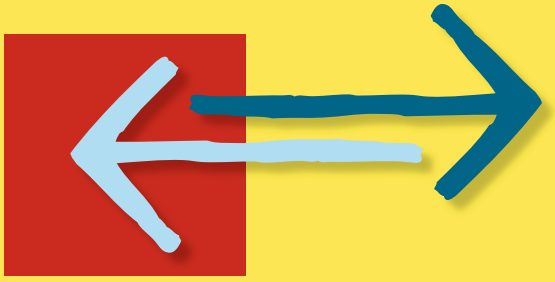
peter leo **GORSKI**
luigi **LO IACONO**
hoai viet **NGUYEN**



# WebSockets

## Moderne **HTML5-Echtzeit-anwendungen** entwickeln

HANSER

Gorski/Lo Iacono/Nguyen

**WebSockets**

Peter Leo Gorski
Luigi Lo Iacono
Hoai Viet Nguyen

# WebSockets

## Moderne HTML5-Echtzeitanwendungen entwickeln

HANSER

Die Autoren:

*Peter Leo Gorski,* Köln
*Prof. Dr.-Ing. Luigi Lo Iacono,* Bonn
*Hoai Viet Nguyen,* Köln



Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.


Lektorat: Sieglinde Schärl
Herstellung: Irene Weilhart
Copy editing: Sandra Gottmann, Münster
Layout: Peter Leo Gorski mit LaTeX
Umschlagdesign: Marc Müller-Bremer, www.rebranding.de, München
Umschlagrealisation: Stephan Rönigk
Druck und Bindung: Kösel, Krugzell
Ausstattung patentrechtlich geschützt. Kösel FD 351, Patent-Nr. 0748702
Printed in Germany

Print-ISBN: 978-3-446-44371-6
E-Book-ISBN: 978-3-446-44438-6

**»Der Weltuntergang steht bevor,
aber nicht so, wie Sie denken.
Dieser Krieg jagt nicht alles in die Luft,
sondern schaltet alles ab.«**


**Tom DeMarco**
**Als auf der Welt das Licht ausging**

ca. 560 Seiten. Hardcover
ca. € 19,99 [D] / € 20,60 [A] / sFr 28,90
ISBN 978-3-446-43960-3
Erscheint im November 2014

**Hier klicken zur Leseprobe**

**Sie möchten mehr über Tom DeMarco und seine Bücher erfahren.
Einfach reinklicken unter www.hanser-fachbuch.de/special/demarco**

# Inhalt

# Vorwort

Netzwerke umgeben uns quasi überall. Viele Dinge des täglichen Geschäfts- und Privatlebens sind ohne Kommunikation über ein digitales Datennetz nicht mehr vorstellbar. Zur Lingua franca der Protokolle der Anwendungsschicht hat sich in den letzten Jahren das HTTP gemausert. Angeschoben durch den immensen Erfolg des Web bedienen sich heute eine Vielzahl anderer Anwendungen dem HTTP. Darunter z.B. die Lokalisierung und Ansteuerung von Geräten im Hausnetz via UPnP, die Speicherung von Daten in der Cloud alla Dropbox und vermehrt auch das TV. Die Gründe hierfür sind vielfältig. Der simple Aufbau von HTTP und die zahlreich verfügbaren Implementierungen spielen dabei sicherlich eine Hauptrolle. In den Nebenrollen finden sich Darsteller wie die durchgängige Firewall-Freundlichkeit und neuerlich auch die ubiquitäre Verbreitung von HTTP in Geräten jeden Couleurs. Ist man mit seinen Anwendungen und Diensten an Reichweite interessiert, führt derzeit kein Weg an HTTP vorbei.

Neben all dieser Euphorie ist es ratsam, alle Eigenschaften des neuen Gefährten zu kennen, da sich daraus auch seine Grenzen ableiten lassen. An der Zustandslosigkeit von HTTP lässt sich dies schön verdeutlichen. Erlaubt es auf der einen Seite ein vereinfachtes Caching und globale Skalierbarkeit durch Content Distribution Networks (CDN), erschwert es die Implementierung von Warenkörben in E-Commerce Anwendungen. Letzteres hat man im Laufe der Zeit gut in den Griff bekommen. Für andere Nachteile haben sich dagegen noch keine Patentrezepte etabliert. Hierzu zählt z.B. die unidirektional ausgelegte Kommunikation. Gemäß der Protokollspezifikation meldet sich immer der Client mit einer Anfrage an den Server, der auf diese mit einer entsprechenden Antwort reagiert. Dass sich der Server selbstinitiativ bei einem Client meldet, ist nicht vorgesehen. Dies liegt auch darin begründet, dass die Verbindung zwischen Client und Server nicht dauerhaft, sondern nur für die Zeit des Austausches von Anfragen und ihrer Antworten besteht. In modernen Anwendungen ist das Pushen von Nachrichten oder Daten vom Server zum Client eine häufige Anforderung, die es technisch umzusetzen gilt. Genau an dieser Stelle setzt die WebSocket-Technologie an.

Dem Wildwuchs an Ansätzen, die zum Aufbrechen der Kommunikationseinbahnstraße von HTTP vorgeschlagen wurden und uneinheitlich zur Verwendung gekommen sind, wird mit WebSockets eine standardisierte Lösung entgegengestellt. Der Hunger nach einem derartigen Standard war groß. Dies lässt sich daran ablesen, dass die Unterstützung von WebSockets durch alle einschlägigen Industriegrößen bekannt gegeben wurde, als sich die Standardisierung noch in den Kinderschuhen befand. Bereits heute lässt sich kein Webbrowser vermissen, der nicht WebSockets mit an Bord hat. Obwohl die Standardisierung

zum aktuellen Zeitpunkt noch nicht abgeschlossen ist, kann davon ausgegangen werden, dass es sich hierbei in nicht allzu langer Zeit um einen etablierten Standard handeln wird.

Das vorliegende Buch stellt diesen neuen Grundpfeiler für moderne Anwendungen auf Grundlage von HTTP vor. Dabei geht es auf alle Aspekte der WebSocket-Technologie in angemessener Tiefe ein. Meine vorweggenommene Motivation wird zu Beginn des Buches, angereichert mit dem notwendigen Background, aufgegriffen und somit auf das eigentliche Thema hingeführt. Dann stehen die WebSockets im Rampenlicht. Mit dem unter den Fittichen der IETF stehendem WebSocket-Protokoll geht es los. Hier wird Bit genau beleuchtet, wie sich die verschiedenen Frames zwischen Client und Server bewegen. Das zeigt unter anderem, wie viel effektiver die Übertragung mittels WebSockets im Vergleich zu HTTP ist, wenn der Overhead der HTTP-Header wegfällt. Im Anschluss lernt man die von der W3C verantwortete JavaScript API kennen, mit der sich die Clientseite einer verteilten WebSocket-Anwendung in einem Browser implementieren lässt. Das ist allerdings noch nicht alles, was für ein ganzheitliches Projekt benötigt wird. Die Serverseite fehlt und hier sieht das Bild nicht ganz so einheitlich aus. Fehlende Standards machen es erforderlich, sich die vom ausgewählten Framework entsprechend bereitgestellten APIs anzueignen. Diesem Umstand Rechnung tragend, werden in dem Buch verschiedene serverseitige Frameworks behandelt, die für Java und JavaScript bereitstehen. Abgerundet wird das Ganze durch zahlreiche Beispiele, die auch größere Zusammenhänge nachvollziehbar und verständlich machen.

Mir hat die Lektüre das notwendige Know-how und das Verständnis in die neuen Möglichkeiten vermittelt, mit dem ich nun spannenden neuen Projekten unter Verwendung von WebSockets entgegenblicke. Ich wünsche Ihnen ebenso viel Erkenntnisgewinn beim Lesen, Ihr

*Alexander Leschinsky* Köln, im November 2014
Geschäftsführer der G&L Systemhaus GmbH

# Danksagung

Die Seiten eines Buches füllen sich maßgeblich durch die Federn der Autoren. Einfluss auf die Formulierungen und Aufbereitungen der produzierten Inhalte haben allerdings eine Vielzahl von weiteren Personen. Auch an diesem Werk haben viele gute Geister konstruktiv mitgewirkt, denen wir hiermit ein herzliches Dankeschön aussprechen möchten (Nennungen in alphabetischer Reichenfolge): Aline Jaritz, Carsten Möhrke, Daniel Torkian, Stephan Mattescheck und Sven Wagner

Ein Dankeschön geht außerdem an die Fachhochschule Köln, an den Carl Hanser Verlag, an unsere Lektorin Sieglinde Schärl, für das reibungslose Management, an Irene Weilhart, für die Hilfe beim Buchsatz und selbstverständlich an Sandra Gottmann, die eine Fülle von Fehlern entdeckt hat, die uns beim Schreiben durch die Finger gerutscht sind.

