Anja Haake, Jörg Haake, Christian Icking, Lihong Ma

Betriebssysteme und Rechnernetze

Kurseinheit 3:
Anwendungen und Transport

Fakultät für
Mathematik und Informatik

FernUniversität in Hagen

Inhalt

Teil I: Betriebssysteme

Teil II: Rechnernetze

Die Autoren

Prof. Dr.-Ing. Anja Haake

Studium der Informatik an der Universität Dortmund zur Dipl.-Inform. 1988, dann Universität Dortmund und Institut für Integrierte Publikations- und Informationssysteme der GMD in Darmstadt, Promotion in Informatik 1996 an der Technischen Universität Darmstadt, danach Innovation Center der TLC GmbH und FernUniversität in Hagen, seit 2006 Professorin für Informatik und Wirtschaftsinformatik an der Fachhochschule Dortmund.

Univ.-Prof. Dr.-Ing. Jörg Haake

Studium der Informatik an der Universität Dortmund zum Dipl.-Inform. 1987, dann BCT GmbH und Institut für Integrierte Publikations- und Informationssysteme der GMD in Darmstadt, Promotion in Informatik 1995 an der Technischen Universität Darmstadt, Bereichsleiter am Fraunhofer Institut für Integrierte Publikations- und Informationssysteme (IPSI) in Darmstadt, seit 2001 Professor für Praktische Informatik an der FernUniversität in Hagen.

apl. Prof. Dr. rer. nat. Christian Icking

Studium der Mathematik und Informatik an der Universität Münster und der Université Pierre und Marie Curie Paris VI zur Maîtrise in Mathematik 1984, dann auch École Nationale Supérieure de Techniques Avançées Paris zum DEA (Diplom) Informatik 1985, danach Universität Karlsruhe, Universität Freiburg, Universität Essen und FernUniversität in Hagen, hier Promotion und Habilitation in Informatik 1994 und 2002, außerplanmäßiger Professor 2010.

Dr. rer. nat. Lihong Ma

Studium der Mathematik an der Universität Yunnan, Bachelor in Mathematik 1984, dann Technische Universität Kunming, Universität Karlsruhe, Universität Freiburg, dort Dipl.-Math. 1990, danach Universität Essen und FernUniversität in Hagen, hier Promotion in Informatik 2000, wissenschaftliche Mitarbeiterin.

Kurseinheit 3

Anwendungen und Transport

Einleitende Bemerkungen

Die beiden Kurseinheiten 3 und 4 beschäftigen sich mit Rechnernetzen. *Rechnernetze* oder *Computernetzwerke* bezeichnen mehrere miteinander verbundene Computer. Mit Hilfe eines Rechnernetzes können Anwendungen, die auf den angeschlossenen Computern laufen, miteinander kommunizieren, z. B. kann eine E-Mail-Anwendung über ein Rechnernetz eine Nachricht an einen Empfänger verschicken. Die an Rechnernetze angeschlossenen Computer, auf denen die Anwendungsprogramme laufen, werden oft als *Hosts* oder *Endsysteme* bezeichnet, und *Server* nennen wir solche, die bestimmte Dienstleistungen für andere Hosts bereitstellen.

Rechnernetze
Computernetzwerke

Hosts
Endsysteme
Server

Rechnernetze sind üblicherweise in mehreren übereinander liegenden Schichten organisiert; dieses Konzept von Abstraktion und Kapselung wird in der Informatik oft verwendet, siehe Abschnitt 1.3.3. Die unterste Schicht behandelt die physikalische Bitübertragung zwischen direkt verbundenen Geräten. Höhere Schichten stellen zusätzliche Funktionen zur Verfügung, wie Fehlerkorrektur oder Routing. Hierbei verwenden höhere Schichten jeweils die Funktionen der darunter liegenden Schicht. Auf der höchsten Schicht finden wir die Anwendungen, mit denen die Benutzer arbeiten, z. B. Web-Browser oder Buchungssysteme.

Häufig werden Rechnernetze in aufsteigender Reihenfolge ihrer Schichten, von Schicht 1 (Bitübertragung) bis zur Anwendungsschicht, eingeführt. Bei diesem Bottom-Up Ansatz bleibt allerdings oft unklar, warum die Funktionen einer Schicht so definiert sind, da ja die sie verwendende darüber liegende Schicht erst später behandelt wird. Darum gehen wir in Kurseinheit 3 und 4 den umgekehrten Weg: beginnend mit der Anwendungsschicht behandeln wir diesen Stoff in der Top-Down-Reihenfolge bis zur Bitübertragungsschicht. Außerdem werden wir frühzeitig ein konkretes Netzwerk – das Internet – einführen, um die praktische Bedeutung der Konzepte klarer darzustellen. Dazu gehört auch ein Überblick über den Aufbau von komplexen Rechnernetzen. Aus Sicht der Komponenten eines Netzwerks und ihrer Verbindungen (der sogenannten Netzwerktopologie) arbeiten wir uns von außen, also von den Rechnern, mit denen die Nutzer arbeiten, nach innen in den Kern des Netzwerks vor.

Schichten

Top-Down

Netzwerkanwendungen

Von diesem Top-Down Ansatz versprechen wir uns eine leichtere Verständlichkeit des Stoffes: Fast alle Anwender sind heute mit Netzwerkanwendungen wie *E-Mail* oder dem *World Wide Web* (*WWW*) vertraut. Wir gehen davon aus, dass die Erläuterung der Funktionsweise von solchen Netzwerkanwendungen zu Beginn des Kurses eine motivierende Wirkung hat. Die Notwendigkeit der Dienstleistungen und Konzepte der unteren Schichten ergibt sich aus den Bedürfnissen der oberen Schichten, d. h. letztendlich aus den Bedürfnissen des Anwenders, die intuitiv einzusehen sind. Darüber hinaus betont der Top-Down Ansatz die Anwendungsschicht, die die größten Wachstumsmöglichkeiten im Bereich der Rechnernetze darstellt. Außerdem bekommen Sie schon früh einen Einblick in die Entwicklung von Netzwerkanwendungen.

Natürlich kann das Thema Rechnernetze nicht in zwei Kurseinheiten erschöpfend behandelt werden. Es ist unser Ziel, Ihnen in diesen beiden Kurseinheiten ein grundlegendes und praktisch orientiertes Verständnis des Aufbaus und der Funktionsweise von Rechnernetzen und der damit zusammenhängenden Probleme zu vermitteln. Außerdem wollen wir die Benutzung von Rechnernetzen zur Realisierung verteilter Anwendungen an einigen Beispielen illustrieren. Wenn Sie dieses Thema im weiteren Verlauf Ihres Studiums vertiefen wollen, dann können Sie eine Reihe weiterführender Kurse belegen.

weiterführende Kurse

Der Kurs *1802 Betriebssysteme* betrachtet den Aufbau und die Funktionsweise moderner Betriebssysteme. Im Kurs *1678 Verteilte Systeme* wird genauer dargestellt, wie verteilte Systeme auf der Basis moderner Betriebssysteme und Rechnernetze entworfen und realisiert werden können. Der Kurs *1690 Kommunikations- und Rechnernetze* betrachtet mit mehr Detail, wie Netzwerke entworfen werden und wie sie funktionieren. Schließlich betrachtet der Kurs *1866/67/68 Sicherheit im Internet*, mit welchen Mechanismen im Internet Sicherheit gegen Spione und Einbrecher gewährleistet werden kann.

Trotz sehr sorgfältigen Korrekturlesens ist leider damit zu rechnen, dass sich Fehler in den Kurs eingeschlichen haben. Sie helfen uns und Ihren Kommiliton(inn)en, wenn Sie uns auf Fehler, Ungenauigkeiten oder Schwierigkeiten beim Durcharbeiten des Kurses hinweisen! Bitte schicken Sie dazu Ihre Hinweise an die Kursbetreuer.

3.1 Einführung

Prozesse

Von der Anwendungsseite aus betrachtet geht es bei dem Thema Rechnernetze um die Kommunikation zwischen Prozessen, die auf verschiedenen Endsystemen laufen. In Kurseinheit 1 und 2 haben wir das Konzept des Prozesses zur Ausführung von Programmen oder Anwendungen in einem Endsystem kennen gelernt. Wenn Prozesse auf dem gleichen Endsystem laufen, kommunizieren sie miteinander mit Hilfe der prozessübergreifenden Kommunikation, wie z. B. über einen gemeinsamen Speicher (vgl. Abschnitt 1.6 und Abschnitt 2.3).

In Kurseinheit 3 und 4 geht es nun darum, wie Prozesse miteinander kommunizieren, die auf unterschiedlichen Endsystemen mit durchaus auch unterschiedlichen Betriebssystemen laufen. Wenn ein Anwender ein Programm startet, dann wird das Programm vom Betriebssystem als Prozess ausgeführt.

Dabei kann ein Programm zur Ausführungszeit durchaus mehrere Prozesse starten. Wenn ein Programm nun Dienste eines anderen Programms aufruft, dann müssen zur Laufzeit die Prozesse, welche die entsprechenden Programme ausführen, miteinander kommunizieren. Laufen diese Prozesse auf verschiedenen Rechnern, z. B. das E-Mail-Leseprogramm auf einem PC und ein E-Mail-Serverprogramm auf einem Server, dann spricht man auch von *verteilten Anwendungen* oder *Netzwerkanwendungen*. Die Konzepte aus dem Gebiet der Rechnernetze können also immer aus der Perspektive betrachtet werden, wie sie dazu beitragen, verteilte Anwendungen zu ermöglichen.

verteilte Anwendungen

Netzwerkanwendungen

Im Folgenden werden Anwendungsbeispiele gegeben sowie die gebräuchlichste Klassifikation für Computernetze nach ihrer Größe und Komplexität eingeführt.

In der Entwicklungsgeschichte der Computertechnologie hat sich neben dem Konzept einer vorwiegend zentralen Datenverarbeitung auch das Konzept einer eher *verteilten Datenverarbeitung* entwickelt. Während bei der zentralen Datenverarbeitung die Verarbeitung der Daten auf einem zentralen Rechner passiert, werden bei der verteilten Datenverarbeitung die Daten auf mehreren Rechnern arbeitsteilig verarbeitet. Die Motive, Rechner über Kommunikationsnetze miteinander zu verbinden, haben sich im Laufe der Zeit geändert. Stand zu Beginn dieser Entwicklung die Datenfernverarbeitung im Vordergrund, erlauben heute internationale Kommunikationsnetze die Kommunikation und Kooperation zwischen weltweit verteilten Rechnern und ihren Benutzern. Der sekundenschnelle Austausch von Nachrichten und der Zugriff auf Daten über Kontinente hinweg ist für viele Anwendungszwecke nicht mehr wegzudenken.

zentrale Datenverarbeitung

verteilte Datenverarbeitung

Ein *klassisches Anwendungsbeispiel* der Datenfernverarbeitung sind Flugbuchungssysteme, bei denen viele Datensichtgeräte der einzelnen Büros einer oder mehrerer Fluggesellschaften mit einem zentralen Datenverarbeitungssystem verbunden sind. Buchungswünsche einzelner Kunden können somit dezentral erfasst, zentral ausgewertet und dezentral bestätigt oder geändert werden. Jedes Büro steht in Kontakt zum Rechner. Der zentrale Rechner kennt ständig den aktuellen Stand aller Buchungen und kann dementsprechend Buchungswünsche bestätigen, ablehnen oder Ersatzvorschläge ausarbeiten. Ferner können zentral Passagierlisten etc. erstellt und dezentral, z. B. auf den betroffenen Flughäfen, ausgegeben werden. Ähnliche Beispiele finden sich im Bereich großer Banken, Versandhäuser usw. Die skizzierten Aufgaben wären ohne die Möglichkeit der Datenfernverarbeitung nicht durchführbar.

klassisches Anwendungsbeispiel

Zur Datenübertragung selbst können bestehende Datennetze oder eigens eingerichtete private Netze verwendet werden, die auch Richtfunkstrecken und Satellitenverbindungen einschließen können.

Die Gründe, Rechner miteinander zu vernetzen, lassen sich grob wie folgt klassifizieren:

- Beim *Betriebsmittel- oder Funktionsverbund (resource sharing)* steht das Ziel im Vordergrund, sämtliche im Netz vorhandenen Betriebsmittel wie Daten, Programme oder Geräte auf jedem am Rechnernetz beteiligten Rechner zugänglich zu machen. Auch das gesamte Kommunikationsnetz kann in diesem Zusammenhang als Betriebsmittel betrachtet

Betriebsmittelverbund

Funktionsverbund

Datenverbund

werden. Besteht das Hauptziel eines Betriebsmittelverbundes darin, auf netzweit verteilte Datenbestände zuzugreifen, spricht man auch von einem *Datenverbund (data sharing)*, wobei auch das mehrfache Halten von Datenbeständen aus Sicherheitsgründen eine Rolle spielen kann.

Last-, Leistungs- und Wartungsverbund

- Der *Last- und Leistungsverbund* gestattet es, anfallende Rechenlasten gleichmäßig auf mehrere Rechner zu verteilen bzw. mehrere Rechner für das Lösen einer einzigen umfangreichen Aufgabe einzusetzen. Ein *Wartungsverbund* schließlich ermöglicht die zentrale Wartung, Störungserkennung und -behebung bei einer Vielzahl räumlich verteilter Rechner.

Rechnernetz als Kommunikationsmedium

- Seit dem internationalen Siegeszug des Internets und insbesondere des auf ihm realisierten World Wide Web (WWW) stehen der Online-Zugriff auf weltweit erfasste Dokumente sowie die Kommunikation und Kooperation zwischen menschlichen Benutzern im Vordergrund des Einsatzes von Rechnernetzen. Der Austausch von Nachrichten durch E-Mail, das Arbeiten an gemeinsamen Projekten, oder das Veranstalten internationaler Videokonferenzen sind aus dem beruflichen und zunehmend auch aus dem privaten Alltag nicht mehr wegzudenken. Das Fachgebiet der Informatik, das sich mit der Unterstützung der Zusammenarbeit von Menschen durch den Einsatz von Computertechnologie befasst, wird als *Computerunterstützte Gruppenarbeit* (*Computer Supported Cooperative Work*, CSCW) bezeichnet.

Je nach ihrer räumlichen Ausdehnung unterteilt man Rechnernetze in

- Lokale Netze (*Local Area Network*, LAN) mit Ausdehnungen von höchstens wenigen Kilometern,

- Regionale Netze (*Metropolitan Area Network*, MAN) mit Ausdehnungen von ca. 100 km und

- Weitverkehrsnetze (*Wide Area Network*, WAN), die ganze Länder und Kontinente und im Weitverkehrsverbund die gesamte Erde umspannen.

Wir werden später sehen, dass an lokale Netze und an Weitverkehrsnetze in der Regel sehr unterschiedliche Anforderungen gestellt werden und sich diese in ihren Architekturen und Betriebsweisen erheblich unterscheiden.

Selbst Rechnernetze unterschiedlichster Konfiguration und Technologie lassen sich mit Hilfe sogenannter Switches oder Bridges, Router und Gateways miteinander vernetzen. Der einzelne Rechner oder Benutzer gewinnt so den Eindruck, er sei über ein einziges erdumspannendes Kommunikationsnetz mit anderen Rechnern oder Benutzern verbunden. Abbildung 3.1 skizziert eine Situation, in der verschiedene Rechner und lokale Netze über ein Netz von Weitverkehrsnetzen miteinander verbunden sind.

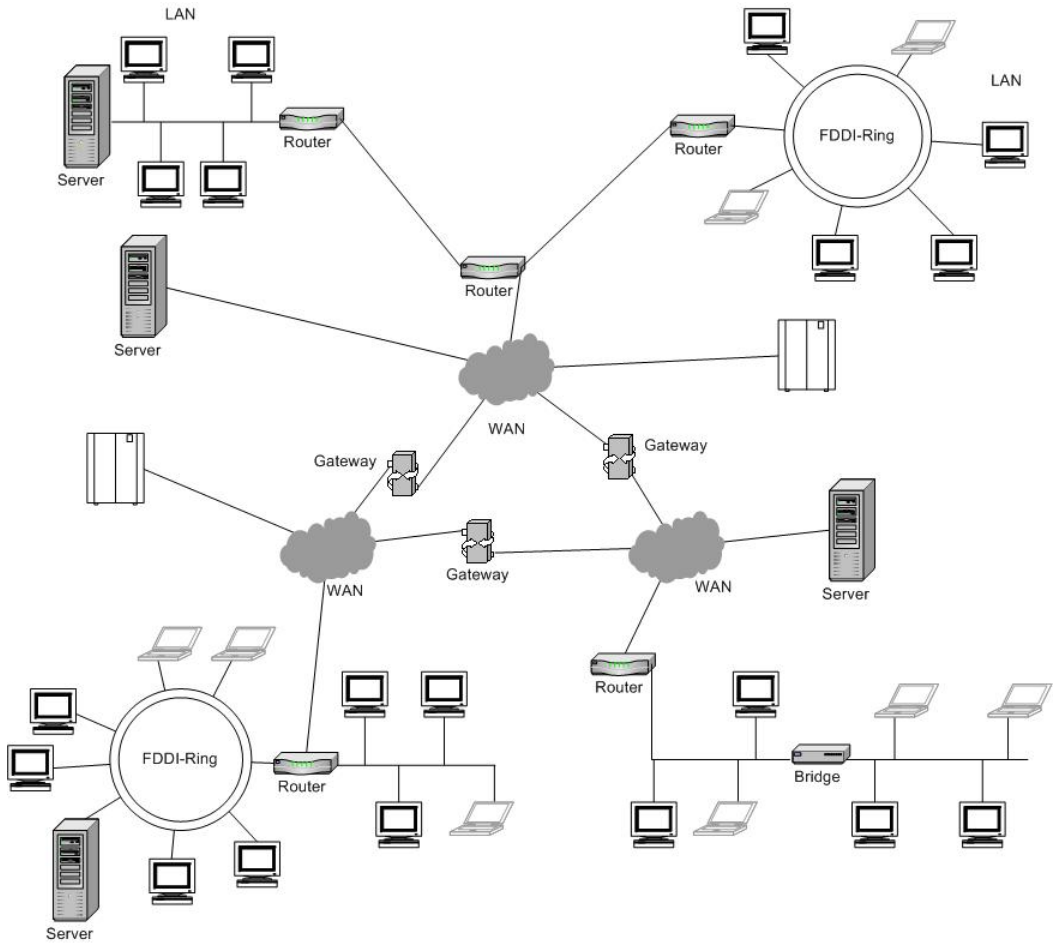

Abbildung 3.1: Weltweiter Verbund von Rechnernetzen.

3.2 Überblick über Rechnernetze

3.2.1 Kommunikation, Protokolle und Schichtenarchitekturen

Prozesse auf zwei unterschiedlichen Endsystemen kommunizieren miteinander durch den Austausch von Nachrichten über ein Computernetzwerk. Ein sendender Prozess erzeugt und sendet *Nachrichten* über das Netzwerk. Ein empfangender Prozess empfängt diese Nachrichten und antwortet möglicherweise, indem er Nachrichten zurück schickt. Die Regeln und Konventionen, auf denen diese Kommunikation basiert, werden *Protokolle* genannt. Ein Protokoll ist eine Vereinbarung zwischen kommunizierenden Parteien über den Ablauf der Kommunikation. Ein Protokoll definiert

Nachrichten

Protokolle

- die Nachrichtentypen (z. B. Anforderungen und Antworten),
- ihre Syntax (d. h. Felder und Feldabgrenzungen in den verschiedenen Nachrichtentypen),

- die Semantik der Nachrichtentypen und der Felder (die im übrigen nur für das Protokoll von Belang sind) und
- die Regeln, wann und wie ein Prozess Nachrichten eines bestimmten Typs sendet bzw. wie er auf diese Nachrichten reagiert (beispielsweise durch das Senden von Antwortnachrichten oder Ausführen von Programmen, wie z. B. Lesen oder Schreiben von Dateien).

Vereinfacht kann man sagen: Protokolle sind Regeln, die das Format, den Inhalt von Feldern und seine Bedeutung sowie die Reihenfolge gesendeter Nachrichten festlegen.

Um die Komplexität überschaubar zu halten, organisieren Netzwerkdesigner Protokolle sowie die Netzwerkhardware und -software, mit der die Protokolle implementiert werden, in Schichten. Bei einer *geschichteten Protokollarchitektur* gehört jedes Protokoll zu einer bestimmten Schicht. Ein Protokoll der Schicht n definiert also genau die Regeln, mit denen zwei Endsysteme Nachrichten auf der Schicht n austauschen. Tatsächlich werden Daten nicht direkt von Schicht n eines Rechners zu Schicht n eines anderen übertragen. Vielmehr nutzt eine Schicht den *Dienst* der direkt darunter liegenden Schicht, den es über einen *Dienstzugangspunkt* erreicht, der auch als *Schnittstelle* (*Interface*) bezeichnet wird. Nur in der untersten Schicht findet die eigentliche Kommunikation statt, die Nachrichtenübertragung.

geschichtete Protokollarchitektur

Dienst Schnittstelle

Ein Dienst, der der direkt darüber liegenden Schicht angeboten wird, ist eine Menge von Programmen oder Operationen, die Aktionen innerhalb der Schicht definieren und die wiederum Dienste der direkt darunter liegenden Schicht verwenden. Die Definition vom Dienst sagt nur, *was* die Schicht leistet, sie sagt aber *nicht*, *wie* die direkt darüber liegenden Schicht auf den Dienst zugreift und *wie* die Schicht arbeitet. Die Schnittstelle zwischen zwei benachbarten Schichten definiert, *wie* die obere Schicht auf den Dienst der unteren zugreift. Ein Protokoll einer Schicht *implementiert* einen Dienst der Schicht.

Dienst: was

Schnittstelle: wie

Protokoll implementiert

Zur Verdeutlichung wollen wir die Kommunikation zwischen zwei Unternehmen per Brief als vereinfachte dreischichtige Architektur betrachten, siehe Abbildung 3.2. Die oberste Schicht bezeichnen wir als Schreib/Leseschicht, die mittlere als Briefkommunikationsschicht und die untere als Posttransportschicht. Herr Müller möchte Frau Schmidt eine Nachricht zukommen lassen. Er schreibt sie auf ein Blatt Papier und legt es in den Postkorb seines Sekretärs und versieht ihn mit dem Empfänger Frau Schmidt. Dieser steckt das Blatt in einen Briefumschlag, findet die Adresse von Frau Schmidt heraus und schreibt sie darauf. Dann frankiert er den Brief und übergibt ihn durch Einwurf in den gelben Postbriefkasten an die Posttransportschicht. Die Post sorgt für die Leerung des Postbriefkastens, versieht dem Brief mit einem maschinenlesbaren Strichcode und transportiert ihn zum Hausbriefkasten des Empfängers. Die Sekretärin von Frau Schmidt öffnet den Brief und legt ihn in die Postmappe, die Frau Schmidt zu bearbeiten hat. Die Dienstzugangspunkte sind hierbei der Postkorb und die Postmappe sowie der gelbe Postbriefkasten und der Hausbriefkasten.

Beispiel Briefkommunikation

Auf der Briefkommunikationsschicht besteht der Dienst darin, dass der Sekretär einen Brief zur Post bringt und die Sekretärin ihn an Frau Schmidt aus-

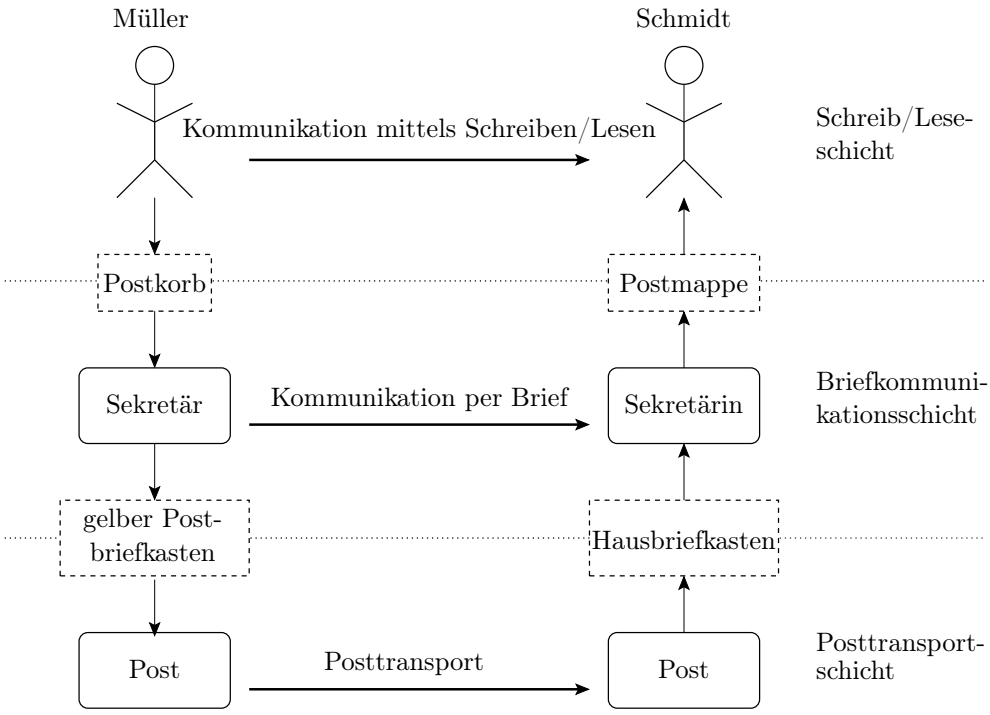

Abbildung 3.2: Briefkommunikation als vereinfachte geschichtete Architektur.

liefert. Wie der Sekretär die Adresse von Frau Schmidt findet, wie er adressiert, frankiert und den Brief zur Post bringt und wie die Sekretärin ihn aufmacht, dazu macht der Dienst keine Angabe. Der Dienst auf der Posttransportschicht besteht darin, einen adressierten und frankierten Brief bei seinem Empfänger abzuliefern. Er sagt aber nichts darüber, wie der Brief befördert wird. Jede Schnittstelle definiert, wie der Dienst der direkt darunter liegenden Schicht benutzt wird.

Das Protokoll der Schreib/Leseschicht ist die geschriebene Sprache, das Protokoll der Briefkommunikationsschicht regelt das Einstecken des Briefs in einen genormten Umschlag sowie die Adressierung und Frankierung beim Absender und das Öffnen beim Empfänger. Bei der Posttransportschicht regelt das Protokoll, wie die Beförderung von Briefen organisiert wird.

Aufbereitung von Nachrichten

In unserem Beispiel sieht man, dass jede Schicht die Nachrichten für die untergeordnete Schicht aufbereiten kann. Dabei muss allerdings sicher gestellt werden, dass die Inhalte der Nachrichten, nämlich die Daten, die von der oberen Schicht übergeben wurden, nicht verändert werden. Eine Schicht darf einzig die Daten, die an ihrem Dienstzugangspunkt eintreffen, mit Headern und Trailern versehen (sozusagen umklammern). Die Header und Trailer fügen der Nachricht Informationen für den Empfänger auf derselben Schicht hinzu. In unserem Beispiel sind Header und Trailer in der Briefkommunikationsschicht der Briefumschlag mit der Adresse des Empfängers und in der Posttransportschicht der Strichcode. Außerdem enthält der Header eine Briefmarke. Als Trailer könnte man die Absenderadresse interpretieren. Außer dem Hinzufügen von Headern und Trailern kann eine Schicht die ihr übergebenen Daten noch zu größeren Paketen zusammenfassen oder in kleinere Pakete aufteilen.

Nehmen wir an, dass in der Posttransportschicht nur Standardbriefe zu 20 g übertragen werden können. Das stellt die Posttransportschicht dadurch sicher, dass zu schwere Briefe nicht ihren Dienstzugangspunkt passieren dürfen. Wenn nun der Briefkommunikationsschicht als Daten die Inhalte des Buches „Computernetze“ übergeben werden, so stellt das die Briefkommunikationsschicht vor ein Problem: das Buch wiegt mit 1295 g (in der Auflage von 2002) erwiesenermaßen mehr als 20 g. Was kann die Briefkommunikationsschicht nun tun, damit sie das Buch doch über die Posttransportschicht versenden kann? Die Antwort ist unter diesen Bedingungen relativ simpel: Die Briefkommunikationsschicht (der Sekretär von Herrn Müller) teilt das Buch in Einheiten zu 20 g auf und schickt jede dieser 65 Einheiten einzeln auf den Weg. Die einzelnen Teile werden noch mit fortlaufenden Nummern versehen, auf Basis derer die Empfängerseite, d. h. die Sekretärin von Frau Schmidt, das Buch wieder zusammensetzen kann.

In unserem Beispiel sehen wir auch, dass die Einzelheiten, wie Briefe durch die Post gesammelt, transportiert und zugestellt werden, uninteressant für die Benutzer der Post sind. Das Konzept der Schichten vereinfacht die Realisierung eines großen und komplexen Systems, denn es ist leichter, die Realisierung des von einer Schicht bereitgestellten Dienstes zu ändern. Solange die Schicht den gleichen Dienst für die darüber liegende Schicht bereitstellt und die gleichen Dienste von der darunter liegenden Schicht benutzt, bleibt der Rest des Systems unverändert. Wenn z. B. die Post ihre Transportmethoden ändert, bleibt der Rest des Briefkommunikationssystems unverändert.

