Betriebssysteme und Netzwerke Vorlesung N02

Artur Andrzejak

Protokollschichten und ihre Dienstmodelle

Netzwerke sind komplex

- Netzwerke haben **viele heterogene Bestandteile**
  - Hosts
  - Router / Switches
  - Unterschiedliche Übertragungsmedien
  - Anwendungen und Endsysteme
- Gibt es eine Möglichkeit, diese komplexe Architektur zu strukturieren?
  - Oder zumindest unsere Darstellung und Diskussion der Netzwerke (zu strukturieren)?
- Eine Möglichkeit ist die **Schichtenarchitektur (layering)**

Protokollschichten

- In Netzwerken bezieht sich die Schichtung überwiegend auf die **Protokolle**
- Jede Schicht bietet ihre Dienste an, indem sie …
  - Innerhalb ihrer Schicht bestimmte Aktionen durchführt
  - Die Dienste der direkt unter ihr liegenden Schicht nutzt
- Wichtig: Schicht k kann nur die Dienste der Schicht (k-1) verwenden, aber nicht von (k-2), (k-3) usw.!

Schicht k nutzt nur Dienste von Schicht (k-1)

Schicht (k-1) nutzt nur Dienste von S. (k-2)

Schicht (k-2) nutzt nur Dienste von S. (k-3)

Internet-Protokollstapel

Video: Internet Protocol [N02b]
- https://www.youtube.com/watch?v=zyL1Fud1Z1c
- Ab ca. 0:38 (min:sec)

- Die Gesamtheit der Protokolle aller Schichten bildet den **Protokollstapel** (**protocol stack**)
  - Internet Protokollstapel besteht aus fünf Schichten

Name	Funktion	Bsp-Protokolle
Anwendungsschicht (application layer)	Netzwerkanwendungen	HTTP, FTP, SMTP
Transportschicht (transport layer)	Überträgt Nachrichten zwischen BS-Prozessen	TCP, UDP
Netzwerkschicht (network layer)	Leitet die Pakete (Data-gramme) zwischen Hosts	IP, Routing-Protokolle
Sicherungsschicht (data link layer)	Leitet die Pakete zwischen Netzwerkknoten (Routern)	PPP, Ethernet
Bitübertragungss. (physical layer)	Überträgt einzelne Bits zwischen Netzwerkknoten	Hängt vom Medium ab

