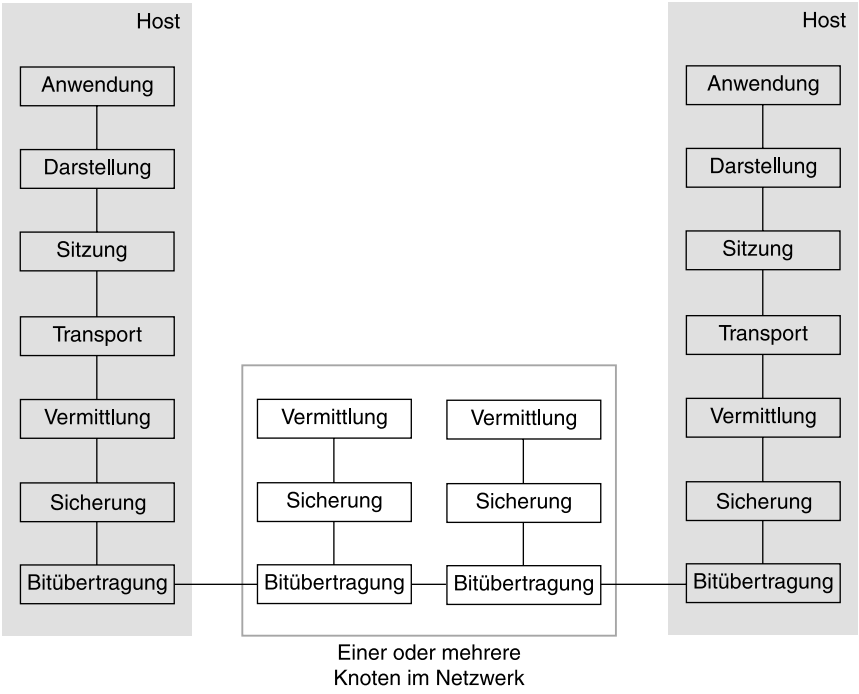

Abb. 1.13: OSI-Netzwerkarchitektur

sowie Switchen des Netzwerks implementiert. Die *Transportschicht* (*Transport Layer*) implementiert alles, was wir bisher mit »Prozess-zu-Prozess-Kanal« bezeichnet haben. Hier nennt man die ausgetauschte Dateneinheit *Nachricht* statt Paket oder Frame. Die Transportschicht und die höheren Schichten laufen normalerweise nur auf den Endsystemen und nicht auf den dazwischenliegenden Switchen oder Routern.

Über die Definition der oberen drei Schichten besteht weniger Einigkeit. Wir springen zur obersten (siebten) Schicht, der *Anwendungsschicht* (*Application Layer*). Ein Protokoll dieser Schicht ist z.B. das FTP (File Transfer Protocol), über das Dateitransfer-Anwendungen interoperieren können. Darunter befindet sich die *Darstellungsschicht* (*Presentation Layer*), die sich mit dem Format der zwischen Partnern ausgetauschten Daten beschäftigt, beispielsweise, ob eine Ganzzahl 16, 32 oder 64 Bit lang ist, das werthöchste Bit zuerst oder zuletzt übertragen wird, oder wie ein Videostrom formatiert werden muss. Schließlich bietet die *Sitzungsschicht* (*Session Layer*) einen Namensraum, der für die Verknüpfung potenziell unterschiedlicher Transportströme, die Teil einer einzelnen Anwendung sind, benutzt wird. Dies kann z.B. die Handhabung eines Audio- und Videostroms sein, die in einer Telekonferenzanwendung kombiniert werden müssen.

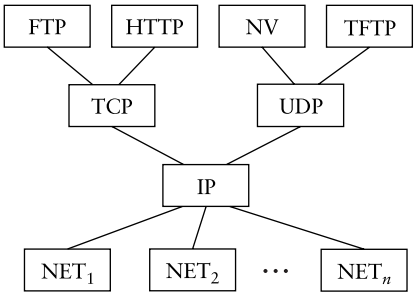

Abb. 1.14: Darstellung der Internet-Architektur als Protokollgraph

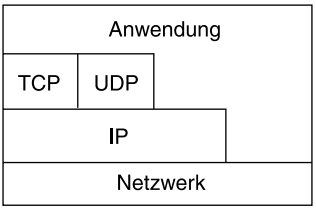

Abb. 1.15: Alternative Betrachtung der Internet-Architektur

1.3.3 Internet-Architektur

Die Internet-Architektur, die man nach ihren beiden wichtigsten Protokollen auch TCP/IP-Architektur nennt, ist in Abb. 1.14 wiedergegeben. Eine alternative Darstellung erscheint in Abb. 1.15. Die Internet-Architektur entstand aus Erfahrungen mit dem älteren paketvermittelten Netzwerk ARPANET. ARPANET und das Internet wurden von der Advanced Research Projects Agency (ARPA), im Rahmen von Forschungsprojekten des US-Verteidigungsministeriums gefördert. Sowohl das ARPANET als auch das Internet gab es schon vor der OSI-Architektur, sodass sie einen entscheidenden Einfluss auf das OSI-Referenzmodell hatten.

Die sieben Schichten des OSI-Modells lassen sich mit einem gewissen Maß an Vorstellungskraft auf die Internet-Architektur übertragen; sie umfasst aber nur vier Schichten. Auf der untersten Ebene gibt es zahlreiche Netzprotokolle, die als NET_1, NET_2 usw. bezeichnet werden. In der Praxis werden diese Protokolle dadurch implementiert, dass man Hardware (z.B. einen Netzadapter) und Software (z.B. einen Gerätetreiber für den Netzadapter) kombiniert. Man findet auf dieser Schicht beispielsweise Ethernet- oder FDDI-Protokolle (Fiber Distributed Data Interface). (Diese Protokolle können ihrerseits mehrere Unterschichten umfassen, die Internet-Architektur gibt für sie aber nichts vor.) Auf der zweiten Schicht gibt es nur das *Internet Protocol* (IP). Dieses Protokoll unterstützt den Zusammenschluss mehrerer Netztechnologien zu einem einzigen logischen Netzverbund (Internetwork). Die dritte Schicht umfasst das TCP (*Transmission Control Protocol*) und das UDP (*User Datagram Protocol*). Diese beiden wichtigen Protokolle stellen alternative logische Kanäle zu Anwendungsprogrammen bereit: TCP bietet einen zuverlässigen Byte-Strom-Kanal und UDP einen unzuverlässigen Datagramm-Kanal (*Datagram* ist lediglich ein anderer Begriff für »Nachricht«). Im Internet-Slang nennt man TCP und UDP auch *Ende-zu-Ende-Protokolle*, man kann sie aber auch als »Transportprotokolle« bezeichnen.

Oberhalb der Transportschicht laufen verschiedene Anwendungsprotokolle, z.B. FTP, TFTP (Trivial File Transport Protocol), Telnet (Remote Login) und SMTP (Simple Mail Transfer Protocol für E-Mail). Sie ermöglichen die Interoperation

populärer Anwendungen. Um den Unterschied zwischen einem Protokoll der Anwendungsschicht und einer Anwendung zu verdeutlichen, stelle man sich die Vielzahl verfügbarer WWW-Browser (z.B. Mosaic, Netscape, Internet Explorer, Lynx usw.) vor. Es gibt eine ebenso große Anzahl an verschiedenen Implementierungen von Webservern. Der Grund dafür, dass man ein beliebiges dieser Anwendungsprogramme benutzen kann, um auf eine bestimmte Seite im Web zuzugreifen, liegt daran, dass sie alle dem gleichen Protokoll auf der Anwendungsschicht gehorchen – dem HTTP (HyperText Transport Protocol). Verwirrend ist manchmal, dass die gleiche Bezeichnung für eine Anwendung und ein Protokoll der Anwendungsschicht verwendet wird (z.B. FTP).

Die Internet-Architektur weist drei Merkmale auf, deren nähere Betrachtung sich lohnt. Wie aus Abb. 1.15 ersichtlich wird, impliziert die Internet-Architektur kein strenges Schichtenmodell. Einer Anwendung steht es frei, die definierten Transportschichten zu umgehen und IP oder eines der darunter liegenden Netzwerke direkt zu nutzen. Anders betrachtet, können Programmierer beliebig neue Kanalabstraktionen oder Anwendungen entwickeln, die auf irgendeinem der vorhandenen Protokolle aufsetzen.

Sieht man sich den Protokollgraphen in Abb. 1.14 genauer an, erkennt man die Form einer Eieruhr – oben breit, in der Mittel schmal und unten wieder breit. Diese Form spiegelt die zentrale Philosophie der Architektur wider. Sie besagt, dass IP als Mittelpunkt der Architektur dient. IP definiert eine gemeinsame Methode für den Austausch von Paketen zwischen unterschiedlichsten Netzwerken. Auf IP können beliebig viele Transportprotokolle aufsetzen, die Anwendungsprogrammen jeweils eine andere Kanalabstraktion bieten. Das heißt beispielsweise, dass die Frage der Zustellung von Nachrichten von einem Host zu einem anderen völlig abgetrennt ist von der Frage der Bereitstellung eines nützlichen Kommunikationsdienstes zwischen Prozessen. Unterhalb von IP lässt die Architektur beliebig viele unterschiedliche Netztechnologien zu, die von Ethernet über FDDI und ATM bis zu Punkt-zu-Punkt-Leitungen reichen.

Ein letztes Merkmal der Internet-Architektur (oder genauer, der IETF-Kultur) betrifft die Entwicklung neuer Protokolle. Möchte jemand einen Vorschlag für ein neues Protokoll in der Architektur berücksichtigt sehen, muss er eine Protokollspezifikation und mindestens eine (vorzugsweise zwei) repräsentative Implementierung der Spezifikation vorlegen. Voraussetzung für die Annahme eines Standards durch die IETF sind funktionierende Implementierungen. Dieses ungeschriebene Gesetz in der Entwicklungsgemeinde trägt zur Gewährleistung bei, dass die Protokolle der Architektur effizient implementiert werden können. Vielleicht wird der Wert, den die Inernet-Kultur für funktionierende Software hat, am besten durch ein Zitat illustriert, welches sich häufig auf den T-Shirts von Teilnehmern an IETF-Meetings findet:

> We reject kings, presidents, and voting. We believe in rough consensus and running code. Dave Clark

1 **Von diesen drei Attributen der Internet-Architektur ist die Philosophie des Sanduhr-Designs wichtig genug, um sie nochmal zu betonen. Die schlanke Taille der Sanduhr stellt eine minimale, sorgfältig gewählte Menge globaler Fähigkeiten dar, die es sowohl höherschichtigen Anwendungen als auch niederschichtigen Kommunikationstechnologien erlaubt, nebeneinander zu existieren, Fähigkeiten gemeinsam zu nutzen und sich rasch weiterzuentwickeln. Das Modell mit der schlanken Taille ist von entscheidender Bedeutung für die Fähigkeit des Internet, sich schnell an neue Benutzerforderungen und sich ändernde Technologien anzupassen.**

1.4 Implementierung von Netzsoftware

Netzarchitekturen und Protokollspezifikationen sind wichtig, aber auch ein guter Entwurf reicht nicht aus, um den phänomenalen Erfolg des Internet zu erklären: Die Anzahl der an das Internet angeschlossenen Computer hat sich seit 1981 jedes Jahr verdoppelt und nähert sich heute etwa 200 Millionen. Schätzungsweise benutzen weit über 600 Millionen Menschen das Internet. Man nimmt an, dass die Anzahl der über das Internet übertragenen Bits die entsprechende Zahl beim konventionellen Telefonnetz etwa seit dem Jahre 2001 übertrifft.

Was erklärt den Erfolg des Internet? Sicherlich gibt es viele Faktoren, die dazu beitragen (einschließlich einer guten Architektur). Was dem Internet zu diesem nie dagewesenen Erfolg verholfen hat, ist aber die Tatsache, dass ein Großteil seiner Funktionalität von Software bereitgestellt wird, die auf jedem Allzweckcomputer läuft. Das bedeutet, dass jederzeit mit einem »geringem Programmieraufwand« neue Funktionalität hinzugefügt werden kann. Aus diesem Grund haben sich neue Anwendungen und Dienste, wie elektronischer Handel, Videokonferenz und Internet-Telefonie, um nur einige zu nennen, in derart raschem Tempo entwickelt.

Ein damit zusammenhängender Faktor ist die enorme Steigerung der Rechenleistung von Personalcomputern. Obwohl Rechnernetze im Prinzip immer in der Lage waren, diese Datenart, z.B. digitale Sprachmuster, digitalisierte Bilder usw., zu befördern, war dieses Potenzial nicht von besonderem Interesse, als die Computer, die Daten sendeten und empfingen, zu langsam waren, um mit den Informationen etwas anfangen zu können. Praktisch alle heutigen Computer sind in der Lage, digitalisierte Sprache in voller Geschwindigkeit wiederzugeben und Video in einer Geschwindigkeit und Auflösung darzustellen, die für einige (bei weitem aber nicht alle) Anwendungen nützlich sind. Die heutigen Netzwerke haben begonnen, Multimedia zu unterstützen, und diese Unterstützung wird sich natürlich verbessern, wenn die Rechnerhardware noch schneller wird.

Wir schlussfolgern daraus, dass es unabdingbar ist zu wissen, wie Netzwerksoftware richtig implementiert wird, um Rechnernetze voll zu verstehen. Vor diesem Hintergrund werden in diesem Abschnitt zuerst einige Punkte vorgestellt, die sich

mit der Implementierung eines auf ein Netzwerk aufsetzenden Anwendungsprogramms befassen. Dann werden Fragen zur Implementierung der im Netzwerk laufenden Protokolle behandelt. In vielerlei Hinsicht ähneln sich Netzanwendungen und Netzprotokolle. Anwendungen nutzen die Dienste des Netzwerks ähnlich wie das Protokoll einer höheren Ebene die Dienste des Protokolls einer niedrigeren Ebene in Anspruch nimmt. Abschließend werden in diesem Abschnitt aber auch ein paar wichtige Unterschiede behandelt.