Dienst

Dienstmodell

Ein *Dienst* wird durch eine Menge von Programmen oder Operationen definiert, die eine Schicht der darüber liegenden Schicht zur Verfügung stellt, dieses Konzept nennen wir das *Dienstmodell*. Ein Dienst sagt aber nichts darüber, wie diese Operationen implementiert (realisiert) werden. Die Implementierung der Dienste einer Schicht benutzt aber die Dienste der darunter liegenden Schicht. Im Beispiel wird in der Briefkommunikationsschicht der Dienst angeboten, dass ein Brief aus einem Postkorb in einer Postmappe landet. In der Transportschicht wird der Brief vom gelben Briefkasten zum Hausbriefkasten befördert.

Ein Protokoll ist eine Menge von Regeln, die Format und Bedeutung der innerhalb einer Schicht ausgetauschten Nachrichten festlegt. Ein Protokoll dient der Erbringung eines gewünschten Dienstes auf der Basis der Nutzung der untergeordneten vorhandenen Dienste. Die Einheiten der kommunizierenden Seiten erbringen mit Protokollen ihre definierten Dienste, z. B. erbringen die Quell- und Zielpostämter das Befördern der Briefe. Man kann die Protokolle verändern, solange dies nichts an den für die Dienstbenutzer sichtbaren Diensten ändert.

Protokolldateneinheiten

Ein Protokoll der Schicht n wird auf die Netzwerkeinheiten verteilt, wir nennen diese Netzwerkeinheiten Quellhost und Zielhost. Die Prozesse, die auf dem Quellhost und Zielhost laufen, kommunizieren miteinander durch Austausch von Schicht-n-Nachrichten. Diese Nachrichten nennt man *Protokolldateneinheiten* (*Protocol Data Units, PDUs*) der Schicht n bzw. n-PDUs. Das Schicht-n-Protokoll definiert den Inhalt und das Format einer n-PDU sowie die Art, in der die n-PDUs zwischen den Netzwerkeinheiten ausgetauscht werden.

Zusammengenommen werden die Protokolle der verschiedenen Schichten als *Protokollstapel* (*Protocol Stack*) bezeichnet, da sie aufeinander aufbauen. Protokollstapel

Wir betrachten nun ein Beispiel eines Netzwerks, das seine Kommunikationsprotokolle in vier Schichten organisiert. Abbildung 3.3 zeigt,dass die Anwendung, die auf der höchsten Schicht (hier also Schicht 4) läuft, eine Nachricht M erzeugt. Jede auf dieser höchsten Schicht erzeugte Nachricht ist eine 4-PDU. Die Nachricht M selbst kann weiter strukturiert sein, so wie eine Struktur in einer Programmiersprache verschiedene Felder enthalten kann, z. B. ein Record in Pascal, ein Struct in C, oder eine Instanz einer Klasse in Java. Die Definition und Interpretation der Felder einer Nachricht sind Sache der Anwendung.

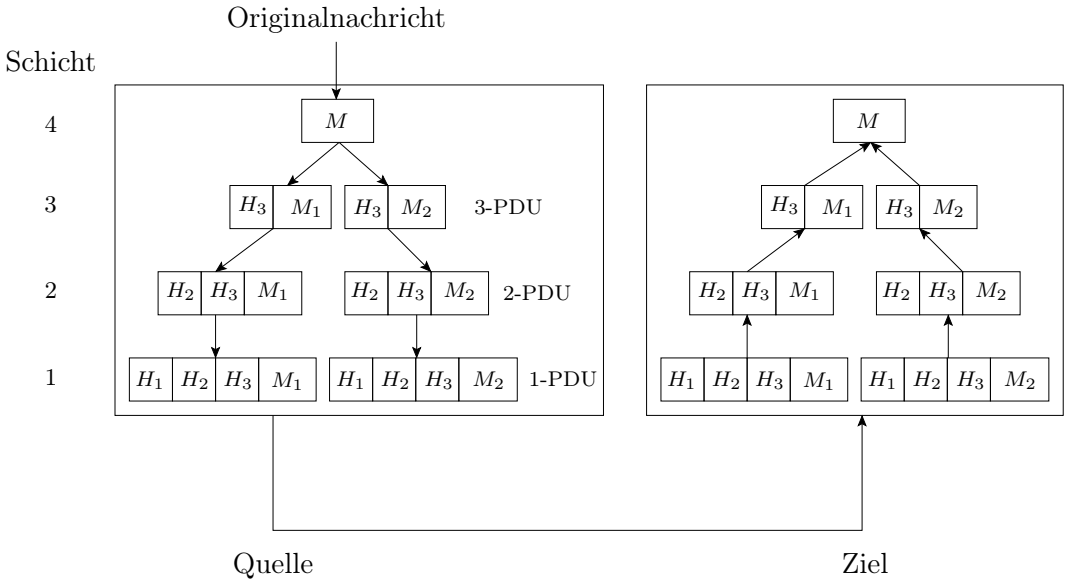

Abbildung 3.3: Verschiedene PDUs auf unterschiedlichen Schichten der Protokollarchitektur.

Im Quellhost wird die Nachricht M im Protokollstapel von der Schicht 4 nach unten an die Schicht 3 *weitergegeben*. Die Schicht 3 im Quellhost teilt die 4-PDU M in zwei Teile, M_1 und M_2, auf und fügt dann zu M_1 und M_2 sogenannte *Header*[1] hinzu, um zwei vollständige 3-PDUs zu erzeugen. Die Header enthalten hier die zusätzlichen Informationen, um es dem Empfänger in Schicht 3 zu ermöglichen, bei Empfang von M_1 und M_2 diese wieder zu M zusammensetzen zu können.[2] In der Quelle wird auf jeder Schicht ein weiterer Header hinzugefügt, bis eine Menge von 1-PDUs fertig ist. Diese 1-PDUs werden vom Quellhost auf einer physikalischen Leitung abgeschickt. Am anderen Ende empfängt der Zielhost 1-PDUs und leitet sie im Protokollstapel nach oben weiter. Auf jeder Schicht wird der entsprechende Header entfernt und die Header

[1] Eine untere Schicht kann die Daten, die sie bekommt, für den weiteren Transport verpacken. Das bedeutet, dass sie um die Daten herum zusätzliche Daten hinzufügen kann. Da die Daten eine Sequenz von Bit sind, kann das Hinzufügen vor den Daten (Header) oder nach den Daten (Trailer) passieren.

[2] Header werden auch dazu verwendet, damit Nachrichten ihr Ziel finden können, oder damit die Nachricht als solche erkennbar ist.

Nachrichten, falls notwendig, zusammengefügt und an die obere Schicht als PDU dieser Schicht weitergereicht. Schließlich wird M aus M_1 und M_2 wieder zusammengesetzt und an die Anwendung weitergeleitet.

Aufteilen und Zusammenfügen von Nachrichten

Eine Nachricht kann während ihres Transportes in tieferen Schichten nicht nur zerlegt (disassembliert) und im Zielsystem wieder zusammengesetzt (reassembliert) werden. Ebenso können verschiedene Nachrichten oder Pakete verkettet und wieder getrennt werden. Ein Beispiel für das Zusammenfassen der Pakete finden wir auf einer Schicht unterhalb der Posttransportschicht. Wenn der Brief sich seinen Weg durch die Verteilzentren sucht, so wird er meist anhand der Postleitzahl zu großen Einheiten in Form von Postsäcken zusammengefasst, die dann gemeinsam durch eine unterliegende Schicht, beispielsweise die Bahn, transportiert werden. Die Rollen sind hier genau gegensätzlich zum vorangegangenen Beispiel: Der Sender fasst viele kleine ihm übergebene Pakete zu einem großen Paket zusammen. Dabei muss er darauf achten, dass die einzelnen Teilpakete voneinander getrennt im großen Paket liegen. Der Sender übergibt das so entstandene Paket an die untergeordnete Schicht. Der Empfänger erhält von der untergeordneten Schicht das große Paket und trennt es wieder in die einzelnen Teilpakete auf.

3.2.2 Aufbau eines Computernetzwerks am Beispiel des Internets

Endsystem

Die Netzwerkanwendungen laufen auf den *Endsystemen*, den sogenannten Hosts. Die Endsysteme sind über *Kommunikationsleitungen* verbunden. Als physische Medien für Kommunikationsleitungen kommen zum Beispiel geführte Medien wie Koaxialkabel, Kupferkabel und Glasfaser oder ungeführte Medien wie Funkwellen im Raum zum Einsatz. Eine wichtige Kenngröße einer Leitung ist die *Übertragungsrate*, die man auch als *Bandbreite* bezeichnet und in Bit/Sekunde (bps, bits per second) misst. In diesem Zusammenhang verwenden wir das Dezimalsystem statt Binärsystem und es gilt:

Übertragungsrate
Bandbreite

$$1 \text{ Kbps} = 10^3 \text{ bps}, 1 \text{ Mbps} = 10^6 \text{ bps und } 1 \text{ Gbps} = 10^9 \text{ bps},$$

da es um die Geschwindigkeit der Übertragung und *nicht* um Speichergrößen geht.

Transitsystem

In einem größeren Computernetzwerk sind die Endsysteme nicht direkt miteinander verbunden. Zwischen ihnen liegen weitere Datenverarbeitungsgeräte, sogenannte *Transitsysteme*. Solch ein Transitsystem ist im Internet z. B. der Router. Ein *Router* nimmt Informationen von einer Eingangsleitung an und sendet sie über eine seiner ausgehenden Kommunikationsleitungen weiter. Das *Internetprotokoll* (*Internet Protocol, IP*) spezifiziert das Format, in dem im Internet Informationen zwischen Routern und Endsystemen ausgetauscht werden. Der Weg, den die Information durch das Rechnernetz von einem Endgerät über verschiedene Transitsysteme zum anderen Endgerät nimmt, bezeichnet man als *Route* oder *Pfad*.

Internetprotokoll
IP

Route
Pfad

Das Internet ist ein Verbund aus privaten und öffentlichen Netzwerken. Deshalb wird es auch als *Netzwerk aus Netzwerken* bezeichnet. Jedes an das Internet angeschlossene Netzwerk muss das IP ausführen und bestimmte Namens-

und Adresskonventionen einhalten, genaueres dazu werden wir vor allem in Kurseinheit 4 kennen lernen. Das wichtigste ist, dass das IP den Zusammenschluss von verschiedenen Netzwerken zu einem *einzigen logischen Netzwerk*, dem sogenannten Internet, macht.

Das Internet verwendet eine hierarchische Topologie. Endsysteme sind über Zugangsnetzwerke an lokale *Internet-Service-Provider* (*ISPs*) angeschlossen, siehe Abbildung 3.4. Ein *Zugangsnetzwerk* kann

Internet-Service-Provider

Zugangsnetzwerk

1. ein lokales Netzwerk (LAN) innerhalb eines Unternehmens oder einer Universität,
2. eine Wählverbindung über Telefonleitung (Modem, ISDN oder DSL) oder
3. ein Zugangsnetzwerk für mobile Endgeräte sein.

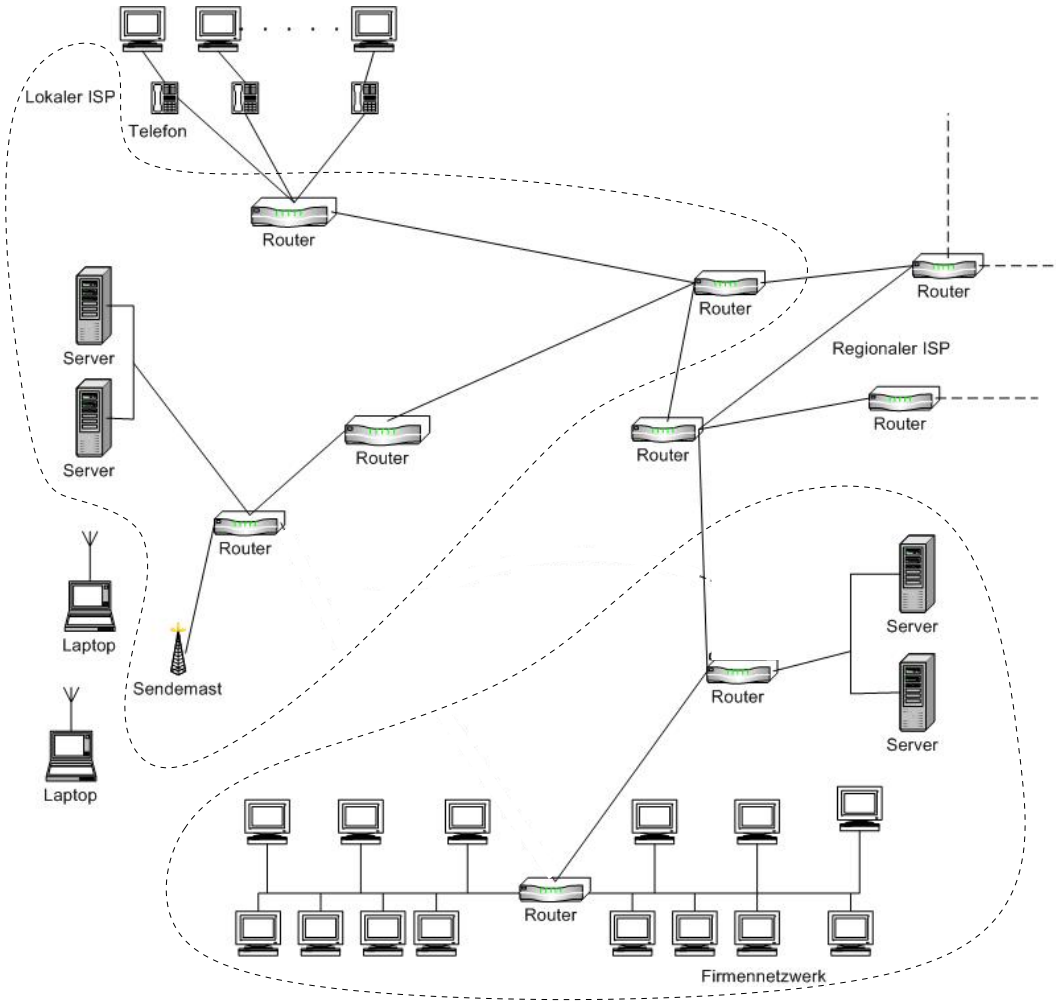

Abbildung 3.4: Ein Ausschnitt aus dem Internet: einige Heimcomputer und ein Firmennetzwerk sind über ISPs an das Internet angeschlossen.

Lokale ISPs sind ihrerseits mit regionalen ISPs verbunden, die wiederum mit nationalen und internationalen ISPs verbunden sind. Die nationalen und

internationalen ISPs sind untereinander auf der höchsten Ebene der Hierarchie verbunden. Neue Netzwerke können jederzeit an die bestehende Konstruktion angefügt werden. Die Basis des Internets sind *Internet-Standards*, die von der *IETF* (Internet Engineering Task Force) entwickelt werden. Sie bilden die Grundlage dafür, dass verschiedene Geräte und Netzwerke miteinander kommunizieren können. Die Standardisierungsdokumente nennt man *RFCs* (Request for Comments). Sie sind keine formellen Standards, werden aber als solche betrachtet. Heute gibt es über 8000 RFCs. Alle RFCs lassen sich im Internet unter der Adresse `http://www.ietf.org/rfc.html` abrufen.

Internet-Standards durch IETF

RFCs

Bei den Endsystemen, d. h. den Geräten, die am äußeren Rand des Internets angekoppelt sind, unterscheidet man gerne *Clients* und *Server*. Mit Server bezeichnet man alle Rechner, die Dienstleistungen für andere Rechner anbieten und mit Clients solche, die diese Dienste in Anspruch nehmen. Oft sind Server leistungsstärkere Maschinen, die an einem zentralen Standort aufgestellt und von einem Systemadministrator verwaltet werden, während Clients eher auf den Schreibtischen oder in den Taschen der Benutzer anzutreffen sind. Diese Unterscheidung ist aber ganz und gar nicht eindeutig, weil auch der schwächste Schreibtischrechner als Server einen Dienst anbieten kann und Server-Maschinen häufig als Clients auf die Dienste anderer Server zugreifen.

Clients

Server

Das Internet bietet verteilten Anwendungen auf den Endsystemen zwei Dienste, realisiert durch die Protokolle TCP [8] beziehungsweise UDP [28]. Während IP zur Weiterleitung von Informationen zwischen Endsystemen und Routern dient, leisten TCP oder UDP die Weiterleitung von Informationen zwischen einzelnen Prozessen, die auf den Endsystemen laufen. TCP und UDP benutzen dazu das IP. Sie liegen also auf einer höheren Schicht als das IP.

Betrachten wir nun den Netzwerkkern, d. h. das Internet ohne die Endgeräte. Der Netzwerkkern besteht aus einer Menge verbundener Router, welche die Endsysteme miteinander verbinden. Für den Aufbau eines Netzwerkkerns existieren zwei prinzipiell verschiedene Ansätze: *Leitungsvermittlung* und *Paketvermittlung*.

In *leitungsvermittelten Netzen* wird für jede Kommunikationsanforderung eine dauerhafte Verbindung aufgebaut. Dazu werden die auf einem Pfad vom Sender zum Empfänger zur Kommunikation zwischen den Endsystemen benötigten Ressourcen (Puffer, Leitungsbandbreite) für die Dauer der Sitzung reserviert, d. h. für den Zeitraum, in dem Sender und Empfänger Nachrichten austauschen wollen. Ein Beispiel dafür ist das alte, analoge Telefonnetz; hier wurde für jedes Telefongespräch eine Leitungsverbindung geschaltet, die erst bei Beendigung des Gesprächs wieder abgebaut wurde.

Leitungsvermittlung

In *paketvermittelten Netzen* werden diese Ressourcen nicht reserviert. Jede Nachricht, die in einer Sitzung gesendet wird, verwendet die Ressourcen nach Bedarf. Es kann daher vorkommen, dass Nachrichten oder Teile davon auf den Zugriff auf eine Verbindungsleitung warten müssen. Sie stehen dann z. B. in einer Warteschlange an.

Paketvermittlung

Das Internet ist ein paketvermitteltes Netz. Die Paketvermittlung erlaubt die gleichzeitige Nutzung von Pfaden oder Teilpfaden durch mehrere Endsysteme. In modernen, paketvermittelten Netzwerken schickt der Quellhost,

auf dem der Sender-Prozess läuft, lange Nachrichten als Folge kleiner *Pakete*. Dazu teilt er die Nachricht in Teile und gibt diese an die darunter liegende Schicht weiter (vgl. Abschnitt 3.2.1).

Übungsaufgabe 3.1 Recherchieren Sie, warum das Internet als ein paketvermitteltes Netz konzipiert worden ist.

`https://e.feu.de/1801-paketvermitteltes-netz`

Warum aber werden die Nachrichten in kleine Pakete aufgeteilt und beim Empfänger wieder zusammengesetzt, kann man nicht jede Nachricht gleich als Ganzes versenden? Diese besonders einfache Form der Paketvermittlung wird als *Nachrichtenvermittlung* bezeichnet; sie hat allerdings gravierende Nachteile: Damit die Nachricht bei der Übertragung immer intakt bleibt, muss jede Zwischenstation zuerst die komplette Nachricht empfangen und zwischenspeichern, bis sie komplett vorhanden ist, bevor sie sie an die nächste Station weiterleitet; außerdem muss bei Übertragungsfehlern immer gleich die ganze Nachricht wiederholt übertragen werden. Wird sie aber in kleine Pakete zerlegt, dann sind viele Zwischenstationen parallel mit der Übertragung beschäftigt, und bei Übertragungsfehlern wird nur das fehlerhafte Paket wiederholt; beides führt zu viel größerem Durchsatz.

Nachrichtenvermittlung

Dem gegenüber steht der Nachteil von kleinen Paketen, dass der Umfang der Header-Information für ein Paket oder eine Nachricht etwa gleich ist (Identität von Sender und Empfänger, Paket- oder Nachrichtenidentifizierung). Damit ergibt sich bei der Paketvermittlung gegenüber der Nachrichtenvermittlung ein höherer Header-Overhead[3] pro Datenbyte.

Paketvermittelte Netze werden in zwei Klassen unterteilt: *Datagramm-Netzwerke* und *VC-Netzwerke* (*Virtual Channels*, virtuelle Kanäle). Sie unterscheiden sich dadurch, dass sie Pakete entweder anhand von Hostzieladressen oder anhand von virtuellen Kanalnummern weiterleiten. Das Internet ist ein Datagramm-Netzwerk, da das IP die Pakete anhand von Zieladressen weiterleitet. X.25 [16], Frame-Relay [19] und ATM (Asynchronous Transfer Mode) sind Beispiele für Paketvermittlungstechnologien, die virtuelle Kanäle verwenden.

Datagramm-Netzwerke
VC-Netzwerke

In einem Datagramm-Netzwerk startet ein Paket in einem Sender-Host, anschließend fließt es durch eine Reihe von Routern und endet in dem Empfänger-Host. Ein Router kann erst dann mit dem Versenden eines Pakets beginnen, wenn er das ganze Paket empfangen hat; dieses Verfahren nennt man *Speichervermittlung* (*Store-and-Forward*). Auf dem Weg von einem Router zum nächsten unterliegt ein Paket verschiedenen Arten von *Verzögerungen*:

Store-and-Forward

- *Verarbeitungsverzögerung*: Die Zeit, die ein Router für die Feststellung benötigt, wohin das Paket weiterzuleiten ist, ggfs. auch die Zeit, um Bitfehler zu entdecken, die durch Störungen auf der Leitung bei der Übertragung auftreten können.

Verarbeitung

[3] Der Begriff Overhead steht für Verwaltungsmehraufwand.

Warteschlange

- *Warteschlangenverzögerung*: Die Zeit, die das Paket in einer Warteschlange eines Routers verbringt, um auf die Übertragung auf einer ausgehenden Verbindungsleitung zu warten.

Übertragung

- *Übertragungsverzögerung*: Die Zeit, die für die Einstellung aller Paketbits in das Übertragungsmedium erforderlich ist, d. h. Größe des Pakets in Bit geteilt durch die Übertragungsrate der Kommunikationsschnittstelle zu dem physischen Medium in bps.

Ausbreitung

- *Ausbreitungsverzögerung*: Die Zeit, die ein Bit braucht, um sich von Router A bis zum nächsten Router B auszubreiten, nachdem es auf die Verbindungsleitung befördert wurde. Die Ausbreitungsverzögerung von A nach B ist die Entfernung zwischen A und B geteilt durch die Ausbreitungsgeschwindigkeit, die vom physischen Medium der Verbindungsleitung abhängt.

Gesamtverzögerung

Zusammen bilden diese Verzögerungen die *Gesamtverzögerung des Routers* für die Übertragung eines Pakets zum nächsten Router.

Die Verzögerung auf einem Pfad errechnet sich als die Summe der Verzögerungen in allen Routern auf dem Pfad. Die Summe aus Verarbeitungs-, Warteschlangen-, Übertragungs- und Ausbreitungsverzögerungen eines Pakets auf dem Pfad von einem Sender zu einem Empfänger bezeichnet man als *Ende-zu-Ende-Verzögerung* des Pakets.

Ende-zu-Ende-Verzögerung

Bei Überlastung eines Netzes kann es vorkommen, dass für ein Paket kein Speicherplatz in der Warteschlange eines Routers vorhanden ist. Dann verwirft der Router das Paket. Das Paket geht damit verloren. Der Anteil verlorener Pakete erhöht sich mit zunehmender Verkehrsintensität. Aus diesem Grund wird die Leistung an einem Router oft nicht in Bezug auf die Verzögerung, sondern in Bezug auf die Wahrscheinlichkeit von *Paketverlusten* gemessen.

Paketverlust

Übungsaufgabe 3.2 Man betrachte eine Nachricht, die 7,5 Mbit groß ist. Angenommen, zwischen Quelle und Ziel befinden sich zwei Router und drei Verbindungsleitungen und jede Verbindungsleitung hat eine Übertragungsrate von 1,5 Mbps. Weiter ignorieren wir die Verarbeitungs-, Warteschlangen- und Ausbreitungsverzögerung. Welche Ende-zu-Ende-Verzögerung ergibt sich,

1. wenn die Nachricht mit der Nachrichtenvermittlung befördert wird, also durch Versenden der 7,5 Mbit langen Nachricht im Ganzen?

2. wenn die Nachricht in kleine Pakete mit einer Länge von je 1,5 Kbit übertragen wird?

`https://e.feu.de/1801-ende-zu-ende-verzoegerung`

3.2.3 Das Internet-Schichtenmodell

Der Internetprotokollstapel besteht aus fünf Schichten,[4] siehe Abbildung 3.5. Die PDUs auf jeder Schicht haben einen eigenen Namen.

	Schicht	Layer	PDU	Protokolle (z. B.)	Kommunikationspartner
5	Anwendung	Application	Nachricht	HTTP, SMTP, IMAP, FTP	Anwendungen
4	Transport	Transport	Segment	TCP, UDP	Prozesse
3	Vermittlung	Network	Datagramm	IP	Hosts
2	Sicherung	Data Link	Rahmen	Ethernet, PPP	Nachbarknoten
1	Bitübertragung	Physical	1-PDU		

Abbildung 3.5: Der Internet-Protokollstapel, PDUs, Protokolle und Kommunikationspartner.

Anwendungsschicht

Die *Anwendungsschicht* ist zuständig für die Unterstützung von Netzwerkanwendungen. Sie beinhaltet viele Protokolle, darunter *HTTP* [11, 15] für das Web, *SMTP* [31] für E-Mail und *FTP* [30] für Filetransfer. In der Internet-Terminologie wird sie auch als *Verarbeitungsschicht* bezeichnet.

Transportschicht

Die *Transportschicht* stellt einen Kommunikationsdienst zwischen zwei Anwendungsprozessen bereit und ermöglicht so eine *Endpunkt-zu-Endpunkt*-Kommunikation von Quell-Host zu Ziel-Host. Es gibt zwei Übertragungsprotokolle: *TCP* und *UDP*. TCP bietet einen zuverlässigen, verbindungsorientierten Dienst, über das ein Bytestrom von einem Rechner im Internet fehlerfrei einem anderen Rechner zugestellt wird. UDP bietet einen unzuverlässigen, verbindungslosen Dienst.

Vermittlungsschicht

IP

IP-Datagramm

Routing-Protokolle

IP-Schicht

Auch die *Vermittlungsschicht* bietet einen Endpunkt-zu-Endpunkt-Kommunikationsdienst. Sie transportiert Datagramme vom Quellhost zum Zielhost. Die Vermittlungsschicht des Internets besteht aus zwei Hauptkomponenten: dem IP und vielen Routing-Protokollen. Das *IP* definiert das Format eines *IP-Datagramms*. Die *Routing-Protokolle* bestimmen, welche Routen die Datagramme zwischen Quelle und Ziel nehmen. Innerhalb eines Netzwerks kann der Netzwerkadministrator jedes beliebige Routing-Protokoll benutzen. Die Vermittlungsschicht, die das IP und viele Routing-Protokolle enthält, wird auch als *IP-Schicht* oder *Internet-Schicht* bezeichnet.

Sicherungsschicht

Die *Sicherungsschicht* ist für die Beförderung ganzer Rahmen von einem Knoten (Host oder Paket-Switch) zu einem *benachbarten* Knoten zuständig, die physikalisch über einen Übertragungskanal, z. B. ein Kabel, eine Telefonleitung oder eine Punkt-zu-Punkt-Funkverbindung, miteinander verbunden sind. Der auf der Sicherungsschicht bereitgestellte Dienst hängt von dem spezifischen Sicherungsschichtprotokoll ab. Ethernet [37] und PPP (Point-to-Point Protocol)

[4] Der Internetprotokollstapel, der in RFC 1122 [12] definiert ist, hat nur vier Schichten. In der Literatur wird aber oft das hier gezeigte 5-Schichten-Hybridmodell benutzt. Dieses Modell heißt Hybridmodell, da es eine Mischung aus Internet-Stack und ISO-OSI ist, siehe Abschnitt 3.2.5.

[36, 35] und in gewissem Umfang auch ATM und Frame-Relay sind Beispiele für die Sicherungsschicht.

Bitübertragungsschicht

Demgegenüber überträgt die *Bitübertragungsschicht* die einzelnen Bits der 1-PDU von einem Router zum nächsten. Die Protokolle dieser Schicht hängen wesentlich von der Verbindungsleitung und vom Übertragungsmedium der Verbindungsleitung ab. Ethernet benutzt z. B. viele Protokolle für die Bitübertragungsschicht: eines für Twisted-Pair Kabel, eines für Koaxialkabel, eines für Glasfaser usw. Je nach Medium wird ein Bit unterschiedlich auf der Verbindungsleitung übertragen.

Netzwerkschnittstellenschicht

Der Begriff *Netzwerkschnittstellenschicht* (*Network Interface Layer*) fasst die Bitübertagungsschicht und die Sicherungsschicht zusammen. Diese Schicht wird auch als *Host-an-Netz-Schicht* (*Host-to-Net Layer*) bezeichnet.