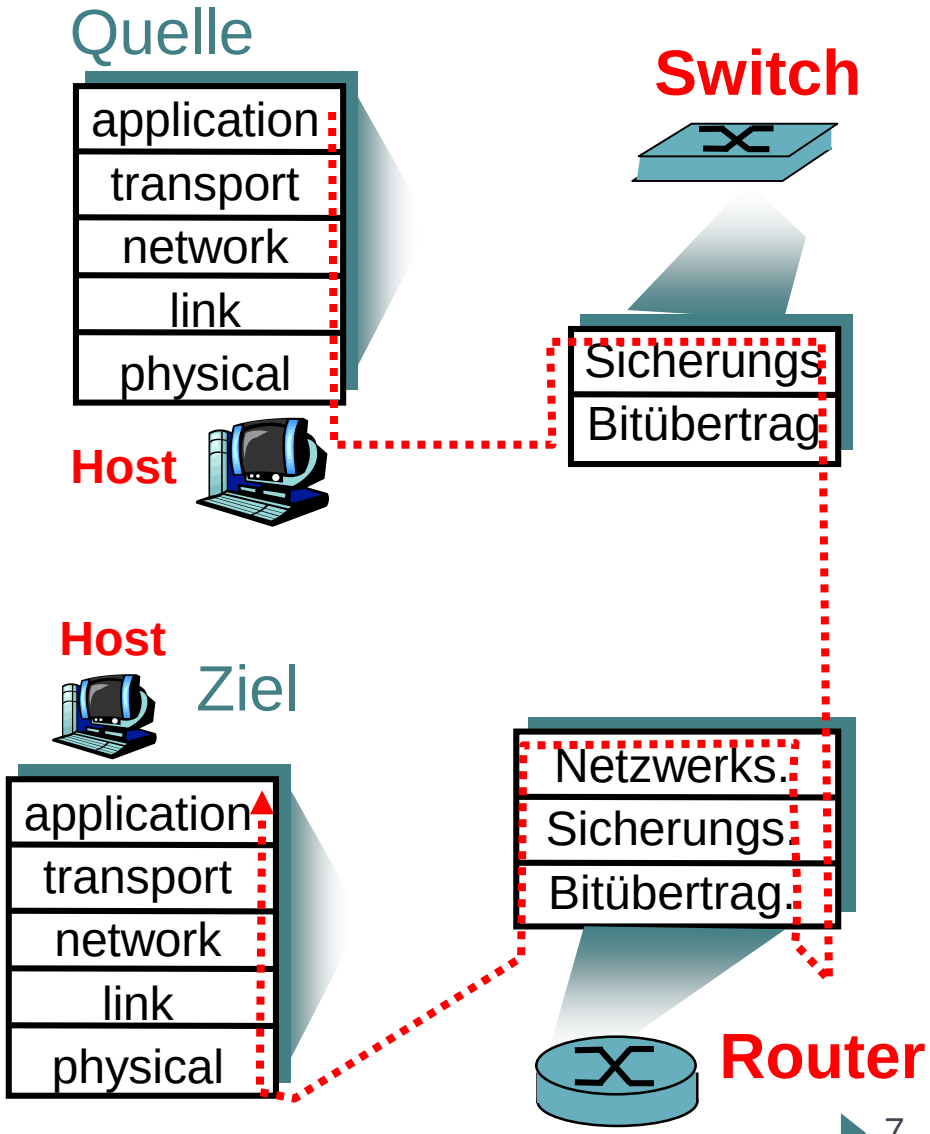
Internet-Protokollstapel /2

- Wie identifiziert man die Schicht?
  - Eine gutes Kriterium sind die Endpunkte der Kommunikation

Name	Endpunkte
Anwendungsschicht (application layer)	Mehrere Programme, jedes kann aus einem oder mehr Prozessen bestehen (z.B. Browser + Server)
Transportschicht (transport layer)	Prozesse (bzw. zugehörige Sockets – APIs des BS) auf einem oder mehreren Hosts
Netzwerkschicht (network layer)	Start- und End-Hosts (es wird nicht zwischen Prozessen auf einem Host unterschieden)
Sicherungsschicht (data link layer)	Zwei direkt verbundene Geräte (Host, Router, Switch) an den beiden Enden einer Teilstrecke
Bitübertragungss. (physical layer)	Elektronik der Leitungen oder Glasfaser an den Enden einer Teilstrecke

Wo werden die Protokolle implementiert?

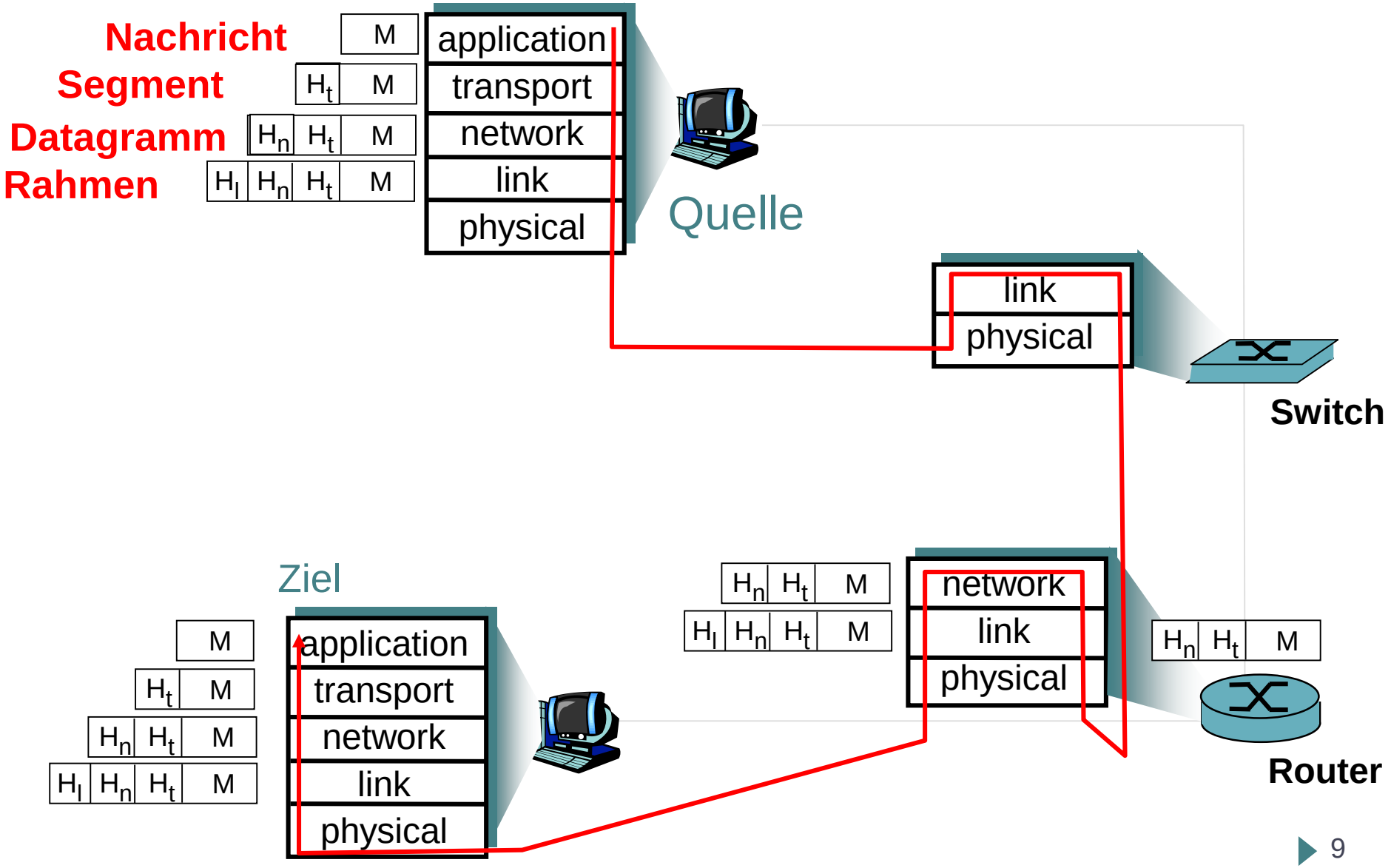
- Hosts (Endpunkte) implementieren alle fünf Schichten
- **Router** bis zur Netzwerkschicht
  - D.h. Router wissen, wo das Ziel des Pakets ist
- **Switches** nur bis zur Sicherungsschicht
  - Sie kennen nur den nächsten Netzwerk-knoten auf dem Weg des Pakets