Möchte man eine Webanwendungen oder gar einen Webservice entwickeln, kommt man unweigerlich mit einer Vielzahl von Web-Technologien, Frameworks und anderen Artefakten in Kontakt. Um dem Leser die WebSocket-Technologie näherzubringen, nehmen auch wir daher Bezug auf viele Zeilen Code und Software, die wir nicht immer selbst entwickelt haben. Deshalb möchten wir den Organisationen, Unternehmen und Entwicklern, einen besonderen Dank aussprechen, insbesondere denen, die Ihre Entwicklungen jedem zur freien Verfügung stellen, um etwas daraus lernen und um gemeinsam Technologien voranbringen zu können (erneut in alphabetischer Reihenfolge): Agendaless Consulting and Contributors (supervisor), Alexis Deveria (caniuse.com), Amazon.com Inc., Apple Inc., Automattic (Socket.io), Brian McKelvey (WebSocket-Node), Canonical Foundation, Eclipse Foundation, Eric Lawrence (Fiddler), Firebug Working Group, GNU/Linux, Google Inc., IETF, ISO, Joyent Inc. (Node.js), Kaazing Corporation (WebSocket.org), LearnBoost (WebSocket.io), Microsoft Corporation, Mozilla Corporation, Opera Software, Oracle Corporation, Patrick Wied (heatmap.js), Play Framework, strongloop (Express.js), Tim Fox & VMWare & Red Hat (vert.x), Twitter Inc. (Bootstrap), W3C, Wireshark-Community und alle, die noch im Buch erwähnt werden und die wir versehentlich an dieser Stelle vergessen haben.

Ein Teil der Grafiken, die wir für dieses Buch angefertigt haben, sind mithilfe von lizenzfreien Cliparts der Internetseite *www.clker.com/* angefertigt worden. Auch dafür sind wir sehr dankbar.

Die Fertigstellung eines Buchs benötigt vor allem viel Zeit, Geduld, Fleiß und Ausdauer. Auch das wäre für uns ohne die Unterstützung der Menschen, die uns am nächsten stehen, nicht möglich gewesen. Vielen Dank an Anh Trung, Antonio, Ba Tho, Barbara, Em Yen, Fabio, Giuliana, Inge, Marco, Me Chau, Peter und Tati.

An letzter Stelle möchten wir natürlich auch Ihnen für den Kauf dieses Buches und Ihr Interesse an den WebSockets danken!

# 1 Einleitung

Einer der technischen Hauptakteure im Web ist HTTP, das *Hypertext Transfer Protocol* [FGM+99], und das ist so, weil HTTP den genauen Kommunikationsablauf zwischen Webbrowser und Webserver festlegt. Genau wie ein Protokoll für eine königliche Gala zu Hofe alle Regeln in Bezug auf Kleidung, Etikette und Umgangsformen zusammenfasst, so legt ein Kommunikationsprotokoll alle Regeln zum Nachrichtenaustausch fest. Anders als bei Hofe sind das bei einem Kommunikationsprotokoll aber vielmehr Regeln in Bezug auf den Aufbau, das Format und die Codierung der ausgetauschten Nachrichten. Außerdem muss in einem Protokoll festgelegt sein, wer die Kommunikation beginnt und wie diese dann Nachricht für Nachricht bis zum Kommunikationsende abläuft. HTTP verwendet hier einen einfachen Request-Response-Nachrichtenaustausch. Dieser wird immer vom Webclient durch eine Anfrage (engl. *Request*) initiiert und entsprechend vom Webserver mit einer Antwort (engl. *Response*) beantwortet. Sobald Sie also z. B. eine Webadresse – die URL – in die Adresszeile eines Webbrowsers eingeben oder einen Bookmark bzw. Link anklicken, setzen Sie damit eine Anfrage an einen Server ab, der diese dann bearbeitet und die Antwort daraufhin zurücksendet (siehe Bild 1.1).

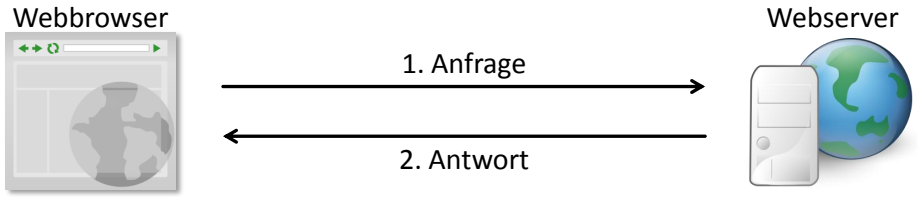



**Bild 1.1** Grundlegender HTTP-Nachrichtenaustausch

In diesem einfachen Request-Response-Nachrichtenaustausch liegen viele Gründe für den großen Erfolg von HTTP und nicht zuletzt des Webs. Dazu zählen u. a. die guten Skalierungseigenschaften, denen wir die Größe des Webs verdanken. Wie Sie sich aber sicher jetzt schon denken, bringt das Nachrichtenpaar nicht nur Vorteile mit sich. Und tatsächlich ist die Protokollregel, dass der Request immer vom Client ausgehen muss, eine Zwangsjacke, die bestimmte Bewegungsfreiheiten in Form von Kommunikationsmustern nicht zulässt. Der Teil einer Webanwendung, der auf dem Server ausgeführt wird, kann sich nämlich nicht von sich aus an einen Client wenden. Diese Eigenschaft des Protokolls schränkt folglich Anwendungsfälle ein, in denen die Serverseite Statusänderungen an die betroffenen Clients weiterreichen können muss. Typische Beispiele hierfür sind Chatanwendun-

gen, Finanzdatenströme, Webmail, Sportticker, das Monitoring von Haushalt und Industrie oder auch Internetauktionen. All diese Anwendungen teilen die Gemeinsamkeit, dass (häufig in relativ kurzen Zeitabschnitten) Daten auf dem Server hinzukommen bzw. sich ändern und dies unmittelbar an die angeschlossenen Clients bekannt gegeben werden muss. Liegt Ihnen eine derartige Anforderung auf dem Tisch, dann würden Sie sich eine standardisierte Technik wünschen, mit der Sie die Schranken der Serverseite öffnen und die entsprechenden Clients ansprechen können.

HTTP bietet für derartige Kommunikationsmuster keine native Unterstützung. Wenn Sie sich in den letzten Jahren aus dieser Zwangsjacke hätten befreien und die Kommunikationseinbahnstraße zur Umsetzung von Serverbenachrichtigungen hätten aufbrechen wollen, so wären Sie auf Grundlage des Verfügbaren auf eine (selbst) konstruierte Lösung angewiesen gewesen. Die Konstruktionen, die findige Webentwickler dabei gefunden haben, sind vielfältig. Prinzipiell können Sie den Standardfunktionsumfang durch Browser-Plug-ins und serverseitige Erweiterungen aufbohren und damit proprietäre Lösungen zur Serverbenachrichtigung implementieren. Der erste Ansatz überhaupt kam tatsächlich auch aus dieser Technologiegattung und bediente sich Java Applets und der LiveConnect-Schnittstelle zur Anbindung an JavaScript-Funktionen im Browser. Seit HTML5 besteht allerdings ein starker Trend weg von Browsererweiterungen à la Plug-in oder Add-on. Schützenhilfe kam zudem aus dem Umfeld der Smartphones und Tablets, das sich gegen eine Unterstützung von Flash und Silverlight entschieden hat, was schließlich selbst Adobe und Microsoft dazu bewegt hat, die Entwicklungen ihrer Technologien für mobile Plattformen einzustellen. Daher wollen wir auch nicht weiter auf das Auslaufmodell der Plug-in-basierten Ansätze eingehen. Weitere Konstruktionen basieren auf Ansätzen, die den Webbrowser in regelmäßigen Zeitabständen beim Webserver nach Veränderungen fragen lassen. Diese aktuell weit verbreiteten Verfahren haben Nachteile, was das Kommunikationsaufkommen anbelangt. Dennoch haben sie unter Umständen ihre Berechtigung. Sie lernen diese Ansätze daher kennen und insbesondere diese zu bewerten, um die richtige Technologie für Ihr Entwicklungsvorhaben auswählen zu können. Nichtsdestotrotz handelt es sich aber weitestgehend um Notlösungen, die durch die in der HTML5-Standardisierung befindlichen alternativen Lösungswege abgelöst werden. Dazu gehören zum einen *Server-Sent Events* (SSE) [Hic12a] und zum anderen *WebSockets* (WS) [FM11a]. Die verschiedenen Bestandteile des Standards, die HTML5 einführt, sind thematisch gruppiert und mit einem Logo versehen worden [W3C14]. SSE und WS sind in der sogenannten Connectivity-Gruppe. Das Logo sehen Sie in einer leicht abgewandelten Form in Bild 1.2.