1.4.1 APIs und Sockets

Ein guter Ausgangspunkt bei der Implementierung einer Netzanwendung ist das vom Netzwerk angebotene Interface. Da die meisten Netzprotokolle in Software implementiert werden (insbesondere die oberen im Protokoll-Stack) und fast alle Computersysteme ihre Netzprotokolle als Teil des Betriebssystems implementieren, meinen wir mit dem Ausdruck »vom Netzwerk angebotenes Interface« im Allgemeinen das Interface, welches das Betriebssystem für sein Kommunikationssubsystem bereitstellt. Dieses Interface wird meist als *Netzwerk-API* (*Application Programming Interface*) bezeichnet.

Obwohl jedes Betriebssystem sein eigenes Netzwerk-API definieren kann (was bei den meisten der Fall ist), hat die Unterstützung einiger weniger dieser APIs im Lauf der Zeit stark zugenommen. Das heißt, sie wurden auf andere Betriebssysteme als nur ihr ursprüngliches portiert. Das genau passierte mit dem *Socket-Interface*, das ursprünglich in der Berkeley-Distribution von Unix enthalten war und heute von praktisch allen gängigen Betriebssystemen unterstützt wird. Die industrieweite Unterstützung eines einzelnen API hat den Vorteil, dass Anwendungen leichter von einem Betriebssystem auf ein anderes portiert werden können. Man darf hier aber nicht vergessen, dass Anwendungsprogramme normalerweise mit vielen Teilen des Betriebssystems interagieren, die nichts mit Vernetzung zu tun haben, z.B. Dateien lesen und schreiben, identische Kopien von Prozessen erzeugen oder Daten grafisch ausgeben. Allein auf Grund dessen, dass zwei Systeme das gleiche Netzwerk-API unterstützen, müssen ihre Dateisystem-, Prozess- und Grafik-Interfaces nicht unbedingt gleich sein. Dennoch bieten uns akzeptierte APIs wie die Unix-Sockets einen guten Ausgangspunkt.

Vor der Beschreibung des Socket-Interface ist es wichtig, zwei grundlegende Konzepte bewusst zu trennen. Jedes Protokoll bietet bestimmte *Dienste*, und das API bietet eine *Syntax*, mit der diese Dienste im betreffenden Betriebssystem aufgerufen werden können. Die Implementierung ist dann für die Abbildung der vom API definierten Operationen und Objekte auf die vom Protokoll definierten abstrakten Dienste zuständig. Ist ein Interface gut definiert, so ist es möglich, seine Syntax zu benutzen, um die Dienste vieler verschiedener Protokolle aufzurufen. Eine derartige Generalität war beim Socket-Interface sicherlich beabsichtigt, wenn es auch weit davon entfernt ist, perfekt zu sein.

Die wichtigste Abstraktion des Socket-Interface ist – wie könnte es anders sein – der Socket. Ein *Socket* kann man sich als Punkt vorstellen, an dem ein lokaler

Anwendungsprozess sich mit dem Netzwerk verbindet. Das Interface definiert Operationen zum Erstellen eines Sockets, Anbinden des Sockets an das Netzwerk, Senden/Empfangen von Nachrichten über den Socket und Schließen des Sockets. Der Einfachheit halber beschränken wir uns hier auf die Nutzung von Sockets mit TCP.
Im ersten Schritt wird mit folgender Operation ein Socket erzeugt:

```
int socket(int domain, int type, int protocol)
```

Diese Operation nimmt drei Argumente an, weil das Socket-Interface für die allgemeine Nutzung ausgelegt wurde, um beliebige Protokoll-Stacks zu unterstützen. Das Argument domain spezifiziert die zu verwendende Protokoll*familie*. PF_INET wird benutzt, um die Internet-Familie zu bezeichnen; PF_UNIX ist eine Alternative, mit der lokale Prozesskommunikation unter Unix bezeichnet wird. Das Argument type zeigt die Semantik der Kommunikation an. SOCK_STREAM wird benutzt, um einen Byte-Strom zu bezeichnen. SOCK-DGRAM ist eine Alternative, die einen nachrichtenorientierten Dienst, z.B. den vom UDP bereitgestellten, bezeichnet. Das Argument protocol identifiziert das benutzte Protokoll. In unserem Fall lautet dieses Argument UNSPEC, weil die Kombination aus PF_INET und SOCK_STREAM bereits eindeutig identifiziert, dass wir TCP benutzen. Schließlich ist der Rückgabewert von socket ein »*Handle*« für den neu erstellten Socket, also ein Bezeichner, mit dem wir künftig auf den Socket verweisen können. Er steht als Argument für nachfolgende Operationen auf diesem Socket zur Verfügung.

Der nächste Schritt hängt davon ab, ob wir es mit einem Client oder Server zu tun haben. Bei einem Server führt der Anwendungsprozess ein *passives* Öffnen durch. Das heißt, der Server teilt mit, dass er für die Annahme von Verbindungen bereit ist, baut aber keine Verbindung auf. Hierfür ruft der Server die folgenden drei Operationen auf:

```
int bind(int socket, struct sockaddr *address, int addr_len)
int listen(int socket, int backlog)
int accept(int socket, struct sockaddr *address, int *addr_len)
```

Wie der Name andeutet, bindet die Operation bind den neu erstellten Socket (socket) an die angegebene Adresse (address). Dies ist die Netzwerkadresse des lokalen Teilnehmers, also des Servers. Man beachte, dass address bei Benutzung der IP-Schicht eine Datenstruktur ist, die sowohl die IP-Adresse des Servers als auch eine TCP-Portnummer beinhaltet. (Wie in Kapitel 5 beschrieben wird, werden Ports benutzt, um Prozesse indirekt zu identifizieren. Sie sind mit den in Abschnitt 1.3.1 beschriebenen Demuxschlüsseln vergleichbar.) Die Portnummer ist normalerweise eine spezifische, für den anzubietenden Dienst wohlbekannte Nummer. Webserver nehmen beispielsweise Verbindungen über Port 80 an.

Die Operation listen definiert, wie viele Verbindungen am bezeichneten Socket (socket) anstehen können. Die Operation accept führt das passive Öffnen aus. Hierbei handelt es sich um eine blockierende Operation, die erst zurückkehrt, wenn ein

entfernter Teilnehmer eine Verbindung aufgebaut hat. Bei ihrer Rückkehr gibt sie einen *neuen* Socket zurück, der der gerade aufgebauten Verbindung entspricht. Das Argument address enthält die Adresse des *entfernten* Teilnehmers. Wenn accept beendet ist, ist der ursprüngliche Socket, der als Argument weitergereicht wurde, immer noch vorhanden und steht nach wie vor zum passiven Öffnen zu künftigen Aufrufen von accept bereit.

Auf der Client-Maschine führt der Anwendungsprozess ein *aktives* Öffnen durch. Das heißt, er ruft die folgende Operation auf, um mitzuteilen, mit wem er kommunizieren möchte:

```
int connect(int socket, struct sockaddr *address, int addr_len)
```

Diese Operation kehrt erst zurück, wenn TCP erfolgreich eine Verbindung aufgebaut hat. Dann kann die Anwendung mit dem Senden von Daten beginnen. In diesem Fall enthält address die Adresse des entfernten Teilnehmers und überlässt es dem System, die lokalen Informationen einzutragen. Während ein Server normalerweise an einem bekannten Port auf Nachrichten wartet (listen), kümmert sich ein Client nicht darum, welchen Port er für sich selbst nutzt. Das Betriebssystem wählt einfach einen freien Port aus.

Nachdem eine Verbindung aufgebaut wurde, rufen die Anwendungsprozesse die folgenden beiden Operationen auf, um Daten zu senden und zu empfangen:

```
int send(int socket, char *message, int msg_len, int flags)
int recv(int socket, char *buffer, int buf_len, int flags)
```

Die erste Operation sendet die betreffende Nachricht (message) über den angegebenen Socket (socket), während die zweite eine Nachricht vom angegebenen Socket (socket) empfängt und im angegebenen Puffer (buffer) ablegt. Beide Operationen erlauben die Benutzung verschiedener Schalter (flags), die bestimmte Einzelheiten der Operation steuern.

1.4.2 Anwendungsbeispiel

Wir beschreiben die Implementierung eines einfachen Client/Server-Programms, das das Socket-Interface benutzt, um Nachrichten über eine TCP-Verbindung zu senden. Das Programm benutzt noch weitere netzwerkspezifische Unix-Utilities, die wir im Verlauf der Beschreibung erklären. Unsere Anwendung gestattet es einem Benutzer an einem Rechner, Text einzugeben und an einen anderen Benutzer an einem anderen Rechner zu senden. Es handelt sich um eine vereinfachte Version des Unix-Programms talk, welches wiederum als Vorläufer von Chat-Rooms im World Wide Web betrachtet werden kann.

Client

Wir beginnen mit dem Client, der den Namen des entfernten Hosts als Kommandozeilenargument übergeben bekommt. Er ruft die Unix-Utility gethostbyname auf, um diesen Namen in die IP-Adresse des entfernten Hosts zu übersetzen. Im nächsten

Schritt wird die Datenstruktur (sin) der Adresse initialisiert, die vom Socket-Interface erwartet wird. Diese Datenstruktur spezifiziert, dass wir den Socket für Verbindungen zum Internet (AF_INET) verwenden werden. In unserem Beispiel benutzen wir TCP-Port 5432 als Server-Port. Dieser Port wurde noch keinem anderen Internet-Dienst zugewiesen. Im letzten Schritt beim Einrichten der Verbindung werden socket und connect aufgerufen. Wenn die Operation connect zurückgekehrt, ist die Verbindung aufgebaut und das Client-Programm tritt in seine Hauptschleife ein, in der Text aus einer Standardeingabe gelesen und über den Socket gesendet wird.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 5432
#define MAX_LINE 256

int
main(int argc, char * argv[])
{
  FILE *fp;
  struct hostent *hp;
  struct sockaddr_in sin;
  char *host;
  char buf[MAX_LINE];
  int s;
  int len;

if (argc==2) {
  host = argv[1];
}
else {
  fprintf(stderr, "usage: simplex-talk host\n");
  exit(1);
}

/* Übersetze Host-Name in IP-Adresse des Partners */
hp = gethostbyname(host);
if (!hp) {
  fprintf(stderr, "simplex-talk: unknown host: %s\n", host);
  exit(1);
}

/* Initialisiere die Datenstruktur für die Adresse */
bzero((char *)&sin, sizeof(sin));
sin.sin_family = AF_INET;
bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
sin.sin_port = htons(SERVER_PORT);
```

```
/* Aktives Öffnen */
if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
  perror("simplex-talk: socket");
  exit(1);
}
if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
  perror("simplex-talk: connect");
  close(s);
  exit(1);
}
/* Hauptschleife: Lies und sende Textzeilen */
while (fgets(buf, sizeof(buf), stdin)) {
  buf[MAX_LINE-1] = '\0';
  len = strlen(buf) + 1;
  send(s, buf, len, 0);
  }
}
```

Server

Der Server ist ebenso einfach. Er initialisiert zuerst die Datenstruktur für die Adresse, indem er seine eigene Portnummer (SERVER_PORT) einträgt. Da das Anwendungsprogramm keine IP-Adresse angibt, kann es Verbindungen auf jeder beliebigen IP-Adresse des lokalen Hosts annehmen. Anschließend führt der Server die Vorbereitungsschritte für ein passives Öffnen aus: Erstellen des Sockets, Binden des Sockets an die lokale Adresse und Einrichten der maximalen Anzahl anstehender Verbindungen. Schließlich wartet die Hauptschleife, bis ein entfernter Host versucht, eine Verbindung zum Server aufzubauen. Geschieht dies, gibt der Server den vom Client über die Verbindung gesendeten Text aus.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 5432
#define MAX_PENDING 5
#define MAX_LINE    256

int
main()
{
  struct sockaddr_in sin;
  char buf[MAX_LINE];
  int len;
  int s, new_s;
```

```
    /* Initialisiere Datenstruktur für die Adresse */
    bzero((char *)&sin, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons(SERVER_PORT);

    /* Richte passives Öffnen ein */
    if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
      perror("simplex-talk: socket");
      exit(1);
    }
    if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
      perror("simplex-talk: bind");
      exit(1);
    }
    listen(s, MAX_PENDING);

    /* Warte auf Verbindung, dann empfange und drucke Text */
    while(1) {
      if ((new_s = accept(s, (struct sockaddr *)&sin, &len)) < 0) {
        perror("simplex-talk: accept");
        exit(1);
      }
      while (len = recv(new_s, buf, sizeof(buf), 0))
        fputs(buf, stdout);
      close(new_s);
    }
  }
```

1.4.3 Aspekte der Protokollimplementierung

Wie eingangs in diesem Abschnitt erwähnt, kann man die Art, wie Anwendungsprogramme mit dem zu Grunde liegenden Netzwerk interagieren, mit der Interaktion der Protokolle auf höherer und niedrigerer Ebene vergleichen. TCP benötigt beispielsweise ein Interface, um abgehende Nachrichten an IP zu senden. IP muss wiederum in der Lage sein, TCP ankommende Nachrichten zuzustellen. Das ist das in Abschnitt 1.3.1 beschriebene Dienst-Interface.

Da wir bereits über ein Netzwerk-API (z.B. Sockets) verfügen, sind wir möglicherweise versucht, das gleiche Interface zwischen jedem Protokollpaar im Protokoll-Stack zu benutzen. Das wäre sicherlich eine Möglichkeit, wird aber in der Praxis nicht so gehandhabt. Das liegt daran, dass das Socket-Interface einige Ineffizienzen aufweist, die von den Protokoll-implementierenden Entwicklern nicht toleriert werden. Anwendungsprogrammierern sind sie ganz bequem, weil sie ihre Programmieraufgabe vereinfachen und die Ineffizienzen jeweils nur einmal toleriert werden müssen. Die Entwickler, die die entsprechenden Protokolle implementieren, sind demgegenüber stark an der Performance interessiert und müssen berücksichti-

gen, dass eine Nachricht durch mehrere Protokollschichten zu befördern ist. Im restlichen Teil dieses Abschnitts werden zwei primäre Unterschiede zwischen Netzwerk-APIs und dem Interface zwischen zwei Protokollen, die sich tiefer im Protokollgraphen befinden, beschrieben. Außerdem werden einige häufig in Protokollimplementierungen benutzte Bibliotheksroutinen vorgestellt.