Kapselung (Encapsulation)

- Jede Schicht fügt ihren eigenen Header hinzu
  - Jedes Paket trägt deshalb zwei Arten von Feldern: **Header-Felder** und **Datenfelder** (**payload fields**)
- Die Anwendungsschicht erzeugt eine Nachricht M
- Die Transportschicht fügt den Header H_t hinzu
  - Es entsteht ein **Segment**
- Die Netzwerkschicht fügt den Header H_n hinzu
  - Es entsteht ein **Datagramm**
- Die Sicherungsschicht fügt den Header H_l (für link) hinzu
  - Es entsteht ein **Rahmen (frame)**

Kapselung - Beispiel



Open Systems Interconnection-Modell (OSI)

- In den späten 1970ern schlug **ISO** (**International Organization for Standardization**) vor, dass Computernetzwerke in sieben Schichten organisiert werden sollten, dem **OSI**-Modell
- Schichten 1..4 + 7 wie im Internet Protokollstapel
- **Neu: Darstellungsschicht**
  - Verschlüsselung, Kompression, Datenformat-umwandlung
- **Neu: Kommunikations-steuerungsschicht**
  - U.a. Synchronisation des Datenaustausches
- Wo sind sie heutzutage?

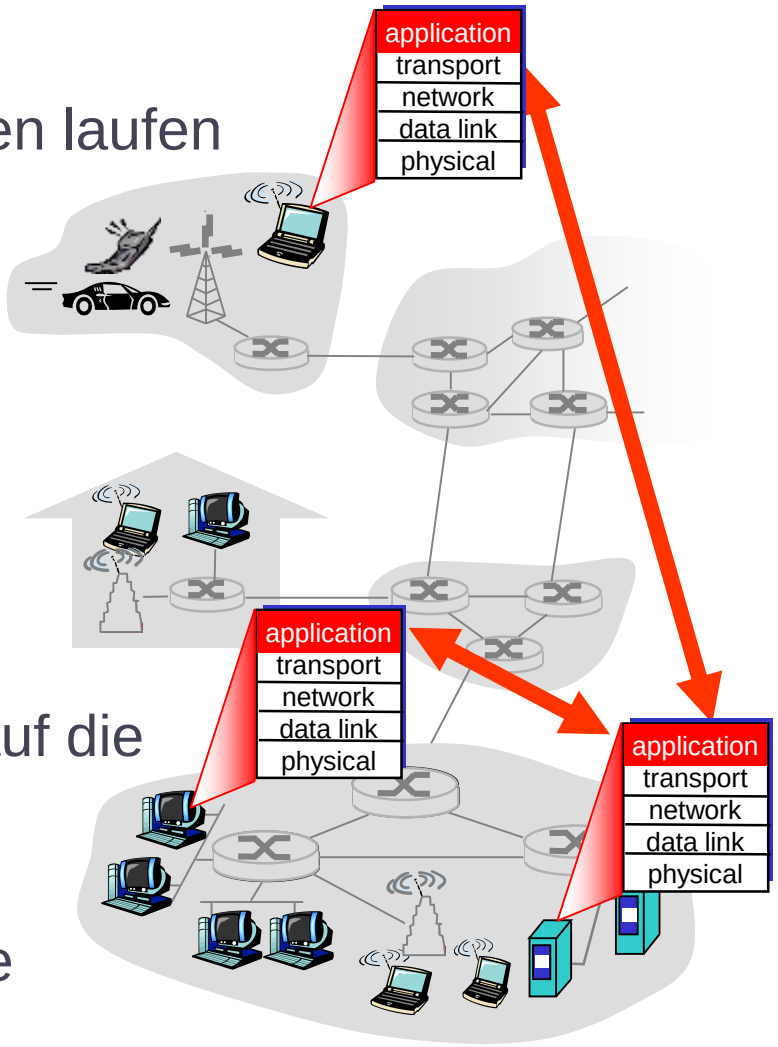
7: Anwendungsschicht
6: Darstellungsschicht
5: Kommunikations-steuerungsschicht
4: Transportschicht
3: Netzwerkschicht
2: Sicherungsschicht
1: Bitübertragungsschicht