**Bild 1.2** Das Logo für die Connectivity-Standards Server-Sent Events und WebSockets, platziert auf dem HTML5-Schild [W3C14]

WebSockets stehen im Fokus dieses Buches, da Sie damit neben den Serverbenachrichtigungen eine Vielzahl weiterer Anwendungsfälle realisieren können, zu denen z. B. Spiele, jegliche Art von Konversationen (Text, Sprache, Sprache mit zusätzlichem Video), Kollaborationen sowie das Benutzermonitoring im Rahmen von Usabilitystudien zählen.

## 1.1 Wie ist das Buch strukturiert?

WebSockets und die damit einhergehenden neuen Entwicklungsmöglichkeiten stehen somit im Zentrum des vorliegenden Buches und an diese wollen wir uns mit Ihnen Schritt für Schritt ranpirschen. In Kapitel 2 führen wir uns dazu zunächst die notwendigen Grundlagen in Bezug auf HTTP zu Gemüte. Kennen Sie diese schon, können Sie getrost die Abkürzung zu Kapitel 3 nehmen, in dem wir uns mit den Mechanismen befassen, mit denen eine höhere Interaktivität und Echtzeitfähigkeit bei Webanwendungen erreicht werden kann. In Kapitel 4 widmen wir uns dann dem WebSocket-Protokoll und in Kapitel 5 geht es anschließend ans Eingemachte. Hier lernen wir die WebSocket API kennen und programmieren mit JavaScript erste Beispiele für WebSocket-Clientanwendungen in Webbrowsern. In Kapitel 6 wenden wir uns den WebSocket Implementierungen auf der Serverseite sowie gebräuchlichen Frameworks zu. Weitere wichtige Aspekte folgen in Kapitel 7, das das Testen von verteilten WebSocket-basierten Applikationen beschreibt und auf Eigenschaften der Performance eingeht. Was niemals vergessen werden darf, sind Sicherheitsaspekte, insbesondere wenn die Anwendung aus verteilten Komponenten zusammengesetzt ist, die über offene Netze miteinander gekoppelt sind, wie das im Web quasi immer der Fall ist. Daher wollen wir uns damit ebenfalls in Kapitel 7 auseinandersetzen und die Besonderheiten von WebSockets in diesem Kontext herausstellen. In Kapitel 8 werden wir dann das Gelernte einsetzen und verschiedene „größere“ und vollständige Anwendungen implementieren. Diese haben wir dabei so gewählt, dass damit ein möglichst großes Spektrum an Möglichkeiten aufgezeigt wird. Wir spannen den Bogen von einer generischen Fernsteuerung für Webanwendungen, über ein klassisches Chatsystem über eine Heatmap für Usability-Tests bis hin zu einer Überwachungskamera per Webcam.

## 1.2 Wer sollte das Buch lesen?

In erster Linie richtet sich das vorliegende Buch an den Entwickler in Ihnen. Möchten Sie mehr über WebSockets wissen und verstehen, was Sie mit diesen so alles in Ihren Webprojekten anstellen können, dann sollten Sie hier fündig werden. Da der Koffer an Werkzeugen, Technologien und Sprachen kaum unterschiedlicher gefüllt ist als bei Webentwicklern, stellen wir auf keine spezifische Umgebung ab, sondern versuchen dieser farbenfrohen Vielfalt gerecht zu werden. So finden sich Beispiele auf Basis von Node.js, vertx und JSR 356 im Buch wieder. Als Programmiersprachen verwenden wir Java und JavaScript. Neben der gängigen Vorstellung und Erläuterung der Buchinhalte sind die zahlreichen Beispielanwendungen in Kapitel 8 eine Stärke des vorliegenden Buches, mit denen das Erlernte geübt und nachvollzogen werden kann. Webentwickler sollten somit in die Lage versetzt werden,

einen umfassenden Einstieg in die Thematik zu bekommen und anhand der exemplarischen Anwendungen einen tief gehenden Einblick in den Einsatz von WebSockets zu erhalten.

Neben Webentwicklern eignet sich das Buch auch für Lehrveranstaltungen an Hochschulen und damit für Studierende verschiedener Fachrichtungen, die mit WebSockets innovative Anwendungen im Web entwickeln wollen. Zudem profitieren Informations- und Softwarearchitekten sowie Konzepter und (technische) Projektleiter vom vorliegenden Buch, da ein Grundverständnis der Kommunikationsmuster, die mit WebSockets realisiert werden können, für Architekturen und Konzepte von Bedeutung sein können. Auch hierfür haben wir versucht, mit den in Kapitel 8 beschriebenen Beispielanwendungen den Horizont aufzuzeichnen.

## 1.3 Wie sollte mit dem Buch gearbeitet werden?

Das Buch führt sukzessive in die Thematik ein und setzt nur wenige Vorkenntnisse voraus. Wenn Sie über kaum bis wenige Erfahrungen in der Webentwicklung verfügen, raten wir Ihnen, sich an diesem Aufbau zu orientieren und sich Kapitel für Kapitel einzuarbeiten. Nach den Grundlagen werden in den einführenden Kapiteln erste kleinere Beispielanwendungen gezeigt und erläutert. Wir empfehlen, diese nachzuprogrammieren und die Quelltexte im Detail nachzuvollziehen. Ist das Fundament gelegt, geht es mit weiterführenden Aspekten weiter, die selektiv durchgearbeitet werden können. Das Durcharbeiten der Beispielanwendungen ist wiederum sehr empfehlenswert. Es trainiert auf der einen Seite die im Buch vermittelten Inhalte. Bei Bedarf können Unklarheiten nochmals dediziert nachgelesen werden. Auf der anderen Seite zeigen sie die Potenziale von WebSockets auf. Leser mit fundiertem Vorwissen können sicherlich die Grundlagenkapitel überspringen und sich direkt mit den originären Inhalten und den Beispielanwendungen befassen.

**http://websocket101.org/**

Wir freuen uns von Ihnen zu hören! Ob Lob oder Kritik, ob Frage oder Anregung, ob Hinweis oder Verbesserung, wir glauben, dass das Buch von all dem profitieren kann, wenn es konstruktiv eingebracht wird. Dazu wollen wir unser Nötigstes tun und freuen uns auf Ihr Mitwirken. Aktuelles wird dabei zeitnah auf der buchbegleitenden Webseite bereitgestellt. Schauen Sie doch häufiger mal auf

*http://websocket101.org/*

vorbei. Dort finden Sie auch unsere Kontaktinformationen.

An dieser Stelle bleibt uns nur noch, Ihnen viel Spaß beim Lesen und Nachprogrammieren sowie viel Erfolg bei den eigenen Entwicklungen zu wünschen.

# 2 HTTP: Hypertext Transfer Protocol

Wenn Sie gefragt werden, was das Web ist, so können Sie darauf viele verschiedene Antworten finden. Aus technischer Sicht lässt sich die Antwort allerdings auf ein Kommunikationsprotokoll zurückführen. Es wird Hypertext Transfer Protocol (HTTP) [FGM+99] genannt.