Prozessmodell

Die meisten Betriebssysteme bieten eine Abstraktion, die man als *Prozess* oder *Thread* bezeichnet. Jeder Prozess läuft größtenteils unabhängig von anderen Prozessen. Das Betriebssystem ist dafür zuständig, dass die entsprechenden Ressourcen wie Adressraum und CPU-Zyklen allen aktiven Prozessen zugeteilt werden. Die Prozessabstraktion vereinfacht die gleichzeitige Ausführung von Aufgaben auf einem Rechner. Beispielsweise muss jede Benutzeranwendung ihren eigenen Prozess ausführen, während verschiedene Aufgaben innerhalb des Betriebssystems andere Prozesse ausführen. Stoppt das Betriebssystem die Ausführung eines Prozesses in der CPU und beginnt es mit einem neuen, nennt man dies *Prozesswechsel* (Context Switch).

Bei der Entwicklung des Kommunikationssubsystems muss man frühzeitig die Frage beantworten: »Wo sind die Prozesse?« Im wesentlichen gibt es dabei zwei Möglichkeiten, wie aus Abb. 1.16 ersichtlich wird. Bei der ersten, die wir *Prozess-pro-Protokoll-Modell* nennen, wird jedes Protokoll durch einen getrennten Prozess implementiert. Das heißt, eine Nachricht, die sich im Protokoll-Stack nach oben oder unten bewegt, wird von einem Prozess/Protokoll zu einem anderen weitergegeben. Der Prozess, der Protokoll *i* implementiert, verarbeitet die Nachricht und gibt sie an Protokoll *i*-1 ab usw. Wie ein Prozess/Protokoll eine Nachricht an das nächste Pro-

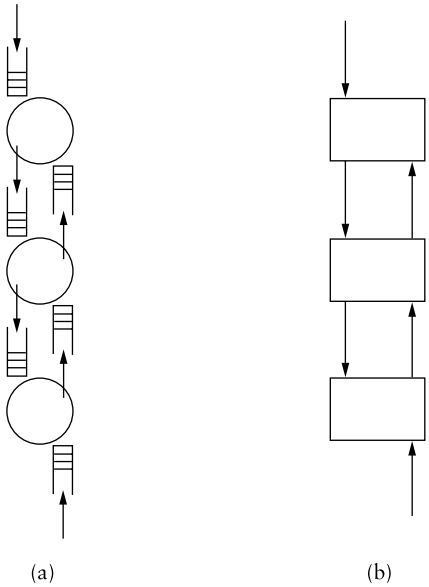

Abb. 1.16: Mögliche Prozessmodelle;
(a) Prozess-pro-Protokoll,
(b) Prozess-pro-Nachricht

zess/Protokoll-Paar abgibt, hängt davon ab, wie das Betriebssystem des Hosts die prozessübergreifende Kommunikation unterstützt. Meist ist ein einfacher Mechanismus vorhanden, mit dem sich eine Nachricht in die Warteschlange eines Prozesses einreihen lässt. Wichtig ist dabei, dass auf jeder Ebene des Protokollgraphen ein Prozesswechsel – eine relativ zeitaufwändige Operation – erforderlich ist.

Die zweite Möglichkeit, die wir *Prozess-pro-Nachricht-Modell* nennen, behandelt jedes Protokoll als statischen Code und verknüpft die Prozesse mit den Nachrichten. Das heißt, wenn eine Nachricht vom Netzwerk ankommt, teilt das Betriebssystem einem Prozess die Zuständigkeit für die Nachricht auf dem Weg nach oben im Protokollgraphen zu. Auf jeder Ebene wird die Prozedur aufgerufen, die das betreffende Protokoll implementiert, bis die letzte Ebene erreicht ist. Bei abgehenden Nachrichten ruft der Prozess der Anwendung die nötigen Prozeduren auf, bis die Nachricht zugestellt wurde. In beiden Richtungen wird der Protokollgraph in einer Folge von Prozeduraufrufen durchschritten.

Das Prozess-pro-Protokoll-Modell ist vom Konzept her leichter verständlich: »Ich implementiere mein Protokoll in meinem Prozess und du implementierst dein Protokoll in deinem Prozess«. Demgegenüber ist das Prozess-pro-Nachricht-Modell aus einem einfachen Grund generell effizienter: Auf den meisten Computern ist ein Prozeduraufruf um eine Größenordnung effizienter als ein Prozesswechsel. Beim ersten Modell ist auf jeder Ebene ein aufwändiger Prozesswechsel nötig, während beim zweiten Modell nur ein Prozeduraufruf pro Ebene erforderlich ist.

Nun stelle man sich die Beziehung zwischen dem oben definierten Dienst-Interface und dem Prozessmodell vor. Bei einer abgehenden Nachricht ruft das Protokoll der höheren Ebene eine send-Operation im Protokoll der niedrigeren Ebene auf. Da das höhere Protokoll beim Aufruf von send über die Nachricht verfügt, lässt sich diese Operation leicht als Prozeduraufruf implementieren. Ein Prozesswechsel ist nicht erforderlich. Bei ankommenden Nachrichten ruft das Protokoll der höheren Ebene die receive-Operation für das niedrigere Protokoll auf und muss dann warten, bis irgendwann eine Nachricht ankommt. Dadurch ist ein Prozesswechsel unabdingbar. Mit anderen Worten: Der laufende Prozess des höheren Protokolls empfängt eine Nachricht von dem Prozess, der im Protokoll der tieferen Ebene läuft. Das ist kein Problem, wenn nur der Anwendungsprozess Nachrichten vom Kommunikationssubsystem empfängt. Es ist ja das richtige Interface für das Netzwerk-API, weil Anwendungsprogramme ohnehin eine prozessorientierte Sicht der Dinge besitzen. Es hat aber große Auswirkungen auf die Leistung, wenn eine solche Prozessumschaltung auf jeder Ebene des Protokoll-Stacks erfolgt.

Aus diesem Grund wird in den meisten Protokollimplementierungen die receive- durch eine deliver-Operation ersetzt. Das heißt, das Protokoll der tieferen Ebene führt einen *Upcall* aus, ruft also eine Prozedur weiter oben im Protokoll-Stack auf, um die Nachricht dem höheren Protokoll zuzustellen. Abb. 1.17 zeigt das daraus resultierende Interface zwischen zwei benachbarten Protokollen, in diesem Fall TCP und IP. Im allgemeinen bewegen sich Nachrichten im Protokoll-Stack nach unten durch eine Reihe von send-Operationen und nach oben durch eine Reihe von deliver-Operationen.

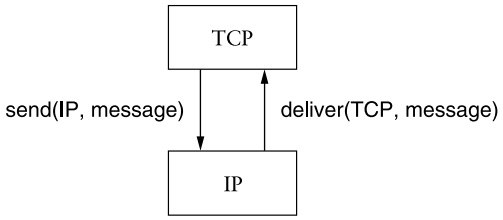

Abb. 1.17: Protokoll-zu-Protokoll-Interface

Nachrichtenpuffer

Eine weitere Ineffizienz des Socket-Interface ist, dass der Anwendungsprozess den Puffer bereitstellt, der die abgehende Nachricht beim Aufruf von send enthält. Genauso stellt er auch den Puffer bereit, in den eine ankommende Nachricht beim Aufruf der receive-Operation kopiert wird. Dies zwingt das oberste Protokoll dazu, die Nachricht vom Puffer der Anwendung in einen Netzwerkpuffer, und umgekehrt, zu kopieren, wie aus Abb. 1.18 ersichtlich wird. Das Kopieren von Daten von einem Puffer in einen anderen ist eine der aufwändigsten Aktionen einer Protokollimplementierung. Das liegt daran, dass die Geschwindigkeit von Speicherelementen bei weitem nicht so rasant fortschreitet wie bei Prozessoren.

Statt die Daten von Nachrichten auf jeder Ebene des Protokoll-Stacks von einem Puffer in einen anderen zu kopieren, definieren die meisten Kommunikationssubsysteme eine Nachrichtenabstraktion, die für alle Protokolle im Protokollgraphen verwendet wird. Diese Abstraktion ermöglicht nicht nur die Weitergabe von Nachrichten nach oben und unten im Protokollgraphen, ohne sie kopieren zu müssen, sondern bietet oft auch eine kopierfreie Möglichkeit, Nachrichten auf andere Weise zu manipulieren, z.B. Einfügen und Entnehmen von Headern, Aufteilen großer Nachrichten in mehrere kleine und Zusammensetzen dieser kleinen zu einer großen Nachricht. Die genaue Form dieser Nachrichtenabstraktion unterscheidet sich je nach Betriebssystem, sie enthält aber generell eine verlinkte Liste von Zeigern (Pointer), ähnlich wie es in Abb 1.19 dargestellt ist. Wir überlassen es dem Leser als Übung, eine allgemeine, kopienfreie Nachrichtenabstraktion zu definieren.

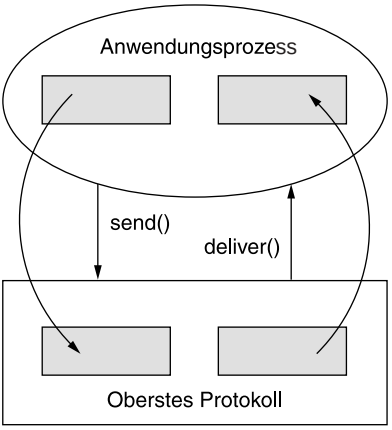

Abb. 1.18:
Kopieren ankommender/abgehender Nachrichten zwischen den Puffern der Anwendung und des Netzwerks

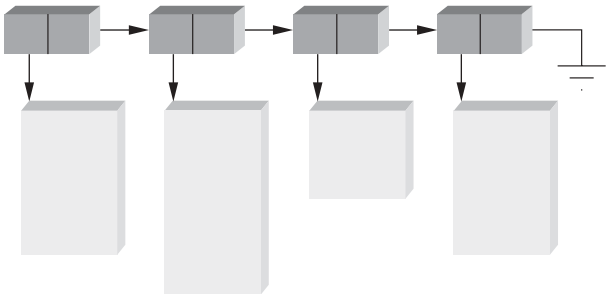

Abb. 1.19:
Beispiel einer Nachrichtendatenstruktur

1.5 Leistung

Bis zu diesem Punkt haben wir uns vorwiegend auf die funktionalen Aspekte eines Netzwerks konzentriert. Allerdings wird wie von jedem Computersystem auch von Rechnernetzen erwartet, dass sie hohe Leistung erbringen, weil die Effektivität von über das Netz verteilten Berechnungen oft direkt von der Effizienz abhängt, mit der das Netzwerk Daten überträgt. Während das in Programmierkreisen althergebrachte Sprichwort, »Mach es zuerst richtig und dann schnell«, auf viele Umgebungen zutrifft, ist es im Netzwerkbereich normalerweise notwendig, von vornherein »mit Blick auf Leistung zu entwickeln«. Man muss daher unbedingt die verschiedenen Faktoren kennen, die sich auf die Netzleistung auswirken.

1.5.1 Bandbreite und Latenz

Die Leistung eines Netzwerks wird auf zwei grundlegende Arten gemessen: *Bandbreite* (auch *Durchsatz* genannt) und *Latenz* (auch als *Verzögerung* bezeichnet). Die Bandbreite eines Netzwerks wird durch die Anzahl von Bits angegeben, die innerhalb eines bestimmten Zeitraums im Netzwerk übertragen werden können. Beispielsweise kann ein Netzwerk über eine Bandbreite von 10 Millionen Bit/s (Mbit/s) verfügen, was bedeutet, dass es 10 Millionen Bits pro Sekunde übertragen kann. Manchmal ist es nützlich, sich Bandbreite in bezug darauf vorzustellen, wie lange es dauert, ein Datenbit zu übertragen. In einem 10-Mbit/s-Netzwerk sind das beispielsweise 0,1 Mikrosekunde (µs).

Bandbreite und Durchsatz

Bandbreite und *Durchsatz* sind zwei der verwirrendsten Themen im Netzwerkbereich. Wir können versuchen, beide Begriffe genau zu definieren. Wichtig ist zu wissen, wie andere sie verwenden, und dass sie oft als Synonyme gebraucht werden. Bandbreite ist praktisch ein Maß der Breite eines Frequenzbands. Eine Telefonleitung in Sprachqualität unterstützt z.B. einen Frequenzbandbereich von 300 bis 3.300 Hz. Man sagt, sie hat eine Bandbreite von 3.300 Hz – 300 Hz = 3.000 Hz. Wird das Wort »Bandbreite« in einer Situation benutzt, in der in Hertz gemessen wird, bezieht es sich wahrscheinlich auf den unterstützten Signalbereich.

Wenn wir von der Bandbreite einer Kommunikationsleitung sprechen, beziehen wir uns normalerweise auf die Anzahl der Bits pro Sekunde, die auf der Leitung übertragen werden können. Die Bandbreite einer Ethernet-Leitung kann z.B. 10 Mbit/s sein. Zwischen der auf der Leitung verfügbaren Bandbreite und der tatsächlich auf der Leitung übertragenen Bits pro Sekunde wird in der Praxis aber unterschieden. Wir verwenden das Wort »Durchsatz« für die *gemessene Leistung* eines Systems. Aufgrund verschiedener Ineffizienzen von Implementierungen können zwei über eine Leitung mit einer Bandbreite von 10 Mbit/s verbundene Knoten einen Durchsatz von nur 2 Mbit/s erreichen. Dies würde bedeuten, dass eine Anwendung auf einem Host Daten in 2 Mbit/s an den anderen Host senden kann.

Schließlich spricht man auch oft von den Bandbreiten*anforderungen* einer Anwendung. Das ist die Anzahl von Bits pro Sekunde, die sie benötigt, um im Netzwerk in akzeptabler Leistung zu übertragen. Bei einigen Anwendungen kann das die maximale Bandbreite sein, die sie bekommen kann. Bei anderen kann das eine feste Größe (vorzugsweise nicht mehr als die verfügbare Leitungsbandbreite) sein. In wieder anderen Fällen kann die Bandbreite mit der Zeit schwanken. Dieses Thema wird später in diesem Abschnitt ausführlicher behandelt.