Grundlagen von Netzwerkanwendungen

Architekturen, Sockets, Protokolle

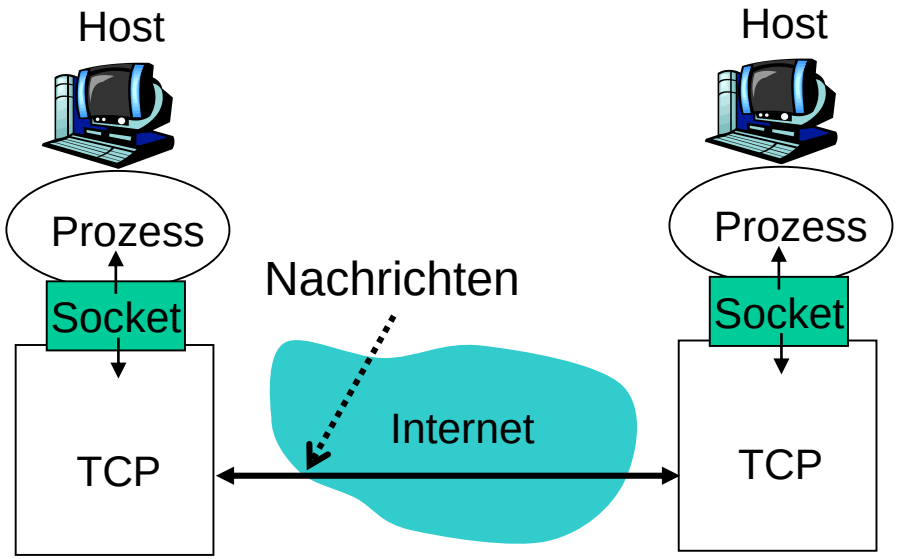
Netzwerkanwendungen sind …

- Programme, die …
  - Auf verschiedenen Endsystemen laufen
  - Über Netzwerk kommunizieren
  - Z.B. Web Server-Programme und Browser-Programme
- Keine Programme für das „Innere“ des Netzwerks nötig
  - Anwendungen verlassen sich auf die Dienste der Transportschicht für die Kommunikation
  - Router / Switches können keine Benutzerprogramme ausführen



Prozesskommunikation

- Aus dem Teil BS wissen wir, dass eine Anwendung als ein oder mehr Prozesse ausgeführt wird
  - Es kommunizieren also nicht die „Rechner“, sondern **Prozesse**
- Befinden sich diese Prozesse auf dem selben Rechner, wird für die Kommunikation **IPC** benutzt (siehe VL 3, 4)
- Sind sie auf verschiedenen Rechnern, wird **Nachrichtenaustausch** (**message passing**) verwendet



Architektur: Client-Server vs. P2P

- **Client-Server**
- Server
  - Immer bereit
  - Permanente IP-Adresse
  - Serverfarmen für die Skalierbarkeit
- Clients
  - Kommunizieren mit dem Server, aber nicht direkt untereinander
  - Können dynamische IP-Adressen haben
  - Nicht immer online

- **Reine P2P-Architektur**
- Beliebige Endsysteme („Peers“) kommunizieren miteinander
- Keine Server
- Peers ändern zwischendurch ihre IP-Adressen und sind nicht die ganze Zeit online
- Probleme?
  - Hochskalierbar, aber schwierig zu verwalten
  - Sicherheitsbedenken