**Fragen, die dieses Kapitel beantwortet:**

- Welche wichtigen Kommunikationsgrundlagen werden zum besseren Verständnis von HTTP benötigt?
- Was sind die wesentlichen Details der Kommunikation im Web und insbesondere von HTTP?
- Welche Einschränkungen hat HTTP in Bezug auf Realzeit-Kommunikation?
- Was für Ad-hoc-Lösungen sind aus diesen Einschränkungen hervorgegangen?

## 2.1 Grundlegendes

Wenn Ihnen Begriffe wie Protokoll, Schicht und Dienst im Kommunikationskontext noch nicht geläufig sind, dann sollten Sie diesem Kapitel aufmerksam folgen. Hier möchten wir Ihnen die grundlegenden Prinzipien und Konzepte moderner Kommunikationssysteme erläutern, da diese zum Verständnis von HTTP und später von WebSockets notwendig sind. Dazu möchten wir Ihnen zunächst das ISO/OSI-Referenzmodell näherbringen, welches das formale Modell aller gegenwärtigen Kommunikationssysteme darstellt. In den darauffolgenden Unterkapiteln zeigen wir Ihnen dann, wie das ISO/OSI-Modell im Internet angewendet wird und wo welche Protokolle was für Dienste realisieren, um schließlich den Austausch von Nachrichten im Web zu ermöglichen. Wenn wir das geschafft haben, steigen wir Schritt für Schritt tiefer in die Details von HTTP ein und leiten den Übergang in die Welt der WebSockets ein, indem wir Möglichkeiten und Einschränkungen der realzeitorientierten Kommunikation im Web diskutieren.

### 2.1.1 ISO/OSI-Referenzmodell für Kommunikationssysteme

Häufig sind „alte“ Standards in der noch jungen Informations- und Kommunikationstechnologie ein Merkmal für deren Relevanz und Bedeutung. Ein herausragendes Beispiel hierfür ist das von der Standardisierungsorganisation ISO (International Organization for Standardization) bereits 1984 als internationaler Standard verabschiedete OSI-Referenzmodell (Open Systems Interconnection) [ISO94].

Ein Referenzmodell beschreibt ganz allgemein die logische Gliederung von Systemen. Dabei müssen Sie durch Abstraktion die Komponenten eines Systems identifizieren und deren Abhängigkeiten zueinander bestimmen. Und mehr noch, ein Referenzmodell versucht dabei die Gemeinsamkeiten und Unterschiede der konkreten Instanziierungen des Modells zu berücksichtigen. Wenn Sie sich mal überlegen, was Sie für Gemeinsamkeiten und Unterschiede in Kommunikationsnetzen zur Telefonie, zu dem Rundfunk oder dem Internet finden und wie Sie diese abstrakten Komponenten in Beziehung bringen können, sind Sie vermutlich schon beim OSI-Modell.

Das OSI-Modell definiert zur Abstraktion von Kommunikationssystemen sieben Schichten, die alle bestimmte festgelegte Aufgaben erledigen (siehe Bild 2.1).

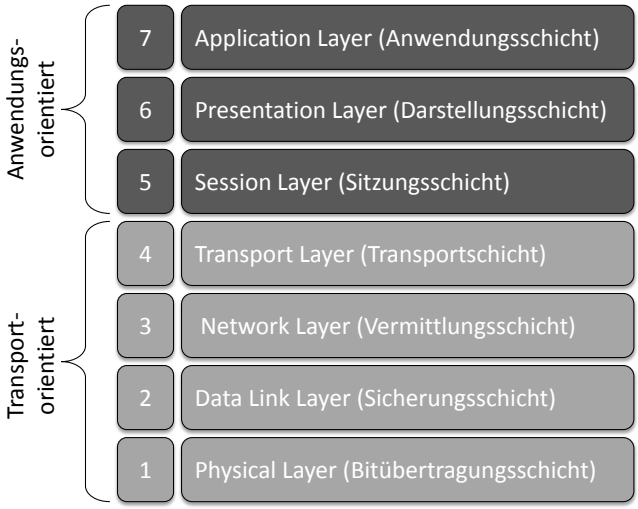



**Bild 2.1** ISO/OSI-Referenzmodell [ISO94]

Starten wir von unten. In der ersten Schicht oder in **Schicht 1** finden Sie alle diejenigen Spezifikationen und Funktionen, die für die physikalische Übertragung der Bits benötigt werden. Daraus ergibt sich auch der Name der Schicht, der im Deutschen passend als Bitübertragungsschicht übersetzt ist. Konkrete Ausprägungen dieser Schicht müssen Definitionen und Festlegungen machen, die von Spezifikationen von Kabeln und Steckern bis hin zu Modulationsverfahren reichen. Diese Schicht ist in der Netzwerkkopplungshardware (z. B. der Netzwerkkarte) implementiert.

Die **Schicht 2** hat die maßgebliche Aufgabe, dafür Sorge zu tragen, dass die zu übertragenden Bits fehlerfrei beim Empfänger ankommen. Sie müssen davon ausgehen, dass bei der Übertragung der Bits in vielfältiger Form Verfälschungen auftreten können. Diese können

in Abhängigkeit des konkreten Übertragungsmediums und -verfahrens sehr unterschiedlich sein. Diese Schicht soll übertragungsbedingte Verfälschungen erkennen und ggf. beheben. Eine weitere wichtige Funktion, die Schicht 2 innehat, ist die Steuerung von (zeitgleichen) Zugriffen auf das gemeinsam genutzte physikalische Übertragungsmedium. Hierzu wird je nach Ausprägung eine Identifikation der Netzwerkkarte benötigt, die als MAC-Adresse bekannt ist. MAC steht hier für Medium Access Control. Als Softwareentwickler kommen Sie in der Regel, wie auch schon für Schicht 1, mit dieser Schicht nicht direkt in Berührung, da diese ebenfalls in der Netzwerkkartenhardware implementiert ist.

Die Vermittlungsschicht ist die **Schicht 3**. Sie können sich vorstellen, dass diese Schicht die Verbindung zum Kommunikationspartner herstellt. Dies kann erneut verschiedene Ausprägungen annehmen. Ist das umgebende Kommunikationssystem ein leitungsvermittelndes Netz, wie es z. B. bei der analogen Festnetztelefonie der Fall ist, dann wird die Verbindung durch Schalten von Leitungen hergestellt. Handelt es sich um ein paketvermittelndes Netz, wie es z. B. das Internet ist, dann wird die Verbindung durch Weiterleiten der Pakete an das Zielsystem hergestellt. In beiden Fällen muss der Weg vom Sender zum Empfänger gesucht werden, was als Routing bezeichnet wird. Um einen Weg finden zu können, muss in Schicht 3 ein netzwerkweites Adressierungsschema vorliegen, womit die Kommunikationsendpunkte eindeutig identifiziert werden können. In den Beispielen, die wir gerade erwähnt haben, wären das die Telefonnummern bzw. IP-Adressen.

Die als Transportschicht benannte **Schicht 4** hat die Aufgabe, die Kommunikation von Endpunkt zu Endpunkt zu kontrollieren und zu steuern. Zu den typischen Dingen, die in diesen Aufgabenbereich fallen, zählen u. a., dass der Sender den Empfänger mit den gesendeten Daten nicht überfrachtet und dass die anwendungsorientierten Schichten 5 bis 7 einen einheitlichen Zugang zum Netzwerk bekommen, unabhängig von den tatsächlichen Gegebenheiten. Bei Ausprägungen der Transportschicht, die einen verlässlichen Transport gewährleisten, werden zudem die Reihenfolge, Vollständigkeit und Fehlerfreiheit der Nachrichten bzw. Nachrichtenpakete sichergestellt. Um auch mehrere verschiedene Dienste auf einem Endsystem für das Netzwerk anbieten zu können, muss auf dieser Ebene ein zusätzlicher Adressierungsmechanismus vorliegen. In der Internet-Protokollfamilie sind das die nummerischen Ports. Für viele Standarddienste sind Default-Ports festgelegt worden [TLM+14]. Für das unverschlüsselte HTTP ist das der Port 80 und für die SSL-/TLS-gesicherte Variante entsprechend der Port 443.