Manchmal soll die Bandbreite eines Netzwerks nicht als Ganzes betrachtet werden, beispielsweise, um sich auf die Bandbreite einer bestimmten Verbindungsleitung oder eines logischen Prozess-zu-Prozess-Kanals zu konzentrieren. Auf der physikalischen Ebene verbessert sich die Bandbreite ständig, ohne dass ein Ende in Sicht wäre. Stellt man sich eine Sekunde als Entfernung vor, die man mit einem Lineal messen kann, und die Bandbreite dahingehend, wie viele Bits in diese Entfernung passen, kann man sich ein Bit als Impuls mit einer bestimmten Breite vorstellen. Ein Bit wäre in einer 1-Mbit/s-Leitung dann 1 µs und in einer 2-Mbit/s-Leitung 0,5 µs breit (siehe Abb. 1.20). Je ausgefeilter die Übertragungs- und Empfangstechnologie, um so schmäler wird ein Bit und um so höher die Bandbreite. Bei logischen Prozess-zu-Prozess-Kanälen wird die Bandbreite noch von weiteren Faktoren beeinflusst, u.a. wie oft die Software, die den Kanal implementiert, jedes Datenbit zu bearbeiten und möglicherweise umzuwandeln hat.

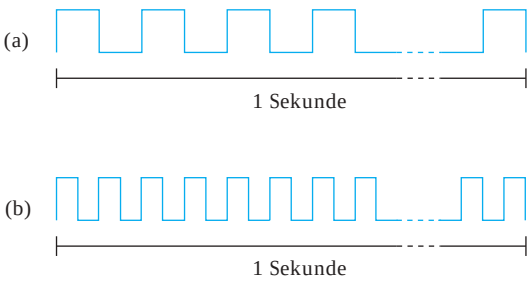

Abb. 1.20: In einer bestimmten Bandbreite übertragene Bits kann man sich mit einer gewissen Breite vorstellen: (a) in 1 Mbit/s übertragene Bits (jedes Bit ist 1 µs breit); (b) in 2 Mbit/s übertragene Bits (jedes Bit ist 0,5 µs breit).

Latenz entspricht als zweites Leistungsmaß der Dauer, bis eine Nachricht von einem Ende eines Netzwerks zum anderen gelangt. (Wie bei der Bandbreite können wir uns auf die Latenz einer einzelnen Leitung oder eines Ende-zu-Ende-Kanals beziehen.) Latenz wird strikt in Zeit gemessen. Ein transkontinentales Netzwerk kann beispielsweise eine Latenz von 24 Millisekunden (ms) aufweisen. Das heißt, eine Nachricht braucht von einem Ende Nordamerikas zum anderen 24 ms. In vielen Situationen genügt es uns aber nicht, die Latenzzeit in einer Richtung zu ermitteln. Vielmehr möchten wir zu wissen, wie lange es dauert, um eine Nachricht von einem Ende eines Netzwerks zum anderen und zurück zu senden. Man nennt dies die *Roundtrip-Zeit* (Round-Trip Time, *RTT*) des Netzwerks.

Die Latenz beinhaltet im Grunde drei Komponenten: Erstens die endliche Ausbreitungsgeschwindigkeit des Lichts. Die Verzögerung entsteht, weil nichts, auch kein Bit in einer Leitung, sich schneller als mit Lichtgeschwindigkeit bewegen kann. Kennt man die Entfernung zwischen zwei Punkten, kann man die durch die Lichtgeschwindigkeit bedingte Verzögerung berechnen. Man sollte dabei allerdings vorsichtig sein, weil sich Licht in verschiedenen Medien mit unterschiedlicher Geschwindigkeit ausbreitet, z.B. $3{,}0 \times 10^8$ m/s im Vakuum, $2{,}3 \times 10^8$ m/s in einem Kabel und $2{,}0 \times 10^8$ m/s in einem Lichtwellenleiter. Zweitens dauert es eine gewisse Zeit, um eine Dateneinheit zu übertragen. Dies ist eine Funktion der Netzbandbreite und der Größe des Pakets, in dem die Daten befördert werden. Drittens kann es im Netz zu Verzögerungen auf Grund von Wartezeiten kommen, weil Switche die Pakete im Allgemeinen eine gewisse Zeit lang speichern müssen, bevor sie sie auf einer abgehenden Leitung weitersenden (siehe Abschnitt 1.2.2). Wir können die gesamte Latenz also wie folgt definieren:

$$\text{Latenz} = \text{Ausbreitungsverzögerung} + \text{Übertragungsverzögerung} + \text{Wartezeit}$$

$$\text{Ausbreitungsverzögerung} = \text{Entfernung}/\text{Lichtgeschwindigkeit}$$

$$\text{Übertragungsverzögerung} = \text{Paketgröße}/\text{Bandbreite}$$

wobei Entfernung die Länge der Leitung, über die Daten gesendet werden, Lichtgeschwindigkeit die effektive Geschwindigkeit des Lichts in dieser Leitung, und Bandbreite die Bandbreite ist, in der das Paket übertragen wird. Enthält die Nachricht nur ein Bit und sprechen wir über eine einzelne Leitung (im Gegensatz zum gesamten Netzwerk), sind die Begriffe Übertragungsverzögerung und Wartezeit nicht relevant, und Latenz entspricht nur der Ausbreitungsverzögerung.

Bandbreite und Latenz zusammen definieren die Leistungsmerkmale einer bestimmten Leitung oder eines Kanals. Ihre relative Bedeutung hängt aber von der Anwendung ab. Bei einigen Anwendungen wird die Latenz von der Bandbreite beherrscht. Ein Client, der eine 1-Byte-Nachricht an einen Server sendet und eine ebensolche Nachricht empfängt, ist Latenz-bestimmt. Wenn bei der Vorbereitung der Antwort keine umfangreiche Berechnung erforderlich ist, ergibt sich eine andere Leistung der Anwendung, je nachdem, ob auf einem transkontinentalen Kanal mit einer RTT von 100 ms oder einem Kanal mit einer 1-ms-RTT, der sich nur über einen Raum erstreckt, übertragen wird. Ob der Kanal über 1 oder 100 Mbit/s ver-

fügt, ist demgegenüber relativ unbedeutend, weil es bei 1 Mbit/s 8 µs dauert, um ein Byte zu übertragen (Übertragung) und 100 Mbit/s Übertragung = 0,08 µs implizieren.

Betrachtet man demgegenüber ein digitales Bibliotheksprogramm, das aufgefordert wird, ein 25 Megabyte (MB) großes Bild einzulesen, wird das Bild um so schneller am Bildschirm angezeigt, je mehr Bandbreite verfügbar ist. Hier wird die Leistung von der Bandbreite des Kanals beherrscht. Zur Verdeutlichung nehmen wir an, dass der Kanal über eine Bandbreite von 10 Mbit/s verfügt. Dann dauert es 20 Sekunden, um das Bild zu übertragen. Relativ unbedeutend ist dabei, ob sich das Bild auf der anderen Seite eines 1-ms- oder 100-ms-Kanals befindet. Der Unterschied zwischen einer Anwortzeit von 20,001 oder 20,1 Sekunden ist verschwindend gering.

Abb. 1.21 bietet einen Überblick darüber, wie Latenz oder Bandbreite die Leistung je nach Situation beeinflussen kann. Das Diagramm zeigt, wie lange es dauert, Objekte unterschiedlicher Größe (1 Byte, 2 KB, 1 MB) über Netzwerke mit RTTs von 1 bis 100 ms und Leitungsgeschwindigkeiten von 1,5 bzw. 10 Mbit/s zu befördern. Wir verwenden eine logarithmische Skala, um die relative Leistung darzustellen. Bei einem 1-Byte-Objekt (z.B. einem Tastenanschlag) bleibt die Latenz gegenüber der RTT fast genau gleich, sodass man nicht zwischen einem 1,5-Mbit/s- und einem 10-Mbit/s-Netzwerk unterscheiden kann. Bei einem 2-KB-Objekt (z.B. einer E-Mail) unterscheidet sich die Leitungsgeschwindigkeit deutlich zwischen einem Netzwerk mit einer 1-ms-RTT und einem mit einer 100-ms-RTT. Bei einem 1-MB-

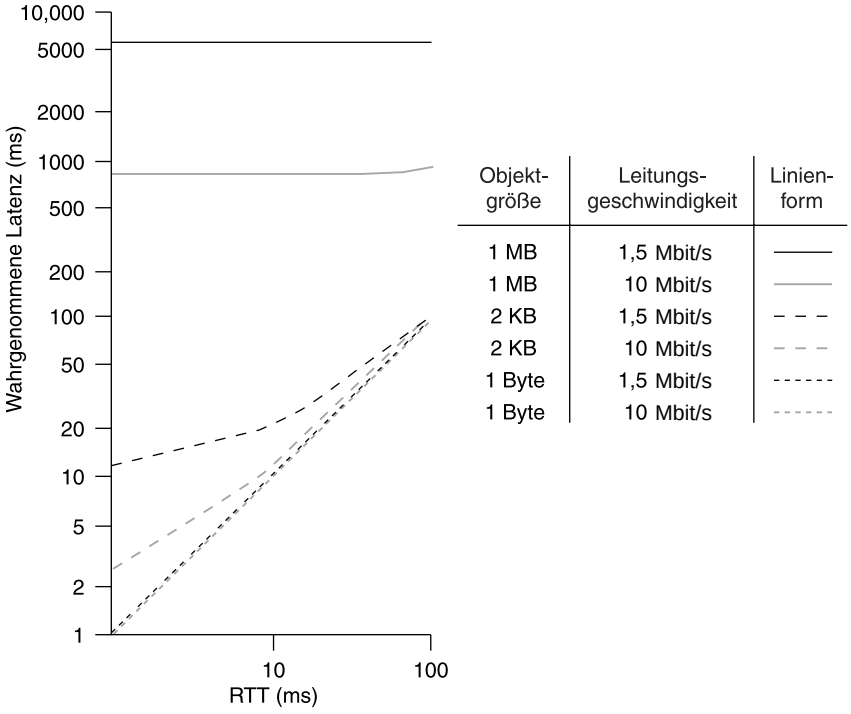

Abb. 1.21: Wahrgenommene Latenz (Antwortzeit) gegenüber Roundtrip-Zeit bei verschiedenen Objektgrößen und Leitungsgeschwindigkeiten

Objekt (z.B. einem digitalen Bild) bewirkt die RTT keinen Unterschied; hier wird die Leistung bei allen RTT-Werten nur von der Leitungsgeschwindigkeit beeinflusst.

Man beachte, dass wir die Begriffe *Latenz* und *Verzögerung* im gesamten Buch allgemein verwenden, d.h. wir bezeichnen damit, wie lange es dauert, um eine bestimmte Funktion, z.B. die Übersendung einer Nachricht oder eines Objekts, auszuführen. Wenn wir uns auf eine bestimmte Zeit beziehen, die es dauert, bis ein Signal von einem Ende einer Leitung zum anderen gelangt, verwenden wir den Begriff *Ausbreitungsverzögerung*. Wir verdeutlichen im jeweiligen Zusammenhang auch, ob wir uns auf Einweglatenz oder Roundtrip-Zeit beziehen.

Nebenbei bemerkt, werden Computer so schnell, dass man, wenn man sie an Netzwerke anschließt, zumindest bildhaft in *Instruktionen pro Kilometer* denken kann. Man stelle sich vor, was passiert, wenn ein Computer, der 200 Millionen Instruktionen pro Sekunde ausführt, eine Nachricht auf einen Kanal mit einer RTT von 100 ms abschickt. (Zur Vereinfachung der Rechenaufgabe gehe man davon aus, dass die Nachricht eine Entfernung von 5.000 Kilometer abdeckt.) Wartet dieser Computer 100 ms lang untätig auf eine Antwort, hat er die Gelegenheit verpasst, weitere 20 Millionen Instruktionen bzw. 4.000 Instruktionen pro Kilometer ausführen zu können. Angesichts einer solchen Verschwendung muss sich die Überquerung des Netzwerks auf jeden Fall lohnen.

1.5.2 Verzögerung-Bandbreite-Produkt

Es ist auch hilfreich, das so genannte *Verzögerung-Bandbreite-Produkt* zu betrachten. Wenn wir uns einen Kanal zwischen zwei Prozessen als hohle Pipeline (siehe Abb. 1.22) vorstellen, bei der die Latenz der Länge und die Bandbreite dem Durchmesser entspricht, ergibt das Verzögerung-Bandbreite-Produkt das Volumen der Pipeline, d.h. die Anzahl der enthaltenen Bits. Anders ausgedrückt: Wenn die Latenz der Länge entspricht, kann man anhand der Breite jedes Bits (ebenfalls in Zeit gemessen) berechnen, wie viele Bits in die Pipeline passen. Ein transkontinentaler Kanal mit einer Einweglatenz von 50 ms und einer Bandbreite von 45 Mbit/s nimmt ein Datenvolumen von

$$50 \times 10^{-3}\ \text{s} \times 45 \times 10^{6}\ \text{Bit/s} = 2{,}25 \times 10^{6}\ \text{Bit}$$

bzw. ca. 280 KB auf. Das bedeutet, dass der Kanal (die Pipeline) in diesem Beispiel so viele Bytes aufnehmen kann wie in den Speicher eines Personalcomputers aus dem Anfang der achtziger Jahre passen.

Es ist wichtig, das Verzögerung-Bandbreite-Produkt zu kennen, wenn man Hochleistungsnetze entwickelt, weil es angibt, wie viele Bits der Sender übertragen

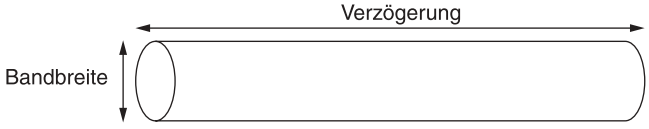

Abb. 1.22: Netzwerk als Pipeline

muss, bis das erste Bit beim Empfänger ankommt. Erwartet der Sender vom Empfänger irgendein Signal, dass die ersten Bits ankommen, und entsteht eine weitere Kanallatenz, bis dieses Signal zum Sender gelangt (wenn wir also nicht nur an der Einweglatenz, sondern auch an der RTT des Kanals interessiert sind), kann der Sender Daten im Umfang von zwei Verzögerung-Bandbreite-Produkten senden, bis er vom Empfänger erfährt, dass alles zum Besten steht. Die Bits in der Pipeline befinden sich auf dem »Transit«. Das heißt, wenn der Empfänger dem Sender bedeutet, er möge mit der Übertragung innehalten, müsste er noch ein Datenvolumen empfangen, das bis zu einem Verzögerung-Bandbreite-Produkt reicht, bis es dem Sender gelingt, zu antworten. In unserem obigen Beispiel entspricht dies einem Datenvolumen von $4{,}5 \times 10^6$ Bit (549 KB). Wenn der Sender demgegenüber die Pipeline nicht füllt, d.h. Daten im Umfang des Verzögerung-Bandbreite-Produkts sendet und dann zunächst unterbricht, um auf ein Signal zu warten, lastet er das Netzwerk nicht voll aus.