Prozesse: Client und Server

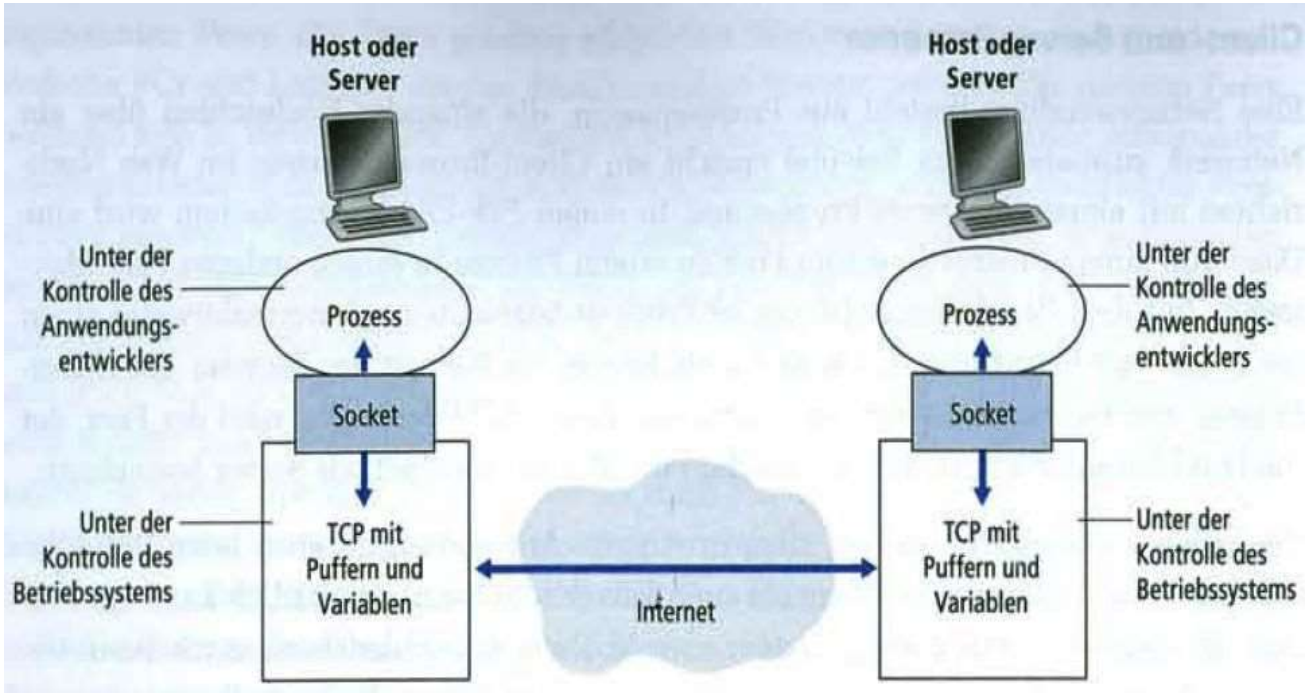
- Eine Netzanwendung besteht aus Prozesspaaren, die einander Nachrichten über Netzwerk zusenden

- Der Prozess, der die Kommunikation *eröffnet* (also erstmals den anderen Prozess zu Beginn der Sitzung kontaktiert), wird als **Client** bezeichnet
- Der Prozess, der darauf wartet, zu Beginn einer Sitzung angesprochen zu werden, ist der **Server**

- Achtung: Hier wird mit Client oder Server die **Funktion** bezeichnet und nicht die Anwendungsarchitektur
  - Auch beim P2P-Filesharing ist der Peer, der die Datei herunterlädt, der Client und der andere Peer der Server

Sockets – Schnittstellen

- Ein Prozess sendet Nachrichten in und empfängt Nachrichten aus dem Netzwerk mittels einer Softwareschnittstelle, die als **Socket** bezeichnet wird
- Die Socket-API erlaubt die Wahl des Transport-protokolls und einiger Parameter (später mehr dazu)



Addressieren von Prozessen

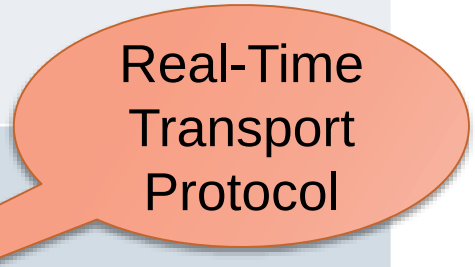
- Um aus dem Internet erreichbar zu sein, braucht jeder Host eine 32-Bit **IP-Adresse** (IPv4)
  - **ipconfig** auf Windows lieferte mir: 147.142.160.86
- Reicht das, um einen Prozess zu adressieren?
  - Nein; man braucht zusätzlich eine **Port-Nummer** (**port number**): Sie identifiziert einen Socket und damit einen Prozess auf dem Host
- Die Portnummer kann frei gewählt werden (1... 2^{16}-1)
  - Es gibt aber Konventionen – Beispiele?
  - HTTP Server: 80
  - Mail Server: 25
  - SSH-Server: 22
  - Wikipedia: *List of TCP and UDP port numbers* (Link)

Zwei Grundlegende Internet-Protokolle

- **TCP** Protokoll
  - **Verbindungsorientiert**: Einrichtung (setup) der Verbindung durch ein **Handshake** ist nötig
    - Zustand nur an den „Enden“ => keine leitungsorientierte Verbindung!
  - **Verlässliche Übertragung**
  - **Überlastkontrolle**: Der Empfänger wird nicht überlastet; Drosseln der Ü-Geschwindigkeit, wenn Netzwerk überlastet
  - **Keine**: Echtzeitgarantien, Sicherheit, min. Durchsatz, Mindestlohn
- **UDP** Protokoll
  - **Verbindungslos**: Kein Handshake nötig
  - **Unzuverlässig**: Keine Garantien auf die Zustellung der Nachricht oder die richtige Reihenfolge ihrer Ankunft
  - **Keine**: Überlastkontrolle, Echtzeitgarantien, Sicherheit
  - Frage: Warum wird UDP überhaupt verwendet?