**Schicht 5** ist mit der Etablierung und Verwaltung von Sitzungen befasst. Sitzungen sollen durch entsprechende Maßnahmen gewährleisten, dass im Falle einer unterbrochenen Kommunikation und insbesondere eines nicht vollständig durchgeführten Datenaustausches dieser zu einem späteren Zeitpunkt an der unterbrochenen Stelle wieder aufgenommen werden kann.

Um zu gewährleisten, dass unterschiedliche Systeme mit den ausgetauschten Daten umgehen können, stellt **Schicht 6** Funktionen bereit, die dafür sorgen, dass die verwendeten Codierungen für die an der Kommunikation beteiligten Partner bekannt und verständlich sind. Sollte es nicht möglich sein, sich auf einen gemeinsamen Codierungsnenner zu verständigen, verfügt die sogenannte Darstellungsschicht zur Not über Konvertierungsroutinen, mit denen eine Anpassung an die spezifischen Gegebenheiten erfolgen kann.

Die Zuckerschicht des Schichtenkuchens stellt die Anwendungsschicht dar. Auf der 7. Ebene des OSI-Turms thronend, fasst **Schicht 7** alle Protokolle zusammen, die die unterschiedlichen Anwendungen benötigen. Hierzu zählen z. B. Protokolle zum Austausch von E-Mails

oder Dateien und für den Zugriff auf entfernte Systeme via Remote Login oder Virtual Terminal und das für uns so wichtige HTTP.

### 2.1.2 Vereinfachtes OSI-Modell

Referenzmodelle zeichnen sich dadurch aus, dass sie die Gemeinsamkeiten, aber auch die Unterschiede zwischen den verschiedenen konkreten Modellinstanzen berücksichtigen. Das bedeutet, dass die Anwendung des OSI-Referenzmodells für ein bestimmtes Kommunikationssystem nicht immer alle Aspekte des Modells beinhalten muss. Als wohl bekanntestes Beispiel können Sie sich hierfür das Internet vorstellen. Wie Sie aus Bild 2.2 ersehen können, stimmen das OSI-Modell und seine Inkarnation im Internet in den ersten vier transportorientierten Schichten überein. Erst in den anwendungsorientierten Schichten 5 bis 7 wird es im Internet schwierig, entsprechende Vertreter zu finden. Daher hat sich für das Internet eine entsprechend vereinfachte Instanz des OSI-Modells herausgebildet, die nur über die Anwendungsschicht verfügt und nicht weiter die Sitzungs- und Darstellungsschicht unterscheidet.

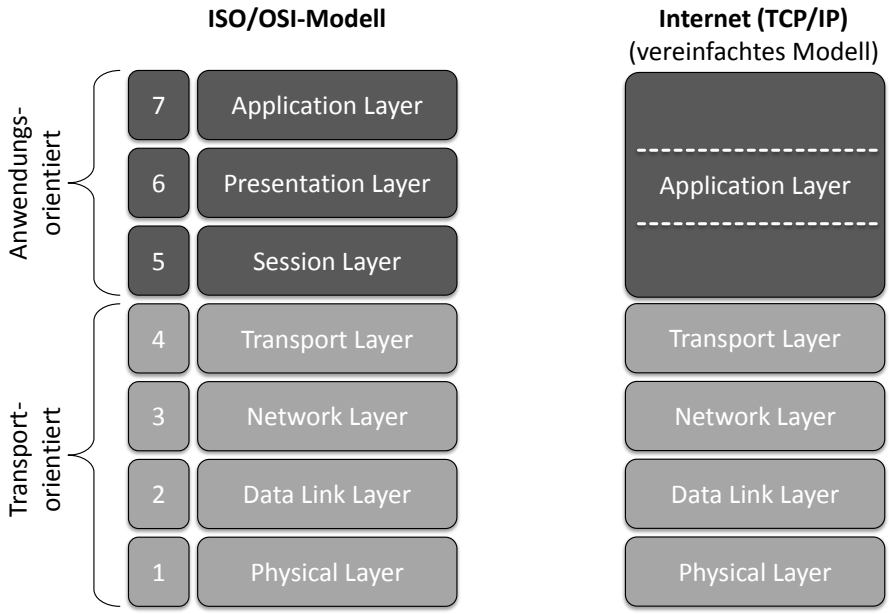



**Bild 2.2** Anwendung des OSI-Modells auf das Internet

Das beschriebene Gebilde, ob als allgemeines Referenzmodell oder als Ausprägung für das Internet, allein reicht noch nicht, um zu verstehen, wie Ihnen ein solcher Schichtenhaufen bei der Entwicklung von komplexen Kommunikationssystemen behilflich sein kann.

### 2.1.3 Dienst

Durch die Gruppierung in sieben Schichten mit abstrakten, aber definierten Aufgaben und Funktionen ist zunächst eine Struktur in das Themenumfeld eingebracht worden. Damit Sie dieses aber tatsächlich in seiner Komplexität beherrschen können, fehlt es noch an Ordnung. Diese wäre Ihnen nicht gegeben, wenn alle Schichten mit allen anderen Schichten interagieren könnten. Wenn dem so wäre, so stünden die Chancen nicht schlecht, in einem schier undurchdringbaren Wirrwarr an Abhängigkeiten zu enden, das Sie kaum noch durchblicken könnten. Wir benötigen also noch weitere Maßnahmen, um aus den sieben Schichten ein wirkungsvolles Modell zu bekommen, das uns tatsächlich bei der Komplexitätsbewältigung von Kommunikationssystemen unterstützt.

Dass die sieben Schichten im OSI-Referenzmodell häufig als Bauklötzchenturm dargestellt werden, der ausgehend vom Fundament der Schicht 1 Stockwerk für Stockwerk in den Himmel ragt und schließlich im Dachgeschoss der Schicht 7 endet, ist kein Zufall. Dies hängt vielmehr mit der Tatsache zusammen, dass sich genau durch diese Anordnung die Abhängigkeiten der Schichten untereinander darstellen lassen. Die Dienste einer Schicht bieten diese genau der darüber liegenden Schicht an und sonst keiner anderen. Abstrakter ist dies nochmals in Bild 2.3 dargestellt.

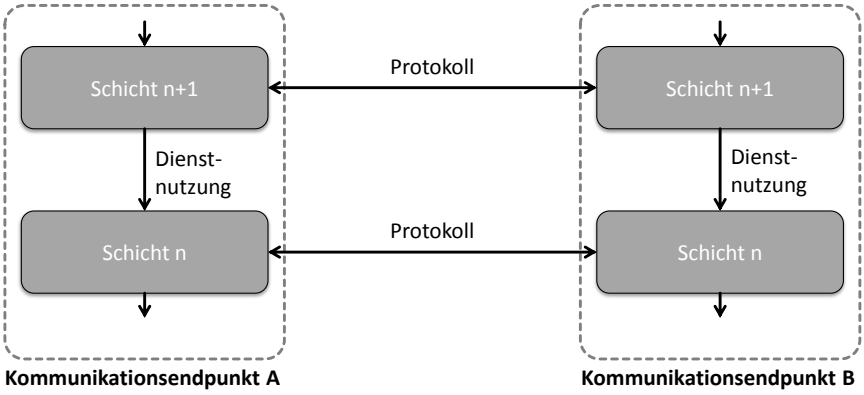



**Bild 2.3** Dienste und Protokolle in Kommunikationssystemen

Auf diese Weise lassen sich die Abhängigkeiten unter den Schichten dramatisch reduzieren und die Kontrolle über den tatsächlichen Kommunikationsablauf bewahren. Sie können sich die Wirkung dieses Prinzips an einigen konkreten Konstellationen verdeutlichen. So ist es z. B. für die Schicht-4-Protokolle TCP und UDP völlig irrelevant, wie sich das Netzwerk unterhalb der eigenen Schicht zusammensetzt. Ob drahtgebunden oder kabellos, ob rasend schnell oder schleichend langsam, ob zuverlässig oder fehleranfällig, für TCP und UDP sind nur die Dienste der Netzwerkschicht ausschlaggebend und damit das, was ihnen IP bietet. Auf dieser Ebene muss nicht noch mit anderen Schichten als der direkt darunter liegenden interagiert werden, um die eigenen Funktionalitäten anbieten zu können. Denken Sie sich eigene Situationen aus und überprüfen Sie damit Ihr Verständnis.