3.2.4 Netzwerkeinheiten und Schichten im Zusammenspiel am Beispiel des Internets

Endsysteme und Transitsysteme sind die wichtigsten Netzwerkeinheiten. Ein Beispiel für eine Art von Transitsystem im Internet sind die Router. Außerdem finden sich im Internet noch Switches oder Bridges. Beide Typen von Geräten sind Arten von *Paket-Switches*. *Router und Switches* (Bridges) unterscheiden sich von Hosts dadurch, dass sie nicht alle Schichten des Protokollstapels implementieren: Switches und Bridges implementieren nur die Schichten 1 und 2, während Router die Schichten 1 bis 3 implementieren. Dagegen implementieren Hosts alle fünf Schichten. Die reduzierte Komplexität von Routern und Switches (Bridges) hängt mit dem Bestreben zusammen, einen großen Teil der Komplexität der Internet-Architektur auf die Ränder des Netzwerks zu legen. Das Weiterleiten einer Nachricht von einem Endsystem zum anderen über eine Bridge und einen Router wird in Abbildung 3.6 illustriert.

Paket-Switches

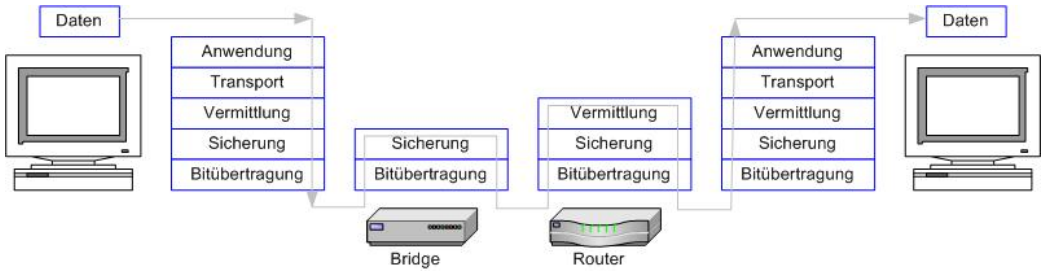

Abbildung 3.6: Informationsfluss zwischen zwei Endsystemen über Router und Bridge im Internet.

3.2.5 Das ISO/OSI-Referenzmodell

Von historischer Bedeutung ist das ISO/OSI-Referenzmodell [4, 5, 6]. Anfang der 80er Jahre entstand die internationale Norm ISO/IS 7498 mit dem Titel „Information Processing Systems: Open Systems Interconnection – Basic Reference Model“ (ISO-RM). Hier wurde ein Rahmen geschaffen, um die Kom-

munikation zwischen Computersystemen zu standardisieren. Das ISO/OSI-Referenzmodell definiert sieben Schichten. Abbildung 3.7 zeigt einen möglichen Informationsfluss und Protokollstapel mit Dienstzugangspunkten anhand dieses Modells.

Das Referenzmodell schreibt keine Implementierung dieser Schichten vor, sondern beschreibt zu jeder Schicht ihre allgemeinen Aufgaben, ihre der nächsthöheren Schicht bereitzustellenden Dienste und die in ihr zu realisierenden Funktionen. Das Modell bildet damit ein *Rahmenwerk zur Implementierung von Protokollen.*

Rahmenwerk für Protokolle

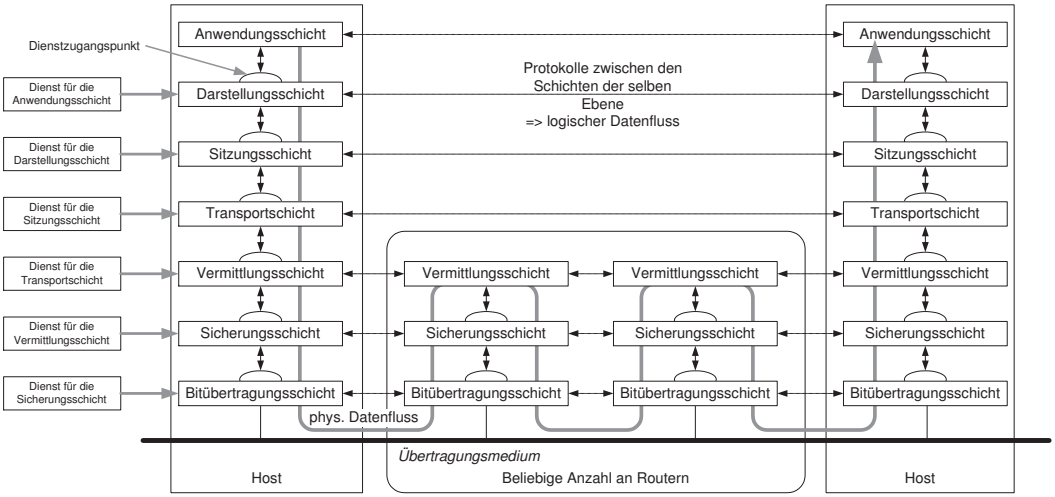

Abbildung 3.7: Möglicher Informationsfluss zwischen zwei Endsystemen über zwei Transitsysteme gemäß dem ISO/OSI-Referenzmodell

Die sieben Schichten lassen sich wie folgt charakterisieren:

7-Schichten-Modell

- Schicht 1: Bitübertragungsschicht (physical layer): stellt physikalische Übertragungskanäle zur Verfügung, die es gestatten, beliebige Bitfolgen zu übertragen.

- Schicht 2: Sicherungsschicht (data link layer): stellt weitgehend sichere Übertragungskanäle für die Übertragung von Datenblöcken zur Verfügung. Hierzu werden Verfahren zur Fehlererkennung und Fehlerkorrektur eingesetzt.

- Schicht 3: Vermittlungsschicht (network layer): stellt logische Übertragungskanäle zwischen Endsystemen zur Verfügung. Ein Beispiel ist das IP im Internet.

- Schicht 4: Transportschicht (transport layer): stellt logische Übertragungskanäle zwischen den auf den Endsystemen aktiven, miteinander kommunizierenden Anwendungsprozessen zur Verfügung. Beispiele sind die schon kurz angesprochenen Protokolle TCP und UDP im Internet.

- Schicht 5: Kommunikationssteuerungsschicht oder auch Sitzungsschicht (session layer): stellt Dienste zur Verfügung, die es den kommunizierenden Prozessen erlauben, ihren Dialog zu kontrollieren und zu synchronisieren.
- Schicht 6: Darstellungsschicht (presentation layer): stellt Dienste zur Verfügung, die die Darstellung von Daten in unterschiedlichen Repräsentationen (z. B. zwischen verschiedenen Zeichencodierungen) umwandeln.
- Schicht 7: Anwendungsschicht (application layer): implementiert die eigentliche Anwendungsfunktionalität.

Die Dienste der unteren drei Schichten ermöglichen den Transport von Daten zwischen Endsystemen, während die Dienste der oberen vier Schichten für eine reibungslosen Verlauf des Dialogs zwischen den Anwendungsprozessen zuständig sind.

Im Vergleich zu diesen sieben Schichten besteht das Internet nur aus fünf Schichten, siehe Abbildung 3.5 und 3.6. Die Dienste und Funktionalitäten der Schichten 5 und 6 des ISO/OSI-Modells sind bereits so anwendungsspezifisch, dass sie in der Regel von den Anwendungsprozessen implementiert werden. So findet sich die Funktionalität der Schicht 6 heutzutage am ehesten in Object Brokern.[5] In diesem Kurs folgt die Darstellung der Konzepte der einzelnen Protokollschichten deshalb dem fünfschichtigen Internet-Schichtenmodell, beginnend mit der Anwendungsschicht.

3.3 Anwendungsschicht

verteilte Anwendungen

Rechnernetze dienen zur *Realisierung verteilter Anwendungen*. Allein durch den Anwendungszweck begründet sich die Notwendigkeit von Rechnernetzen. Die Anwendungsprozesse auf den unterschiedlichen Endsystemen kommunizieren miteinander durch den Austausch von Nachrichten über das Computernetzwerk. Die Protokolle auf der Anwendungsschicht legen das Format und die Reihenfolge der ausgetauschten Nachrichten fest und definieren die aus der Übertragung oder dem Empfang einer Nachricht resultierenden Konsequenzen.

Anwendung versus Protokoll der Anwendungsschicht

Bei verteilten Anwendungen muss man die *Netzwerkanwendungen* von den *Protokollen auf der Anwendungsschicht* unterscheiden. Die gebräuchlichste Netzwerkanwendung heutzutage ist wohl das *World Wide Web* (WWW).[6] Diese Anwendung erlaubt das *Holen von Dokumenten*, die auf anderen Endsystemen liegen. Um dieses zu leisten, besteht das WWW aus mehreren Komponenten:

1. Web-Browser als Benutzerschnittstelle, z. B. Firefox, Google Chrome, Safari, Microsoft Edge, Opera, etc.,

[5] Object Broker realisieren über eine objekt-orientierte Client/Server Architektur verteilte Dienste. Die Objekt Management Group [9] ist ein Forum, das diese Technologie unterstützt. Der wichtigste Standard in diesem Zusammenhang ist Corba [3].

[6] Das World Wide Web ist nicht das Internet! Es kann auch auf einem anderen Netzwerk als dem Internet betrieben werden. Und das WWW ist natürlich nur eine Anwendung von vielen im Internet.

2. *HTML* (*HyperText Markup Language*) [2], einem Standard für Dokumentenformate, damit der Browser die Dokumente anzeigen kann,

3. Web-Servern, die die Dokumente bereithalten, z. B. Apache-, Microsoft-Netscape-Server, und

4. *HTTP* (*HyperText Transfer Protocol*, [11, 15]), einem Protokoll der Anwendungsschicht.

HTTP definiert die Nachrichtentypen und die Art der Weiterleitung von Nachrichten zwischen Web-Browsern und Web-Servern. Es ist der Grund, warum Browser und Server sich weltweit verstehen. Das Protokoll ist in den RFC 1945 und 2616 [11, 15] standardisiert und öffentlich zugänglich: Jeder Web-Browser kann von jedem Web-Server Dokumente anfordern, solange er sich an die Konventionen hält, die in dem Protokoll festgelegt sind. Umgekehrt kann jeder Web-Server Dokumentanfragen korrekt beantworten, solange er sich an die Festlegungen des Protokolls hält.[7]

3.3.1 Client/Server-Modell

Für die Implementierung der Anwendungen brauchen wir eine Architektur, die festlegt, welche Komponenten es gibt und wie sie miteinander kommunizieren. Bei den modernen Architekturen für Netzwerkanwendungen gibt es das *Client/Server-Modell* und *Peer-to-Peer-Modell*.

Client/Server und Peer-to-Peer

Im Client/Server-Modell werden Prozesse in zwei Gruppen eingeteilt. Es gibt einen *Server-Prozess*, der einen bestimmten Dienst implementiert und bereitstellt. Es gibt viele *Client-Prozesse*, die jweils einen Dienst von dem Server anfordern, indem ihm eine Anforderung gesendet und dann auf die Antwort des Servers gewartet wird, siehe Abbildung 3.8. Anders ausgedrückt: Der Client ist eine Anwendung, die auf einer Maschine des Anwenders läuft, Dienste beim Server anfordert und deren Ergebnisse vom Server zurück erhält. Zum Beispiel fordern Web-Clients, in der Regel implementiert in Web-Browsern, Dokumente beim Web-Server an und stellen diese für ihre Benutzer dar. Der Web-Server unterstützt das Abholen von Dokumenten.

Der Server kann als fortwährend ablaufender Prozess charakterisiert werden, der den folgenden Code ausführt:

[7] HTTP regelt das Anfordern und Holen von Dokumenten. Die Darstellung der Dokumente in den Web-Browsern ist in einem anderen Standard, nämlich HTML, geregelt. Das Konzept von Hypertext, dass eine Seite auf eine andere verweisen kann, wurde vom Professor für Elektrotechnik Vannevar Bush im Jahr 1945 lange vor Erfindung des Internets entwickelt. HTML ist insbesondere kein Kommunikationsprotokoll und wird deshalb hier auch nicht behandelt. Grundlage von HTML bzw. dessen Nachfolger XML [1] ist vielmehr die Dokumentenbeschreibungssprache SGML [10, 7].

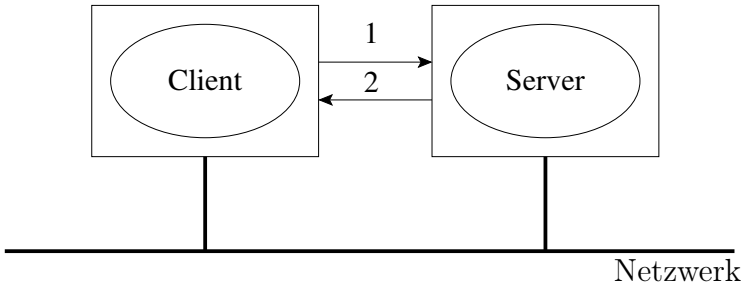

Abbildung 3.8: Client-Server Architektur.

while true **do** {
    Warte, bis Anforderung von einem Client C eintrifft;
    Führe Anforderung aus;
    Sende Antwort an Client C;
}

Der Client ist üblicherweise in die Anwendung eingebettet. Ein Web-Browser implementiert beispielsweise die Funktionalität von HTTP, die ein Client zur Kommunikation mit dem Web-Server benötigt, die sogenannte Client-Seite von HTTP. Der Web-Server implementiert analog die Server-Seite von HTTP.

Übungsaufgabe 3.3 Warum ist der Server im Client/Server-Modell ein kontinuierlich ablaufender Prozess?

`https://e.feu.de/1801-client-server`

Viele heutige populäre Anwendungen, wie z. B. Internet Telefonie, IPTV und Filesharing, verwenden die Peer-to-Peer-Architektur, bei der es keine strenge Trennung von Server und Clients gibt. Die Teilnehmer sind Peers (Gleichgestellte, Kollege) und jeder kann einen Dienst anbieten oder nutzen.

3.3.2 Protokolle der Anwendungsschicht

Dieser Abschnitt stellt exemplarisch verschiedene Protokolle der Anwendungsschicht vor, um verschiedene *Designprinzipien von Protokollen* der Anwendungsschicht zu beleuchten. Wir verwenden dazu standardisierte, öffentliche Protokolle,[8] siehe Tabelle 3.1. Konzepte und Designentscheidungen existierender Protokolle zu kennen, hilft bei der Beurteilung von Protokollen und ist nützlich für den Entwurf und die Implementierung eigener Protokolle.

HTTP

Wesentliche Prinzipen von Protokollen der Anwendungsschicht lassen sich am eingangs erwähnten HTTP erklären. Dazu müssen wir zuerst verstehen,

[8] Es gibt natürlich viele proprietäre Protokolle, die aus wirtschaftlichen Interessen nicht veröffentlicht werden.

Anwendung	Protokoll
World Wide Web	HTTP [11, 15]
E-Mail	SMTP [31], IMAP [14], POP3 [27]
File Transfer	FTP [30]
News	NNTP [21]
Remote Login	Telnet [29]
Streaming Multimedia	HTTP, RTP[34], RTMP
Internet-Telephonie	SIP [33], RTP[34], proprietär (z. B. Skype)

Tabelle 3.1: Häufig benutzte verteilte Anwendungen mit zugehörigem Protokoll der Anwendungsschicht.

dass im WWW Objekte, d. h. zum Beispiel Webseiten oder Bilder, eindeutige Bezeichner haben. Hierfür dient der *URL* (*Uniform Resource Locator*). Ein URL identifiziert eine Ressource (z. B. eine Webseite) auf eindeutige Weise. Ein URL besteht aus drei Teilen: dem Protokoll, dem DNS-Namen oder der IP-Adresse des Rechners (den eindeutigen Namen des Hosts), auf dem sich die Seite befindet, und einem lokalen eindeutigen Namen der Seite. Dazu wird das Namensschema `<protocol id>:<protocol specific address>` verwendet, siehe Tabelle 3.2.

URL

Protokoll	`<protocol id>:<protocol specific address>`
HTTP	`http://<IP adresse>/<lokaler Pfad>`
SMTP	`mailto:`E-Mail-Adresse
FTP	`ftp://<IP adresse>/<lokaler Pfad>`
NTTP	`news://<IP adresse>/<Newsgroup-Id>`
Telnet	`telnet://<IP adresse>:<Port Nr.>`
Lokale Dateien	`file:<lokaler Pfad>`

Tabelle 3.2: Unterstützte Protokolle der Anwendungsschicht im Internet. URLs in diesen Formaten können in den meisten Web-Browsern direkt in das Adressfeld eingegeben werden.

Übungsaufgabe 3.4 Um eine Webseite über das Internet zu holen, muss sie identifiziert werden. Dafür benötigt man die folgenden drei Informationen:

1. Wie heißt die Webseite?
2. Wo befindet sie sich?
3. Wie wird auf sie zugegriffen?

Sind diese Informationen in einem URL der Form

`http://<IP adresse>/<lokaler Pfad>`

für einen Benutzer vorhanden?

https://e.feu.de/1801-url

3.3.2.1 HTTP

Zunächst betrachten wir nun die Nachrichtenformate von HTTP. HTTP definiert lediglich zwei Nachrichtentypen, eine HTTP-Anfragenachricht und eine HTTP-Antwortnachricht. Hier ist eine typische Anfragenachricht:

Anfrageformat für HTTP

```
GET /ks/1801/ HTTP/1.1
Host: www.fernuni-hagen.de
Connection: close
User-agent: Mozilla/4.0
(extra carriage return, line feed)
```

Die Anfragenachricht ist ein ASCII-Text, der von Menschen gelesen werden kann. Er genügt einem bestimmten Format. Die erste Zeile mit dem `GET`-Befehl heißt *Anfragezeile* und hat immer die drei Parameter: Methode (hier GET), Dateiname (hier: `/ks/1801/`) und HTTP-Version (hier: HTTP/1.1). Die GET-Methode wird benutzt, um eine Datei anzufordern, die durch `/ks/1801/` identifiziert wird. Die drei weiteren Zeilen in der Anfragenachricht sind die *Header-Zeilen*. Die Header-Zeile `Host` gibt den Web-Server `www.fernuni-hagen.de` an, auf dem die Datei `/ks/1801/` liegt. Alle Zeilen sind durch Carriage Return und Line Feed (Zeilenende) getrennt. Am Ende kommt ein zusätzliches Carriage Return und Line Feed, um das Ende der Nachricht zu markieren.

Zum Verständnis von Anfragenachrichten ist noch eine Präzisierung bezüglich der Funktionsweise des Webs notwendig: Web-Seiten werden üblicherweise als Dokumente bezeichnet. Diese Seiten bestehen aus Objekten. Ein Objekt ist einfach eine Datei, z. B. eine HTML- oder GIF-Datei, die mit einem eigenen URL zugänglich ist. Viele Web-Seiten bestehen aus einer Basisdatei, die Referenzen auf weitere Objekte, z. B. Bilder, enthält. Um eine Web-Seite aufzubauen, muss ein Browser also gegebenenfalls mehr als ein Objekt von Servern anfordern. Für jedes dieser Objekte muss eine eigene HTTP-Anfragenachricht erzeugt und abgeschickt werden.

Objekte und Objektreferenzen im WWW

In obigem Beispiel fordert der Client das Objekt `1801` im Verzeichnis `/ks/` vom Web-Server `www.fernuni-hagen.de` an. Aus der Zeile, die die Benutzer zur eindeutigen Identifizierung von Web-Seiten angeben, werden für eine HTTP-Anfragenachricht zwei Parameter gewonnen: die Eingabe `www.fernuni-hagen.de/ks/1801/` liefert für die Anfragezeile das Dokument `/ks/1801/` und für die erste Header-Zeile den Hostnamen `www.fernuni-hagen.de`.

Übungsaufgabe 3.5 Warum taucht im URL

`http://www.fernuni-hagen.de/ks/1801/`

bei der Identifizierung der Webseite `1801` im Verzeichnis `/ks/` vom Web-Server `www.fernuni-hagen.de` nicht die IP-Adresse auf, sondern der Name `www.fernuni-hagen.de`?

`https://e.feu.de/1801-domain-name`

Der dritte Parameter HTTP/1.1 der Anfragezeile gibt an, mit welcher Version von HTTP der Webbrowser arbeitet[9]. Wie Tabelle 3.1 bereits vermuten ließ, gibt es offenbar zwei Versionen von HTTP, HTTP-Version 1.0 [11] und HTTP-Version 1.1 [15].

Beide Versionen von HTTP benutzen das Protokoll TCP der Transportschicht, um Nachrichten auszutauschen. TCP stellt der Anwendung eine *TCP-Verbindung* zum Partnerprozess zur Verfügung. TCP garantiert für seine TCP-Verbindungen, dass Nachrichten zuverlässig beim Empfänger ankommen, mehr dazu in Abschnitt 3.4. HTTP-Version 1.0 arbeitet nur mit nicht persistenten Verbindungen. *Nicht-persistente Verbindungen* werden nach der Lieferung des angefragten Objektes sofort wieder abgebaut: Mit der Anfragenachricht fordert der Client ein bestimmtes Objekt an. Sobald der Server das angefragte Objekt in einer HTTP-Antwortnachricht verpackt, also praktisch in den Briefumschlag gesteckt und abgeschickt hat, befiehlt er dem TCP, die Verbindung nach Ablieferung der Antwortnachricht zu schließen.

nicht-persistente Verbindung

HTTP-Version 1.1 unterstützt auch persistente Verbindungen.[10] Bei *persistenten Verbindungen* überlässt der Server dem Client die Entscheidung, wann die Verbindung wieder abgebaut werden soll. Bei komplexen Web-Seiten kann der Client also eine Verbindung dazu benutzen, mehrere Objekte nacheinander von demselben Server zu laden, ohne für jedes Objekt eine neue Verbindung aufbauen zu müssen. Der HTTP-Server schließt die Verbindung nur, wenn sie für eine bestimmte Zeitdauer nicht benutzt wird. Diese Zeit ist konfigurierbar und heißt *Timeout-Intervall*.

persistente Verbindung

Timeout-Intervall

Persistente Verbindungen werden zudem entweder ohne oder mit Pipelining verwendet. Bei *persistenten Verbindungen ohne Pipelining* wartet der Client auf die Antwort einer Anfrage, bevor er die nächste Anfrage an den Server schickt. Deshalb hat der Server für diesen Client keine Aufträge zu erfüllen, während das unter Umständen große Antwortobjekt sich auf dem Weg vom Server zum Client befindet (vgl. Verzögerungen in Abschnitt 3.2.2). Um Server und Netzwerk besser auszulasten und auch um dem Benutzer eines Web-Browsers schnelleren Zugriff auf schon geladene Teile des Dokuments zu geben,

persistente Verbindungen ohne Pipelining

persistente Verbindungen mit Pipelining

[9] HTTP-Version 1.1 ist abwärtskompatibel mit der Version 1.0, d. h. ein Web-Server, der HTTP-Version 1.1 ausführt, versteht auch HTTP-Anforderungen, die gemäß der Version 1.0 formuliert sind. Umgekehrt passt sich ein Web-Browser, der Version 1.1 benutzt, Web-Servern an, die Version 1.0 betreiben.

[10] Diese sind sogar der Standard.

kann der Client, sobald er auf eine Referenz eines Objekt stößt, Anfragen an einen Server senden, ohne auf vorherige Antworten zu warten. Diese Art der Nutzung der Verbindung nennt man *persistente Verbindungen mit Pipelining*.

In obiger Anfragenachricht wünscht der Client durch die Angabe in der dritten Zeile `connection: close` ausdrücklich eine nicht persistente Verbindung, obwohl HTTP-Version 1.1 persistente Verbindungen unterstützt. Die vierte Zeile gibt den Browser-Typ an, in diesem Fall das Mozilla/4.0, ein Netscape-Browser. Dieses kann nützlich sein, da manche Server unterschiedliche Versionen eines Objektes für unterschiedliche Browser bereithalten. Alle diese Versionen werden aber nur mit einem URL bezeichnet. Wird dem Server kein Browsertyp mitgeteilt, liefert er die auf ihm konfigurierte Default-Version des angefragten Objektes.

Eine mögliche Antwort auf obige Anfrage könnte wie folgt aussehen:

Antwortformat für HTTP

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 08 Aug 2004 12:00:15 GMT
Server: Apache/2.0.35 (Unix)
Last-Modified: Mon, 24 Jun 2004 09:23:24 GMT
Content-Length: 483
Content-Type: text/html
...
```

Die erste Zeile, die *Statuszeile*, umfasst immer drei Felder: Version (hier HTTP-Version 1.1), Statuscode (`200` bedeutet, dass alles in Ordnung ist), und Phrase (hier `OK`: die Anfrage wurde gemäß den Anforderungen abgearbeitet). Danach folgen wieder *Header-Zeilen*, die dem Web-Browser Details über die Lieferung geben: Zeile 3 enthält das Lieferdatum, d. h. wann die Antwortnachricht generiert wurde, Zeile 4 den Server und die Zeilen 5 bis 7 beschreiben das gelieferte Objekt selbst. Schließlich folgen im sogenannten Entity Body die eigentlichen Daten.[11]

Web-Caches

Ergänzend möchten wir noch auf eine Besonderheit des World Wide Web eingehen, nämlich die *Web-Caches*. Das sind Web-Server, die von (lokalen) Organisationen eingesetzt werden, um Kopien der zuletzt angeforderten Objekte zu speichern. Die Web-Browser der Benutzer werden so konfiguriert, dass sie zunächst beim Web-Cache nach Objekten anfragen. Verfügt der Web-Cache

[11] Sie können sich eine solche Antwortnachricht auf den Bildschirm holen, indem Sie HTTP über Telnet benutzen, z. B.: Tippen Sie `telnet://www.fernuni-hagen.de:80` in das Web-Browser-Adressfeld oder öffnen Sie ein Kommandozeilenfenster (Shell oder Eingabeaufforderung in einem Terminalfenster) und geben Sie `telnet www.fernuni-hagen.de 80` (Return) ein. Auf die Systemantwort `Escape character...` geben Sie
`GET /ks/1801/ HTTP/1.0` (Return) (Return) ein.

über das Objekt, so liefert er es direkt zurück. Andernfalls fungiert der Web-Cache als Web-Client und fordert das Objekt seinerseits vom dem Originalhost[12] an, hinterlegt eine Kopie und liefert dem Web-Browser das gewünschte Objekt. Web-Caches können also gleichzeitig als Server und Client operieren. Sie werden durch sogenannte *Proxy-Server* implementiert.[13] Die drei häufigsten Gründe für die Einführung von Web-Caches sind:

Proxy-Server

1. Die Reaktionszeit auf eine Client-Anfrage kann reduziert werden. Dieser Effekt wird noch verstärkt, wenn eine Hochgeschwindigkeitsverbindung zwischen Web-Cache und Hosts besteht, was in vielen Organisationen der Fall ist.

2. Der Verkehr von einer Institution in das öffentliche Internet kann reduziert werden, so dass die Institution mit weniger Bandbreite an das öffentliche Netz angeschlossen sein muss.

3. Ein Host soll nicht im Internet als Client eines Web-Servers erkennbar sein. Er benutzt einen vertrauenswürdigen Proxy-Server, so dass der Web-Server nicht erfährt, von welchem Host aus die Anfrage wirklich kommt, der Client bleibt also dem Server gegenüber anonym.

Übungsaufgabe 3.6 Warum bleibt ein Client, der über einen Proxy-Server einen Webserver kontaktiert, dem Webserver gegenüber anonym?

`https://e.feu.de/1801-anonymitaet`

3.3.2.2 FTP

Weitere Konzepte von Protokollen der Anwendungsschicht lassen sich an *FTP*, dem *File Transfer Protocol* [30] erklären. Während HTTP alle Informationen, die Client und Server zur Kommunikation benötigen, über nur eine TCP-Verbindung sendet, benutzt FTP zeitweise zwei Verbindungen für die Kommunikation zwischen Client und Server. Mit FTP können Benutzer Dateien von oder zu einem entfernten Host übertragen. Dazu muss der Benutzer sich dem entfernten Host bekannt machen und Zugang erhalten. Hierzu wird eine TCP-Verbindung als sogenannte *Steuerverbindung* aufgebaut. Die Steuerverbindung bleibt während der gesamten FTP-Sitzung des Benutzers erhalten. Über diese Verbindung kann der Benutzer durch Verzeichnisbäume navigieren und den Transfer von Dateien anstoßen. Wird der Transfer einer Datei angestoßen, so wird die Datei aber nicht über die bisherige Verbindung übertragen,

FTP

[12] Beim kooperativen Caching kann es sogar vorkommen, dass der Web-Cache sich noch an einen anderen, günstiger gelegenen Web-Cache wendet.

[13] Durch seine Vermittlerposition kann der Proxy-Server selbst die Anfragen und Antworten verändern, oder eigene Funktionalität hinzufügen. Im Falle des Web-Caches speichert der Proxy nicht nur oft benutzte Seiten, um sie direkt an den Anfordenden zu schicken, sondern kann beispielsweise auch übersetzen (vgl. den Übersetzungsdienst von Google, *www.google.de*).

sondern über eine zweite TCP-Verbindung, die *Datenverbindung*. Sie wird nur für den Transfer dieser einen Datei eingerichtet und danach wieder abgebaut.

Out-of-Band versus In-Band

Man sagt, FTP sendet seine Steuerinformation *out-of-Band*. HTTP ist ein Beispiel für ein Protokoll, das seine Steuerinformationen *in-Band* sendet. Bei HTTP stehen alle Steuerinformationen in den Headerfeldern der Anfrage- und Antwortnachrichten zusammen mit den nachfolgenden Daten.

Protokoll mit Zustand versus zustandsloses Protokoll

Ein FTP-Server muss zudem über den Zustand aller seiner Benutzersitzungen Buch führen. Insbesondere muss er für jeden Benutzer das aktuelle Verzeichnis verfolgen, während die Benutzer durch die entfernten Verzeichnisbäume navigieren, und die Benutzer den richtigen Steuerverbindungen zuordnen. Die Gesamtzahl der Sitzungen, die ein FTP-Server gleichzeitig unterstützen kann, ist hierdurch eingeschränkt. Web-Server hingegen speichern keine Informationen über die Clients, die Dienste angefordert haben. HTTP ist deshalb ein sogenanntes *zustandsloses Protokoll*.

3.3.2.3 E-Mail

E-Mail SMTP

Das *E-Mail*-System des Internets basierte ursprünglich nur auf *SMTP*, dem *Simple Mail Transfer Protocol* [31], das schon seit 1982 existiert und wesentlich älter ist als z. B. HTTP. Dieses SMTP ist ein sehr einfach gehaltenes Protokoll zum Versenden von E-Mails zwischen sogenannten Mail-Servern mittels TCP, das nur besonderen Wert auf Zuverlässigkeit bei Ausfall von Leitungen oder Hosts legt, an Authentifizierung und ähnlichen Luxus wurde damals jedoch nicht gedacht.[14]

MTA

Auf allen Mail-Servern, die heute üblicherweise von ISPs betrieben werden, läuft ein besonderer Prozess, der *Message Transfer Agent* (*MTA*), der E-Mails annimmt (oder die Annahme verweigert) und entweder in lokalen Mailboxen für seine Benutzer abspeichert oder per SMTP an andere Mail-Server weiterleitet.

POP3 IMAP MUA

Um es den Endbenutzern möglichst bequem zu machen, sind heute weitere Protokolle im Einsatz. Zum Abholen der E-Mails vom Mail-Server auf den eigenen Rechner wird *POP3* (*Post Office Protocol Version 3* [27]) benutzt, zum komfortablen Verwalten von Mailboxen, die auf dem Server bleiben, dient *IMAP* (*Internet Message Access Protocol* [14]). Der Benutzer hat auf seinem Rechner ein meist sehr mächtiges Programm, den *Mail User Agent* (*MUA*, z. B. Mozilla Thunderbird oder Microsoft Outlook), der einmal auf die verwendeten Protokolle und Benutzerdaten eingestellt werden muss. Zum Versenden der E-Mails wird auch vom MUA weiter SMTP benutzt, seit einiger Zeit aber mit Authentifizierung.

[14] So kann mit SMTP jeder Teilnehmer über jeden Mail-Server beliebig viele E-Mails mit beliebigen Absenderadressen verschicken, und dem Problem von SPAM (unerwünschte Massen-E-Mail) ist damit der Boden bereitet. Zwar wurden im Laufe der Zeit verschiedene Kontrollmechanismen eingebaut, aber das SPAM-Problem ist bis heute noch lange nicht gelöst, weil die SPAM-Versender eigene, unkontrollierte Server und sogar Internet-Domänen betreiben.

Übungsaufgabe 3.7 Recherchieren Sie, welche Vorteile das Protokoll IMAP gegenüber dem Protokoll POP3 für das Abholen von E-Mails hat.

`https://e.feu.de/1801-imap-versus-pop3`

HTTP

Auch über *HTTP*, also mittels Web-Browser, kann man bei vielen ISPs seine E-Mails lesen und schreiben, z. B. im Internet-Café, und ist damit völlig unabhängig von einem eigenen Host oder MUA.

Eine starke Einschränkung von SMTP ist, dass die komplette Nachricht nur aus 7-Bit-ASCII-Zeichen bestehen soll, dass also noch nicht einmal deutsche Umlaute und andere Sonderzeichen, geschweige denn Schriftzeichen aus anderen Zeichensätzen erlaubt sind. Hierfür und für das Versenden von beliebigen Dateien als Mailanhang wurde ein Kodierungsstandard eingeführt, an den sich alle MUAs halten sollen, die *Multipurpose Internet Mail Extensions* (*MIME*, siehe [17, 18]). Dadurch wird sichergestellt, dass zwischen SMTP-Servern nur 7-Bit-Nachrichten ausgetauscht werden, der Benutzer aber in seinem MUA die Nachrichten mit allen verwendeten Schriftzeichen und Anhängen betrachten kann.

MIME

Bemerkenswert bei E-Mail ist, dass Mail-Server in der Regel gleichzeitig als Client und als Server agieren: sie senden und empfangen E-Mail für ihre Benutzer. Im Gegensatz zu HTTP ist SMTP ein *Push-Protokoll*, d. h. der sendende Mail-Server *schiebt* die E-Mail unaufgefordert auf den empfangenden Mail-Server. Ein Web-Server dagegen sendet niemals unaufgefordert Daten. Bei HTTP *zieht* die Anfragenachricht die Information bei Bedarf vom Web-Server; HTTP ist ein *Pull-Protokoll*.

Push-Protokoll versus Pull-Protokoll

3.3.3 Schnittstelle zur Transportschicht

Im vorherigen Abschnitt wurden bereits TCP-Verbindungen angesprochen. TCP ist ein Protokoll der Internet-Transportschicht, mit dem Nachrichten zwischen Anwendungsprozessen ausgetauscht werden können. Wie wird dieser Dienst in Anspruch genommen? Hierzu existiert eine Schnittstelle zur Transportschicht, die auch als *API (Application Programming Interface)* zwischen der Anwendung und der Transportschicht bezeichnet wird. Diese Schnittstelle bietet als Dienstzugangspunkt zu dem Dienst einen sogenannten *Socket*[15]. Web-Clients und -Server kommunizieren also, indem sie ihre HTTP-Nachrichten an einen Socket übergeben. Das Socket-Interface in einem Client ist die Tür zur TCP-Verbindung zu einem Server, der ebenfalls über einen Socket eine Tür zu dieser Verbindung hat.

API

Socket

Neben TCP bietet die Transportschicht im Internet noch ein zweites Protokoll: UDP. UDP wird über eine andere Art von Socket in Anspruch genommen; zu den Unterschieden mehr in Abschnitt 3.4. Für die Anwendungsschicht ist

[15] Der englische Begriff Socket steht für Steckdose oder Fassung. In unserem Fall bezeichnet Socket bildlich gesprochen den Anschluss an das Kommunikations- statt Stromnetz, siehe Abbildung 3.9. Er entspricht dem gelben Briefkasten der Post in unserem Einführungsbeispiel.

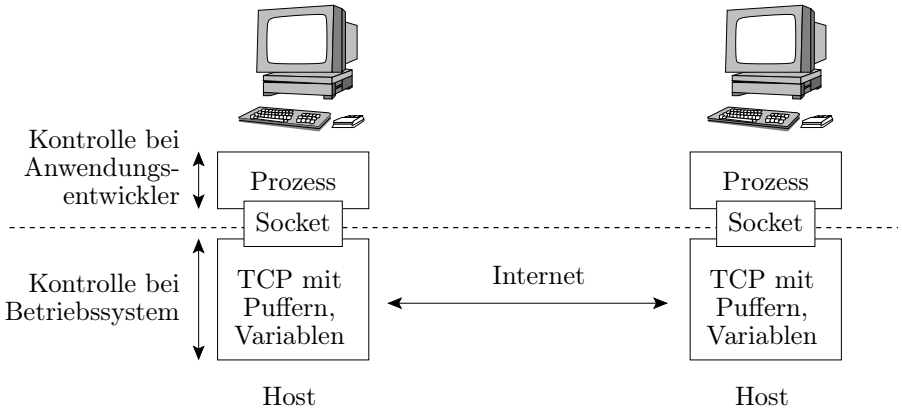

Abbildung 3.9: Anwendungsprozesse, Sockets und das zugrundeliegende Transportprotokoll.

wichtig zu bedenken: Während der Anwendungsentwickler die volle Kontrolle über die Anwendungsschichtseite eines Sockets hat, hat er kaum Kontrolle über die Transportschichtseite des Sockets, siehe Abbildung 3.9. Dem Entwickler bleibt im wesentlichen die Wahl des Transportprotokolls und die Möglichkeit, einige Parameter der Transportschicht zu setzen, zum Beispiel die maximale Puffer- und Nachrichtengröße.

Anforderungen an die Transportschicht

Welche Dienste benötigt eine Anwendung überhaupt von der Transportschicht? Die Anforderungen an einen Dienst lassen sich grob in drei Gruppen einteilen. Sie betreffen die Zuverlässigkeit (Datenverlust), Bandbreite und die Zeit. Wir werden die Punkte kurz beleuchten. Eine vollständige Klassifizierung würde den Rahmen dieses Kurses jedoch sprengen.

Zuverlässigkeit versus Verlusttoleranz

Anwendungen wie E-Mail, Filetransfer oder das Laden von Dokumenten in einen Web-Bowser tolerieren keine *Datenverluste*. Eine E-Mail sollte beispielsweise vollständig sein, wenn der Empfänger sie erhält. Im Gegensatz dazu sind Multimedia-Anwendungen, die Daten in Echtzeit übertragen, bis zu einem gewissen Grade verlusttolerant. Fehlen bei einer Live-Videoübertragung einige Bilder, so kommt es vielleicht zu einigen Ruckeleffekten, aber die Videoübertragung selbst ist immer noch nützlich für den Benutzer.

bandbreitensensitive versus elastische Anwendungen

Echtzeit Multimedia-Anwendungen benötigen jedoch in der Regel eine gewisse Bandbreite. Wird Sprache für die Internet-Telefonie in einer gewissen Rate kodiert, so muss die entsprechende *Bandbreite* auch zur Verfügung stehen. Andernfalls kann man nicht mehr von einer Echtzeitanwendung sprechen. Ein Telefongespräch ist zum Beispiel nicht mehr möglich, wenn die Interaktion der Gesprächteilnehmer zu stark gestört wird. Kann die Anwendung nicht auf platzsparendere Kodierungen für kleinere Bandbreiten ausweichen, macht gegebenenfalls die gesamte Anwendung für den Benutzer keinen Sinn mehr. E-Mail, Filetransfer und Web-Transfer sind dagegen elastische Anwendungen. Im ungünstigen Fall wird die Geduld der Benutzer strapaziert, aber die intendierte Nutzung der übertragenen Daten wird prinzipiell nicht gefährdet. Die Anwendungen profitieren natürlich von mehr Bandbreite.

zeitsensitive Anwendungen

Als dritte Gruppe der Dienstanforderungen lassen sich Anforderungen an die *Zeit*, d. h. akzeptable Ende-zu-Ende-Verzögerungen stellen. Interakti-

ve Echtzeitanwendungen wie die Internet-Telefonie erfordern Ende-zu-Ende-Verzögerungen in einer Größenordnung von ein paar hundert Millisekunden (ms) oder weniger, sonst entstünden zu lange Pausen im Gesprächsverlauf. Tabelle 3.3 fasst die Anforderungen einiger Netzwerkanwendungen an die Transportschicht zusammen.

Anwendung	Datenverlust	Bandbreite	zeitsensitiv
Filetransfer	Kein Verlust	Elastisch	nicht
E-Mail	Kein Verlust	Elastisch	nicht
Web-Transfer	Kein Verlust	Elastisch	nicht
Echtzeitaudio	Verlusttolerant	wenige Kbps – 1 Mbps	einige 100 ms
Echtzeitvideo	Verlusttolerant	10 Kbps – 5 Mbps	einige 100 ms
Konsum von gespeichertem Audio/Video	Verlusttolerant	gleich wie Echtzeitaudio/ -video	wenige Sekunden

Tabelle 3.3: Anforderungen ausgewählter Netzwerkanwendungen an die Transportschicht (basierend auf Abbildung 2.4 aus [22]).

3.3.4 Adressierung von Prozessen

Damit Prozesse auf verschiedenen Endsystemen miteinander kommunizieren können, müssen sie sich gegenseitig kennen. Im Postdienstbeispiel muss der Briefschreiber auf der Briefkommunikationsschicht der Posttransportschicht die Adresse des Adressaten mitteilen. Die Adresse besteht aus dem Namen des Adressaten und einer Beschreibung seines Postkastens. Genauso müssen Netzwerkanwendungen auf der Anwendungsschicht der Transportschicht folgendes mitteilen:

1. eine ID, welche die Identität des empfangenden Prozesses auf dem Zielhost bezeichnet, und
2. einen Namen oder die Adresse des Hostrechners.

Portnummer

Ein empfangender Prozess wird durch eine *Portnummer* eines Protokolls der Anwendungsschicht identifiziert, wenn er gerade eine Anwendung ausführt, die auch das Protokoll verwendet. Jedes Protokoll der Anwendungsschicht besitzt eine Portnummer. Sie besteht aus 16 Bit (d. h. eine Zahl von 0 bis 65535). Ein Host hat damit 65536 Ports oder mögliche ansprechbare Prozesse. Die Portnummern 0 bis 1023 sind für bekannte Anwendungsprotokolle reserviert. Der RFC 1700 [32] listet diese *bekannten Portnummern* (*well-known port numbers*) auf. HTTP benutzt zum Beispiel Portnummer 80 und FTP Portnummer 21, d. h. den Web-Serverprozess auf einem Host findet man am Port 80, den FTP-Serverprozess am Port 21. Stellt man einen eigenen Serverdienst einer verteilten Anwendung bereit, so sollte man eine Portnummer zwischen 1024 und 65535 als Zugang festlegen und in der Benutzerinformation dokumentieren. Der bereitgestellte Dienst ist dann für potentielle Nutzer auffindbar und

ansprechbar. Natürlich muss ein Sender-Prozess zur Identifizierung auch eine Portnummer besitzen, der zwischen 1024 und 65535 liegt.

IP-Adressen

Hosts werden im Internet durch sogenannte *IP-Adressen* identifiziert. IP-Adressen sind 32 Bit lang. Die 32 Bit werden als 4 Byte interpretiert, die von links nach rechts gelesen eine hierarchische Struktur repräsentieren. IP-Adressen werden dezimal als vier durch Punkte getrennte Zahlen von 0 bis 255 geschrieben, beispielsweise 132.176.71.2. Diese Notation heißt auch Punktdezimalnotation (dotted-decimal notation). IP-Adressen sind global eindeutig. Deshalb können diese Adressen nicht vom Host beliebig gewählt werden. Die 32 Bit werden in einen *Netzwerk-* und einen *Host-Teil* der IP-Adresse aufgeteilt. Eine häufige Aufteilung ist,[16] dass die ersten 24 Bit das Netzwerk bestimmen, in dem sich der Host befindet, so wie der größte Teil unserer Postadresse durch den Staat, den Wohnort, und die Strasse feststeht. Lediglich die letzten 8 Bit sind dann der Host-Teil der IP-Adresse (entsprechend der Hausnummer).

3.3.5 Domain Name System

Den meisten Lesern werden die mnemonischen Bezeichnungen für Internet-Hosts, wie z. B. *www.fernuni-hagen.de* geläufiger sein. Diese alphanumerischen Hostnamen variabler Länge sind von Routern schwer zu verarbeiten. Router benutzen deshalb ausschließlich die IP-Adressen zum Weiterleiten von Nachrichten. Das *Domain Name System* (DNS) des Internets übernimmt die Aufgabe, die mnemonische Hostnamen auf IP-Adressen abzubilden. Protokolle der Anwendungsschicht wie HTTP, SMTP und FTP benutzen den DNS-Dienst, um zu einem vom Benutzer eingegebenen Hostnamen die IP-Adresse zu ermitteln. Dabei ist DNS selbst ein Protokoll der Anwendungsschicht.

Domain Name System

Übungsaufgabe 3.8 Mit dem Kommando `nslookup` oder `dig` können Sie die IP-Adresse eines Hosts finden. Versuchen Sie die IP-Adressen der folgenden Server zu finden.

- `www.fernuni-hagen.de`
- `www.uni-hagen.de`
- `mailstore.fernuni-hagen.de`

`https://e.feu.de/1801-ip-lookup`

Hierarchie lokale Name-Server

Das DNS realisiert eine verteilte Datenbank, die als eine *Hierarchie* von Name-Servern implementiert ist. Auf der untersten Stufe stehen die *lokalen Name-Server*, die vom ISP betrieben und den Hosts im lokalen Netz jeweils beim Internet-Zugang zugewiesenen werden. Lokale Name-Server versuchen, DNS-Anfragen für den anfragenden Host zu erledigen und die endgültige Antwort an ihn zurück zu liefern. Jede Institution, die eine Second-Level-Domain

[16] Kurseinheit 4 wird auf andere Formate von IP-Adressen noch detaillierter eingehen.

wie `fernuni-hagen.de` betreibt, stellt auch mindestens einen zuständigen autoritativen Name-Server zur Verfügung, der alle Anfragen zu dieser Domain beantworten kann. Viele Name-Server fungieren gleichzeitig als lokale und autoritative Name-Server. Darüber stehen die Top-Level-Name-Server, die für Domains wie `.de` zuständig sind und insbesondere den jeweils richtigen autoritativen Name-Server einer Institution kennen. Ganz oben in der Hierarchie des Internets gibt es etwa ein Dutzend *Root-Name-Server*, die mindestens alle Top-Level-Name-Server kennen. Alle Name-Server arbeiten mit einem *Cache*, der die Anfragen und Antworten der letzten Zeit speichert, so dass die darüber liegenden Server entlastet werden. So wird der lokale Name-Server fast nie die Root-Name-Server konsultieren, und auch die Top-Level-Server nur dann, wenn nach einer bisher nicht bekannten Subdomain gefragt wird.

autoritativer Name-Server
Top-Level-Name-Server
Root-Name-Server
Cache

Der lokale Name-Server arbeitet nach dem Prinzip der *rekursiven Namens-Auflösung*: er übernimmt die Anfrage eines Hosts, erledigt alles Notwendige zur Beantwortung und liefert die endgültige Antwort zurück, falls möglich. Die Arbeitsweise der Name-Server auf den höheren Stufen nennen wir dagegen *iterative Namens-Auflösung*: Sie antworten nur mit dem Verweis auf einen anderen (niedrigeren) Server, der die Frage weiterbearbeiten kann. An diesen ist dann dieselbe Anfrage noch einmal zu richten. Das iterative Prinzip ist für die viel beschäftigten Name-Server vorteilhaft, weil sie so ohne jede Verzögerung die Anfragen endgültig erledigen können.

rekursive Namens-Auflösung
iterative Namens-Auflösung

Abbildung 3.10 zeigt ein Beispiel für eine DNS-Anfrage: Host D benötigt die IP-Adresse von Host E (Schritt 1). Der lokale Name-Server L kann die Anfrage nicht sofort selbst beantworten, fungiert deshalb als Client und wendet sich an den Root-Name-Server R, der auf den zuständigen Top-Level-Name-Server T verweist. Dieser wiederum kennt den richtigen autoritativen Name-Server A. Also schickt L seine Anfrage ein letztes Mal los, dieses Mal an A, der endlich die gewünschte IP-Nummer von E liefert. Am Ende erhält D von L die gewünschte IP-Adresse von E (Schritt 8).

3.3.6 Beispiel einer Client/Server-Anwendung

Die bisher betrachteten Konzepte möchten wir in der Praxis an einem Beispiel einer Client/Server-Anwendung demonstrieren. Das Beispiel zeigt, wie ein Client- und ein Server-Prozess über TCP-Sockets miteinander kommunizieren. Wir verwenden hier Java, weil Java sich als Programmiersprache für die Entwicklung von Netzwerkanwendungen im Internet etabliert hat. Sie müssen Java nicht kennen, um das Programm zu verstehen. Betrachten Sie die Zeilen als Pseudo-Code. Selbstverständlich kann man das Beispiel in jeder anderen Sprache programmieren, in der ein API zu einer Transportschicht existiert.

Das Beispiel ist dem Buch von Kurose und Ross [22] entnommen. Es hat folgende Struktur:

1. Ein Client liest eine Zeile von seiner Standardeingabe (Tastatur) und sendet die Zeile über einen Socket an den Server.

2. Der Server liest eine Zeile von seinem Verbindungssocket.

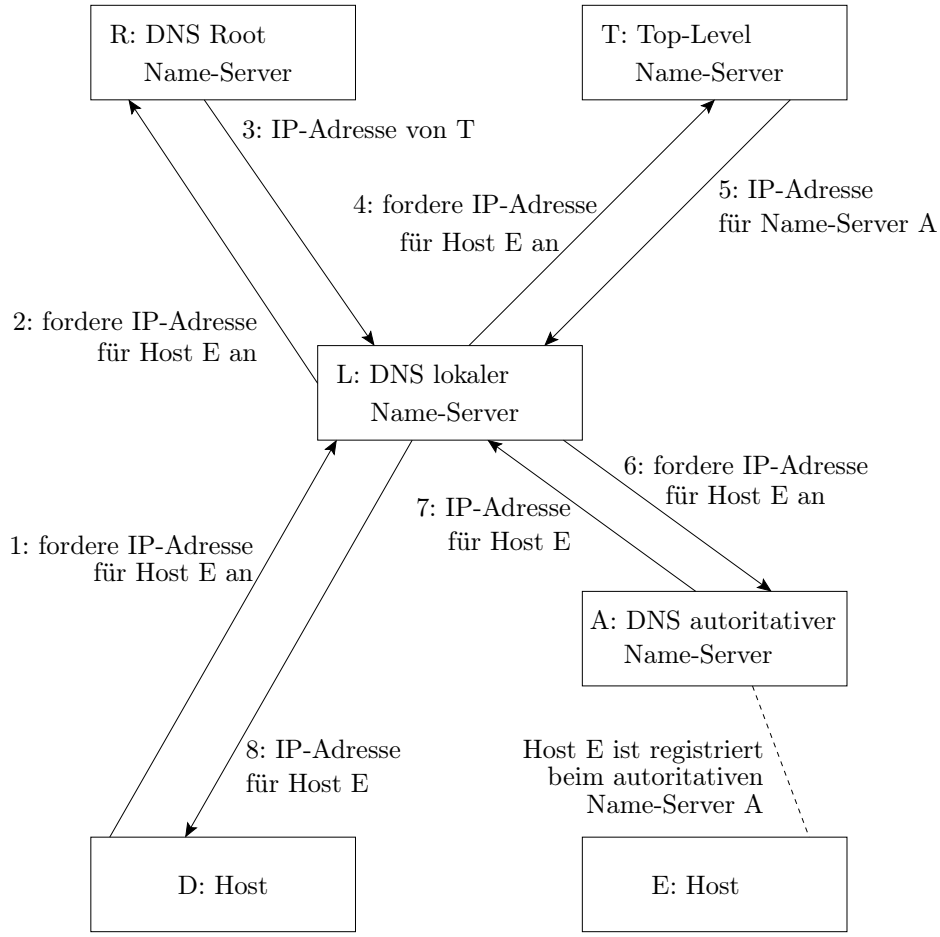

Abbildung 3.10: Ein Beispiel für eine DNS-Anfrage: Host D benötigt die IP-Adresse von Host E. Hier ist der vollständige Weg über den Root-Name-Server R, den Top-Level-Server T und den autoritativen Name-Server A dargestellt, in den meisten Fällen ist ein Teil der Information schon im Cache vorhanden und wird deshalb nicht abgefragt.

3. Der Server konvertiert die Zeile in Großbuchstaben.

4. Der Server sendet die modifizierte Zeile über seinen Verbindungssocket an den Client.

5. Der Client liest die modifizierte Zeile von seinem Socket und gibt sie auf seiner Standardausgabe (Monitor) aus.

```
// ConversionClient-Programm:

// Liest eine Zeile von der Tastatur, und sendet sie an den Server und
// bekommt von ihm eine Zeile zurück.
// Speichern als: ConversionClient.java
// Kompilieren: javac ConversionClient.java
// Starten: java ConversionClient
// In Zeile 11 muss der richtige Servername stehen bzw. "localhost",
```

```
// falls der Server auf demselben Rechner läuft.
// Der Server muss natürlich vorher gestartet sein.

01 import java.io.* ;
02 import java.net.*;
03 class ConversionClient {
04     public static void main(String[] argv) throws Exception
05     {
06         String sentence;
07         String modifiedSentence;
08         BufferedReader inFromUser =
09             new BufferedReader(
10                 new InputStreamReader(System.in));
11         Socket clientSocket = new Socket ("servername", 6789);
12         DataOutputStream outToServer =
13             new DataOutputStream(
14                     clientSocket.getOutputStream());
15         BufferedReader inFromServer =
16             new BufferedReader(new InputStreamReader(
17                     clientSocket.getInputStream()));
18         sentence = inFromUser.readLine();
19         outToServer.writeBytes(sentence + '\n');
20         modifiedSentence = inFromServer.readLine();
21         System.out.println("FROM SERVER:" +
22                             modifiedSentence);
23         clientSocket.close();
24     }
25 }
```

In den ersten beiden Zeilen werden die Bibliotheken für die Benutzereingabe und -ausgabe (01) und die Nutzung des API der Internet-Transportprotokolle (02) geladen. Nach dem Programmkopf des Clientprogramms und der Deklaration zweier Variablen `sentence` und `modifiedSentence` wird in den Zeilen 08 bis 10 ein Zeichenstrom `inFromUser` für die Annahme von Benutzereingaben eingerichtet. Dann wird in Zeile 11 ein Socket `clientSocket` vom Typ TCP zu dem gewünschten Konvertierungsdienst geöffnet. Der Dienst wird auf dem Host `servername` an Port 6789 erwartet. Die Variablen `outToServer` und `inFromServer` werden mit den Datenströmen des Socket hin zum Server (12-14) und zurück vom Server an den Client (15-17) initialisiert. Sie sind das eigentliche Socket-Interface, die *Türen*, durch die die Informationen fließen. Die Benutzereingabe wird nun aus dem Zeichenstrom vom Benutzer eingelesen, der Variablen `sentence` zugewiesen (18) und an den Server übergeben (19). Danach wird auf die Antwort des Servers gewartet, und die ankommenden Zeichen vom Server fließen in die Zeichenkette `modifiedSentence` (20). Das Ergebnis des Dienstes wird für den Benutzer ausgegeben (21 und 22) und die TCP-Verbindung geschlossen (23).

Die Benutzung des TCP-Dienstes geschieht in Java also durch die Anforde-

rung eines Objektes vom Typ Socket (Zeile 11) und die nachfolgende Nutzung des Sockets zum Schreiben (19) und Lesen (20) von Daten. Dazu notwendige Datenströme müssen vorher initialisiert werden (12-14 und 15-17).[17]

Das Gegenstück zum Client ist das Server-Programm:

```
// ConversionServer-Programm:

// Speichern als: ConversionServer.java
// Kompilieren: javac ConversionServer.java
// Starten: java ConversionServer