Beispiele von Anwendungen und Protokollen

Anwendung	Anwendungs-schicht-Protokoll	Zugrunde-liegendes Internet-Protokoll
Email-Dienst	SMTP [RFC 2821]	TCP
Remote-Terminalzugang	Telnet [RFC 854]	TCP
World Wide Web	HTTP [RFC 2616]	TCP
Dateitransfer	FTP [RFC 959]	TCP
Internettelefonie (VoIP)	SIP, RTP oder proprietär (z. B. Skype)	Normalerweise UDP



Das Web und HTTP

Übersicht und Begriffe

- Das **World Wide Web** (**WWW** bzw. **Web**) ist die bekannteste Netzwerkanwendung (unter vielen)
  - Für viele ein Synonym für das Internet (das ist falsch)
- Idee: Ein Browser (Client) kann vom Webserver eine **Webseite** durch das **HTTP**-Protokoll erhalten
- Eine Webseite besteht aus einer **Basis-HTML-Datei**, die ggf. auf mehrere Objekte verweist
  - Objekte sind HTML-Dateien, (JPEG)-Bilder, Applets,…
- Jedes Objekt ist durch eine **URL** (**uniform resource locator**) eindeutig adressiert
  - URL besteht aus dem Hostnamen (host name) vor dem 1. Schrägstrich („/“) und Pfadnamen des Objekts danach

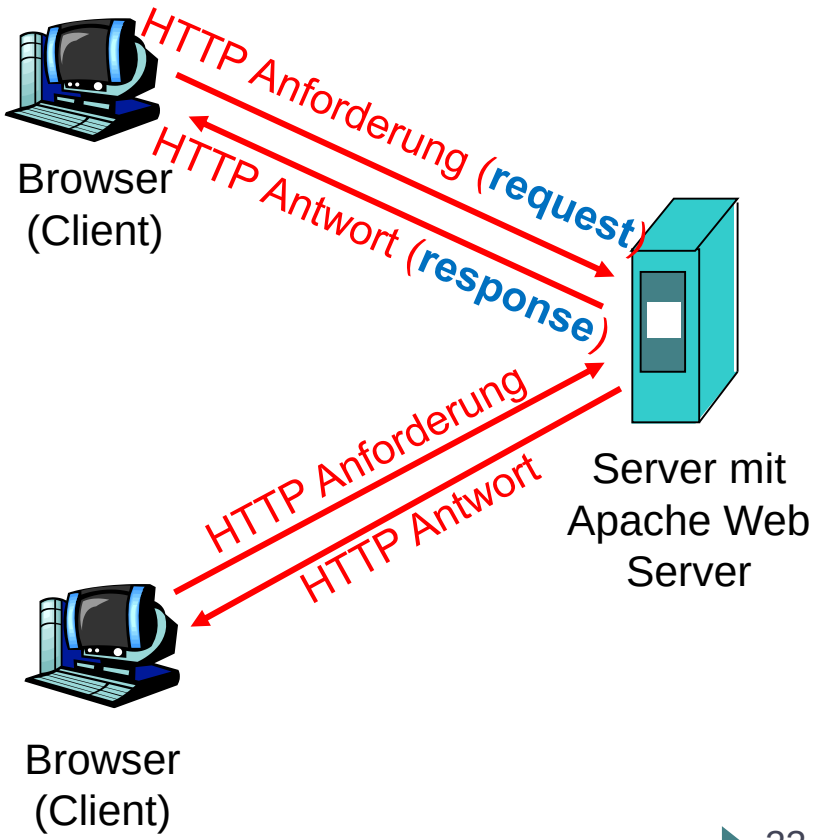
Video - HTTP Übersicht

- Video aus Coursera: http://1drv.ms/1FQTsFr
- Internet B - Technology/B09 - Application Layer:
  - Von 6:06 bis ca. 7:37 (min:sec)
  - Von 9:05 bis ca. 10:50 (min:sec)

HTTP - Hypertext Transfer Protocol

- HTTP verwendet TCP
  - Client (Browser) iniitert eine TCP-Verbindung zum Port 80 des Servers
  - Server akzeptiert TCP Verbindung
  - HTTP Nachrichten werden zwischen Browser und Server ausgetauscht
  - TCP Verbindung wird geschlossen
    - Meist vom Client

- HTTP ist **zustandslos**
  - Server behält keinen „HTTP-basierten“ Zustand zwischen Anfragen