### 2.1.4 Protokoll

Neben den Diensten und den Abhängigkeitsstrukturen zwischen den Schichten im OSI-Modell, kommt in Bild 2.3 zudem der Begriff des Protokolls vor. Nur was ist das bloß?

Wie ein Protokoll bei Hofe die Umgangs- und Benimmregeln festlegt, legt ein Kommunikationsprotokoll eine Menge von Regeln fest, die beschreiben, wie Nachrichten ausgetauscht werden, um eine bestimmte Aufgabe zu erledigen . Sie können noch weitere anschauliche Parallelen finden, mit denen Sie sich dem Protokollbegriff annähern können. Stellen Sie sich z. B. die Konvention vor, wie eine Person eine zweite Person in einem Straßencafé höflich nach einem freien Sitzplatz fragt (siehe Bild 2.4).

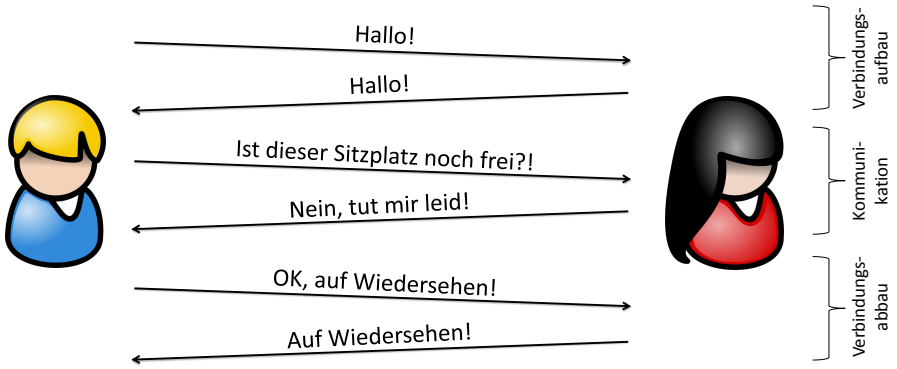



**Bild 2.4** Veranschaulichung des Protokollbegriffs

In dem in Bild 2.4 dargestellten Protokoll ist festgelegt, dass derjenige die Verbindung aufbaut, der den freien Sitzplatz im Straßencafé sucht. In unserer Abbildung ist das der junge Mann auf der linken Seite. Dieser beginnt also mit dem Verbindungsaufbau, indem er die junge Frau mit einem freundlichen „Hallo!“ anspricht. Er konnte die junge Frau mit seiner Begrüßung erreichen und diese ist auch gesprächsbereit und antwortet in der Folge ebenfalls mit einem freundlichen „Hallo!“. Hiermit ist der Kommunikationskanal zwischen den beiden Gesprächspartnern aufgebaut und die eigentliche Kommunikation kann über die etablierte Verbindung vollzogen werden. Die eigentliche Frage, die dem jungen Mann am Herzen liegt, folgt sogleich und beinhaltet die Frage nach einem freien Sitzplatz. Auf seine Frage erwidert die junge Frau, dass der Platz nicht frei ist. Damit ist die gerade begonnene Konversation bereits wieder an ihrem Ende angekommen. Zu einer höflichen Anfrage gehört, das sich der Fragende für die Auskunft bedankt und verabschiedet. Dies erfolgt auch mit dem letzten Nachrichtenpaar, dass damit auch die etablierte Kommunikationsverbindung beendet.

An dieser alltäglichen Lebenssituation können Sie wunderbar nachvollziehen, was alles Gegenstand eines Kommunikationsprotokolls ist. Die Abfolge der auszutauschenden Nachrichten wird festgelegt und damit auch, welcher der beiden Kommunikationspartner die Konversation initiiert und welche Nachrichtensequenz dann bis zur Beendigung der Kommunikation stattfindet. Zudem werden der Aufbau und Inhalt der Nachrichten festgelegt und wie diese codiert sind (in unserem Beispiel höflich). Und tatsächlich folgen Protokollstandards diesen Maßgaben aus dem realen Leben, um technische Kommuni-

kationssysteme umzusetzen. Wir haben für Sie zu einem späteren Zeitpunkt in diesem Kapitel – bei der detaillierten Beschreibung von HTTP – eine weitere Veranschaulichung eingebaut, bei der Sie ein Déjà-vu erleben sollten. Wenn Sie keines gehabt haben sollten, dann schreiben Sie uns!

Protokolle gibt es viele und sie sind immer einer Schicht des OSI-Modells zugeordnet. Bild 2.5 zeigt nochmals die Ausgestaltung des OSI-Modells für das Internet und fügt relevante Protokolle der jeweiligen Schicht an.

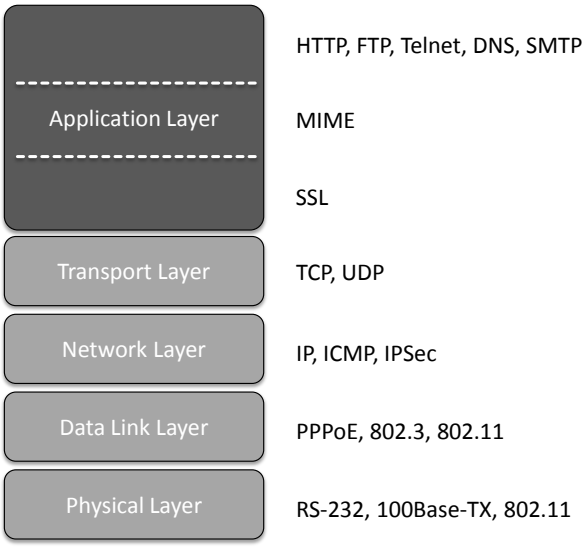



**Bild 2.5** Ausgewählte Protokolle bei der Internetkommunikation

In den ersten beiden Schichten haben wir die gebräuchlichen Standards Ethernet (IEEE 802.3) und WLAN (IEEE 802.11) angegeben. Schicht 3 wird im Internet von IP dominiert. Auf der Transportschicht haben Sie die Wahl zwischen den zuverlässigen Transportdiensten von TCP oder dem unzuverlässigen Pendant UDP. In der Anwendungsschicht findet sich eine ganze Reihe von Protokollen, die z. B. für den entfernten Login (Telnet, SSH), für E-Mail (SMTP, IMAP, POP3), für den Datentransfer (FTP, SFTP) und für das Browsen im Web (HTTP) spezifiziert wurden.

### 2.1.5 Kommunikationsfluss

Warum und wie jetzt das auf den ersten Anblick recht umfangreiche und komplex wirkende OSI-Referenzmodell genau das Gegenteil erreichen will, nämlich die Komplexität moderner Kommunikationssysteme beherrschbar zu machen, können Sie sich an einem einfachen Beispiel vor Augen führen. Stellen Sie sich ein lokales Netzwerk vor, wie es in Bild 2.6 dargestellt ist und das einen Webserver über einen WLAN-Router mit einem Notebook vernetzt. Der Webserver ist dabei via Kupferkabel gemäß den Ethernet-Standards IEEE 802.3 [IEE12a] und das Notebook entsprechend kabellos gemäß den WLAN-Standards IEEE 802.11 [IEE12b] am Router angeschlossen.