01 import java.io.* ;
02 import java.net.* ;
03 class ConversionServer {
04     public static void main(String[] argv) throws Exception
05     {
06         String clientSentence;
07         String capitalizedSentence;
08         ServerSocket welcomeSocket = new ServerSocket (6789);
09
10         while (true) {
11             Socket connectionSocket = welcomeSocket.accept();
12             BufferedReader inFromClient =
13                 new BufferedReader (new InputStreamReader (
14                         connectionSocket.getInputStream()));
15             DataOutputStream outToClient =
16                 new DataOutputStream(
17                         connectionSocket.getOutputStream());
18             clientSentence = inFromClient.readLine();
19             capitalizedSentence =
20                 clientSentence.toUpperCase() + '\n';
21             outToClient.writeBytes(capitalizedSentence);
22         }
23     }
24 }
```

In Zeile 08 wird der Dienst an Port 6789 bereit gestellt und läuft ab dann potenziell für immer (Zeile 10) (siehe Abschnitt 3.3.1, Charakterisierung von Serverprozessen). Der `welcomeSocket` ist eine Tür, an der auf Clients gewartet wird. Fragt ein Client den Dienst nach, wird ein neuer Socket `connectionSocket` erzeugt (Zeile 11), um den Kunden zu bedienen. Dazu werden Datenströme vom Client `inFromClient` (12-14) und zurück `outToClient` (15 - 17) erzeugt, und so die Datenströme vom Client zu Server (12-14) und vom Server zu Client (15-17) miteinander verbunden. Die Variable `inFromClient` ist dabei

[17] Möchte man einen UDP-Dienst benutzen, so müsste man ein Objekt vom Typ `DatagramSocket` erzeugen.

das Gegenstück zu der Variable `outToServer` des Client-Programms und die Variable `outToClient` das Gegenstück zu der Variablen `inFromServer` des Client-Programms. Über den in den Server eingehenden Datenstrom wird der zu übersetzende Satz entgegen genommen (18), in Großbuchstaben konvertiert (19 und 20) und das Ergebnis an den Client zurückgeschickt (21). An dem Beispiel wird auch deutlich, dass bei TCP die Struktur der Anwendungsnachricht und insbesondere die Erkennung des Nachrichtenendes Sache der Anwendung ist. Hier ist eine Nachricht durch genau eine Zeile (vgl. Zeile 18 im Server-Programm und Zeile 19 im Client-Programm) definiert.

Übungsaufgabe 3.9 In Abbildung 3.4 ist zu sehen, dass ein Router eines Firmennetzwerks eine *Schnittstelle* zwischen diesem privaten Netzwerk und dem Internet ist. Welche Aufgaben muss dieser Router erfüllen?

`https://e.feu.de/1801-router-aufgaben`

3.4 Transportschicht

3.4.1 Überblick

Die Aufgabe der Transportschicht ist die Bereitstellung von Kommunikationsdiensten für die Anwendungsprozesse, die auf den verschiedenen Endgeräten eines Computernetzwerks laufen. Hierbei geht es um die Unterstützung *logischer Kommunikation* zwischen Anwendungsprozessen auf verschiedenen Hosts, die in der Regel nicht direkt, sondern über eine Menge von dazwischenliegenden Routern und Kommunikationsverbindungen miteinander verbunden sind. Abbildung 3.11 illustriert die Nutzung der Transportschicht durch zwei Anwendungen auf verschiedenen Hosts, die über das Rechnernetz verbunden sind. Transportprotokolle werden auf den Endgeräten (Hosts) implementiert, nicht in Routern! Router dienen lediglich zum Transport von Paketen zwischen Hosts. Sie implementieren die Dienste der Schichten 3 bis 1 (Vermittlungsschicht bis Bitübertragungsschicht).

logische Kommunikation zwischen zwei Prozessen

Transportprotokolle nutzen jedoch die Dienste der Vermittlungsschicht zur Realisierung logischer Kommunikation zwischen Prozessen (vgl. hierzu auch das Beispiel der Briefkommunikation mittels der Posttransportschicht in Abschnitt 3.2.1). In der Regel laufen auf einem Host mehrere Anwendungsprozesse, die über das Rechnernetz kommunizieren wollen. Da auf Schicht 3 nur die Kommunikation zwischen Hosts unterstützt wird, muss ein Transportprotokoll Nachrichten verschiedener Prozesse über denselben Host senden und empfangen. Hierzu muss das Transportprotokoll den Nachrichtenstrom, den es von der Vermittlungsschicht erhält, auf die verschiedenen Empfängerprozesse verteilen. Dies nennt man *Demultiplexen*. Damit dies funktioniert, muss das sendende Transportprotokoll geeignete Kontrollinformationen zu den Nachrichten, die es vom Anwendungsprozess erhält, hinzufügen, bevor es sie an die Vermittlungsschicht zum Versenden übergibt. Dies nennt man *Multiplexen*.

Demultiplexen

Multiplexen

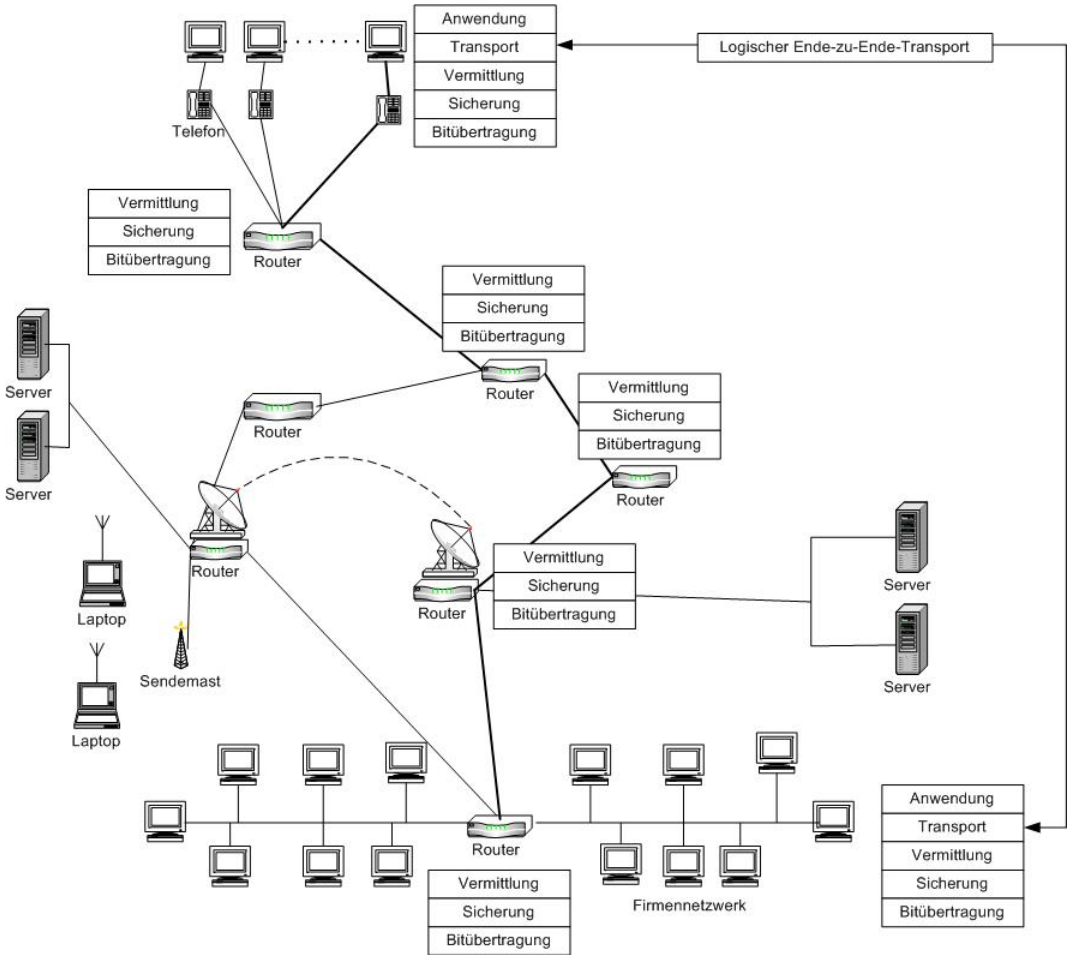

Abbildung 3.11: Bereitstellung logischer Kommunikation zwischen zwei Anwendungen.

Das Multiplexen/Demultiplexen muss jedes Transportprotokoll leisten, da Transportprotokolle gerade die Host-zu-Host-Übertragung der Vermittlungsschicht um die Prozess-zu-Prozess-Kommunikation erweitern. Die Anwendungen stellen aber noch andere Anforderungen an die Transportschicht, wie wir in Abschnitt 3.3.3 gesehen haben. Das Internet bietet zwei unterschiedliche Transportprotokolle, die sich hinsichtlich der ersten Anforderung, der *Zuverlässigkeit des Dienstes*, unterscheiden:

Zuverlässigkeit des Dienstes

- Das UDP-Protokoll erlaubt das Senden einzelner Nachrichten zwischen Prozessen. Es macht dabei keine Garantien bezüglich der Ankunft (Datenverlust) und der Reihenfolge der Nachrichten. UDP funktioniert praktisch wie die gelbe Post: Der Sender baut keine Verbindung zum Empfänger auf, sondern schickt die Nachrichten einfach los. Der Empfänger weiß also gar nicht, dass Nachrichten an ihn gesendet werden.[18] UDP wird deshalb als *verbindungsloser Dienst* bezeichnet.

verbindungsloser Dienst

[18] Wenn die Kommunikation funktionieren soll, muss der Empfänger aber prinzipiell bereit sein, Nachrichten zu empfangen.

- Das TCP-Protokoll garantiert, dass die von einem Senderprozess zu einem Empfängerprozess übertragenen Nachrichten irgendwann in der richtigen Reihenfolge und vollständig beim Empfängerprozess ankommen. Der Sender baut vor dem Übertragen der Daten eine virtuelle Verbindung zum Empfänger auf, ähnlich dem Telefonieren: Erst wenn der Empfänger den Hörer abnimmt, kann der Sender dem Empfänger die Nachricht mitteilen. TCP wird deshalb als *verbindungsorientierter Dienst* bezeichnet.

Garantie der Zuverlässigkeit

verbindungsorientierter Dienst

Für die anderen beiden Anforderungen von Anwendungen, Bandbreite und Zeit, gibt es im derzeitigen Internet keine Unterstützung. Die neue Version des Internets (IP Version 6) stellt hierfür neue Konzepte zur Verfügung, die in der einschlägigen Literatur behandelt werden (z. B. [23, 39]).

3.4.2 Multiplexen und Demultiplexen von Anwendungen

Jedes Transportprotokoll nutzt die darunter liegende Schicht zum Senden und Empfangen von Nachrichten. Solche Schicht-4-Nachrichten werden im folgenden als Segmente bezeichnet; siehe Abbildung 3.5 auf Seite 115. Im Internet unterstützt die Vermittlungsschicht den Nachrichtenaustausch zwischen Hosts. Jeder Host wird eindeutig durch seine IP-Adresse identifiziert. Ankommende Nachrichten (Segmente), die bei einem Host von der Vermittlungsschicht an die Transportschicht abgeliefert werden, müssen nun an die richtigen Anwendungsprozesse weitergeleitet werden. Wie wir in Abschnitt 3.3.4 kennen gelernt haben, werden Prozesse durch Portnummern identifiziert. Unter Verwendung von IP-Adressen und Portnummern funktioniert das *Multiplexen* und *Demultiplexen* wie folgt, siehe Abbildung 3.12.

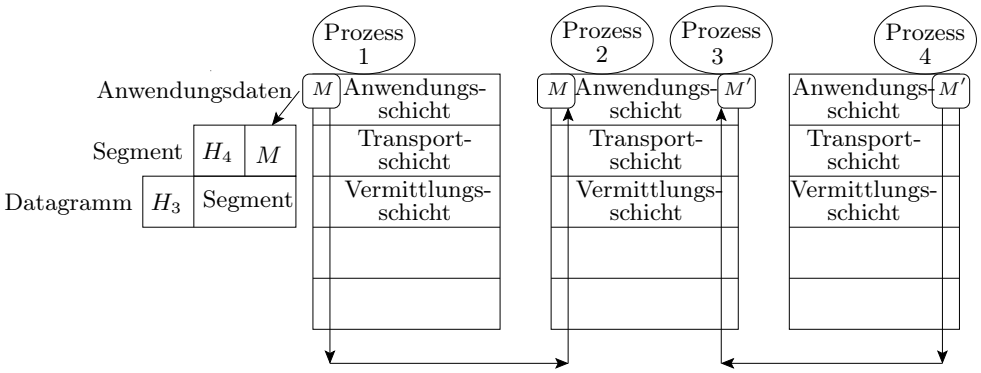

Abbildung 3.12: Multiplexen und Demultiplexen von Anwendungen: Prozess 1 sendet eine Nachricht M an Prozess 2 und Prozess 4 die Nachricht M' an Prozess 3.

Das sendende Transportprotokoll fügt die zu sendenden Anwendungsdaten M mit einem Segmentheader H_4 zu einem Segment zusammen. Im Segmentheader gibt es insbesondere zwei Felder, die für die Identifizierung des Sender- und Empfängerprozesses verwendet werden: SourcePort und DestinationPort. Das Hinzufügen der Portnummern des Quellprozesses und des

Multiplexen

Zielprozesses gehört zum *Multiplexen* (*Multiplexing*). Das Segment wird dann an die Vermittlungsschicht zur Versendung weitergegeben.

Demultiplexen

Das empfangende Transportprotokoll erhält ein Segment von der Vermittlungsschicht. Mittels der Felder SourcePort und DestinationPort im Segmentheader identifiziert das Transportprotokoll den korrekten Empfängerprozess und stellt das Segment zu. Diesen Schritt nennt man *Demultiplexen* (*Demultiplexing*). Zum Demultiplexen reicht insbesondere die Angabe der Portnummer des Zielprozesses alleine nicht aus. Da mehrere Anwendungsprozesse desselben Typs (die ja dieselbe Portnummer nutzen) auf einem Host laufen können, wird auch die Angabe der Portnummer des Senderprozesses benötigt, um die logische Kommunikationsverbindung eindeutig zu definieren, siehe Abbildung 3.13.

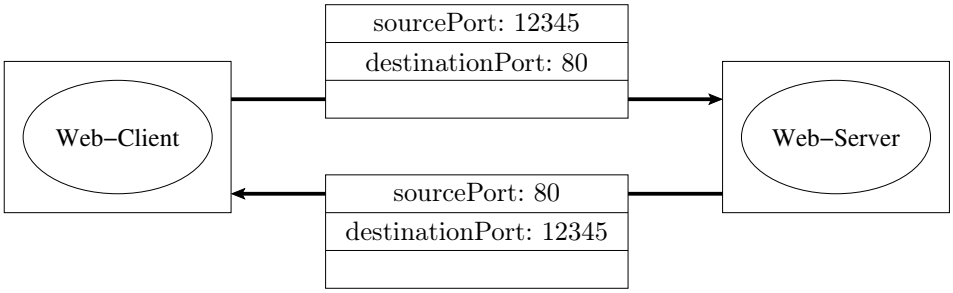

Abbildung 3.13: Nutzung von SourcePortnummern und DestinationPortnummern.

Als Beispiel zu Multiplexen- und Demultiplexen betrachten wir einen Web-Server und mehrere Web-Clients, die HTTP-Anfragen zum Server senden.

Bevor ein Client eine HTTP-Anfrage sendet, muss er eine Verbindung mit dem Server aufbauen, wobei dort ein neuer Prozess mit Portnummer 80 erzeugt wird. Nun kann der Client eine Anfrage verpackt in Segmente zum Server senden. Die Portnummern von Quelle und Ziel im Header eines Segments ermöglichen das Demultiplexen.[19]

Rolle der Portnummer

Wenn aber Web-Clients auf verschiedenen Hosts denselben SourcePort gewählt haben, dann ermöglicht die eindeutige IP-Adresse im IP-Header des Datagramms die eindeutige Zuordnung des Segments zum korrekten Serverprozess. Dies wird in Abbildung 3.14 illustriert: Web-Server B kann die Anfragen mit SourcePort 12345 der Web-Clients A und C durch die Angabe der sourceIP von A und C unterscheiden.

Rolle der IP-Adresse

[19] Während die Portnummer des Zielprozesses vom sendenden Anwendungsprozess mitgeteilt werden muss, wird die SourcePortnummer des sendenden Prozesses automatisch gesetzt. Das Protokoll nimmt einfach die nächste freie Portnummer.

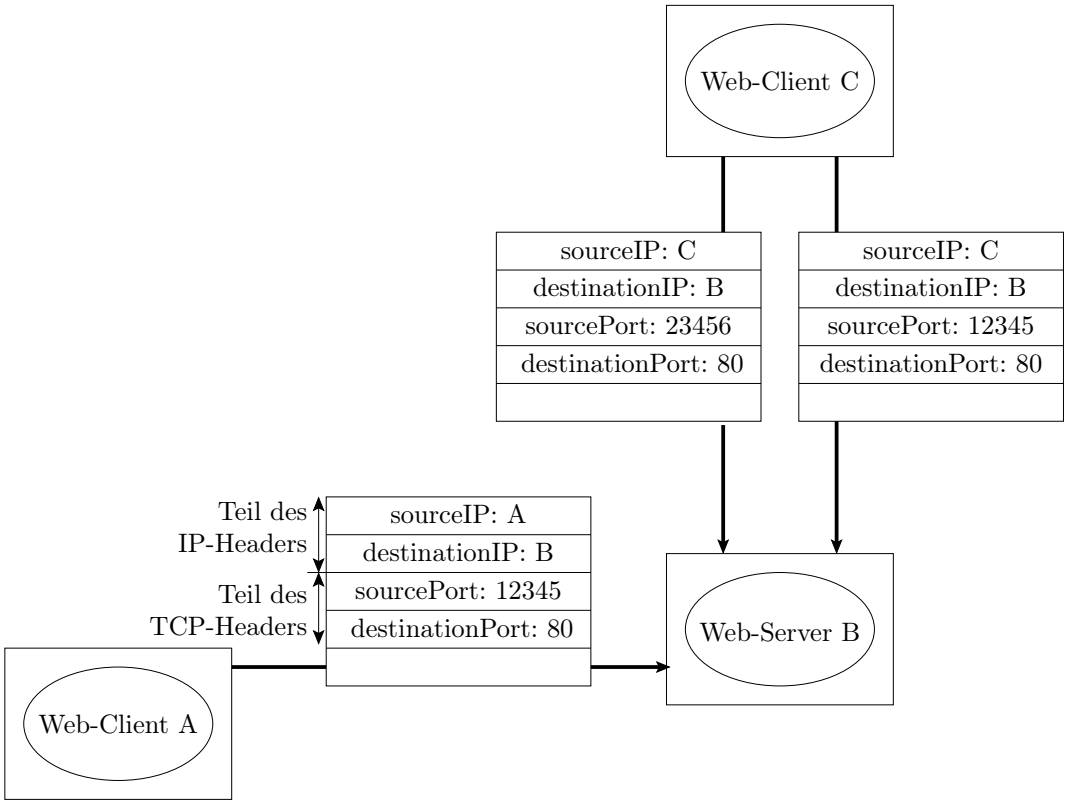

Abbildung 3.14: Kommunikation von Web-Clients unter Verwendung einer identischen SourcePortnummer mit einem Webserver.

3.4.3 Verbindungslose Kommunikation mit UDP

UDP (*User Datagram Protocol*) ist ein einfaches Transportprotokoll, das in RFC 768 definiert ist. UDP ist ein verbindungsloser Dienst, d. h. ein Senderprozess kann direkt UDP-Segmente verschicken, ohne vorher eine Verbindung mit dem Empfängerprozess aufgebaut zu haben. Dies entspricht der Briefzustellung per Post, bei der ein Absender ja auch einen Brief verschicken kann, ohne dass der Empfänger etwas davon weiß. UDP realisiert einen unzuverlässigen Transportdienst, d. h.

Verbindungsloser Dienst

1. Nachrichten können verloren gehen.
2. Nachrichten, die in einer Reihenfolge vom Senderprozess an den Empfängerprozess gesendet werden, können in einer anderen Reihenfolge beim Empfängerprozess abgeliefert werden.

UDP macht keine Aussage über die Übertragungsverzögerung bei der Nachrichtenübertragung. UDP wird bevorzugt für einmalige Abfragen und Anwendungen in Client/Server-Umgebungen benutzt, in denen die Schnelligkeit der Zustellung wichtiger ist als die Genauigkeit, z. B. die Übertragung von Sprache oder Video. Ein Beispiel für die Anwendung von UDP ist das Senden von DNS-Anfragen. Nach der DNS-Spezifikation kann DNS auch über TCP laufen, aber es wird fast immer über UDP realisiert, da eine DNS-Anfrage relativ

klein ist und in ein UDP-Segment passt. So kann ein DNS-Server jede hereinkommende Anfrage gleich beantworten und sofort wieder vergessen. Wenn eine DNS-Anfrage erfolglos ist, kann der DNS-Client die Anfrage bei einem anderen Name-Server versuchen oder die anfragende Anwendung informieren, dass der Server nicht erreichbar ist.

Dienstnutzung

Abbildung 3.15 zeigt eine typische Client-Server-Anwendung, die UDP zur Kommunikation verwendet. Ein Anwendungsprozess erzeugt zuerst einen Dienstzugangspunkt zum UDP-Dienst. Dies ist üblicherweise ein Socket vom Typ Datagramm-Socket, der dem Anwendungsprozess eine eindeutige IP-Adresse und Portnummer zuordnet. Ein Beispiel für die Verwendung von Sockets für den TCP-Dienst haben wir in Abschnitt 3.3.6 kennen gelernt. Der Anwendungsprozess kann nun von dem Socket Nachrichten empfangen oder verschicken. Bei Ende des Anwendungsprozesses wird der Socket wieder freigegeben.

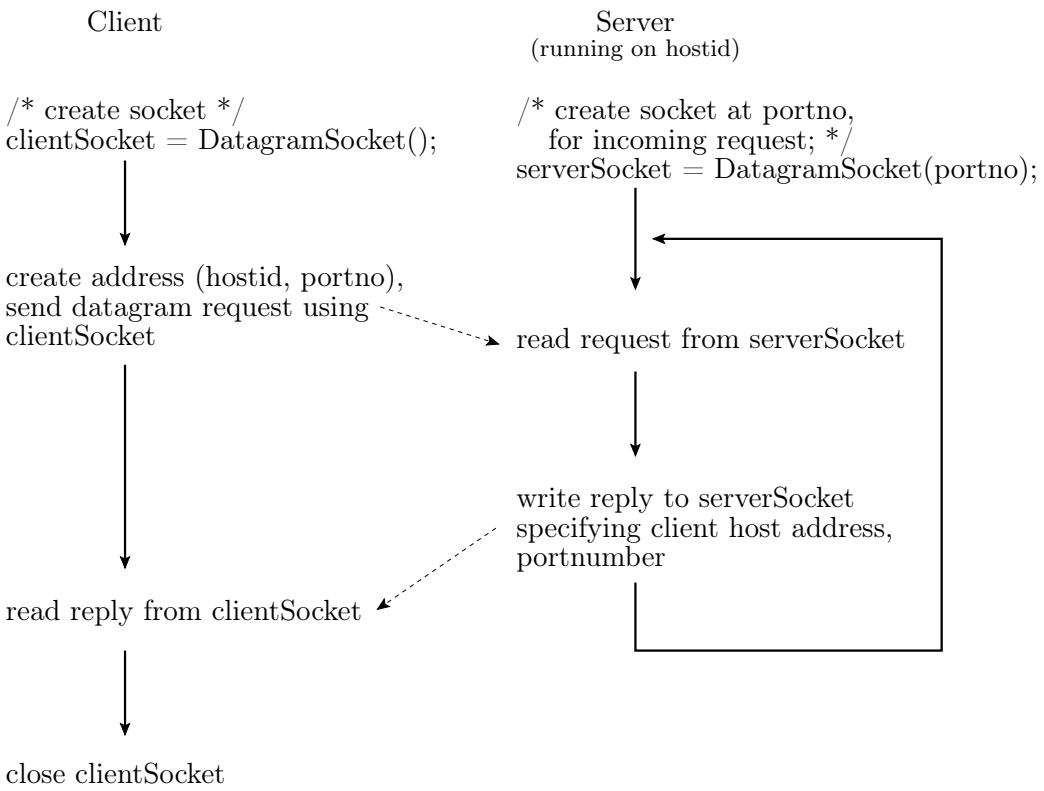

Abbildung 3.15: Auf UDP basierende Client-Server-Anwendung.

Segmentstruktur

UDP-Segmente bestehen aus einem Header mit vier Feldern und einem Datenfeld mit den Anwendungsdaten. Im Header, siehe Abbildung 3.16, stehen die SourcePortnummer (2 Byte) und die DestinationPortnummer (2 Byte) zur Identifikation von Sender- und Empfängerprozess. Hinzu kommen die Länge des Segmentes (inklusive Header) in Byte (2 Byte) und eine Prüfsumme (check sum), die zur Fehlererkennung bei fehlerhafter Übertragung des Headers bzw. des Datenfeldes dient.

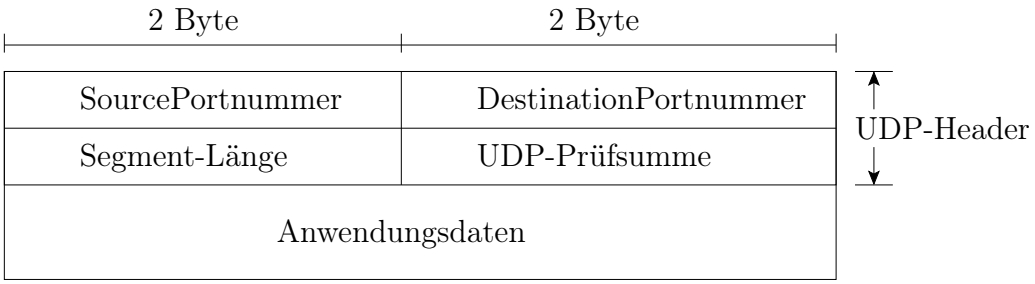

Abbildung 3.16: UDP-Segmentstruktur mit einem 8-Byte-Header.

Übungsaufgabe 3.10

1. Was ist die maximale Länge eines UDP-Segments?
2. Recherchieren Sie die maximale Länge eines Hostnamens und beantworten Sie die Frage, warum eine DNS-Anfrage in ein UDP-Segment passt, wie oben schon behauptet wurde.

`https://e.feu.de/1801-udp`

Fehlererkennung

UDP führt eine simple *Fehlererkennung* ein, da nicht alle Vermittlungsprotokolle eine Fehlererkennung oder -korrektur aufweisen. Falls UDP einen Fehler erkennt, dann wird das Segment entweder verworfen oder mit einer Warnung an den Anwendungsprozess weitergereicht. Zur Fehlererkennung speichert UDP eine Prüfsumme (2 Byte) im Header des UDP-Segments. Die UDP-Prüfsumme wird berechnet als 1er-Komplement[20] der 1er-Komplement-Summe[21] aller 16-Bit-Wörter des Segments. Das UDP-Protokoll auf dem Empfänger-Host wird dann die gleiche Operation auf dem Segment durchführen, aber inklusive der Prüfsumme; hier sollte dann das Resultat 1111111111111111 herauskommen. Ist dagegen mindestens ein Bit gleich 0, dann ist ein Übertragungsfehler aufgetreten.

Übungsaufgabe 3.11 Berechnen Sie die Prüfsumme der folgenden Wörter, wobei hier zur Vereinfachung nur 8-Bit-Wörter (statt 16 Bit wie bei UDP) verwendet werden sollen.

01110101, 11110000, 00111111, 11111000, 11001001.

`https://e.feu.de/1801-checksum`

[20]Das 1er-Komplement einer Bitsequenz wird durch Umkehren aller Bits erzeugt: Aus einer 0 wird eine 1 und umgekehrt.

[21]Die 1er-Komplement-Summe ist die Summe aller Wörter, wobei der Überlauf selbst wieder als Wort hinzuaddiert wird. Als Beispiel für diese Berechnung siehe die Lösung von Übungsaufgabe 3.11.

3.4.4 Prinzipien zuverlässigen Datentransfers

zuverlässiger Datenübertragungsdienst

Eines der grundsätzlichen Probleme bei Rechnernetzen ist das Problem der zuverlässigen Datenübertragung. Grundsätzlich möchten Prozesse einen *zuverlässigen Datenübertragungsdienst*, schließlich nimmt man Verluste bei Nachrichten nur ungern auf sich. Dieser Dienst soll also einen *zuverlässigen Übertragungskanal* für Nachrichten realisieren. Er soll garantieren, dass

- die Daten unverändert (z. B. ohne Bitfehler),
- vollständig und
- in der Reihenfolge beim Empfängerprozess abgeliefert werden, in der sie vom Senderprozess verschickt wurden.

zuverlässiges Datenübertragungsprotokoll

Übertragungskanal

Das Bedürfnis nach einem zuverlässigen Datenübertragungsdienst kann prinzipiell auf jeder Schicht in einem Netzwerk auftreten. Ein solcher zuverlässiger Übertragungskanal für Nachrichten wird dann durch ein *zuverlässiges Datenübertragungsprotokoll* implementiert, das einen darunter liegenden unzuverlässigen Übertragungskanal verwendet. Das Problem bei der Implementierung eines solchen Protokolls liegt in der potentiellen Unzuverlässigkeit der darunter liegenden Schicht. Im Beispiel des Internet realisiert z. B. das TCP-Protokoll einen zuverlässigen Übertragungskanal in der Transportschicht. Dazu baut TCP auf dem IP-Protokoll der Vermittlungsschicht auf, das selbst wiederum ein unzuverlässiges Host-zu-Host-Protokoll ist. Im Allgemeinen kann die Schicht unterhalb der beiden zuverlässig kommunizierenden Endpunkte aus einer einzigen physikalischen Verbindung bestehen, oder ein komplexes globales Rechnernetz sein.

Dienst als abstrakter Datentyp

Man kann sich einen Dienst generell auch als abstrakten Datentyp (siehe Kurse *Datenstrukturen I, Datenstrukturen II, Datenstrukturen*) vorstellen. Der Dienst definiert dann eine Datenstruktur und die darauf arbeitenden Operationen. Ein Protokoll ist dann die Implementierung des Dienstes bzw. des abstrakten Datentyps, der den Dienst spezifiziert.

Ein zuverlässiger Datenübertragungsdienst kann auf einem unzuverlässigem Medium im Prinzip wie in Abbildung 3.17 implementiert werden.