Man beachte, dass wir uns größtenteils auf das RTT-Szenario beziehen, wenn wir einfach vom Verzögerung-Bandbreite-Produkt sprechen, ohne ausdrücklich darauf hinzuweisen, dass dieses Produkt mit zwei multipliziert wird. Ob mit »Verzögerung« im Begriff »Verzögerung-Bandbreite-Produkt« Einweglatenz oder RTT gemeint ist, geht jeweils aus dem Zusammenhang hervor.

Wie groß ist ein Mega?

Es gibt mehrere Fallen, auf die man aufpassen muss, wenn man mit den üblichen Einheiten im Netzwerkbereich arbeitet: MB, Mbit/s, KB und Kbit/s. Als erstes müssen wir sorgfältig zwischen Bits und Bytes unterscheiden. In diesem Buch verwenden wir durchgängig ein kleines *b* für Bits und ein großes *B* für Bytes. Das zweite soll sicherstellen, dass Sie die entsprechende Definition von Mega (M) und Kilo (K) verwenden. *Mega* z.B. kann 2^{20} oder 10^6 bedeuten. Ähnlich kann *Kilo* entweder 2^{10} oder 10^3 sein. Noch schlimmer ist, dass wir im Netzwerkbereich beide Definitionen benutzen. Hier ist der Grund.

Die Netzwerkbandbreite, die oft in Mbit/s angegeben wird, hängt normalerweise von der Geschwindigkeit des Taktgebers ab, der die Übertragung der Bits taktet. Ein in 10 MHz laufender Taktgeber wird für die Übertragung von Bits in 10 Mbit/s benutzt. Da der Teil *Mega* in MHz 10^6 Hertz bedeutet, wird Mbit/s ebenfalls als 10^6 Bit pro Sekunde (und vergleichbar damit Kbit/s in 10^3 Bit/s) definiert. Wenn wir andererseits von einer zu übertragenden Nachricht sprechen, geben wir ihre Größe in Kilobyte an. Da Nachrichten im Speicher eines Computers gespeichert und Speicher generell in Potenzen von Zwei gemessen werden, bedeutet das *K* in KB 2^{10} (und vergleichbar damit MB 2^{20}). Fügt man beides zusammen, spricht man meist vom Senden einer 32-KB-Nachricht über einen 10-Mbit/s-Kanal. Dies bedeutet, dass $32 \times 2^{10} \times 8$ Bit in einer Rate von 10×10^6 Bit pro Sekunde übertragen werden. Soweit nicht ausdrücklich anders angegeben, benutzen wir diese Interpretation im gesamten Buch.

Das Gute daran ist, dass wir oft mit einer schnellen Überschlagsrechnung zufrieden sind. In diesem Fall ist es absolut angemessen, dass ein Byte 10 Bits enthält (sodass die Konvertierung zwischen Bits und Bytes einfacher ist) und dass 10^6 eigentlich 2^{20} entspricht (sodass es einfacher ist, zwischen den beiden Definitionen von Mega zu unterscheiden). Der erste Punkt führt einen Fehler von 20% und der zweite einen von nur 5% ein.

Um Ihre Rechenarbeit zu erleichtern, sei festgestellt, dass 100 ms eine angemessene Zahl für eine Roundtrip-Zeit ist, zumindest wenn es sich bei dem fraglichen Land um die USA handelt. Demgegenüber ist 1 ms eine gute Zahl für eine RTT in einem LAN. Im ersten Fall erhöhen wir die durch Lichtgeschwindigkeit in Glasfaser implizierte Roundtrip-Zeit von 48 ms auf 100 ms, weil es – wie gesagt – weitere Verzögerungsquellen gibt, z.B. die Verarbeitungszeit in den Switchen des Netzwerks. Sie können auch sicher sein, dass der von der Glasfaser zwischen zwei Punkten eingeschlagene Pfad keine gerade Linie darstellt.

1.5.3 Hochgeschwindigkeitsnetze

Die in den heutigen Netzwerken verfügbaren Bandbreiten steigen in atemberaubender Geschwindigkeit, und es herrscht ungebrochener Optimismus darüber, dass sich die Netzbandbreite laufend verbessert. Dies veranlasst Netzwerkentwickler, sich mit dem Gedanken zu befassen, was am Ende dieser Entwicklung steht oder, anders ausgedrückt, welche Auswirkung eine unendliche Bandbreite auf das Netzdesign hat.

Hochleistungsnetze führen zweifellos zu einer dramatischen Änderung der für Anwendungen verfügbaren Bandbreite. In vielerlei Hinsicht hat ihre Auswirkung auf unsere Vorstellung von Netzwerken aber mit etwas zu tun, das sich im Zuge steigender Bandbreiten *nicht* ändert: mit der Lichtgeschwindigkeit. Um Scotty aus *Star Trek* zu zitieren: »Du kannst die Gesetze der Physik nicht ändern.« Mit anderen Worten: »Hohe Geschwindigkeit« bedeutet nicht, dass sich die Latenz in gleichem Umfang wie die Bandbreite verbessert. Die transkontinentale RTT einer 1-Gbit/s-Leitung beträgt genauso 100 ms wie bei einer 1-Mbit/s-Leitung.

Um die Bedeutung ständig steigender Bandbreite angesichts fester Latenzzeiten richtig einschätzen zu können, stelle man die Überlegung an, was nötig ist, um eine 1-MB-Datei über ein 1-Mbit/s-Netzwerk im Gegensatz zu einem 1-Gbit/s-Netzwerk bei jeweils einer RTT von 100 ms zu übertragen. Beim 1-Mbit/s-Netzwerk dauert die Übertragung der Datei 80 Roundtrip-Zeiten, während die gleiche Datei nicht einmal annähernd 1 RTT der 1-Gbit/s-Leitung, die ein Verzögerung-Bandbreite-Produkt von 12,5 MB hat, füllt.

Aus Abb. 1.23 wird der Unterschied zwischen den beiden Netzwerken deutlich. Die 1-MB-Datei sieht wie ein Datenstrom aus, der über ein 1-Mbit/s-Netzwerk

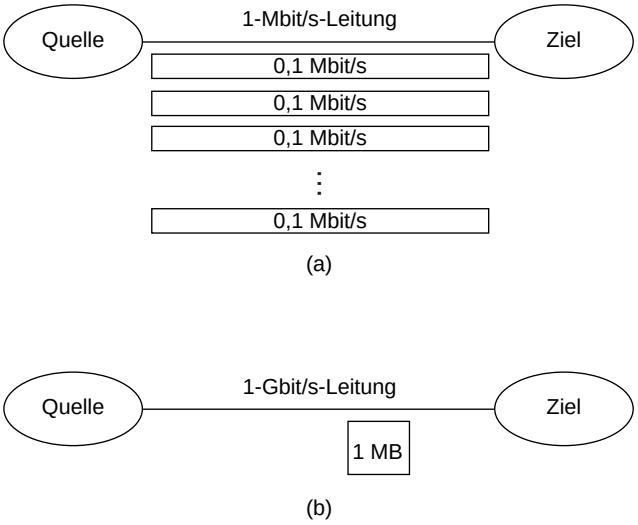

Abb. 1.23: Verhältnis zwischen Bandbreite und Latenz; bei einer 1-MB-Datei entspricht (a) die 1-Mbit/s-Leitung 80 mit Daten gefüllten Pipelines und (b) die 1-Gbit/s-Leitung einer zu einem 1/12 gefüllten Pipeline

übertragen werden muss, während sie bei einem 1-Gbit/s-Netzwerk den Anschein eines einzelnen Pakets erweckt. Um diesen Punkt zu verdeutlichen, stelle man sich vor, dass eine 1-MB-Datei für ein 1-Gbit/s-Netzwerk das ist, was ein 1-KB-*Paket* für ein 1-Mbit/s-Netzwerk ist.

1 **Man kann die Situation auch aus einem anderen Blickwinkel betrachten: Innerhalb jeder RTT können in einem Hochgeschwindigkeitsnetz mehr Daten übertragen werden, sodass eine einzelne RTT zu einem bedeutenden Zeitumfang wird. Während man über den Unterschied zwischen einem Dateitransfer, der 101 statt 100 RTT dauert (ein relativer Unterschied von nur 1%) kaum nachdenkt, wird einem die Bedeutung des Unterschiedes zwischen 1 und 2 RTT klar: eine Steigerung von 100%. Folglich beherrscht nun nicht der Durchsatz, sondern die Latenz unsere Vorstellung von Netzwerkdesign.**

Die vielleicht verständlichste Art, den Unterschied zwischen Durchsatz und Latenz zu verdeutlichen, bietet eine Rückkehr zu den Grundlagen. Der effektive Ende-zu-Ende-Durchsatz, der in einem Netzwerk erreicht werden kann, wird durch das einfache Verhältnis

$$\text{Durchsatz} = \text{TransferGröße}/\text{TransferZeit}$$

angegeben, wobei TransferZeit nicht nur die Elemente der an früherer Stelle erklärten einwegigen Latenz, sondern auch jegliche zusätzliche Zeit für die Anfrage oder den Aufbau des Transfers beinhaltet. Im allgemeinen stellen wir dieses Verhältnis als

$$\text{TransferZeit} = \text{RTT} + 1/\text{Bandbreite} \times \text{TransferGröße}$$

dar. Wir benutzen RTT in dieser Berechnung für die Berücksichtigung einer Anfragenachricht, die über das Netzwerk gesendet wird, und die zurückgeschickten Daten.

Man betrachte z.B. einen Fall, in dem ein Benutzer eine 1-MB-Datei über ein 1-Gbit/s-Netzwerk mit einer Roundtrip-Zeit von 100 ms abrufen möchte. Transfer-Zeit beinhaltet die Übertragungszeit für 1 MB (1/1 Gbit/s × 1 MB = 8 ms) und die 100-ms-RTT, also eine Transferzeit von insgesamt 108 ms. Das bedeutet, dass der effektive Durchsatz

1 MB/108 ms = 74,1 Mbit/s

und nicht 1 Gbit/s beträgt. Selbstverständlich verbessert sich der effektive Durchsatz bei Übertragung einer größeren Datenmenge. Im Grenzfall führt dann eine unendlich große Transfergröße dazu, dass der effektive Durchsatz gegen die Netzbandbreite konvergiert. Muss man andererseits mehr als 1 RTT aufwenden, um beispielsweise fehlende Pakete erneut zu übertragen, verschlechtert man den effektiven Durchsatz jeglichen Transfers mit endlicher Größe, was sich am deutlichsten bei kleinen Transfers bemerkbar macht.

1.5.4 Leistungsanforderungen von Anwendungen

In diesem Abschnitt wurde Leistung bisher nur aus Sicht des Netzwerks betrachtet, d.h. was eine bestimmte Verbindungsleitung oder ein Kanal unterstützen kann. Dabei wurde stillschweigend davon ausgegangen, dass die Anwendungsprogramme einfache Bedürfnisse haben – sie möchten soviel Bandbreite in Anspruch nehmen, wie das Netzwerk bereitstellen kann. Dies ist bei dem oben erwähnten digitalen Bibliotheksprogramm, das ein 25-MB-Bild abruft, sicherlich richtig. Je mehr Bandbreite verfügbar ist, um so schneller kann das Programm das Bild für den Benutzer ausgeben.

Einige Anwendungen sind demgegenüber in der Lage, eine obere Grenze der von ihnen benötigten Bandbreite anzugeben. Videoanwendungen sind dafür ein gutes Beispiel. Angenommen, Sie möchten ein Bild übertragen, das ein Viertel der Größe eines üblichen Fernsehbilds misst, d.h. eine Auflösung von 352 mal 240 Pixel besitzt. Wird jedes Pixel mit 24 Datenbits dargestellt, wie dies bei einer Farbtiefe von 24 Bit der Fall ist, hätte jeder Frame eine Größe von

(352 × 240 × 24 Bit) / 8 = 247,5 Kilobyte (KB).

Eine Videoanwendung, die 30 Frames pro Sekunde übertragen soll, erfordert also eine Durchsatzrate von 75 Mbit/s. Die Fähigkeit des Netzwerks, noch mehr Bandbreite bereitzustellen, ist für eine solche Anwendung nicht von Belang, weil sie innerhalb eines bestimmten Zeitraums eben nur eine gewisse Datenmenge übertragen muss.

Leider ist die allgemeine Situation nicht ganz so einfach wie bei diesem Beispiel. Aufgrund des geringen Unterschieds zwischen zwei aufeinander folgenden Frames in einem Videostrom ist es möglich, das Video zu komprimieren. Beispielsweise dadurch, dass nur jeweils die Unterschiede zwischen aufeinander folgenden Frames übertragen werden. Dieses komprimierte Video fließt nicht in einer konstanten Rate, sondern schwankt im Lauf der Zeit entsprechend verschiedener Faktoren, z.B. Akti-

onsumfang und Details im Bild und verwendetem Kompressionsalgorithmus. Deshalb kann man zwar die durchschnittliche Anforderung an die Bandbreite abschätzen, die unmittelbare Rate kann jedoch eventuell höher oder niedriger sein.

Die wichtigste Frage hierbei ist das Zeitintervall, für das der Durchschnitt berechnet wird. Angenommen, die hier beispielhaft betrachtete Videoanwendung kann soweit komprimiert werden, dass sie im Durchschnitt nur 2 Mbit/s benötigt. Überträgt sie zuerst 1 Megabit in einem und dann 3 Megabit im nächsten 1-Sekunden-Intervall, überträgt sie innerhalb dieses 2-Sekunden-Intervalls mit einer durchschnittlichen Rate von 2 Mbit/s. Dies ist für einen Kanal, der für die Unterstützung von höchstens 2 Megabit pro Sekunde ausgelegt wurde, kaum ein Trost. Offensichtlich genügt es nicht immer, lediglich die durchschnittliche Anforderung einer Anwendung an die Bandbreite zu kennen.