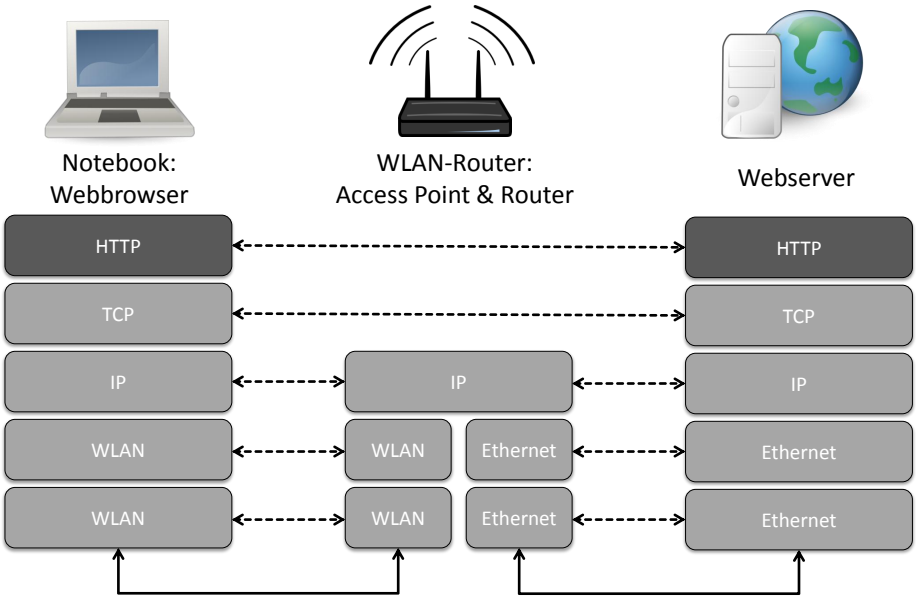



**Bild 2.6** Beispielhafter Kommunikationsfluss zwischen Client und Server

Setzt ein Webbrowser auf dem Notebook eine Anfrage an den Webserver im lokalen Netz ab, wird die HTTP-Anfrage logisch direkt an den Webserver gerichtet, der daraufhin mit einer HTTP-Antwort darauf reagiert. Dies ist die Protokollsicht. Tatsächlich nutzt der Webbrowser den Dienst der darunter liegenden Transportschicht und bittet diesen, eine zuverlässige TCP-Verbindung zu dem in der URL angegebenen Host aufzubauen und den mit dem Port angegebenen Dienst aufzurufen. Die Transportschicht ermittelt die IP-Adresse zum vorliegenden Hostnamen und verpackt die Daten, die sie von der Anwendungsschicht bekommen hat, in TCP-Pakete, die neben den Daten aus der Anwendungsschicht zusätzliche Protokolldaten enthalten, die für die zuverlässige Zustellung der Daten benötigt werden. Die Netzwerkschicht ermittelt die Route, die die Pakete zum Ziel nehmen müssen. An dieser Stelle wird insbesondere ermittelt, an welche Netzwerkkarte die Daten übergeben werden sollen. Im Beispiel in Bild 2.6 ist das die im Notebook eingebaute WLAN-Karte. Ist die Netzwerkkarte ermittelt, werden an diese die IP-Pakete übergeben, in die die Netzwerkschicht die TCP-Pakete verpackt hat. In der Netzwerkkarte angekommen, werden in Abhängigkeit der vorliegenden Netzwerktechnologie der Zugriff auf das Medium (Kupferkabel, Lichtwelle, Funk) sowie die Umwandlung der Daten in übertragungsfähige Signale vorgenommen. Auf diesem Weg gelangen die Daten schließlich physikalisch auf den nächsten Vermittlungsknoten auf der Route zum adressierten Ziel. Im Beispiel ist das der lokale WLAN-Router. Hier werden bis einschließlich zur Schicht 3 die Schritte rückgängig gemacht. Die zurückgewonnenen IP-Pakete enthalten die IP-Adresse des Zielsystems und können anhand dieser weitergeleitet werden. Da der Webserver im LAN angesprochen wird, legt der WLAN-Router diese auf die lokale Ethernetkarte. Die IP-Pakete werden erneut für den physikalischen Transport über das Kupferkabel zur Ethernetkarte im Webserver an-

gepasst und in entsprechende Signale überführt. Im Webserver angekommen, werden die Pakete Schicht für Schicht nach oben gereicht. Die Transportschicht gewährleistet, dass die TCP-Pakete vollständig und fehlerfrei sowie in der richtigen Reihenfolge vorliegen. Daraus werden dann die von der Anwendungsschicht der Clientseite abgeschickten Daten (z. B. die HTTP-Anfrage) rekonstruiert. Die rekonstruierten Daten können dann von der Anwendung verarbeitet werden. Die HTTP-Antwort geht den gleichen Weg zurück.

## 2.2 HTTP en détail

Im OSI-Referenzmodell finden Sie HTTP in der Anwendungsschicht. Einige der Regeln, die ein Kommunikationsprotokoll ausmachen, haben wir bereits im vorangegangenen Kapitel besprochen. Dazu zählt insbesondere die Reihenfolge der zwischen Client und Server ausgetauschten Nachrichten. Diese Regeln, die ein Protokoll ausmachen, schauen wir uns im Folgenden anhand von HTTP im Detail an.

### 2.2.1 Kommunikationsablauf

Die erste Frage, die Sie sich dabei vermutlich gerade stellen, ist, wer die Kommunikation zwischen den beiden beteiligten Akteuren beginnt. Tatsächlich ist in der HTTP-Spezifikation festgelegt, dass immer der Client die Kommunikation initiiert. Diese erfolgt durch das Absetzen einer Anfrage an den Server.

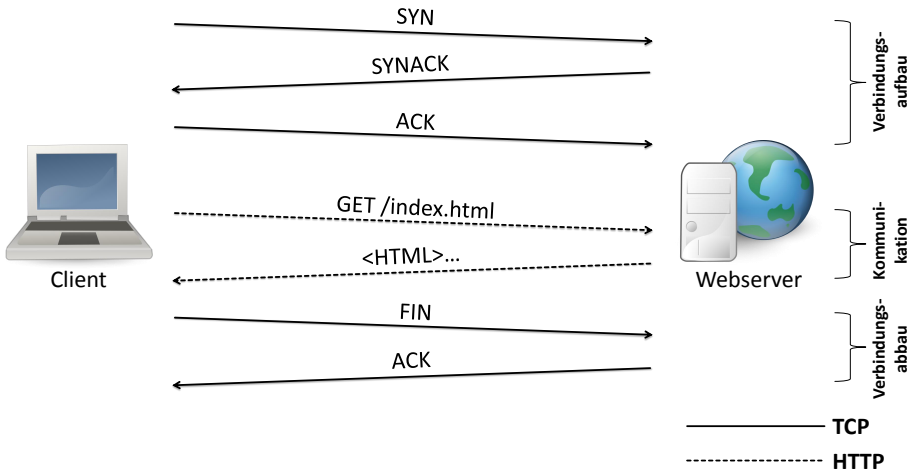



**Bild 2.7** HTTP über TCP

Diesen Ablauf können Sie auch einfach mit dem Telnet-Clientprogramm nachvollziehen (in jedem guten Betriebssystem enthalten ). Das Telnet-Clientprogramm etabliert nämlich

nichts anderes als eine TCP-Verbindung zu einem entfernten Rechner. Konnte diese hergestellt werden, befindet sich im Normalfall auf der Gegenseite ein Telnet-Server, der die Eingaben des Benutzers im Telnet-Client entgegennimmt, lokal ausführt und die Ausgabe an das Clientprogramm zurücksendet. Der Client erlaubt es, einen vom Telnet spezifizierten Standardport 22 abweichenden Port für den Verbindungsaufbau anzugeben. Setzen Sie diesen nun, wie in Bild 2.8 demonstriert, auf den HTTP-Standardport, so wird eine TCP-Verbindung zum angegebenen Webserver aufgebaut (die ersten drei Nachrichten in Bild 2.7). Was die Abbildung nicht mehr zeigt, ist die Antwort des Webservers, der die angefragte HTML-Seite zurückliefert. Die Kommunikationsverbindung wird direkt nach Erhalt der HTTP-Antwort vom Webserver geschlossen.


**Bild 2.8** Die Telnet-Probe

An diesem Experiment erkennen Sie auch schon eine wesentliche Eigenschaft von HTTP. Es ist textbasiert, sonst wären Sie nicht in der Lage gewesen, die Anfrage einfach so über Ihre Tastatur in die offene TCP-Verbindung der Telnet-Sitzung zu tippen. Andere Tools, mit denen Sie derartige Tests durchführen können, reichen von allgemeinen Netzwerktools wie z. B. Netcat[1] bis hin zu spezialisierten Werkzeugen speziell für HTTP wie z. B. cURL[2].