HTTP-Nachrichtenformat

- Zwei Typen von HTTP-Nachrichten
  - Anforderung (**Request**)
  - Antwort (**Response**)
- HTTP-Request-Nachricht
  - Besteht aus einer **Request-Zeile** (Anforderungszeile)
    - 3 Felder: Methoden-, URL- und HTTP-Versionsfeld
  - Sowie **Header-Zeilen** (Kopfzeilen), hier: Webserver-Adresse, Browsertyp, Verbindungstyp, bevorz. Sprache

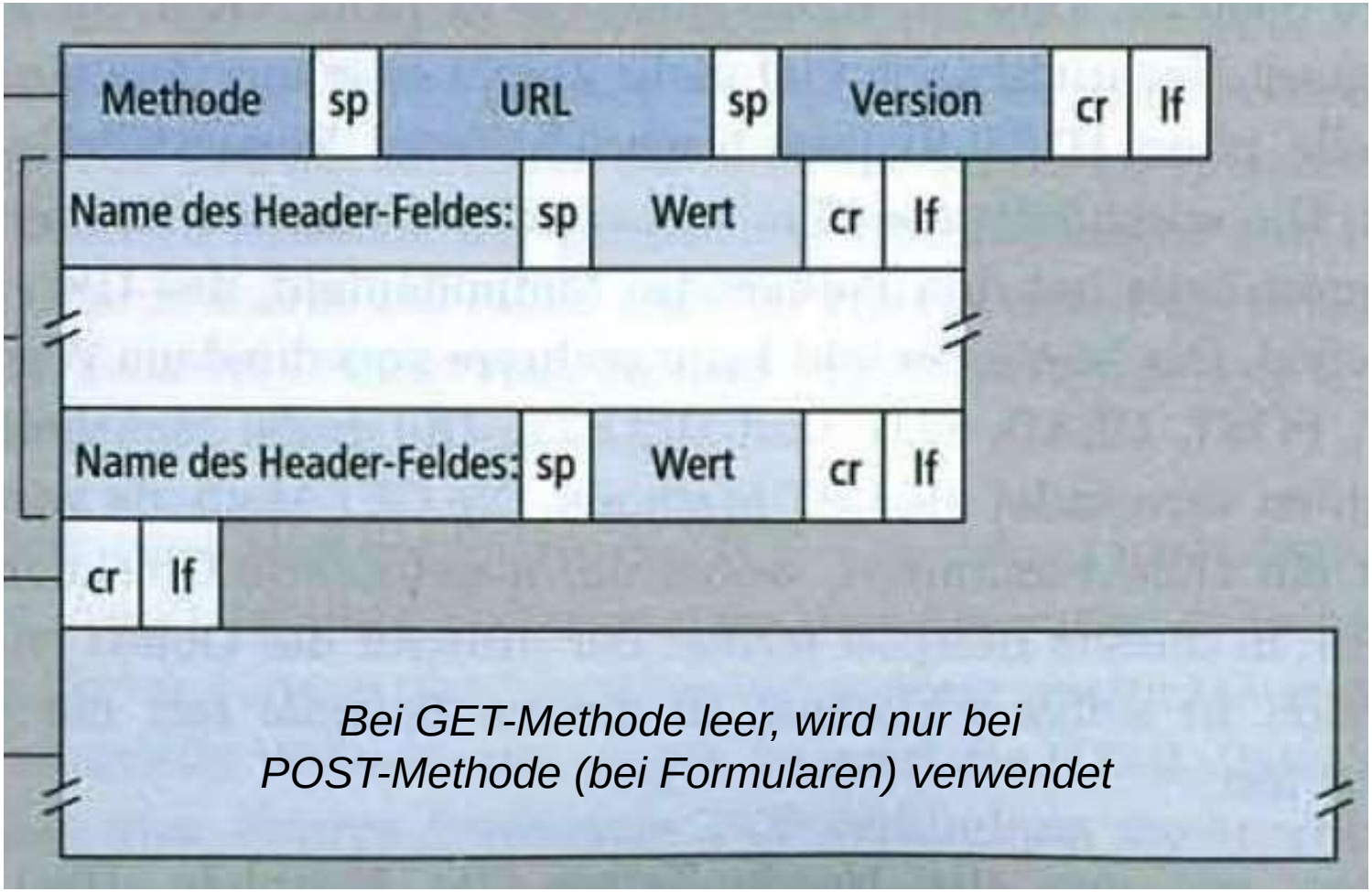
Request-Zeile → `GET /somedir/page.html HTTP/1.1`