Im allgemeinen ist es aber möglich, für die Größe eines Burst, den eine Anwendung wie diese aller Wahrscheinlichkeit nach sendet, eine obere Grenze zu setzen. Unter »Burst« versteht man entweder eine Spitzenrate, die über eine gewisse Zeitdauer aufrechterhalten wird oder mit der gesendet werden kann, bis eine gewisse Menge an Daten übertragen wurde. Ist diese Spitzenrate höher als die verfügbare Kanalkapazität, müssen die überschüssigen Daten irgendwo zwischengespeichert und später übertragen werden. Weiß der Netzwerkentwickler, in welchem Umfang Bursts auftreten können, kann er ausreichend Speicherkapazität zuweisen. Dieses Thema wird ausführlich in Kapitel 6 behandelt.

Ebenso wie die Anforderungen einer Anwendung an die Bandbreite anders gelagert sein können als nach dem Motto, »soviel sie nur bekommen kann«, können die Anforderungen einer Anwendung an die Verzögerung komplexer sein als schlicht »so gering wie möglich«. Bei der Verzögerung spielt es manchmal weniger eine Rolle, ob die Einweglatenz des Netzwerks 100 ms oder 500 ms beträgt, als vielmehr, dass die Latenz von Paket zu Paket variiert. Diese Schwankung in der Latenz nennt man *Jitter*.

Betrachten wir den Fall, bei dem die Quelle einmal alle 33 ms ein Paket sendet, wie das bei einer Videoanwendung, die 30 mal pro Sekunde Frames überträgt, der Fall ist. Kommen die Pakete im Abstand von 33 ms am Ziel an, können wir schlussfolgern, dass die Verzögerung jedes Pakets im Netzwerk genau gleich war. Ist der Abstand zwischen den am Ziel ankommenden Paketen – der so genannte *Interpacket-Gap* – variabel, muss natürlich auch die Verzögerung der einzelnen Pakete variabel gewesen sein. Das Netzwerk hat also Jitter in den Paketstrom eingeführt (siehe Abb. 1.24). Eine solche Schwankung kommt im Allgemeinen nicht auf einer

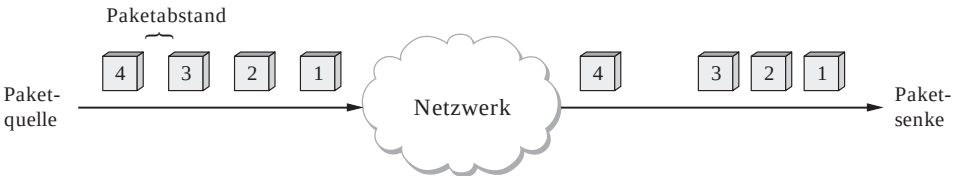

Abb. 1.24: Durch das Netzwerk erzeugter Jitter

einzelnen Verbindungsleitung, sondern nur dann vor, wenn Pakete in einem paketvermittelten Netzwerk mehrere Hops durchlaufen und dabei unterschiedliche Wartezeiten in Kauf nehmen müssen. Diese durch Wartezeiten entstandene Verzögerung entspricht der dritten – zeitlich schwankenden – Komponente der weiter oben definierten Latenz.

Um die Relevanz von Jitter besser zu verstehen, gehen wir davon aus, dass die im Netz übertragenen Pakete Video-Frames enthalten, und dass der Empfänger alle 33 ms einen neuen empfangen muss, damit er diese Frames am Bildschirm anzeigen kann. Kommt ein Frame zu früh an, kann der Empfänger ihn speichern, bis es Zeit ist, ihn darzustellen. Kommt ein Frame aber zu spät an, fehlt dem Empfänger der entsprechende Frame, um den Bildschirm rechtzeitig aufzufrischen. Dadurch wird die Videoqualität beeinträchtigt; es entsteht ein Ruckeleffekt. Man beachte dabei, dass Jitter nicht unbedingt vermieden werden muss; es genügt zu wissen, wie stark es ist. Das hat einen bestimmten Grund: Wenn der Empfänger die obere und untere Grenze der Latenz eines Pakets kennt, kann er den Zeitpunkt, an dem er mit der Wiedergabe des Videos beginnt (also den ersten Frame darstellt), solange hinauszögern, bis sichergestellt ist, dass er zu jedem Zeitpunkt aktuelle Frames präsentieren kann. Er speichert den Frame in einem Puffer, verzögert ihn also und glättet damit effektiv das Jitter. Dieses Thema wird ausführlich in Kapitel 9 behandelt.

1.6 Zusammenfassung

Rechnernetze wie das Internet haben im Laufe des letzten Jahrzehnts ein explosives Wachstum erfahren und sind heute in der Lage, Millionen von Benutzern eine breite Dienstpalette – fernen Dateizugriff, digitale Bibliotheken, Videokonferenzen – zu bieten. Ein Großteil dieses Wachstums ist auf den Allzweckcharakter von Rechnernetzen zurückzuführen, insbesondere aber auf die Fähigkeit, durch Entwicklung von Software, die auf preisgünstigen leistungsstarken Computern läuft, die Funktionalität des Netzwerks zu erweitern. Vor diesem Hintergrund liegt das vorrangige Ziel dieses Buchs darin, Rechnernetze in einer Art darzustellen, dass sich Leser nach der Lektüre in der Lage fühlen, mit einem schlagkräftigen Team von Programmierern an ihrer Seite von Grund auf ein voll funktionstüchtiges Rechnernetz entwickeln zu können. In diesem Kapitel wurde der Grundstein für die Realisierung dieses Ziels gelegt.

Der erste Schritt, den wir in Richtung dieses Ziels unternommen haben, ist die sorgfältige Identifizierung dessen, was wir von einem Netzwerk erwarten. Zuerst muss ein Netzwerk kostengünstige Konnektivität zwischen Computern bereitstellen. Dies wird durch einen Zusammenschluss von Knoten und Verbindungsleitungen und durch gemeinsame Nutzung dieser Hardwarebasis mittels statistischem Multiplexen erreicht. Daraus ergibt sich ein paketvermitteltes Netzwerk, für das wir dann eine Reihe von Diensten definieren, mit denen Prozesse miteinander kommunizieren können.

Im zweiten Schritt definieren wir eine aus Schichten bestehende Architektur, die uns als Entwurf für unser Design dient. Die zentralen Objekte dieser Architektur sind die Netzprotokolle. Protokolle bieten einen Kommunikationsdienst für die Pro-

tokolle der höheren Schichten und definieren sowohl Form als auch Bedeutung von Nachrichten, die sie mit ihren auf anderen Rechnern laufenden Partnern austauschen. In diesem Zusammenhang haben wir die OSI- und die Internet-Architektur als die beiden bekanntesten kurz beschrieben. Dieses Buch basiert strukturell und bezüglich der gewählten Beispiele vorrangig auf der Internet-Architektur.

Im dritten Schritt werden die Protokolle des Netzwerks und die Anwendungsprogramme – normalerweise als Software – implementiert. Sowohl die Protokolle als auch die Anwendungen benötigen ein Interface, über das sie die Dienste anderer Protokolle des Kommunikationssubsystems aufrufen können. Das Socket-Interface ist das am häufigsten benutzte Interface zwischen Anwendungsprogrammen und dem Kommunikationssubsystem. Innerhalb des letzteren wird aber meist ein Interface benutzt, das sich vom Socket-Interface geringfügig unterscheidet. Das Kommunikationssubsystem stellt außerdem eine Menge von Operationen bereit, die bei der Entwicklung von Protokollen immer wiederkehrende Aufgaben unterstützen sollen.

Schließlich muss das Netzwerk als Ganzes hohe Leistungsfähigkeit aufweisen, wobei wir vorrangig an Latenz und Durchsatz als den beiden wichtigsten Kenngrößen interessiert sind. Wie wir in späteren Kapiteln sehen werden, spielt das Produkt dieser beiden Größen – das so genannte Verzögerung-Bandbreite-Produkt – oft eine entscheidende Rolle beim Entwurf eines Netzwerks.

Offene Fragen

Allgegenwärtige Vernetzung

Es besteht kaum Zweifel darüber, dass Rechnernetze für sehr viele Menschen bald fester Bestandteil des täglichen Lebens werden. Was vor über 20 Jahren als Experiment mit Systemen wie dem ARPANET begann, als Großrechner über Telefonleitungen verbunden wurden, hat sich zum großen Geschäft gewandelt. Und wo viel Geld im Spiel ist, stellen sich viele Mitspieler ein. In diesem Fall sind das zum einen die Computerindustrie, die sich immer mehr mit der Unterstützung paketvermittelter Netzwerkprodukte beschäftigt, zum zweiten die Telefonbetreiber, die einen Markt für die Übertragung aller Datenarten, nicht nur Sprache, entdeckt haben, und zum dritten die Kabelfernsehbetreiber, die derzeit das unterhaltungsorientierte Segment dieses Markts beherrschen.

Wenn wir davon ausgehen, dass das Ziel »allgegenwärtige Vernetzung« ist, also die Vernetzung jedes Haushalts, muss man zuerst das Problem aufgreifen, wie die nötigen Verbindungsleitungen bereitzustellen sind. Viele sehen die Lösung letztendlich im Glasfaseranschluss jedes Haushalts. Bei einem geschätzten Kostenumfang von $1.000 pro Haus und 100 Millionen Privathäusern ergibt dies jedoch allein in den USA einen Investitionsaufwand von $100 Milliarden. Die derzeit meist diskutierten Alternativen nutzen entweder den vorhandenen Fernsehkabelanschluss oder

die Kupferkabel des Telefonnetzes. Beide Ansätze weisen spezifische Schwächen auf. So sind die heutigen Kabelanschlüsse z.B. asymmetrisch. Das heißt, man kann jedem Haushalt zwar 150 Kanäle bereitstellen, die zurückgerichtete Bandbreite ist aber sehr begrenzt. Solch eine Asymmetrie bedeutet, dass es wenige Informationsanbieter und sehr viele Informationskonsumenten gibt: Viele Menschen würden jedoch fordern, dass in einer Demokratie jeder die gleichen Möglichkeiten zum Anbieten von Informationen erhalten sollte. Die DSL-Technologie (Digital Subscriber Line) muss nicht asymmetrisch sein, kann aber über die vorhandenen Telefonleitungen auch nur einer eingeschränkten Zahl von Verbrauchern Verbindungen mit hoher Bandbreite bieten.

Wie der Kampf zwischen Computerherstellern, Telefongesellschaften und Kabelbetreibern um Marktanteile letztendlich ausgeht, lässt sich kaum vorhersagen. (Wenn wir hier die Antwort geben könnten, würden wir für dieses Buch wesentlich mehr verlangen.) Wir wissen lediglich, dass es viele technische Hindernisse – Fragen über Konnektivität, Dienstebenen, Leistung, Zuverlässigkeit und Fairness – zu überwinden gilt, die zwischen dem heutigen Entwicklungsstand und dem globalen, allgegenwärtigen heterogenen Netzwerk stehen, das wir für möglich und wünschenswert halten. Genau das sind die Herausforderungen, die den Schwerpunkt dieses Buchs bilden.

1.7 Weiterführende Literatur

Rechnernetze sind nicht die erste kommunikationsorientierte Technologie, die ihren Weg in den Alltag unserer Gesellschaft gefunden hat. In der ersten Hälfte des 20. Jahrhunderts wurde das Telefon eingeführt, und in den fünfziger Jahren hielt das Fernsehen Einzug in die Haushalte. Wenn wir die Zukunft von Netzwerken betrachten, d.h. in welchem Umfang sie sich ausbreiten und wie wir sie benutzen werden, ist ein Rückblick in seine Geschichte aufschlussreich. Unser erster Literaturhinweis ist ein guter Ausgangspunkt dafür (das ganze Buch ist den ersten 100 Jahren der Telekommunikation gewidmet).

Bei der zweiten und dritten Quelle handelt es sich um fundamentale Beiträge über die OSI- bzw. Internet-Architektur. Die Arbeit von Zimmerman liefert eine Einführung in die OSI-Architektur, und bei der von Clark handelt es sich um eine Retrospektive. Die beiden letzten Quellen befassen sich nicht ausdrücklich mit Vernetzung, sollten aber zur Pflichtlektüre jedes Systementwicklers gehören. Die Arbeit von Saltzer et al. behandelt eine der am häufigsten angewandten Regeln des Systemdesigns – das *Ende-zu-Ende-Argument*. Mashey beschreibt die den RISC-Architekturen zu Grunde liegenden Konzepte. Wir werden in den nächsten Kapiteln feststellen, dass wohlüberlegte Strukturierung von Funktionalität in einem komplexen System den Kernpunkt eines Systemdesigns bildet.

- Pierce, J.: Telephony – A personal view, *IEEE Communications*, 22(5), 116–120, Mai 1984.

- Zimmerman, H.: OSI reference model – The ISO model of architecture for open systems interconnection, *IEEE Transactions on Communications*, COM-28(4), 425–432, April 1980.
- Clark, D.: The design philosophy of the DARPA Internet protocols, *Proceedings of the SIGCOMM '88 Symposium*, S. 106–114, August 1988.
- Saltzer, J., Reed, D. und Clark, D.: End-to-end arguments in system design, *ACM Transactions on Computer Systems*, 2(4), 277–288, November 1984.
- Mashey, J.: RISC, MIPS, and the motion of complexity, *UniForum 1986 Conference Proceedings*, S. 116–124, 1986.

Mehrere Lehrbücher bieten eine Einführung in Computervernetzung: Stallings bietet eine enzyklopädische Abhandlung des Themas mit Schwerpunkt auf den niedrigeren Schichten der OSI-Hierarchie [Sta00a], Tanenbaum benutzt die OSI-Architektur als Organisationsmodell [Tan02], Comer gibt einen Überblick über die Internet-Architektur [Com00] und Bertsekas und Gallager behandeln Vernetzung aus Sicht der Leistungsmodellierung [BG92].