### 2.2.2 Textbasiert

Die Codierung der Protokollelemente erfolgt als Text. Sie können sich vorstellen, dass das den Nachteil hat, aus Sicht der zu übertragenden Datenmenge nicht optimal zu sein. In diesem Kontext sind binäre Codierungen sicherlich leichtgewichtiger. Im Vergleich zu binären Protokollen ist aber der große Vorteil textbasierter Protokolle, wie es HTTP ist, dass die Protokollelemente einfach zu erzeugen und zu lesen sind. Hierdurch erleichtert sich die Entwicklung ungemein. Wenn Sie schon mal einen Bug in einem binären Protokollformat aufspüren mussten, werden Sie uns in diesem Punkt ohne Weiteres zustimmen.

Um zu verdeutlichen, wie die Textbasiertheit die Entwicklung mit HTTP vereinfacht, wollen wir mit Ihnen einen einfachen HTTP-Client implementieren. Das Beispiel mit dem Telnet-Client aus Abschnitt 2.2.1 wollen wir mit einer weitverbreiteten Programmierschnittstelle, den Sockets, in ein eigenes Programm integrieren. Sockets bieten eine abstrakte Schnittstelle, mit der Sie eine TCP-Verbindung zu einem Server aufbauen und verwalten können. Das folgende Beispiel zeigt, wie Sie mit Java einen Socket (also einen TCP-Kanal) zu einem Server öffnen können. Da Sockets praktisch in jeder Programmiersprache

1 *http://de.wikipedia.org/wiki/Netcat*
2 *http://de.wikipedia.org/wiki/CURL*

verfügbar sind, können Sie das konzeptionelle Verfahren auch in Ihrer Lieblingssprache vorfinden und entsprechend umsetzen.

Wir ersetzen den Telnet-Client zunächst durch einen TCP-Kanal, den wir programmatisch via Sockets öffnen. Das Beispiel ist absichtlich simplifiziert und vernachlässigt zur besseren Lesbarkeit z. B. die Behandlung von Exceptions, um den Blick für das Wesentliche nicht zu versperren.

**Listing 2.1** Erzeugen eines Socket-Objekts

```
1 String serverHostname = "google.de";
2 Socket webServer = new Socket(serverHostname, 80);
```

Haben wir auf diese Weise ein Socket-Objekt erzeugt, benötigen wir noch einen InputStream, mit dem wir Daten empfangen, und einen OutputStream, mit dem wir Daten senden können. Die generischen Oberklassen können natürlich direkt verwendet werden. Dies wäre aber nur für Binärdaten sinnvoll und ratsam. Für textbasierte Daten, wie sie für HTTP benötigt werden, sind Klassen wünschenswert, mit denen wir mit den gewohnten und lieb gewonnenen `print()`- und `println()`-Methoden arbeiten können. Die `PrintWriter`-Klasse stellt diese bereit und kann auf Grundlage eines OutputStreams instanziiert werden. Selbiges gilt für die Klasse `BufferedReader`.

**Listing 2.2** Erzeugen eines Input- und OutputStreams

```
1 PrintWriter out = new PrintWriter(webServer.getOutputStream(), true);
2 BufferedReader in = new BufferedReader(new InputStreamReader(
      webServer.getInputStream()));
```

Steht der TCP-Kanal, können wir die HTTP-Anfrage aus Bild 2.8 absetzen. Genau die Tatsache, dass HTTP textbasiert ist, ermöglicht es uns nun, das Protokollelement als einfachen String über den PrintWriter in den Kommunikationskanal zu schreiben.

**Listing 2.3** Erzeugen einer HTTP-Anfrage

```
1 out.print("GET /index.html HTTP/1.0\r\n");
2 out.print("\r\n");
```

Das kommt uns doch sehr bekannt vor. Bevor Sie sich jetzt aber anfangen zu wundern, wozu denn die zusätzlichen Escape-Sequenzen (`\r\n`) im String benötigt werden, möchten wir Sie um einige wenige Absätze Geduld bitten, nach denen das Geheimnis gelüftet wird.

Das Argument mit dem höheren Bedarf an Transfervolumen durch die Codierung als Text verliert zudem an Gewicht, da es durch zusätzliche Mechanismen abgeschwächt wird. Bevor die HTTP-Anfrage über TCP gesendet wird, kann diese komprimiert werden. Auf der Gegenseite wird sie dann entsprechend entpackt, bevor die HTTP-Verarbeitung beginnt. Als Komprimierungsverfahren schreibt HTTP hierfür gzip vor [FGM+99, Kapitel 3].

Schließlich gilt nicht für alle Teile eines HTTP-Protokollelements, dass sie textbasiert sein müssen. Stellen Sie z. B. ein Pixelbild im GIF-, JPG- oder PNG-Format über einen Webserver zum Abruf bereit, so wird dieses nicht in Textform übertragen. Da derartige Daten nativ in einem Binärformat vorliegen, würde das sonst bedeuten, dass diese zunächst in ei-

ne Textrepräsentation überführt werden müssten. Hierfür stünde Ihnen z. B. die Base64-Codierung zur Verfügung. Mit dieser könnten Sie binäre Daten in Textdaten transformieren. Der Nachteil hierbei ist, dass die Base64-Codierung aus jeweils 6 Bit der Ursprungsdaten ein Textzeichen bestimmt, das wiederum mit 8 Bit codiert ist. Hieraus können Sie leicht erkennen, dass durch die Base64-Codierung ein Datenvolumen-Overhead entsteht, der nicht unwesentlich ist. Dieser liegt bei etwa 33 % (8 / 6 = 1,33). Daher sieht HTTP für den Transfer von Nutzdaten eine Ausnahme zur Textform vor. Sie steht eng in Verbindung mit dem Aufbau der Anfragen und Antworten, die wir uns im Folgenden genauer anschauen.

### 2.2.3 Aufbau von Request-Nachrichten und Protokollelemente

Der allgemeine Aufbau einer HTTP-Anfrage ist in drei verschiedene Bereiche aufgeteilt, die durch die folgende allgemeine Notation beschrieben sind [FGM+99]:

**Listing 2.4** Notation des Aufbaus einer HTTP-Anfrage

```
<Method> <URL> <HTTP Version><CR><LF>
[Header Field Name: Value <CR><LF>]
...
[Header Field Name: Value <CR><LF>]
<CR><LF>
[Payload]
```

Alle Zeilen werden per Spezifikation durch die zwei Steuerzeichen Carriage Return (`<CR>`) und Line Feed (`<LF>`) abgeschlossen. Durch diese Festlegung werden insbesondere mögliche Inkompatibilitäten zwischen verschiedenen Systemen, die den Zeilenumbruch verschieden handhaben (Windows: `<CR><LF>`, Mac OS bis Version 9: `<CR>`, Mac OS X und Linux: `<LF>`), aus dem Weg geräumt.

Den wichtigsten Teil einer HTTP-Anfrage finden Sie unmittelbar in der ersten Zeile, die sogenannte Anfragezeile (engl. *Request Line*), die Sie für eine gültige Anfrage auch zwingend angeben müssen.

```
<Method> <URL> <HTTP Version><CR><LF>
```

Hier müssen Sie zunächst eine sogenannte Methode (`<Method>`) angeben. Sie können sich unter dieser Methode einen Schalter vorstellen, mit dem Sie verschiedene Protokollelemente an den Server senden bzw. verschiedene Aktionen auf dem Server auslösen können. Die bekannteste Methode ist die *GET*-Methode. Wie Sie sicherlich schon aus dem Namen geschlossen haben, bewirkt diese Methode den Abruf einer Ressource von einem Server. Um die angefragte Ressource zu benennen, muss als Nächstes die `<URL>`-Komponente der Anfragezeile mit der URL befüllt werden, die die angefragte Ressource identifiziert. Zur Erinnerung, eine URL hat den folgenden Aufbau:

*<scheme>://<user>@<domain>:<port>/<path>?<query>#<fragment>*

Im Web wird der mit `scheme` bezeichnete Teil mit dem Kürzel `http` ersetzt. Der `user` fällt im Web ersatzlos weg. Die `domain` entspricht dem Domainname der Website. Ist diese über