Zuerst ruft ein Senderprozess mittels der Funktion `rdt_send(daten)` die Senderseite des zuverlässigen Datentransferprotokolls auf, um *Daten* an den Empfängerprozess zu senden (`rdt` steht für „reliable data transfer“). Das zuverlässige Datentransferprotokoll verschickt die Daten mittels der darunter liegenden unzuverlässigen Netzwerkschicht an das auf der Empfängerseite laufende zuverlässige Datentransferprotokoll. Dazu verwendet es die von der unterliegenden unzuverlässigen Schicht bereitgestellte Funktion `udt_send(paket)` (`udt` steht für „unreliable data transfer“). Auf der Empfängerseite empfängt das Datentransferprotokoll die Pakete mittels der Funktion `rdt_receive(paket)`.

Zusätzliche Kommunikation von Daten und Kontrollinformation

Bei Fehlern in der Datenübertragung ist weitere bilaterale Kommunikation von Daten und Kontrollinformation zwischen der Senderseite und der Empfängerseite des zuverlässigen Datenübertragungsprotokolls notwendig, um die Fehler zu beheben. Wenn die Nachricht komplett übertragen wurde, kann der Empfänger des zuverlässigen Datentransferprotokoll mittels der Funktion

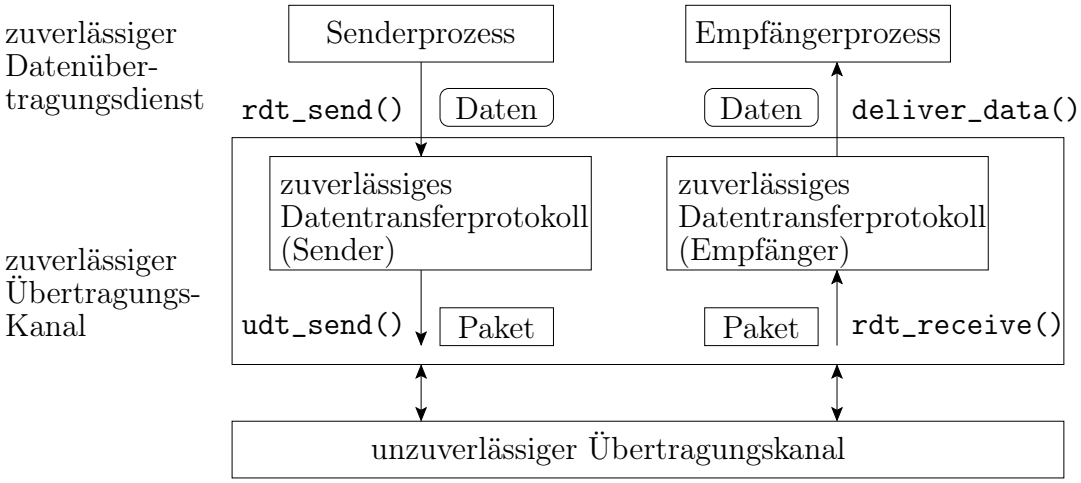

Abbildung 3.17: Implementierung eines zuverlässigen Datenübertragungsdienst. Die Pfeile drücken einen Datenfluss aus.

`deliver_data(daten)` die Nachricht an den Empfängerprozess liefern. Aus der Sicht von Senderprozess und Empfängerprozess handelt es sich um einen zuverlässigen Übertragungskanal, der aber in Wirklichkeit vom zuverlässigen Datenübertragungsprotokoll mit Hilfe des unterlagerten unzuverlässigen Übertragungskanals realisiert wurde.

Übungsaufgabe 3.12 Von welchen Schichten werden die Funktionen `rdt_send()` und `udt_send()` in Abbildung 3.17 zur Verfügung gestellt?

`https://e.feu.de/1801-zuverlaessige-datenuebertragung`

Kommunikationsprotokolle werden in der Regel als *Zustandsautomaten* beschrieben. Hierbei beschreiben Knoten die Zustände und Pfeile die Übergänge zwischen Zuständen. An jedem Pfeil steht das Ereignis, das den Übergang auslöst, und die Operationen, die beim Übergang ausgeführt werden. Ereignis und Operationen sind durch einen waagerechten Strich getrennt. Den einfachsten Fall eines Datenübertragungsprotokolls für einen zuverlässigen Übertragungskanal zeigt Abbildung 3.18:[22] Hier gehen wir davon aus, dass der genutzte Übertragungskanal auch zuverlässig ist, d. h. `rdt_receive()` und `udt_send()` verlieren und verfälschen keine Daten.

Zustandsautomaten

Bei der Übertragung von Daten über einen unzuverlässigen Übertragungskanal muss ein Transportprotokoll generell mit den folgenden Problemen umgehen:

- Einzelne Bits in einer Nachricht können fehlerhaft übertragen werden. — Bitfehler
- Ganze Nachrichten können verloren gehen. — Nachrichtenverlust

[22] Die Operation `make_paket(paket,data)` erzeugt ein neues `paket` mit dem Inhalt `data`. Analog schreibt `extract_data(paket,data)` den Dateninhalt des Paketes in die Variable `data`.

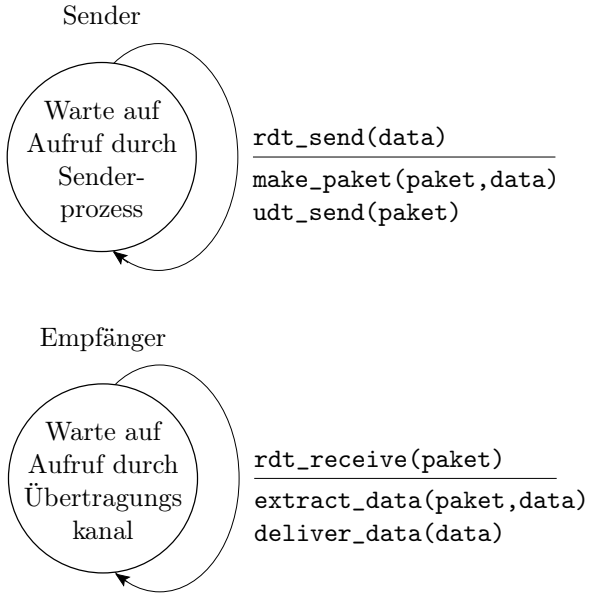

Abbildung 3.18: Datenübertragungsprotokoll für einen zuverlässigen Übertragungskanal.

Überflutung

- Falls die Datenrate, in der ein Empfänger Daten empfangen kann, niedriger sein kann, als die Rate, in der ein Sender Daten senden kann, dann muss zusätzlich das Problem der Überflutung des Empfängers mit Nachrichten gelöst werden.

Im folgenden werden wir Lösungen für die ersten beiden Probleme kennen lernen. Um Überflutungen zu vermeiden, sind Techniken der Flusskontrolle notwendig, siehe Abschnitt 3.4.5.6 und [23].

unidirektionale Kommunikation

Zur Vereinfachung der Darstellung betrachten wir im folgenden nur die unidirektionale Kommunikation von einem Senderprozess zu einem Empfängerprozess. Bidirektionale Kommunikation zwischen zwei Prozessen kann dann als Kombination von Sender- und Empfängerseite in einem Protokoll-Client implementiert werden.

3.4.4.1 Zuverlässige Datenübertragung über einen verlustfreien Kanal mit Bitfehlern

Betrachten wir zuerst den hypothetischen Fall, dass Datenpakete über einen verlustfreien Kanal mit Bitfehlern, d. h. ohne Verlust und in der richtigen Reihenfolge, aber ggf. mit veränderten Bits übertragen werden. Es stellt sich also das Problem, wie Bitfehler vom Protokoll zu behandeln sind. Wenn zwei Personen über einen solchen Kanal, z. B. eine schlechte Telefonverbindung, einen längeren Text diktieren, dann würden sie wahrscheinlich das folgende Protokoll verwenden: der Zuhörer bestätigt nach jedem Satz, den er gehört, verstanden und aufgezeichnet hat, den Empfang, indem er „OK“ sagt. Falls der Empfänger einen Satz nicht versteht, z. B. wegen einer Störung, dann wird er den Sprecher um Wiederholung bitten. In diesem Protokoll werden sowohl *positive Bestätigungen* (*Acknowledgements-ACK*) als auch *negative Bestätigun-*

positive und negative Bestätigungen

gen (*Negative Acknowledgements-NACK*) verwendet. In der Literatur heißen zuverlässige Datenübertragungsprotokolle, die auf Wiederholungen basieren, auch *ARQ-Protokolle* (*Automatic Repeat reQuest*).

Grundsätzlich werden in einem ARQ-Protokoll drei Fähigkeiten benötigt, um Bitfehler zu behandeln:

ARQ-Protokoll

1. *Fehlererkennung*: Um Bitfehler zu erkennen, müssen zusätzliche Informationen in einem Paket übertragen werden (z. B. die Prüfsumme).

2. *Rückmeldung vom Empfänger*: Da Senderprozess und Empfängerprozess in der Regel auf verschiedenen Hosts laufen, kann der Sender nur mittels expliziter Nachrichten etwas über den Zustand des Empfängers erfahren. In obigem Beispiel müssen die positiven (*ACK*) und negativen (*NACK*) Bestätigungen vom Empfänger zum Sender kommuniziert werden.

3. *Wiederholung*: Ein fehlerhaft empfangenes Paket muss vom Sender noch einmal übertragen werden.

Die Senderseite eines Protokolls für einen verlustfreien Kanal mit Bitfehlern hat zwei Zustände, siehe Abbildung 3.19. In einem Zustand wartet das Protokoll auf Daten, die übertragen werden sollen, in dem anderen Zustand wartet das Protokoll auf eine Empfangsbestätigung des Empfängers. Falls ein NACK empfangen wird, dann sendet das Protokoll das letzte Paket noch einmal und bleibt im Wartezustand auf die Bestätigung für dieses Paket. Falls ein ACK empfangen wird, dann wurde das letzte Paket korrekt empfangen und das Protokoll kehrt in den Zustand des Wartens auf neue Daten des Senderprozesses zurück.[23]

Senderseite

In dem oben beschriebenen Protokoll wartet der Sender mit der Übertragung des nächsten Pakets solange bis eine Bestätigung für das aktuelle Paket eingetroffen ist. Deshalb heißt dieses Protokoll auch *Stop-and-Wait-Protokoll*.

Stop-and-Wait-Protokoll

Die Empfängerseite des Stop-and-Wait-Protokolls hat nur einen Zustand, in dem das Protokoll auf die Ankunft eines neuen Pakets wartet, siehe Abbildung 3.20. Das Protokoll prüft dann, z. B. anhand der Prüfsumme, ob das Datenpaket einen Bitfehler aufweist. Liegt ein Bitfehler vor, dann wird ein NACK verschickt, ansonsten werden die Anwendungsdaten extrahiert, an den Empfängerprozess weitergereicht, und ein ACK an den Sender zurückgeschickt.

Empfängerseite

Unglücklicherweise ist das oben skizzierte Protokoll nicht in der Lage, mit Bitfehlern in den Bestätigungen umzugehen. Dieses Problem kann prinzipiell auf zwei Arten gelöst werden:

fehlerhafte Bestätigungen

1. Durch Hinzufügen einer genügend langen Prüfsumme kann der Sender nicht nur einen Bitfehler in einer Bestätigungsnachricht erkennen, sondern diesen auch beheben. Dies löst das Problem eines fehlerhaften Kanals, der keine Pakete verliert.

Bitfehlerbehebung

[23] Außer dem Zustandsübergang ist keine weitere Aktion notwendig, deshalb stehen unter dem Strich auch keine Aktionen.

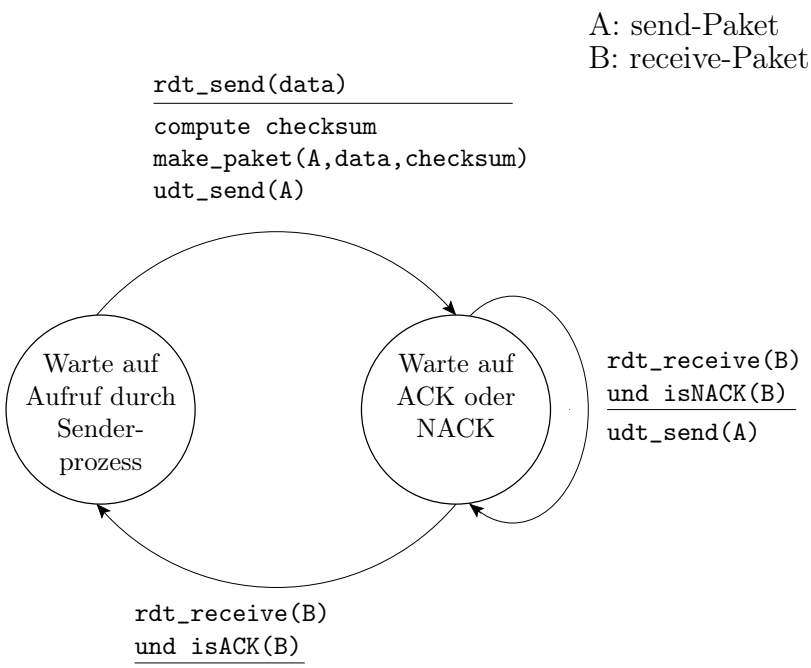

Abbildung 3.19: Senderseite eines Stop-and-Wait- Protokolls für einen verlustfreien Kanal mit Bitfehlern.

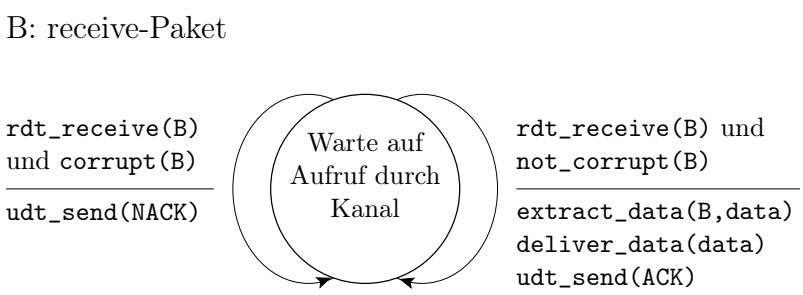

Abbildung 3.20: Empfängerseite eines Stop-and-Wait-Protokolls für einen verlustfreien Kanal mit Bitfehlern.

wiederholtes Senden

Sequenznummer

2. Bei Empfang eines verfälschten ACK oder NACK kann der Sender nicht entscheiden, ob der Empfänger ein ACK oder NACK senden wollte. In diesem Fall kann der Sender einfach das aktuelle Datenpaket erneut senden. Dies führt aber zu *duplizierten Paketen* im Nachrichtenstrom vom Sender zum Empfänger. Der Empfänger hat nun das Problem, dass er nicht weiß, ob das letzte gesendete ACK/NACK korrekt vom Sender empfangen wurde (und ob daher das aktuelle Paket eine Wiederholung ist oder nicht). Üblicherweise wird dieses Problem durch Hinzufügen eines neuen Feldes *Sequenznummer* im Paketheader gelöst. Der Sender nummeriert nun alle gesendeten Pakete und trägt in dieses neue Feld die Sequenznummer des Pakets ein. Damit kann der Empfänger einfach prüfen, ob ein empfangenes Paket eine Wiederholung mit alter Sequenznummer oder ein neues Paket ist. Für den einfachen Fall unseres Stop-and-Wait-Protokolls genügt ein einzelnes Bit zur Nummerierung, um die Pakete zu unterscheiden. Da wir annehmen, dass der Kanal keine Pakete ver-

liert, müssen ACK/NACK-Pakete nicht explizit die Sequenznummer des Pakets enthalten, auf das sie sich beziehen. Der Sender kann annehmen, dass sich jedes ACK/NACK, ob verfälscht oder nicht, immer auf das letzte übertragene Paket bezieht.

Stop-and-Wait-Protokoll mit Sequenznummern

Abbildung 3.21 zeigt das Zustandsübergangsdiagramm der Senderseite eines Stop-and-Wait-Protokolls mit Sequenznummern. Das Protokoll kommt ohne negative Bestätigungen (NACK) aus. Wenn der Empfänger ein Paket nicht korrekt empfangen hat, sendet der Empfänger einfach ein ACK für das letzte korrekt empfangene Paket. Wenn der Sender nun zwei ACKs für dasselbe Paket P erhält, weiß er, dass der Empfänger das nachfolgende Paket von P nicht korrekt empfangen hat. Unser verbessertes Protokoll hat nun doppelt so viele Zustände wie das einfache Protokoll in Abbildung 3.19 und 3.20. Der Grund hierfür ist, dass der Zustand nun speichern muss, ob das nächste zu sendende Paket die Sequenznummer 0 oder 1 haben soll.

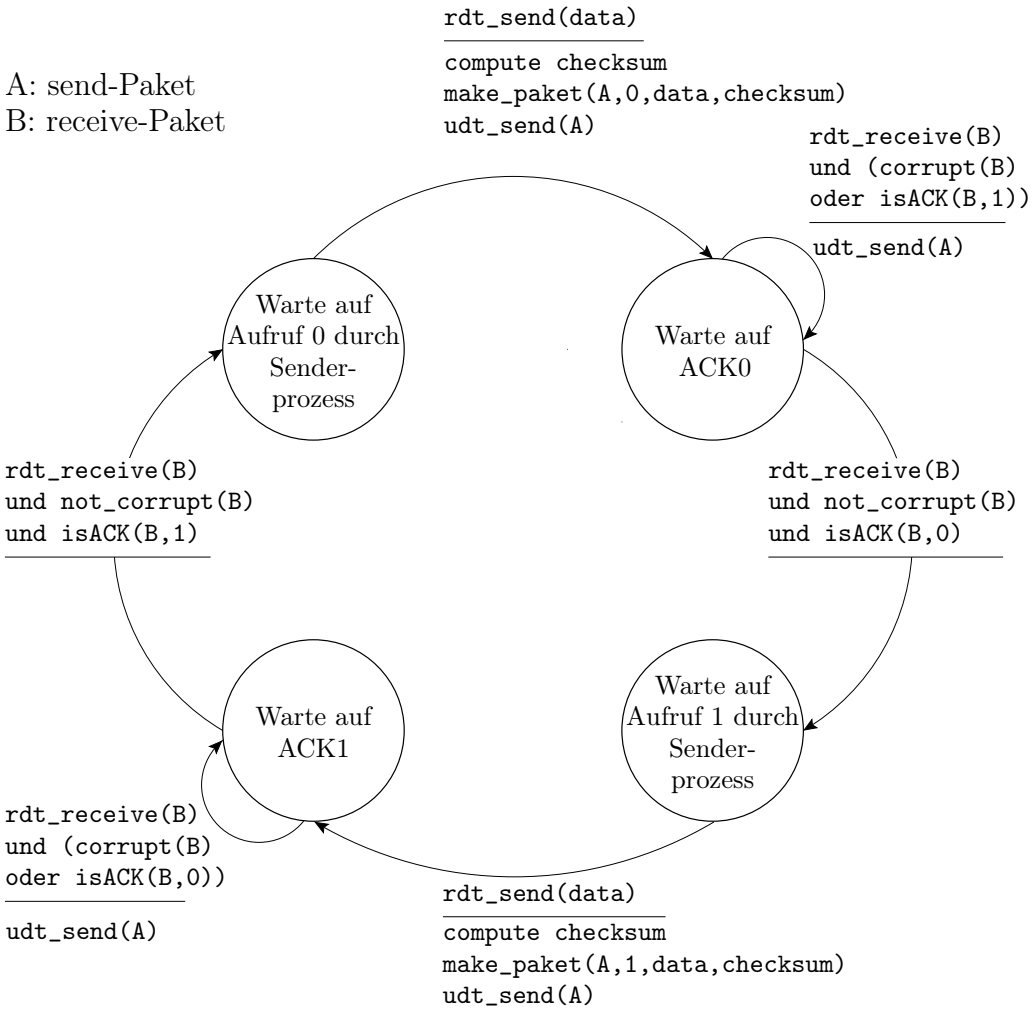

Abbildung 3.21: Senderseite eines Stop-and-Wait-Protokolls mit Sequenznummern, die zwischen 0 und 1 wechseln.

Abbildung 3.22 zeigt das Zustandsübergangsdiagramm der Empfängerseite eines Stop-and-Wait-Protokolls mit Sequenznummern.

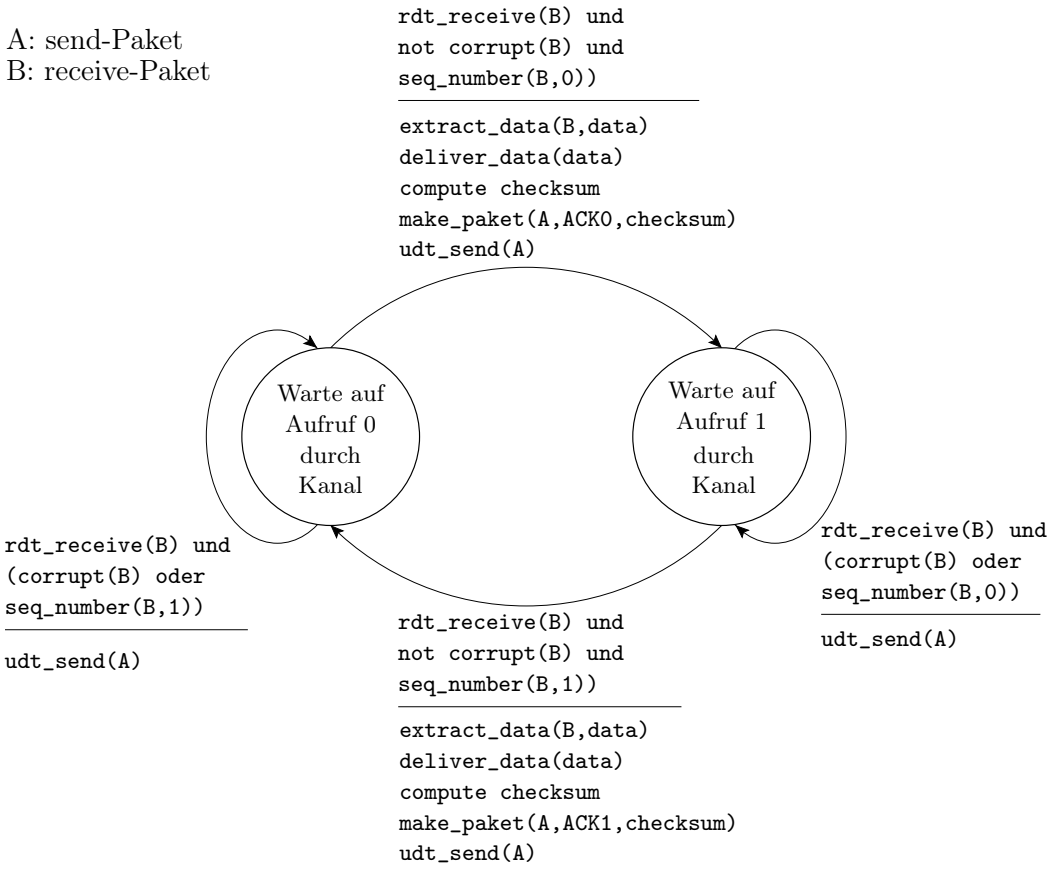

Abbildung 3.22: Empfängerseite eines Stop-and-Wait-Protokolls mit Sequenznummern.

3.4.4.2 Zuverlässige Datenübertragung über einen verlustbehafteten Kanal mit Bitfehlern

Nehmen wir nun an, dass der Kanal Pakete in der richtigen Reihenfolge überträgt, aber sowohl Bitfehler verursachen als auch Pakete verlieren kann. Dies tritt bei den heute verwendeten Netzwerktechnologien durchaus häufiger auf. Ein zuverlässiges Datenübertragungsprotokoll muss nun zwei neue Aufgaben lösen:

1. Erkennung von Paketverlusten,

2. Behandlung von Paketverlusten.

Mit Hilfe der uns schon bekannten Techniken, wie z. B. Prüfsummen, Sequenznummern, Bestätigungsnachrichten (ACK) und Wiederholungen, können wir Paketverluste behandeln. Aber wie kann ein Paketverlust bemerkt werden?

In diesem Abschnitt werden wir eine Lösung vorstellen, bei welcher der Sender die Aufgabe der Erkennung und Behandlung von Paketverlusten löst.

Nehmen wir an, der Sender überträgt ein Paket und entweder dieses Paket oder das dazugehörige ACK geht verloren. In beiden Fällen kommt keine Antwort des Empfängers beim Sender an. Falls der Sender nun bereit ist, lang genug zu warten, so dass er sicher sein kann, dass ein Paket verloren wurde, dann kann der Sender nach Ablauf der Wartezeit einfach das Paket erneut übertragen. Dieses Protokoll löst unser Problem.

Die Frage ist nun: Wie lange muss der Sender warten, um sicher zu sein, dass ein Paket verloren gegangen ist? Im Prinzip müsste der Sender mindestens die Zeit abwarten, die ein Paket vom Sender zum Empfänger und zurück benötigt. Diese Zeit beinhaltet auch Zeiten, in denen Pakete im Internet in Routern oder Gateways[24] gepuffert werden, und die Zeit der Verarbeitung des Pakets bei der Empfängerseite des Protokolls. Diese worst-case Verzögerung lässt sich in der Praxis kaum schätzen. Hinzu kommt, dass unser Protokoll möglichst rasch einen Paketverlust behandeln sollte, anstatt sehr lange darauf zu warten, dass ein Paketverlust tatsächlich sehr sicher aufgetreten ist. Deshalb wird in der Praxis eine Zeitschranke gewählt, deren Überschreitung einen Paketverlust anzeigt, obwohl eventuell gar keiner aufgetreten ist. Sobald der Sender innerhalb der Zeitschranke kein ACK für das Paket erhält, wird er das Paket wiederholen, auch wenn tatsächlich gar kein Paket oder ACK verlorengegangen sein mag, z. B. wegen einer Überlastung des Internets. Dies wird in den Fällen, in denen das Paket gar nicht verloren gegangen ist, zu Duplikaten führen. Solche Duplikate kann der Empfänger aber anhand der Sequenznummern erkennen, bestätigen und verwerfen.

Damit der Sender das Überschreiten der Zeitschranke erkennen kann benötigt er einen *Zeitgeber* (*Countdown–Timer*). Der Sender muss in der Lage sein

Countdown-Timer

- den Zeitgeber bei jedem gesendeten Paket zu setzen,
- bei Ablauf des Zeitgebers das verlorengegangene Paket zu wiederholen, und
- bei Ankunft einer Bestätigungsnachricht (ACK) den Zeitgeber zu stoppen.

Die Zeit vom Starten des Countdown–Timers bis zum Ablauf nennt man auch *Timeout-Intervall*. Die Existenz von Duplikaten der Datenpakete oder der Bestätigungsnachrichten kompliziert für den Sender die Behandlung von Bestätigungsnachrichten. Um festzustellen, zu welchem Paket eine Bestätigungsnachricht gehört, muss die Sequenznummer des bestätigten Pakets vom Empfänger in ein neues Feld (*Bestätigungsfeld*) des ACK-Pakets geschrieben werden. Anhand dieser Sequenznummer kann der Sender feststellen, welches Paket vom Empfänger bestätigt wurde.

Timeout-Intervall

Bestätigungsfeld

[24] Ein Gateway ist ein spezieller Typ von Router. Während Router in einem Netzwerk dasselbe Routing-Protokoll verwenden, verbinden Gateways Netzwerke, die verschiedene Routing-Protokolle benutzen.

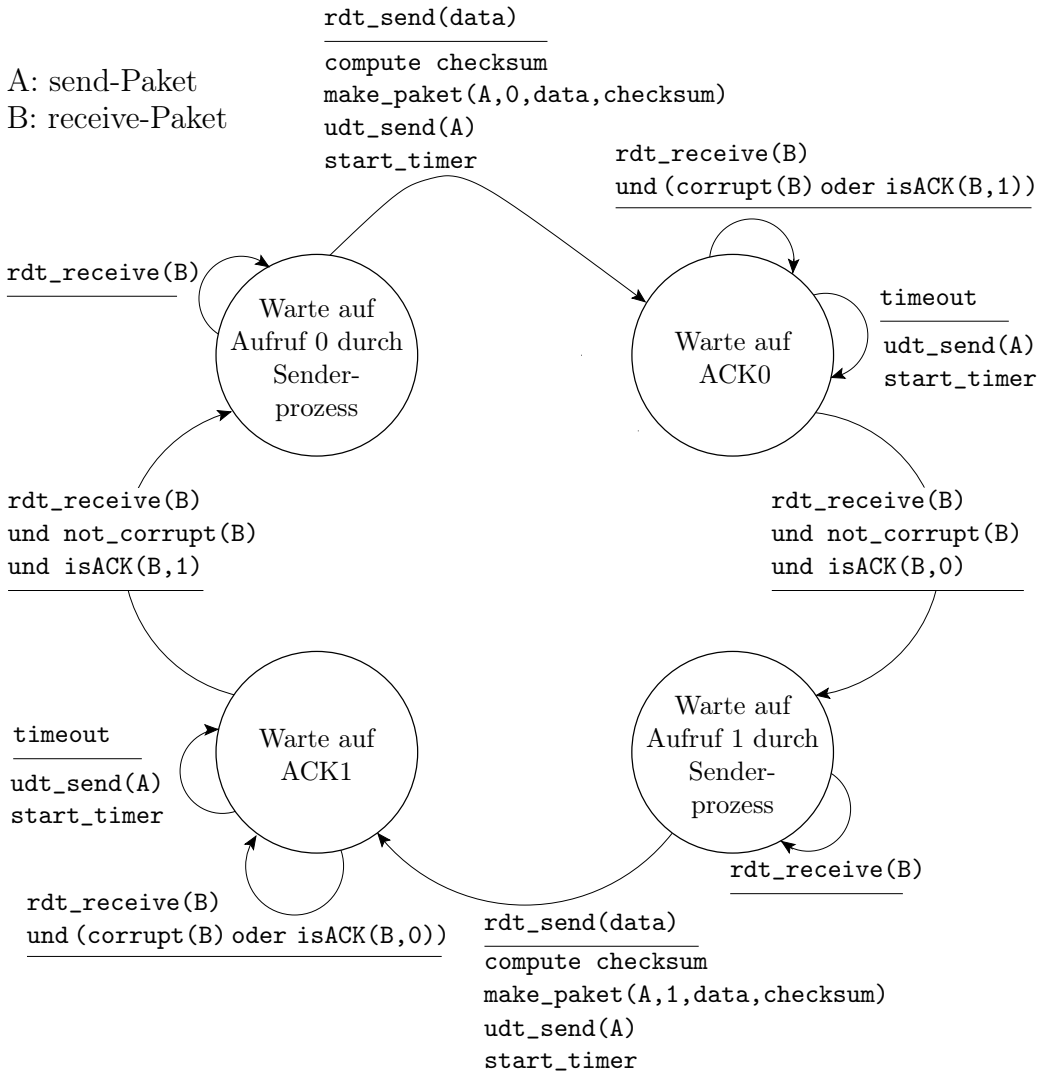

Abbildung 3.23: Senderseite des Alternating-Bit-Protokolls.

Abbildung 3.23 zeigt die Senderseite des oben beschriebenen Protokolls. Der aufmerksame Leser wird feststellen, dass in Abbildung 3.23 keine Zeitgeber angehalten werden, z. B. wenn ein Acknowledgement empfangen wird. Dies ist in einem Zustandsübergangsdiagramm auch nicht notwendig, da ja das Protokoll in einen neuen Zustand übergeht, in dem das Ablaufen des Zeitgebers einfach ignoriert wird. Deshalb muss der Zeitgeber in dieser Darstellung auch nicht explizit angehalten werden.

Alternating-Bit-Protokoll

Das in Abbildung 3.23 gezeigte Protokoll erfüllt nun alle Anforderungen an ein Protokoll für die zuverlässige Datenübertragung über einen verlustbehafteten Kanal mit Bitfehlern. In der Literatur wird dieses Protokoll auch als *Alternating-Bit-Protokoll* bezeichnet, da die Sequenznummern der Pakete immer zwischen 0 und 1 alternieren.

Damit ein solches Protokoll mit Timeout effizient funktioniert, muss die Länge des Timeout-Intervalls sorgfältig gewählt werden, wie die nächste Aufgabe zeigt.

Übungsaufgabe 3.13 Wir betrachten das Alternating-Bit-Protokoll, siehe Abbildung 3.23.

1. Wir verwenden ein extrem kurzes Timeout-Intervall. Funktioniert das Protokoll überhaupt noch? Wenn ja, welche Nachteile ergeben sich?

2. Jetzt setzen wir auf ein sehr (bzw. zu) langes Timeout-Intervall. Das Protokoll funktioniert, aber unter welchen Umständen ist dies nachteilig?

3. Welche Informationen braucht man, um die „richtige“ Länge für das Timeout-Intervall festzulegen?

`https://e.feu.de/1801-modifikation-alternating-bit`

Abbildung 3.24 zeigt in einem sogenannten *Zeitdiagramm* einige typische Situationen bei Ablauf des Protokolls. Häufig werden diese Situationen in solchen Zeitdiagrammen dargestellt. Die Zeit verläuft in solchen Diagrammen von oben nach unten. Die Spalten entsprechen hierbei den Aktionen der Akteure (Senderseite, Empfängerseite). Zwischen den beiden Akteuren stellen Pfeile den Nachrichtenfluss dar, und die Pfeilbeschriftungen identifizieren die gesendeten Nachrichten. Die eckigen Klammern von `send paket1` zu `timeout` deuten an, wann der Zeitgeber abgelaufen ist.

Zeitdiagramm

Leider ist die Leistung des Alternating-Bit-Protokolls in der Praxis kaum ausreichend, da es ein Stop-and-Wait-Protokoll ist. Nach jeder Übertragung eines Pakets wartet das Protokoll auf die Bestätigung. In der Praxis werden deshalb Protokolle eingesetzt, die mehr als ein Paket senden dürfen, bevor eine Bestätigung des Empfängers eintreffen muss. Hierzu ist es notwendig, Pakete durchzunummerieren. Die *Fenstergröße* legt fest, wie viele Pakete gesendet werden dürfen, ohne dass die vorhergehenden Pakete bestätigt werden müssen. Ein sogenanntes *Sendefenster* enthält zu jedem Zeitpunkt sämtliche Paketnummern, die der Sender aktuell beim Versenden von Paketen benutzen darf, siehe auch Abbildung 3.25. Der Empfänger bestätigt in der Regel nicht den Empfang jedes einzelnen Paketes, sondern den Empfang einer korrekt empfangenen Paketfolge (*kumulative Bestätigung*). Dazu sendet er die Nummer des letzten in der Folge korrekt empfangenen Pakets.[25] So arbeitende Protokolle heißen *Fenster-Protokolle*. In diesem Sinne kann z. B. das Alternating-Bit-Protokoll als Fensterprotokoll mit der Fenstergröße 1 bezeichnet werden. Für Protokolle mit einer Fenstergröße größer als 1 sind aber weitere Vorkehrungen für Fehlerbehandlungen zu treffen: Werden Nachrichten oder Quittungen fehlerhaft übertragen oder gehen sie verloren, so kommen zusätzlich zu Zeitüberwachungen, die jetzt individuell pro Paket erfolgen müssen, zwei aktive Eingriffsmöglichkeiten des Empfängers zum Einsatz:

Fenstergröße

Sendefenster

kumulative Bestätigung

Fenster-Protokolle

[25] In vielen Implementierungen wird auch die Nummer des nächsten in der Folge erwarteten Pakets gesendet.

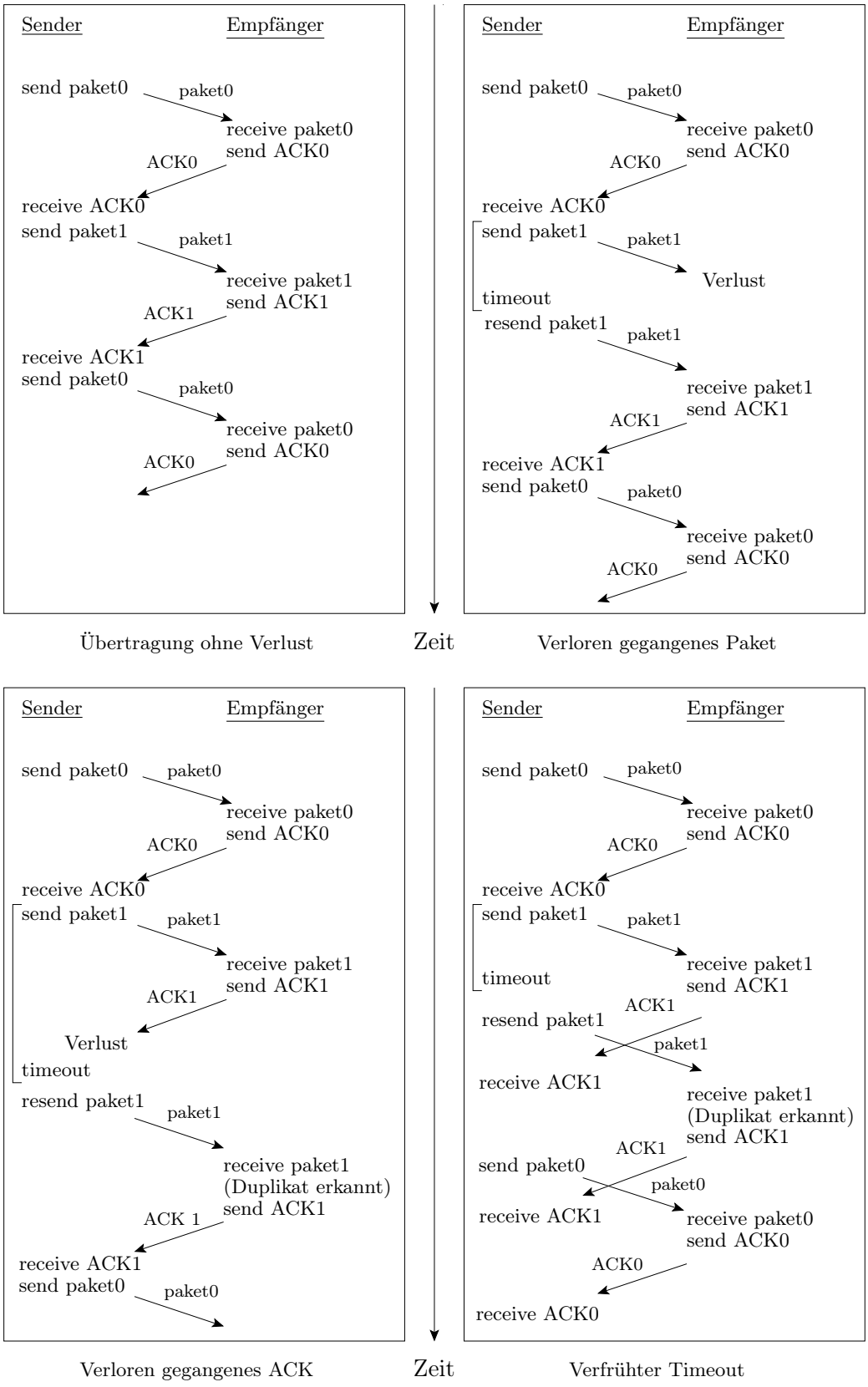

Abbildung 3.24: Typische Situationen beim Alternating-Bit-Protokoll.

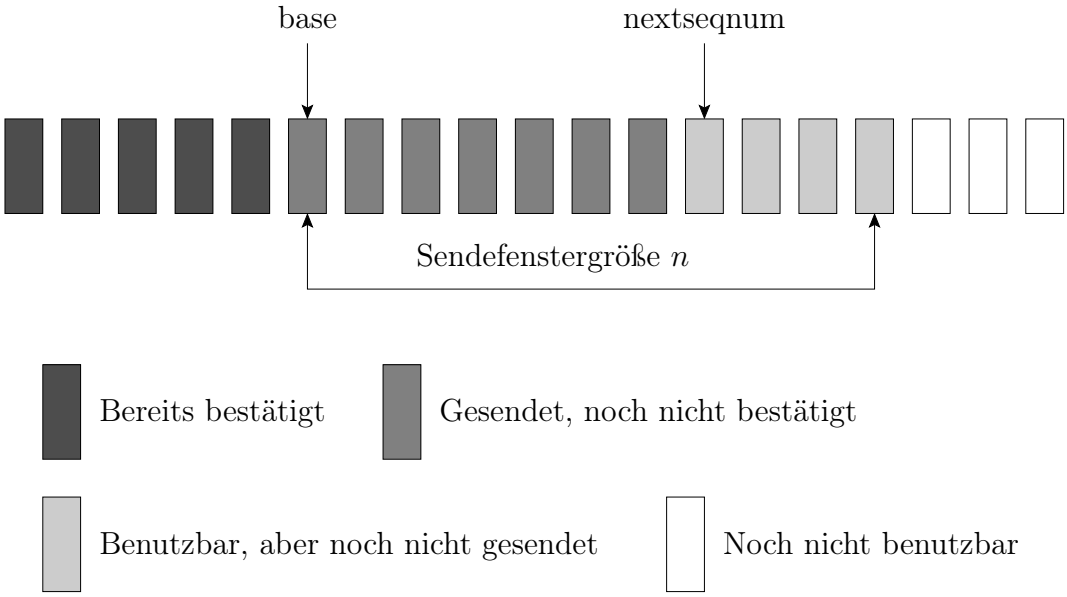

Abbildung 3.25: Sequenznummern im go-back-n aus Sendersicht. Die Sequenznummern der Pakete beim Sender werden in vier Gruppen eingeteilt: Die Pakete bis Nummer nextseqnum $-$ 1 wurden übertragen, davon wurden die bis base $-$ 1 schon bestätigt. Pakete mit Sequenznummern von nextseqnum bis base $+ n -$ 1 dürfen noch gesendet werden.

1. Er kann den Sender auffordern, das fehlerhafte Paket und alle danach bereits gesendeten $n - 1$ Folgepakete erneut zu senden (*go-back-n*, auch *Sliding-Window-Protokoll* genannt).

go-back-n

Sliding-Window-Protokoll

2. Er kann den Sender auffordern, nur das fehlerhafte Paket erneut zu senden (*selective-reject/repeat*).

In diesem Kurs werden diese Ansätze nicht weiter behandelt. Wir gehen nur insoweit darauf ein, dass das im folgenden vorgestellte Protokoll aus der Praxis, also TCP, Konzepte der Fensterprotokolle wie Sequenznummern und kumulative Bestätigungen benutzt. Zur weiteren Vertiefung sei auf die Literatur [23, 39] oder auf weiterführende Kurse (z. B. *Kommunikations- und Rechnernetze*) verwiesen.

3.4.5 Verbindungsorientierte Kommunikation mit TCP

Nachdem wir in den vorangegangenen Abschnitten die Grundlagen von Transportprotokollen und den verbindungslosen Transportdienst UDP im Internet kennen gelernt haben, wollen wir uns nun mit dem zweiten Transportdienst im Internet, dem verbindungsorientierten *Transportprotokoll TCP* (*Transmission Control Protocol*, [8]), beschäftigen.

TCP

Punkt-zu-Punkt

Vollduplex-Datenübertragung

TCP realisiert einen zuverlässigen Datentransfer zwischen zwei Anwendungsprozessen. TCP arbeitet *immer Punkt-zu-Punkt*, d. h. zwischen *genau zwei* Anwendungsprozessen. TCP unterstützt *Vollduplex-Datenübertragung* (*full-duplex*) zwischen genau einem Sender und einem Empfänger. Das heißt beide Anwendungsprozesse können über eine Verbindung gleichzeitig Daten an die andere Seite schicken.[26] Anwendungsprozesse nutzen TCP, indem sie zuerst eine Verbindung zwischen Sender und Empfänger aufbauen. Nachdem die Verbindung aufgebaut ist, kann der Senderprozess beliebige Bytefolgen über die Verbindung an den Empfänger schicken. TCP garantiert, dass die identische Bytefolge beim Empfänger ankommt. Insofern funktioniert TCP wie eine Datei, in die der Sender Daten schreibt und aus welcher der Empfänger Daten liest (analog zu einer Pipe in Unix). TCP vermittelt nicht, wie UDP, einzelne Nachrichten zwischen den Anwendungsprozessen, sondern einen Datenstrom. Wenn die Anwendungsprozesse den Datenstrom zur Übertragung von Nachrichten, also Blöcken von Informationen, nutzen wollen, so müssen sie Beginn und Ende von Nachrichten selbst im Datenstrom codieren, so dass der Empfänger Beginn und Ende einer Nachricht erkennen kann. Wir konnten dieses Vorgehen schon am Anwendungsbeispiel in Abschnitt 3.3.6 sehen, wo das Ende einer Nachricht durch ein '$\backslash n$' (Zeilenende-Zeichen) angezeigt wird. Auch bei HTTP in Abschnitt 3.3.2 lässt sich dies erkennen. Damit der HTTP-Server die eingehende GET-Anfragenachricht bearbeiten konnte, muss er zuerst die gesamte Nachricht vom Socket lesen. Das Ende der Nachricht erkennt der Server an der Zeichenfolge CR LF CR LF, also zweimal das Paar Carriage Return, Line Feed hintereinander. Damit ist klar, wann eine Nachricht zu Ende ist und eine neue Nachricht beginnt.

Wenn ein Anwendungsprozess die Kommunikation beenden will, dann baut er die TCP-Verbindung wieder ab. Sobald die Verbindung abgebaut ist, können keine Daten mehr über die Verbindung übertragen werden.

verbindungsorientiertes Protokoll

Aufgrund der zentralen Bedeutung der Verbindung heißt TCP auch *verbindungsorientiertes Protokoll*.

3.4.5.1 Verbindungsmanagement

Verbindungsaufbau

Bevor Daten auf einer TCP-Verbindung übertragen werden können, muss diese Verbindung zwischen Sender-Anwendungsprozess und Empfänger-Anwendungsprozess aufgebaut werden. Analog zu UDP wird auch hier ein Socket verwendet, um eine Verbindung von einem Client-Prozess zu einem Server-Prozess zu erzeugen. Dabei gibt der Client-Prozess die IP-Adresse und die Portnummer des Server-Prozesses an. Mit dieser Information kann das TCP auf dem Client-Host (im folgenden Client-TCP genannt) eine Verbindung zum TCP auf dem Server-Host (im folgenden Server-TCP) aufbauen. Dabei geht der Client in drei Schritten vor; siehe Abbildung 3.26, die den Verbindungsaufbau in einem *Zeitdiagramm* darstellt.

[26] Bei einer Halbduplexübertragung kann eine Seite nicht gleichzeitig senden u. empfangen.

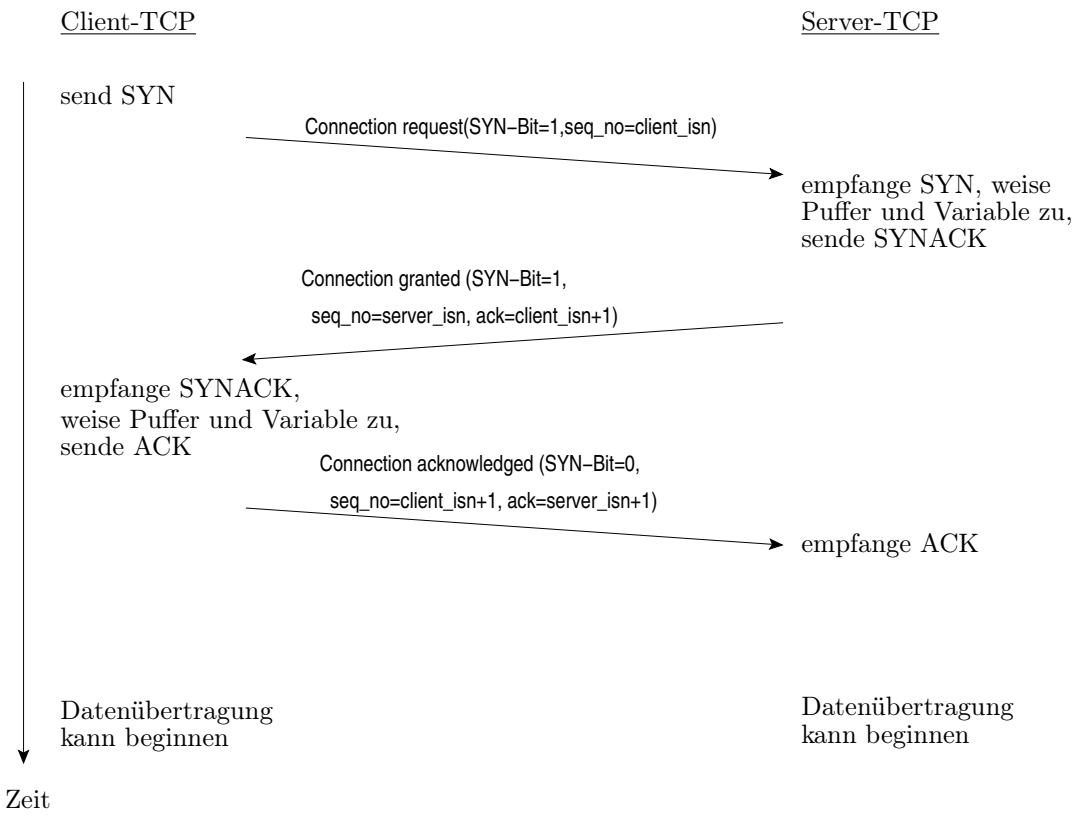

Abbildung 3.26: Zeitdiagramm des TCP-Verbindungsaufbaus.

1. Das Client-TCP sendet ein sogenanntes SYN-Segment[27] an das Server-TCP. Dieses Segment enthält keine Anwendungsdaten, aber es hat ein spezielles Header-Feld, das SYN-Bit, auf 1 gesetzt. Zusätzlich enthält das Sequenznummer-Feld des SYN-Segments eine initiale Sequenznummer (`client_isn`), die das Client-TCP gewählt hat.

2. Sobald das SYN-Segment beim Server-TCP ankommt, erzeugt es eine neue Verbindung und weist ihr entsprechende Puffer und Variablen zu. Dann schickt das Server-TCP ein SYNACK-Segment[27] an das Client-TCP zurück. Dieses Segment enthält noch keine Anwendungsdaten, aber es enthält drei wichtige Informationen im Segment-Header: das SYN-Bit ist auf 1 gesetzt, das Acknowledgement-Feld ist auf die gerade empfangene Sequenznummer `client_isn+1` gesetzt,[28] und das Sequenznummer-Feld ist auf eine initiale Sequenznummer des Servers (`server_isn`) gesetzt, die das Server-TCP gewählt hat. Dieses SYNACK-Segment bestätigt dem Client-TCP, dass das Server-TCP die Verbindung mit der initialen Sequenznummer `client_isn` des Client-TCP akzeptiert hat und dass die initiale Sequenznummer des Servers `server_isn` ist.

[27] SYN ist die Abkürzung von *synchronize sequence numbers*, ACK steht für *acknowledge*.

[28] TCP sendet als Bestätigungsnummer die Nummer des Bytes, das es als nächstes erwartet, siehe auch Abschnitt 3.4.5.2.

3. Sobald das Client-TCP das SYNACK-Segment empfängt, erzeugt es ebenfalls eine entsprechende Verbindung und weist ihr entsprechende Puffer und Variablen zu. Dann schickt das Client-TCP eine Bestätigungsnachricht. In diesem Segment ist das Acknowledgement-Feld auf `server_isn`+1 und das SYN-Bit auf 0 gesetzt.

Sobald diese drei Nachrichten erfolgreich ausgetauscht wurden, ist die Verbindung etabliert und die Anwendungsprozesse können Daten über die Verbindung übertragen. Weil beim Verbindungsaufbau genau drei Nachrichten ausgetauscht werden, heißt dieses Verfahren auch *Drei-Wege-Handschlag* (*Three-ways handshake*).

Drei-Wege-Handschlag

Wenn der Client die Verbindung schließen will, dann sendet er ein *close* Kommando an das Client-TCP. Er geht dann wie folgt vor:

1. Das Client-TCP sendet ein sogenanntes FIN-Segment[29] an das Server-TCP. Dieses Segment enthält keine Anwendungsdaten, aber es hat ein spezielles Header-Feld, das FIN-Bit, auf 1 gesetzt.

2. Wenn das Server-TCP ein FIN-Segment empfängt, dann schickt es eine Bestätigung, ein ACK-Segment, an das Client-TCP.

3. Dann schickt das Server-TCP ein eigenes FIN-Segment, mit FIN-Bit auf 1 gesetzt, an das Client-TCP.

Schließen einer Verbindung

4. Wenn das Client-TCP das FIN-Segment empfängt, dann schickt es eine Bestätigung, ein ACK-Segment, an das Server-TCP. Nach einer bestimmten Wartezeit wird die Verbindung gelöscht und die zugewiesenen Resourcen (Variablen, Puffer) werden freigegeben.

Abbildung 3.27 fasst die Zustandsübergänge, die Client-TCP und Server-TCP typischerweise durchlaufen, zusammen.

Nachdem das Client-TCP das letzte FIN des Server-TCP empfangen und bestätigt hat, wartet es noch eine gewisse Zeit (die implementationsabhängig zwischen 30 Sekunden und 2 Minuten liegt) und gibt dann die Verbindungsressourcen frei. Diese Wartezeit ist notwendig, damit das Client-TCP die letzte Bestätigung wiederholen kann, falls sie verloren geht. Die obige Diskussion des Verbindungsmanagements von TCP hat den normalen Ablauf des Verbindungsaufbaus und -abbaus geschildert. Sonderfälle, wie z. B. den gleichzeitigen Wunsch von Client-TCP und Server-TCP, die Verbindung abzubauen, wurden nicht behandelt. Interessierte Leser werden dazu auf die gängige Literatur verwiesen (z. B. Stevens [38]).

[29] FIN steht für *finish*.

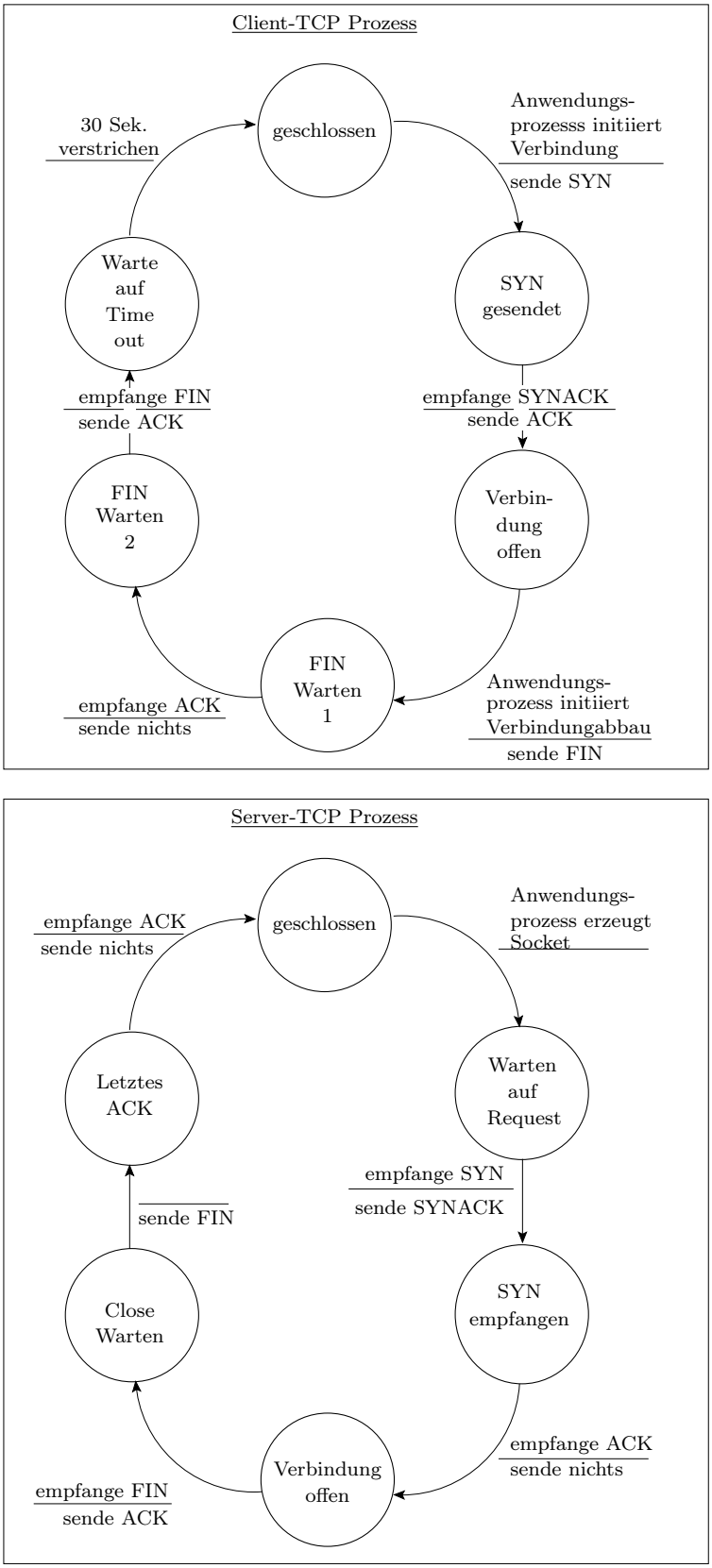

Abbildung 3.27: Zustandsübergänge von Client-TCP und Server-TCP.

Übungsaufgabe 3.14 Beim Aufbau einer TCP-Verbindung zwischen einem Client und einem Server wird ein Drei-Wege-Handschlag durchgeführt, d. h. drei Segmente werden ausgetauscht, wobei auch die initialen Sequenznummern festgelegt werden, siehe Abbildung 3.26.

1. Warum genügt nicht ein Zwei-Wege-Handschlag, d. h. warum kann der Client nicht einfach an den Server ein SYN-Segment schicken, um den Aufbau einer Verbindung anzufordern, und der Server bestätigt mit einem SYNACK-Segment?

2. Warum ist die Angabe der initialen Sequenznummern beim Aufbau der TCP-Verbindung erforderlich?

`https://e.feu.de/1801-3-wege-handshake`

3.4.5.2 Datentransfer in TCP

Sobald eine TCP-Verbindung aufgebaut ist, können sich die beiden Anwendungsprozesse gegenseitig Daten schicken. In dem Programmbeispiel in Abschnitt 3.3.6 entspricht der Verbindungsaufbau der Konstruktion des Sockets in Zeile 11 des Client-Programms auf Seite 132. Sobald der Client-Anwendungsprozess Daten in den Socket schreibt (Zeile 19), werden diese vom Client-TCP behandelt, siehe Abbildung 3.28. Das Client-TCP leitet diese Daten direkt in den Sendepuffer der Verbindung weiter, der beim Verbindungsaufbau dieser Verbindung zugeordnet wurde.

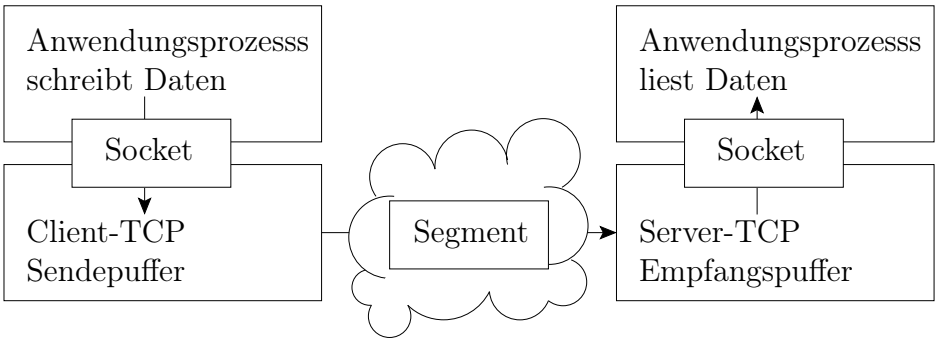

Abbildung 3.28: TCP Sende- und Empfangspuffer.

TCP nimmt eine Datenmenge aus dem Sendepuffer und fügt sie in ein neues Segment ein. Ein TCP-Segment besteht aus einem 20-Byte-Header-Feld, einem Optionsfeld und Anwendungsdaten, siehe Abbildung 3.29. Die maximale Anzahl von Bytes, die in einem Segment übertragen werden können, ist implementationsabhängig und sollte passend zur Datagrammgröße der Vermittlungsschicht (IP-Protokoll) gewählt werden. Übliche *maximale Segmentgrößen* (*Maximum Segment Size*, MSS) sind z. B. 1460 Byte, 536 Byte, oder 512 Byte – jedoch maximal 64 KB in IP Version 4.

maximale Segmentgröße

Abbildung 3.29: Struktur eines TCP-Segments. Die Sequenznummer und die Bestätigungsnummer sind 32 Bit lang. Die 16-Bit-Fenstergröße wird für die Flusskontrolle benutzt. Die Bits RST, SYN und FIN werden zur Signalisierung von Kommandos für den Verbindungsauf- und -abbau genutzt.

Das IP-Protokoll überträgt das TCP-Segment als IP-Datagramme und liefert die gesendeten Daten an das Server-TCP. Das Server-TCP speichert die empfangenen Daten im Empfangspuffer der Verbindung. Aus diesem Empfangspuffer kann der Server-Anwendungsprozess Daten lesen (im Programmierbeispiel entspricht dies der Zeile 18 im Server-Programm auf Seite 134). Da TCP bidirektionale Kommunikation unterstützt, hat jede Seite der Verbindung eigene Sende- und Empfangspuffer, die Variablen `inFromClient`, `outToClient`, `outToServer`, `inFromServer` in Abschnitt 3.3.6.

Übungsaufgabe 3.15 Die Sequenznummer eines TCP-Segments ist 32 Bit lang und deshalb durch 2^{32} begrenzt, siehe auch Abbildung 3.29. Warum können trotzdem theoretisch beliebig viele Segmente über eine TCP-Verbindung gesendet werden? `https://e.feu.de/1801-sequenznummern`

3.4.5.3 Sequenz- und Bestätigungsnummern

Da das IP-Protokoll ein unzuverlässiges Datenübertragungsprotokoll ist, muss TCP zusätzlich Datenverluste erkennen und beheben und die gleichbleibende Reihenfolge der übertragenen Daten garantieren, vergleiche die Situationen in Abbildung 3.17 auf Seite 143.

TCP realisiert einen Datenstrom von Bytes zwischen Sender und Empfänger. Jedes Byte in diesem Bytestrom wird nummeriert, diese Nummer nennen wir die Bytestromnummer. Die *Sequenznummer* eines Segments ist definiert als die Bytestromnummer des ersten Bytes im Segment. Betrachten wir das Beispiel eines Anwendungsprozesses, der über eine TCP-Verbindung eine Datei

Sequenznummern

von 10.000 Byte an den Empfängerprozess verschicken will, wobei die maximale Segmentgröße MSS 1.000 Byte Anwendungsdaten beträgt. Der Anwendungsprozesses schreibt dann die 10.000 Byte in den Socket, der mit der Verbindung assoziiert ist. Das Client-TCP konstruiert dann 10 Segmente und befüllt deren Datenfelder mit den Anwendungsdaten. Wenn das erste Byte mit i nummeriert wird ($i =$ `client_isn` $+ 1$, siehe Abschnitt 3.4.5.1), enthält das erste Segment die Bytes i bis $i + 999$, das zweite Segment die Bytes $i + 1000$ bis $i + 1999$ usw. Damit erhält das erste Segment die Sequenznummer i, das zweite Segment die Sequenznummer $i + 1000$ usw., die im Header des entsprechenden TCP-Segments eingetragen werden.

Bestätigungsnummer

TCP verwendet *Bestätigungsnummern* (Acknowledgements), um zu signalisieren, welches Byte als nächstes vom Sender erwartet wird. Wenn also das Server-TCP die ersten beiden Segmente korrekt empfangen hat und das dritte anfordern möchte, dann erwartet es als nächstes Byte dasjenige mit der Bytestromnummer $i + 2000$. Deshalb gibt das Server-TCP die Nummer $i + 2000$ als Bestätigungsnummer im nächsten Segment an, welches das Server-TCP an das Client-TCP verschickt. Falls das Server-TCP Segmente empfängt, die Daten enthalten, die erst später im Datenstrom kommen (z. B. passiert dies, wenn ein vorheriges Segment verlorengegangen ist oder später ankommen wird), dann verschickt das Server-TCP als Bestätigungsnummer trotzdem nur die Nummer des nächsten noch fehlenden Bytes im Bytestrom. TCP wendet also die kumulative Bestätigung von bis dahin korrekt empfangenen Bytes an.

kumulative Bestätigung

Was passiert jedoch mit den Daten der Segmente, die erst nach der Lücke kommen? Implementationsabhängig kann das Server-TCP diese Daten puffern und darauf warten, dass die fehlenden Daten ankommen, und dann bis zum dann fehlenden nächsten Byte bestätigen, oder aber verwerfen und damit das Client-TCP später zur Wiederholung zwingen.

Übungsaufgabe 3.16 Was passiert, wenn im Netzwerk noch verspätete Pakete einer mittlerweile schon abgebauten Verbindung ankommen? Wann könnte das ein Problem sein?

`https://e.feu.de/1801-latecomer`

3.4.5.4 Fehlerkorrektur

TCP benutzt zur Übertragung seiner Segmente den unzuverlässigen Übertragungsdienst des IP-Protokolls. Insbesondere garantiert IP nicht, dass Datensegmente unverfälscht ankommen. Deshalb fügt TCP eine eigene Prüfsumme zu jedem Segment hinzu, siehe Abbildung 3.29, und überträgt verfälschte Daten erneut, siehe Abschnitt 3.4.5.5. Der Algorithmus für die Bildung der Prüfsumme ist einfach: Sie ist genau das 1er-Komplement der 1er-Komplement-Summe aller 16-Bit-Wörter des Segments. Zur Kontrolle addiert der Empfänger alle 16-Bit-Wörter einschließlich des Headers. Wenn sich dabei die Binärzahl 1111111111111111, bestehend aus 16 gesetzten Bits, ergibt, dann ist wahrscheinlich kein Fehler bei der Übertragung aufgetreten, siehe auch Abschnitt 3.4.3.

3.4.5.5 Zuverlässiger Datentransfer in TCP

TCP realisiert einen zuverlässigen Datenübertragungsdienst, d. h. Daten werden

1. unverändert,
2. vollständig und
3. in der Reihenfolge beim Empfängerprozess abgeliefert, in der sie vom Senderprozess versendet wurden.

Um festzustellen, ob die Daten in einem TCP-Segment verändert wurden, berechnet der Empfänger die 1er-Komplement-Summe, siehe Abschnitt 3.4.3, auf dem empfangenen Segment inklusive der Prüfsumme im TCP-Segment-Header, siehe auch Abbildung 3.29. Die Prinzipien, die wir in Abschnitt 3.4.4 kennen gelernt haben, gelten auch hier. Zur Bestimmung eines Paketverlusts und des korrekten Empfangs wird bei der Übertragung ein *einziger* Timer und insbesondere die kumulative Bestätigung wie beim Sliding-Window-Protokoll verwendet, siehe Seite 153. Für die richtige Reihenfolge besitzt jedes Segment eine Sequenznummer, die genau die erste Bytenummer des Segments ist. Die Bestätigungsnummer im TCP-Header ist immer die Sequenznummer des Segments, das der Empfänger als nächstes erwartet.

Als Beispiel schauen wir ein *vereinfachtes* TCP-Senderprogramm an,

1. das keine Flusskontrolle (siehe Abschnitt 3.4.5.6) realisiert,
2. das Daten vom Anwendungsprozess erhält, die kleiner als die maximale Segmentgröße sind, und
3. das nur unidirektionalen Datentransfer vom Sender zum Empfänger durchführt.

Das Programm muss die drei folgenden Ereignisse bewältigen:

1. TCP erhält Daten vom Anwendungsprozess.
2. Ein Timer ist abgelaufen.
3. Ein ACK-Segment ist angekommen.

Programm Sender