Will man Rechnernetze in einem größeren Zusammenhang betrachten, sind zwei Bücher absolute Pflichtlektüre: *The Early History of Data Networks* von Holzmann und Pehrson [HP95] mit Blick in die Vergangenheit. Die Leser werden zu ihrer Überraschung feststellen, dass viele heutige, im vorliegenden Buch beschriebene Konzepte im 17. Jahrhundert erfunden wurden. Das zweite ist *Realizing the Information Future: The Internet and Beyond* vom Computer Science und Telecommunications Board des National Research Council [NRC94], in dem ein Blick in die Zukunft gewagt wird.

Will man die Geschichte des Internet von Anfang an nachvollziehen, lohnt sich die Durchsicht der Internet-Reihe *Request for Comments* (RFC). Diese Dokumente, die von der TCP-Spezifikation bis zu Aprilscherzen wirklich alles über das Internet enthalten, sind unter http://www.ietf.org/rfc.html verfügbar. Die Protokollspezifikationen für TCP, UDP und IP befinden sich z.B. in RFC 793, 768 bzw. 791.

Um einen besseren Einblick in die Internet-Philosophie und -Kultur zu erhalten, sollte man unbedingt die beiden folgenden Werke lesen: Padlipsky liefert eine gute Beschreibung der Anfänge, z.B. einen mit scharfer Feder gezogenen Vergleich der Internet- und OSI-Architektur [Pad85]. Boorsooks Artikel [Boo95] ist eine aktuellere Beschreibung dessen, was sich hinter den Kulissen der Internet Engineering Task Force tut.

Über die verschiedenen Aspekte von Protokollimplementierungen gibt es eine Fülle von Arbeiten. Einen guten Ausgangspunkt bietet die Beschreibung des Streams-Mechanismus von Unix-System V [Rit84] bzw. des x-Kernels [HP91]. Darüber hinaus beschreiben [LMKQ89] und [SW95] die populäre Berkeley-Unix-Implementierung von TCP/IP.

Aus eher allgemeiner Sicht werden Fragen über die Strukturierung und Optimierung von Protokollimplementierungen von folgenden Autoren behandelt: Clark

befasst sich als einer der ersten Autoren mit der Beziehung zwischen modularem Design und Protokolleistung [Cla82] und in [Cla85] mit der Strukturierung von Protokollcode durch so genannte Upcalls. Der Protokoll-Overhead von TCP wird in [CJRS89] betrachtet. [WM87] befasst sich mit der Effizienzsteigerung durch angemessene Design- und Implementierungsentscheidungen.

In verschiedenen Arbeiten werden spezifische Techniken und Mechanismen für die Verbesserung der Leistung von Protokollen vorgestellt. [HMPT89] beschreibt einige im x-Kernel benutzte Mechanismen, [MD93] verschiedene Implementierungen von Demuxtabellen, [VL87] den für die Verwaltung von Protokollereignissen benutzten Timing-Wheel-Mechanismus und [DP93] eine effiziente Strategie zur Pufferverwaltung. Die Leistung von Protokollen auf parallelen Prozessoren (hierbei ist das zentrale Problem die gegenseitige Blockade) ist Themenschwerpunkt in [BG93] und [NYKT94].

Da viele Aspekte einer Protokollimplementierung von der Kenntnis der Grundlagen von Betriebssystemen abhängen, empfehlen wir die Lektüre von Finkel [Fin88], Bic und Shaw [BS88] und Tanenbaum [Tan01] als Einführung in Betriebssystemkonzepte.

Jedes Kapitel dieses Buchs endet mit einem Abschnitt über weiterführende Literatur und einer Reihe von URLs, die Informationen über das jeweilige Thema enthalten. Wir beschränken uns dabei auf Websites, die Software anbieten, einen Dienst bereitstellen oder über die Aktivitäten einer Arbeitsgruppe oder eines Standardisierungsgremiums berichten. Da es gut möglich ist, dass sie nicht über einen unbegrenzten Zeitraum hinweg zur Verfügung stehen, führen wir also nur Websites mit Material auf, das über andere Quellen nicht leicht zugänglich ist. Für dieses Kapitel haben wir folgende WWW-Quellen zusammengestellt:

- http://www.dpunkt.de/lehrbuch/netzwerke: Informationen zu diesem Buch, einschließlich Beilagen, Ergänzungen usw. (siehe auch http://www.mkp.com)
- http://www.acm.org/sigcomm/sos.html: Status verschiedener Netzwerkstandards, darunter die von IETF, ISO und IEEE
- http://www.ietf.org/: Informationen über die IETF und ihre Arbeitsgruppen
- http://www.cs.columbia.edu/~hgs/netbib/: Bibliographie netzspezifischer Forschungsarbeiten

1.8 Übungen

1. Stellen Sie über anonymes FTP eine Verbindung zu ftp.isi.edu (Verzeichnis in-notes) her und laden Sie den RFC-Index sowie die Protokollspezifikationen für TCP, IP und UDP herunter.

2. Sehen Sie sich auf der Webseite http://www.cs.princeton.edu/nsg um. Sie können dort aktuelle Forschungsarbeiten im Vernetzungsbereich an der Princeton-

Universität nachlesen. Die Seite enthält ein Bild des Autors Larry Peterson und einen Link, über den Sie zu einem Bild des Autors Bruce Davie gelangen.

3. Benutzen Sie ein Web-Suchwerkzeug, um nützliche allgemeine, nicht kommerzielle Informationen über folgende Themen zu finden: MBone, ATM, MPEG, IPv6 und Ethernet.

4. Die Unix-Utility whois kann benutzt werden, um den Domain-Namen einer Firma, oder umgekehrt, zu finden. Lesen Sie die Dokumentation über whois und experimentieren Sie damit. Versuchen Sie als Einstieg whois princeton.edu und whois princeton.

5. Berechnen Sie die Zeit, die insgesamt erforderlich ist, um eine 1000-KB-Datei in den folgenden Fällen zu übertragen. Gehen Sie von einer RTT von 100 ms, einer Paketgröße von 1 KB und von anfangs 2×RTT für das vor der Übertragung ablaufende »Handshaking« aus.

   a. Die Bandbreite beträgt 1,5 Mbit/s und Datenpakete können fortlaufend gesendet werden.

   b. Die Bandbreite beträgt 1,5 Mbit/s, nach der Übertragung jedes Pakets müssen wir aber eine RTT warten, bis das nächste gesendet werden kann.

   c. Die Bandbreite ist »unendlich«, was bedeutet, dass wir von einer Übertragungszeit von Null ausgehen, und dass bis zu 20 Pakete pro RTT gesendet werden können.

   d. Die Bandbreite ist unendlich, und während der ersten RTT können wir ein Paket (2^{1-1}), während der zweiten zwei Pakete (2^{2-1}), während der dritten vier (2^{3-1}) usw. senden. (Der Grund für die exponentielle Zunahme wird in Kapitel 6 erklärt.)

✓ 6. Berechnen Sie die Zeit, die insgesamt erforderlich ist, um eine 1,5-MB-Datei in den folgenden Fällen zu übertragen. Gehen Sie von einer RTT von 80 ms, einer Paketgröße von 1 KB und von anfangs 2×RTT für das vor der Übertragung ablaufende »Handshaking« aus.

   a. Die Bandbreite beträgt 10 Mbit/s und Datenpakete können fortlaufend gesendet werden.

   b. Die Bandbreite beträgt 10 Mbit/s, nach der Übertragung jedes Pakets müssen wir aber eine RTT warten, bis das nächste gesendet werden kann.

   c. Die Bandbreite ist »unendlich«, was bedeutet, dass wir von einer Übertragungszeit von Null ausgehen und dass bis zu 20 Pakete pro RTT gesendet werden können.

   d. Die Übertragungszeit sei Null wie in (c), aber während der ersten RTT können wir ein Pakete, während der zweiten zwei Pakete, während der dritten

vier (2^{3-1}) senden usw. (Der Grund für die exponentielle Zunahme wird in Kapitel 6 erklärt.)

7. Betrachten Sie ein LAN mit einer maximalen Entfernung von 2 km. Bei welcher Bandbreite wäre die Ausbreitungsverzögerung (mit einer Geschwindigkeit von 2×10^8 m/s) bei 100-Byte-Paketen identisch mit der Übertragungsverzögerung? Wie hoch wäre sie bei 512-Byte-Paketen?

✓ 8. Betrachten Sie ein LAN mit einer maximalen Entfernung von 50 km. Bei welcher Bandbreite wäre die Ausbreitungsverzögerung (mit einer Geschwindigkeit von 2×10^8 m/s) bei 100-Byte-Paketen identisch mit der Übertragungsverzögerung? Wie hoch wäre sie bei 512-Byte-Paketen?

9. Welche Merkmale von Postadressen würde man höchstwahrscheinlich in ein Adressierschema für ein Netzwerk aufnehmen? Welche Unterschiede würden Sie erwarten? Welche Eigenschaften des Telefonnummernsystems würde man in das Adressierschema eines Netzwerks aufnehmen?

10. Adressen zeichnen sich vor allem dadurch aus, dass sie eindeutig sind. Wenn zwei Knoten die gleiche Adresse hätten, könnte man sie nicht unterscheiden. Welche weiteren Merkmale wären für Netzwerkadressen nützlich? Können Sie sich Fälle vorstellen, in denen Adressen eines Netzwerks (bzw. des Post- oder Telefonsystems) *nicht* eindeutig sind?

11. Führen Sie einen Beispielfall an, bei dem Multicast-Adressen vorteilhaft wären.

12. Auf welche Unterschiede in Verkehrsmustern ist die Tatsache zurückzuführen, dass STDM kostengünstiges Multiplexen für ein Sprachtelefonnetz und FDM kostengünstiges Multiplexen für Fernseh- und Radionetze bieten, obwohl wir beides für ein allgemeines Rechnernetz als nicht kostengünstig verwerfen?

13. Wie »breit« ist ein Bit in einer 1-Gbit/s-Verbindungsleitung? Wie lang ist ein Bit in einem Kupferkabel, wenn die Ausbreitungsgeschwindigkeit $2{,}3\times10^8$ m/s beträgt?

14. Wie lange dauert es, um *x* KB über eine *y*-Mbit/s-Verbindungsleitung zu übertragen? Geben Sie Ihre Lösung als Verhältnis von *x* und *y* an.

15. Angenommen, zwischen der Erde und einer neuen Mondkolonie wird eine Punkt-zu-Punkt-Leitung mit 100 Mbit/s eingerichtet. Die Entfernung vom Mond zur Erde beträgt ungefähr 385.000 km, und Daten fließen in Lichtgeschwindigkeit – 3×10^8 m/s – über die Leitung.

    a. Berechnen Sie die minimale RTT für die Verbindungsleitung.

b. Unter Verwendung der RTT als Verzögerung berechnen Sie das Verzögerung-Bandbreite-Produkt für die Verbindungsleitung.

c. Welche Bedeutung hat das in b. berechnete Verzögerung-Bandbreite-Produkt?

d. Eine in der Mondstation installierte Kamera nimmt Bilder von der Erde auf und speichert sie in digitalem Format auf Platte. Die Bodenstation möchte das neueste Bild, das 25 MB umfasst, herunterladen. Wie viel Zeit verstreicht mindestens vom Absenden der Anfrage nach den Daten bis zum Ende der Übertragung?

✓ 16. Angenommen, zwischen der Erde und einem Marslandefahrzeug wird eine Punkt-zu-Punkt-Leitung mit 128 kbit/s eingerichtet. Die Entfernung vom Mars zur Erde beträgt ungefähr 55×10^9 m, und Daten fließen in Lichtgeschwindigkeit – 3×10^8 m/s – über die Leitung.

a. Berechnen Sie die minimale RTT für die Verbindungsleitung.

b. Berechnen Sie das Verzögerung-Bandbreite-Produkt für die Verbindungsleitung.

c. Eine auf dem Fahrzeug installierte Kamera nimmt Bilder von der Umgebung auf und sendet sie zur Erde. Wie viel Zeit verstreicht mindestens vom Aufnehmen des Bildes bis zum Ende der Übertragung an die Bodenkontrolle? Gehen Sie von einer Bildgröße von 5 MB aus.

17. Erläutern Sie zu jeder der folgenden, auf einem entfernten Datei-Server laufenden Operationen, ob sie eher auf Verzögerung oder eher auf Bandbreite empfindlich reagiert:

a. Öffnen einer Datei.

b. Lesen des Inhalts einer Datei.

c. Auflisten des Inhalts eines Verzeichnisses.

d. Anzeigen der Attribute einer Datei.

18. Berechnen Sie die Latenz (vom ersten gesendeten bis zum letzten empfangenen Bit) für folgende Fälle:

a. 10-Mbit/s-Ethernet mit einem einzigen Speichervermittler (Store-and-Forward-Switch) auf der Strecke und einer Paketgröße von 5.000 Bit. Gehen Sie davon aus, dass jede Verbindungsleitung eine Ausbreitungsverzögerung von 10 µs einführt, und dass der Switch sofort, nachdem er das Paket vollständig empfangen hat, mit der erneuten Übertragung beginnt.

b. Wie in a., jedoch mit drei Switchen.

c. Wie in a., jedoch mit einem Switch, der »Cut-Through« implementiert, also in der Lage ist, mit der erneuten Übertragung des Pakets bereits zu beginnen, nachdem er beispielsweise die ersten 200 Bit empfangen hat.

✓ 19. Berechnen Sie die Latenz (vom ersten gesendeten bis zum letzten empfangenen Bit) für folgende Fälle:

a. 1-Gbit/s-Ethernet mit einem einzigen Speichervermittler (Store-and-Forward-Switch) auf der Strecke und einer Paketgröße von 5.000 Bit. Gehen Sie davon aus, dass jede Verbindungsleitung eine Ausbreitungsverzögerung von 10 µs einführt und dass der Switch sofort, nachdem er das Paket vollständig empfangen hat, mit der erneuten Übertragung beginnt.

b. Wie in a., jedoch mit drei Switchen.

c. Wie in b., jedoch mit einem Switch, der »Cut-Through« implementiert, also in der Lage ist, mit der erneuten Übertragung des Pakets bereits zu beginnen, nachdem er beispielsweise die ersten 128 Bit empfangen hat.

20. Berechnen Sie die effektive Bandbreite für folgende Fälle: Gehen Sie für a. und b. davon aus, dass permanent Daten zum Senden bereitstehen, und berechnen Sie für c. einfach den Durchschnitt aus 12 Stunden.

a. 10-Mbit/s-Ethernet und drei Speicher-Switche, wie in Übung 18b. Die Switche können über eine Verbindung senden, während sie über die andere empfangen.

b. Wie in a., jedoch muss der Sender auf ein 50 Byte großes Bestätigungspaket warten, nachdem er jeweils ein 5000 Bit großes Datenpaket gesendet hat.

c. Zustellung von 100 Compact-Disks (mit je 650 MB) über Nacht (12 Stunden).

21. Berechnen Sie das Verzögerung-Bandbreite-Produkt für die folgenden Verbindungsleitungen. Benutzen Sie eine Einwegverzögerung, gemessen ab dem ersten gesendeten bis zum ersten empfangenen Bit.

a. 10-Mbit/s-Ethernet mit einer Verzögerung von 10 µs.

b. 10-Mbit/s-Ethernet mit einem einzigen Speicher-Switch wie in Übung 18a, einer Paketgröße von 5.000 Bit und einer Ausbreitungsverzögerung von 10 µs pro Verbindungsleitung.

c. T1-Leitung mit 1,5 Mbit/s und einer transkontinentalen Einwegverzögerung von 50 ms.

d. T1-Leitung mit 1,5 Mbit/s über einen Satelliten im geosynchronen Orbit in einer Höhe von 35.900 km. Die einzige Verzögerung ist hier die Ausbreitungsverzögerung der Lichtgeschwindigkeit.

22. Hosts A und B sind je über 10-Mbit/s-Verbindungsleitungen an einen Switch S angeschlossen (siehe Abb. 1.25). Die Ausbreitungsverzögerung in jeder Verbindungsleitung beträgt 20 µs. S ist ein Store-and-Forward-Switch. Er beginnt mit der erneuten Übertragung eines empfangenen Pakets 35 µs nach dessen vollständigem Empfang. Berechnen Sie die Zeit, die insgesamt erforderlich ist, um 10.000 Bit von A nach B zu übertragen, wenn

    a. das Paket als Ganzes übertragen wird.

    b. zwei 5.000-Bit-Pakete direkt nacheinander gesendet werden.

Abb. 1.25: Diagramm für Übung 22

23. Ein Host möchte einem anderen Host eine 1-MB-Datei zusenden. Die Kompression der Datei beansprucht eine CPU-Zeit von 1 Sekunde bei 50% und 2 Sekunden bei 60%.

    a. Berechnen Sie die Bandbreite, in der beide Kompressionsoptionen die gleiche Gesamtzeit für Kompression und Übertragung beanspruchen.

    b. Erläutern Sie, warum Latenz keinen Einfluss auf Ihre Lösung hat.

24. Ein bestimmtes Kommunikationsprotokoll produziert einen Overhead von 100 Byte für Header und Rahmung pro Paket. Wir senden mit diesem Protokoll 1 Million Daten-Bytes. Ein Daten-Byte wurde allerdings fehlerhaft übertragen, sodass das gesamte Paket, in dem es sich befindet, verlorengeht. Ermitteln Sie die Gesamtzahl an Overhead- + Verlust-Bytes jeweils für eine Paketgröße von 1.000, 5.000, 10.000 und 20.000 Byte. Welche Größe ist optimal?

25. Nehmen Sie an, Sie möchten eine Datei der Größe n Byte über ein Netz übertragen, das aus der Quelle, dem Empfänger, 7 Punkt-zu-Punkt-Verbindungen und 5 Switches besteht. Jede Verbindung habe eine Ausbreitungsverzögerung von 2 ms, eine Bandbreite von 4 Mbit/s und unterstütze sowohl Leitungs- als auch Paketvermittlung. Daher können Sie entweder die Datei in 1-KB-Pakete aufteilen oder eine Verbindung über die Switche aufbauen und die Datei als Ganzes senden. Gehen Sie davon aus, dass die Pakete je 24 Byte an zusätzlichen Header-Informationen und 1000 Byte an Nutzdaten haben, dass die Store-and-Forward-Prozedur, nachdem das Paket im Switch eingetroffen ist, eine Verzögerung von 1 ms bedeutet und dass Pakete kontinuierlich geschickt werden können, ohne auf die Bestätigung warten zu müssen. Bei der Leitungsvermittlung ist zunächst eine 1 KB-Nachricht erforderlich, die hin und her transportiert werden muss, wobei sie eine Verzögerung von 1 ms an jedem Switch erfährt, nachdem sie dort vollständig empfangen worden ist, während die zu übertragenden Daten keiner Verzögerung an den Switches unterliegen. Sie können auch von einer Dateigröße von Viel-fachen von 1000 Byte ausgehen.

a. Bei welcher Dateigröße (n Byte) ist die insgesamt übertragene Datenmenge für Leitungsvermittlung kleiner als für Pakettransport?

b. Für welche Dateigröße (n Byte) ist die Gesamtverzögerung, die entsteht, bevor die Datei insgesamt empfangen wurde, für Leitungsvermittlung kleiner als für Paketvermittlung?

c. Wie wichtig ist für diese Ergebnisse die Anzahl der Switche, die Bandbreite der Übertragungswege bzw. das Verhältnis zwischen Paket- und Header-Größe?

d. Was glauben Sie: Wie genau ist dieses Modell in Bezug auf die relativen Vorzüge der beiden Vermittlungsarten? Werden wichtige Randbedingungen ignoriert, die gegen die eine oder andere Art sprechen? Falls ja: Welche könnten das sein?

26. Betrachten Sie ein in geschlossener Schleife ausgelegtes Netzwerk (z.B. Token-Ring) mit einer Bandbreite von 100 Mbit/s und einer Ausbreitungsgeschwindigkeit von 2×10^8 m/s. Welchen Umfang müsste die Schleife unter der Annahme haben, dass die Knoten keine Verzögerung einführen, um genau ein 250-Byte-Paket aufzunehmen? Welcher Umfang ist nötig, wenn sich alle 100 m ein Knoten befindet und jeder Knoten eine Verzögerung von 10 Bit einführt?

27. Vergleichen Sie die Kanalanforderungen für Sprachverkehr mit den Anforderungen für die Echtzeitübertragung von Musik hinsichtlich Bandbreite, Verzögerung und Jitter. Was müsste man verbessern? Um ungefähr wie viel? Könnte man eine Kanalanforderung lockern?

28. Gehen Sie bei den folgenden Fällen davon aus, dass keine Datenkompression angewandt wird, was in der Praxis fast nie der Fall ist. Berechnen Sie für a. bis c. die für die Übertragung in Echtzeit erforderliche Bandbreite:

a. Video mit einer Auflösung von 640 × 480 Pixel, 3 Byte/Pixel, 30 Frames/Sekunde.

b. Video mit einer Auflösung von 160 × 120 Pixel, 1 Byte/Pixel, 5 Frames/Sekunde.

c. Musik von einer CD-ROM; eine CD enthält Musik für eine Spieldauer von 75 Minuten und hat eine Kapazität von 650 MB.

d. Ein Fax überträgt ein 8 × 10 Zoll großes Schwarzweißbild mit einer Auflösung von 72 Pixel pro Zoll (ppi). Wie lange dauert die Übertragung mit einem 14,4-Kbit/s-Modem?

✓ 29. Nehmen Sie – wie in der vorhergehenden Übung – im Folgenden an, dass dabei keine Datenkomprimierung stattfindet. Bestimmen Sie die Bandbreite, die erforderlich ist, um diese Daten in Echtzeit zu übertragen:

a. HDTV-Video mit einer Auflösung von 1920 × 1080 Pixel, 24 Bit/Pixel und 30 Frames/s.

b. Telefondaten von 8-Bit-Einheiten bei 8 kHz.

c. GSM-Mobiltelefondaten von 260-Bit-Einheiten mit 50 Hz.

d. HDCD-Audiodaten von 24-Bit-Einheiten bei 88,2 kHz.

30. Erläutern Sie die relativen Leistungsanforderungen der folgenden Anwendungen hinsichtlich durchschnittlicher Bandbreite, Spitzenbandbreite, Latenz, Jitter und Verlusttoleranz:

a. Datei-Server

b. Druck-Server

c. Digitale Bibliothek

d. Routineüberwachung von entfernten Wetterinstrumenten

e. Sprache

f. Videoüberwachung eines Warteraums

g. Fernsehsendung

31. Angenommen ein gemeinsam genutztes Medium M bietet Hosts A_1, A_2, ..., A_N im Rundumverfahren (»Round-Robin«) Gelegenheit, ein Paket zu übertragen. Hosts, die nichts zu übertragen haben, geben M sofort frei. In welcher Hinsicht unterscheidet sich dies von STDM? Wie verhält es sich mit der Netzauslastung dieser Methode im Vergleich zu STDM?

★ 32. Betrachten Sie ein einfaches Protokoll für die Übertragung von Dateien über eine Verbindungsleitung. Nach einer gewissen Anfangsverhandlung sendet A Datenpakete mit einer Größe von 1 KB an B. Darauf antwortet B mit einer Bestätigung. A wartet immer auf jede Bestätigung, bevor er das nächste Datenpaket sendet. Man nennt dies *Stop-and-Wait*. Bei überfälligen Paketen wird davon ausgegangen, dass sie verlorengegangen sind, und folglich erneut übertragen werden müssen.

a. Erläutern Sie für den Fall, bei dem keine Pakete verlorengegangen sind und keine Übertragung wiederholt werden muss, warum es nicht nötig ist, eine »Sequenznummer« in die Paket-Header einzubeziehen.

b. Die Verbindungsleitung kann gelegentlich Pakete verlieren, die ankommenden Pakete werden aber immer in der Reihenfolge empfangen, in der sie gesendet wurden. Genügt in diesem Fall eine 2-Bit-Sequenznummer (N mod 4), damit A und B eventuell verlorene Pakete erkennen und erneut senden können? Wäre eine 1-Bit-Sequenznummer ausreichend?

c. Die Verbindungsleitung stellt Pakete außer der Reihe zu, wobei ein Paket bis zu 1 Minute nach darauf folgenden Paketen eintreffen kann. Auf welche Weise ändern sich dadurch die Anforderungen an die Sequenznummer?

★ 33. Hosts A und B sind über eine Verbindungsleitung miteinander verbunden. Host A überträgt fortwährend die aktuelle Zeit von einer Hochpräzisionsuhr in einer gleichbleibenden Rate, die so hoch ist, dass die gesamte verfügbare Bandbreite verbraucht wird. Host B liest diese Zeitwerte und schreibt sie jeweils paarweise mit seiner eigenen Zeit, die er einer lokalen, auf die von A synchronisierten Uhr entnimmt. Führen Sie qualitative Beispiele von B's Ausgabe unter folgenden Bedingungen auf:

a. Hohe Bandbreite, hohe Latenz, niedriges Jitter

b. Geringe Bandbreite, hohe Latenz, hohes Jitter

c. Hohe Bandbreite, niedrige Latenz, niedriges Jitter, gelegentlicher Datenverlust
Beispielsweise kann eine Verbindungsleitung ohne Jitter, einer zum Schreiben jedes zweiten Zeittakts ausreichend hohen Bandbreite und einer Latenz von 1 Zeittakt etwas wie (0000, 0001), (0002, 0003), (0004, 0005) ergeben.

34. Verwenden Sie das als Beispiel im Buch angegebene Socket-Programm simplex-talk. Starten Sie einen Server und einen Client in getrennten Fenstern. Während dieser erste Client läuft, starten Sie 10 weitere Clients, die sich auf den gleichen Server aufschalten. Diese Clients sollten im Hintergrund gestartet und ihre Eingaben von einer Datei umgeleitet werden. Was passiert mit diesen 10 Clients? Schlägt ihre connect()-Operation fehl, läuft ein Timeout ab oder sind sie erfolgreich? Blockieren irgendwelche anderen Aufrufe? Beenden Sie nun den ersten Client. Was passiert? Versuchen Sie das auch mit einem auf 1 gesetzten Server-Wert MAX_PENDING.

35. Ändern Sie das Socket-Programm simplex-talk so ab, dass der Server jedes Mal, wenn der Client eine Zeile an ihn sendet, die Zeile an den Client zurückschickt. Client und Server müssen jetzt recv() und send() abwechselnd aufrufen.

36. Verändern Sie das Socket-Programm simplex-talk so, dass es statt TCP jetzt UDP als Transportprotokoll benutzt. Hierfür müssen Sie für den Client und für den Server SOCK_STREAM auf SOCK_DGRAM abändern. Anschließend entfernen Sie im Server die Aufrufe von listen() und accept() und ersetzen Sie die beiden verschachtelten Schleifen am Ende durch eine einzige Schleife, die recv() mit Socket s aufruft. Beobachten Sie das Verhalten, wenn sich zwei solche UDP-Clients gleichzeitig mit den gleichen UDP-Server verbinden, und vergleichen Sie es mit dem TCP-Verhalten.

37. Untersuchen Sie die verschiedenen Optionen und Parameter, die Sie bei einer TCP-Verbindung verändern können. (Starten Sie man tcp in Unix). Experimentieren Sie mit unterschiedlichen Parameter-Einstellungen, um den Einfluss auf die TCP-Performance zu untersuchen.

38. Die Unix-Utility ping kann benutzt werden, um die RTT zu anderen Internet-Hosts festzustellen. Lesen Sie die Seite über ping und benutzen Sie die Utility, um die RTT zu www.cs.princeton.edu in New Jersey und www.cisco.com in Kalifornien zu ermitteln. Messen Sie die RTT-Werte zu unterschiedlichen Tageszeiten und vergleichen Sie die Ergebnisse. Worauf sind die Unterschiede Ihrer Meinung nach zurückzuführen?

39. Die Unix-Utility traceroute bzw. ihr Windows-Gegenstück tracert kann benutzt werden, um die Reihenfolge von Routern, die eine Nachricht passieren muss, festzustellen. Ermitteln Sie mit einer der beiden Utilities den Weg von Ihrem Standort zu einem anderen. Wie korreliert die Anzahl von Hops im Vergleich zu den mit ping ermittelten RTT-Zeiten? Wie korreliert die Anzahl von Hops mit der geographischen Entfernung?

40. Benutzen Sie traceroute, um einige der Router Ihres Unternehmens zu identifizieren (bzw. zu zeigen, dass keine benutzt werden).