```
initseqnum := die initiale Sequenznummer;
sendbase := initseqnum + 1;   /* die Sequenznummer, die vom
                                 Empfänger erwartet wird,
                                 siehe auch Abbildung 3.25. */
nextseqnum := sendbase;       /* die Sequenznummer des nächsten
                                 zu sendenden Pakets. */

while true do {
```

- Falls das TCP Daten vom Anwendungsprozess erhält, dann
  - Erzeuge ein TCP-Segment mit der Sequenznummer nextseqnum und den Anwendungsdaten als Inhalt;
  - Starte den Timer, falls der Timer nicht läuft;
  - Übergebe das Segment an das IP-Protokoll zur Übertragung;
  - nextseqnum := nextseqnum + Länge (Anwendungsdaten);
- Falls der Timer einen Timeout auslöst, dann
  - Wiederhole die Übertragung des noch nicht bestätigten Segments mit der kleinsten Sequenznummer;
  - Starte Timer;
- Falls eine Bestätigung (ACK) mit der Bestätigungsnummer[30] N empfangen wird, dann
  - Falls $N >$ sendbase dann /* alle Bytes bis $N-1$ wurden korrekt empfangen. */
    - sendbase := N;
    - Starte erneut den Timer, falls sendbase $<$ nextseqnum; /* es gibt noch nicht bestätigte Segmente. */
  - Falls $N <$ sendbase, dann wird die Bestätigung ignoriert;
  - Sonst /* $N =$ sendbase, der Empfänger erwartet das Segment mit der Sequenznummer sendbase. */
    - Erhöhe Anzahl der empfangenen duplizierten ACK für das Segment mit Sequenznummer N;
    - Falls Anzahl der empfangenen duplizierten ACK gleich drei ist, wiederhole die Übertragung des Segments und setze die Anzahl zurück auf 0; /* Fast Retransmit */

} end-of-while

Das Programm zeigt, wie der TCP-Senderprozess die drei Ereignisse behandelt:

1. Er empfängt Daten vom Anwendungsprozess, verpackt diese in Segmente und übergibt sie zur Übertragung an die Vermittlungsschicht. Der Timer wird gestartet, falls er momentan nicht aktiv ist. Dies kann der Fall sein, wenn beispielsweise alle bisher gesendeten Segmente schon bestätigt wurden.

2. Wenn der Timer abläuft, dann wiederholt der Senderprozess die Übertragung des unbestätigten Segments mit der kleinsten Sequenznummer.

[30] Hier steht im ACK-Feld die Sequenznummer des nächsten in der Folge vom Empfänger erwarteten Segments.

3. Wenn ein Segment mit Bestätigungsnummer N im ACK-Feld ankommt, dann prüft der Sender zuerst, ob es sich um eine Bestätigung für ein bis jetzt noch nicht bestätigtes Segment handelt. Dies ist genau dann der Fall, wenn N größer ist als die Sequenznummer des letzten vom Empfänger bestätigt Byte. Der Sender weiß dann, dass der Empfänger alle Daten korrekt empfangen hat, die vor der Sequenznummer N liegen. Deshalb kann der Sender seine Zustandsvariable sendbase, die die Sequenznummer des letzten vom Empfänger noch nicht bestätigten Bytes enthält, auf den neuen Wert N setzen.

   Falls aber vorher schon einmal eine Bestätigung mit demselben Wert N im ACK-Feld empfangen wurde (d. h. $N \leq$ sendbase), dann handelt es sich um ein sogenanntes *Duplicate ACK*. In diesem Fall ignoriert der Sender die Bestätigung, falls $N <$ sendbase ist. Falls $N =$ sendbase, weiß der Sender, dass der Empfänger das Segment mit der Sequenznummer N nicht korrekt empfangen hat, siehe auch Abbildung 3.25. Der Sender erhöht deshalb einen Zähler für die Anzahl der Duplicate ACK für das betroffene Segment. Tatsächlich entspricht ein Duplicate ACK einer negativen Bestätigung.

   Duplicate ACK

   Wenn der Sender drei Duplicate ACKs mit dem Wert $N =$ sendbase erhalten hat, dann nimmt er an, dass das Segment mit Sequenznummer N verloren gegangen ist und überträgt es erneut – auch wenn der Timer noch nicht abgelaufen sein sollte. Der Zähler für die Anzahl der Duplicate ACKs für das wieder übertragene Segment wird auf Null zurückgesetzt. Man spricht deshalb auch von einem *Fast Retransmit*.

   Fast Retransmit

Übungsaufgabe 3.17 Im TCP-Senderprogramm wird erst eine schnelle Wiederholung der Übertragung des Segments N (*Fast Retransmit*) durchgeführt, nachdem drei duplizierte ACK empfangen wurden. Warum wird nicht nach dem ersten Empfang des Duplikat-ACKs für ein Segment schon ein Fast-Retransmit ausgeführt?

`https://e.feu.de/1801-fast-retransmit`

3.4.5.6 Flusskontrolle

Das bis jetzt diskutierte TCP-Protokoll weist noch ein Problem auf. Jeder TCP-Prozess hat für jede Verbindung einen Empfangspuffer, in den die korrekt angekommenen Anwendungsdaten in der richtigen Reihenfolge eingefügt werden. Aus diesem Puffer liest der Anwendungsprozess Daten und macht dadurch wieder Platz für neu ankommende Daten. Falls aber der Senderprozess schneller Daten sendet, als sie vom Empfängerprozess gelesen werden, wird ein endlich großer Puffer nach einiger Zeit überlaufen. Dieses Problem kann häufig auftreten, da der Empfängerprozess durch andere Aktivitäten am Lesen der Daten gehindert werden kann, oder sogar gerade gar nicht aktiv sein kann (weil z. B. das Betriebssystem den Prozess gerade nicht ausführt).

Flusskontrolldienst

Um dieses Problem zu lösen, bietet TCP einen *Flusskontrolldienst* (*Flow Control Service*). Mit diesem Dienst kann der Empfänger die Senderate des Senders so verringern, dass der Empfangspuffer nicht überläuft. Dazu hat jeder

Empfangsfenster

TCP-Prozess eine Variable, die man *Empfangsfenster* (*receive window*) nennt, siehe Abbildung 3.30. Dieses Empfangsfenster zeigt an, wie viel freier Pufferplatz noch beim Empfänger vorhanden ist. Wann immer der Sender Daten sendet, zieht er den zur Speicherung notwendigen Platz vom Empfangsfenster ab. Falls kein Patz mehr da ist, sendet der Sender keine weiteren Daten. Damit dies funktioniert, teilt der Empfänger dem Sender in jedem Segment, das er an den Sender schickt, in dem WINDOW-Feld des TCP-Headers mit, wie viel freien Platz er im Puffer hat. Nun kann der Sender jeweils maximal so viele Daten senden, wie noch Pufferplatz übrig ist. Wenn der Empfängerpuffer voll ist, wartet der Sender, bis er vom Empfänger ein Segment erhält, in dem wieder neuer freier Platz gemeldet wird. Damit dies auch dann funktioniert, wenn der Empfänger gerade keine Anwendungsdaten an den Sender zu schicken hat, muss der Empfänger in diesem Fall ein leeres Segment ohne Dateninhalt an den Sender schicken, in dem er den freien Empfangspufferplatz mitteilt. Sonst wäre der Sender blockiert, und könnte trotz freien Empfangspuffers keine Daten mehr senden.

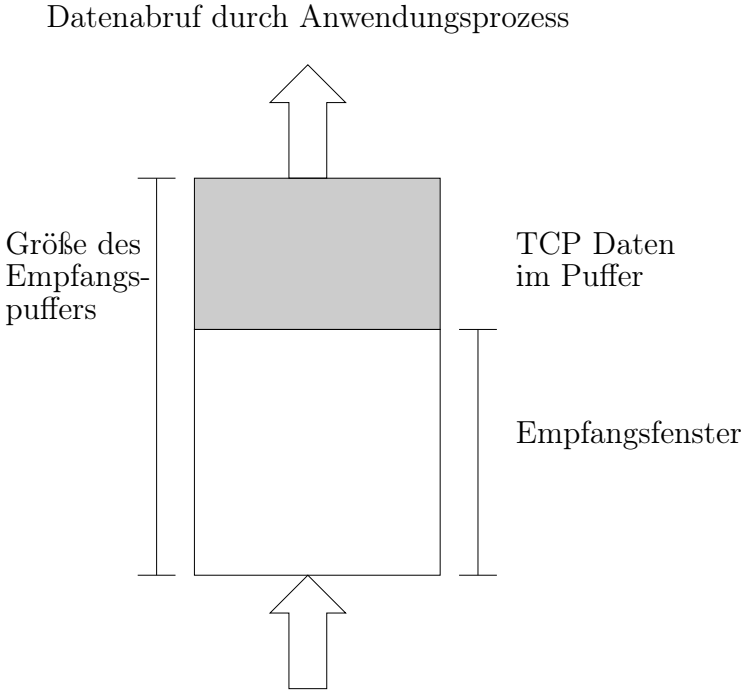

Abbildung 3.30: Empfangsfenster.

Überlastkontrolle

Es gibt noch einen weiteren Fall, in dem die Verringerung der Senderate des Senders sinnvoll ist. Falls das Netzwerk überlastet ist, macht es Sinn, die Senderate zu verringern. Dies nennt man auch *Überlastkontrolle* (*Congestion Control*). Obwohl die Lösung des Überlastproblems auch die Reduzierung der Senderate erfordert, handelt es sich um ein unterschiedliches Problem, da es

durch andere Ursachen entsteht.

3.4.6 Vergleich UDP-TCP

Die Transportsschicht des Internet bietet mit UDP und TCP zwei verschiedene Transportdienste an. TCP bietet einen zuverlässigen Datenübertragungsservice, während UDP nur einen unzuverlässigen Datenübertragungsservice bietet. Wann sollte ein Anwendungsentwickler nun welches Protokoll benutzen?

UDP weist neben der eingeschränkten Zuverlässigkeit auch eine Reihe von Vorteilen auf:

Diskussion von UDP

- UDP ist ein verbindungsloses Protokoll und kommt daher ohne Verbindungsaufbau und Verbindungsabbau aus;
- UDP speichert keinen Verbindungszustand und kommt daher mit einer einfacheren Implementierung aus. Ein Serverprozess, der mit UDP arbeitet, kann daher deutlich mehr Clients bedienen;
- UDP kommt mit einem Header von nur 8 Byte aus. Dies verringert den administrativen Aufwand gegenüber TCP, das einen 20 Byte langen Header aufweist. Pro Segment kann UDP 12 Byte mehr übertragen;
- UDP erlaubt Anwendungen so viele Daten pro Zeiteinheit zu versenden, wie sie Daten generieren (abhängig von Algorithmus, CPU etc.) und über die Netzwerkkarte in das Internet einspeisen können. Dies bedeutet jedoch nicht, dass alle diese Daten auch beim Empfänger ankommen. Insbesondere kann es aufgrund von Überlast zu Nachrichtenverlusten kommen.

Übungsaufgabe 3.18 Warum kann es bei Nutzung von UDP zu Nachrichtenverlusten kommen, wenn das Netzwerk überlastet ist?

`https://e.feu.de/1801-nachrichtenverluste-udp`

Aufgrund obiger Vorteile wird UDP z. B. von den folgenden Internetanwendungen benutzt:

Anwendungsgebiet von UDP

- Remote file server (NFS: Network File System)
- Streaming Multimedia Anwendungen, die einen schnellen Datentransfer brauchen und mit Nachrichtenverlusten leben können (z. B. Video-Übertragungen)
- Internet Telephony Anwendungen, die bei Nachrichtenverlusten eine etwas schlechtere (aber oft noch verständliche) Sprachqualität liefern können

- Network Management Anwendungen (SNMP: Simple Network Management Protocol, [13]) kontrollieren Netzwerke, die bei Überlastung des Netzes noch am ehesten UDP-Kommunikation durchführen können
- Das Routing Protocol (RIP: Routing Internet Protocol, [20], [24]) versendet periodisch neue Routing-Tabellen (siehe Kurseinheit 4), die in der Vermittlungsschicht des Internet zur Steuerung des Paketflusses eingesetzt werden. Bei Verlust einer neuen Routing-Tabelle kommt das nächste Update schnell.
- Das Domain Name System (DNS, [25, 26]) übersetzt symbolische Namen in IP-Adressen, siehe Abschnitt 3.3.4. Bei Verlust einer Anfrage wird die kurze Anfrage erneut gestellt oder an einen anderen Server gerichtet.

Diskussion von TCP

TCP realisiert einen zuverlässigen bidirektionalen Datenübertragungsdienst mit Flusskontrolle. Deshalb entlastet TCP den Anwendungsprogrammierer von der expliziten Behandlung von Bit-Fehlern, Paketverlusten, Paketvertauschungen und Pufferüberläufen im Anwendungsprogramm. Mit diesen angenehmen Leistungen sind aber auch ein paar Nachteile verbunden:

- TCP ist ein verbindungsorientiertes Protokoll. Damit einher geht eine initiale Verzögerung bei der Datenübertragung (verursacht durch das 3-Wege-Handschlag-Verfahren beim Verbindungsaufbau). Dies ist eine der Hauptursachen für die Wartezeiten beim WWW, denn bei HTTP Version 1.0 wird für jede Anforderungsnachricht eine neue Verbindung aufgebaut.
- TCP speichert den Verbindungszustand in der TCP-Implementierung auf Sender- und Empfängerseite. Diese Informationen (Puffer, Zustandsvariable) werden zur Verwaltung der Verbindung und zur Erbringung der Diensteigenschaften benötigt. Daher kann ein Server, der mit TCP arbeitet, weniger Clients bedienen als ein gleich ausgestatteter Server, der mit UDP arbeitet.
- Der TCP Header ist 20 Byte lang gegenüber 8 Byte bei UDP. TCP verbraucht also mehr Bandbreite für Verwaltungsinformation als UDP.
- TCP bietet eine Flusskontrolle, die den Senderprozess am Verschicken zu vieler Nachrichten hindert. Dies ist dann ein Nachteil, wenn eine Echtzeitanwendung zwar Paketverluste tolerieren kann, aber immer eine minimale Senderate haben muss (z. B. bei der Übertragung von Live Video).

Anwendungsgebiete von TCP

TCP wird z. B. von den folgenden Internetanwendungen benutzt:

- Electronic Mail (SMTP [31])
- Remote Terminal Access (Telnet [29])
- WWW (HTTP [15])
- File Transfer (FTP [30])

Literatur

[1] Extensible Markup Language (XML) 1.0.
http://www.w3.org/TR/REC-xml.

[2] HTML 5: A vocabulary and associated APIs for HTML and XHTML.
http://www.w3.org/TR/html5/.

[3] Informationen zum CORBA-Standard. http://www.corba.org.

[4] ISO 8824:1987, Information Processing Systems – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1). International Organization for Standardization,
https://www.iso.org/standard/16287.html.

[5] ISO/IEC 7498-1:1994, Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model. International Organization for Standardization,
https://www.iso.org/standard/20269.html.

[6] ISO/IEC 8824-1:1998, Information Technology – Abstract Syntax Notation One (ASN.1): Specification of Basic Notation. International Organization for Standardization,
https://www.iso.org/standard/32298.html.

[7] SGML: General Introductions and Overviews.
http://xml.coverpages.org/general.html.

[8] Transmission Control Protocol (TCP).
http://www.rfc-editor.org/rfc/rfc793.txt.

[9] Webseite der Object Management Group. http://www.omg.org/.

[10] ISO 8879:1986, Information Processing – Text and office systems – Standard Generalized Markup Language (SGML). International Organization for Standardization,
https://www.iso.org/standard/16387.html, 1986.

[11] T. Berners-Lee, R. Fielding, H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. http://www.rfc-editor.org/rfc/rfc1945.txt.

[12] R. Braden. Requirements for Internet Hosts: Communication Layers.
http://www.rfc-editor.org/rfc/rfc1122.txt, 1989.

[13] J. Case, M. Fedor, M. Schoffstall, J. Davin. Simple network management protocol. http://www.rfc-editor.org/rfc/rfc1157.txt.

[14] M. Crispin. Internet Message Access Protocol (IMAP): Version 4rev1. http://www.rfc-editor.org/rfc/rfc2060.txt, 1996.

[15] R. Fieldings, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. http://www.rfc-editor.org/rfc/rfc2616.txt.

[16] H. C. Folts. *X.25 and Related Protocols*. McGraw-Hill, 1984.

[17] N. Freed, N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Body. http://www.rfc-editor.org/rfc/rfc2045.txt, 1996.

[18] N. Freed, N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. http://www.rfc-editor.org/rfc/rfc2046.txt, 1996.

[19] W. Goralski. *Frame Relay for High-Speed Networks*. John Wiley, New York, 1999.

[20] C. Hedrick. Routing Information Protocol (RIP). http://www.rfc-editor.org/rfc/rfc1058.txt, 1988.

[21] B. Kantor, P. Lapsley. Network News Transfer Protocol (NNTP), a Proposed Standard for the Stream-Based Transmission of News. http://www.rfc-editor.org/rfc/rfc977.txt, 1986.

[22] J. F. Kurose, K. W. Ross. *Computernetze: Ein Top-Down-Ansatz*. Pearson Studium, 6. Auflage, 2014.

[23] J. F. Kurose, K. W. Ross. *Computer Networking: A Top-Down Approach*. Pearson International Edition, seventh edition, 2017.

[24] G. Malkin. RIP Version 2, Carrying Additional Information. http://www.rfc-editor.org/rfc/rfc1723.txt, 1994.

[25] P. Mockapetris. Domain Names: Concepts and Facilities. http://www.rfc-editor.org/rfc/rfc1034.txt, 1987.

[26] P. Mockapetris. Domain Names: Implementations and Specification. http://www.rfc-editor.org/rfc/rfc1035.txt, 1987.

[27] J. Myers, M. Rose. Post Office Protocol Version 3 (POP3). http://www.rfc-editor.org/rfc/rfc1939.txt.

[28] J. Postel. User Datagram Protocol (UDP). http://www.rfc-editor.org/rfc/rfc768.txt, August 1980.

[29] J. Postel, J. Reynolds. Telnet Protocol Specification. http://www.rfc-editor.org/rfc/rfc854.txt, May 1983.

[30] J. Postel, J. Reynolds. File Transfer Protocol (FTP). http://www.rfc-editor.org/rfc/rfc959.txt, October 1985.

[31] J. B. Postel. SMTP: Simple Mail Transfer Protocol. http://www.rfc-editor.org/rfc/rfc821.txt, August 1982.

[32] J. Reynolds, J. Postel. Assigned numbers. http://www.rfc-editor.org/rfc/rfc1700.txt.

[33] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler. SIP: Session initiation protocol. http://www.rfc-editor.org/rfc/rfc3261.txt, June 2002.

[34] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. http://www.rfc-editor.org/rfc/rfc3550.txt, July 2003.

[35] W. Simpson. The Point-to-Point Protocol (PPP). http://www.rfc-editor.org/rfc/rfc1661.txt, July 1994.

[36] J. D. Spragins. *Telecommunications Protocols and Design.* Addison-Wesley, Reading, MA, 1991.

[37] C. Spurgeon, J. Zimmerman. *Ethernet: The Definitive Guide.* O'Reilly Media, 2nd edition, 2014.

[38] W. R. Stevens. *TCP/IP Illustrated, Band 1: The Protocols.* Addison-Wesley, Reading, MA, 1994.

[39] A. S. Tanenbaum. *Computernetzwerke.* Pearson Studium, 5., aktualisierte Auflage, 2012.