Header-Zeilen

```
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

(extra carriage return, line feed)

Allgemeine Form einer Request-Nachricht



Typen von HTTP-Request Methoden

- **GET** vs. **POST**-Methodentypen
  - Bei POST enthält sog. *Entity-Body* diejenigen Daten, die der Benutzer in die Formularfelder eingegeben hat
- Alternativ (insb. bei GET) können die eingegebenen Daten in die angeforderte URL geschrieben werden
  - Z.B. bei Eingaben „Spinat, Ingwer“ könnte die URL lauten: www.essen.de/Rezeptsuche?Spinat&Ingwer
- **HEAD**: wie GET, aber es wird kein Objekt zurückgegeben
- **PUT**: ermöglicht das Hochladen eines Objektes auf den Server
- **DELETE**: ermöglicht das Löschen eines Objektes auf dem Server

HTTP-Response-Nachricht

Einleitende Statuszeile
(Protokollversion
Statuscode
Statusnachricht)

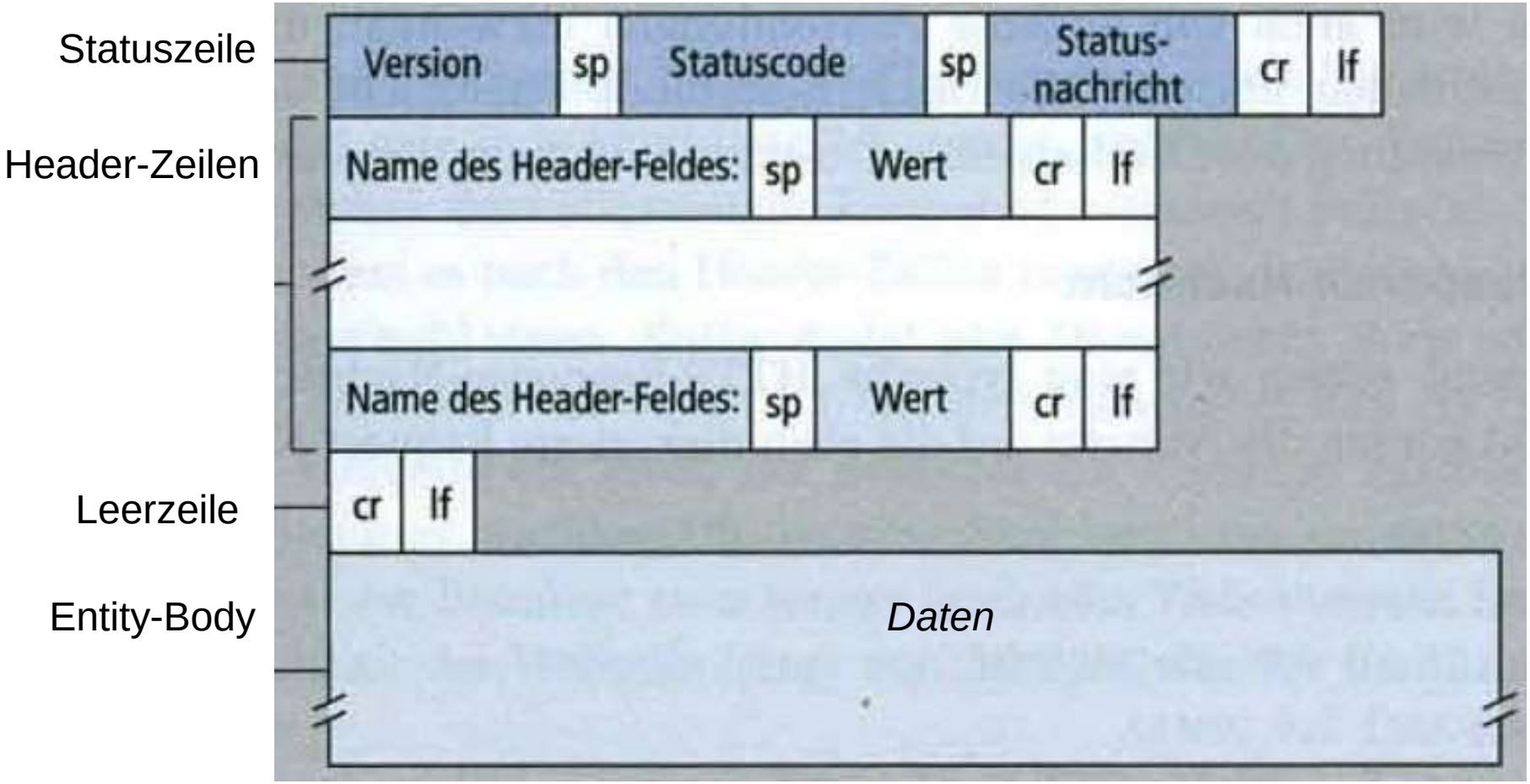
Header-Zeilen

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html
```

Daten – **Entity Body**

```
data data data data data ...
```

Allgemeine Form einer Response-Nachricht



Statuscodes und Statusnachrichten

- Einige zusammengehörige Statuscodes und Statusnachrichten sind:
  - **200 OK**: Die Anforderung hatte Erfolg und die Information wird in der Antwort zurückgegeben
  - **301 Moved Permanently**: Das angeforderte Objekt wurde dauerhaft entfernt; die neue URL ist in der Header-Zeile „Location" der Response-Nachricht angegeben
  - **400 Bad Request**: Dies ist ein generischer Fehlercode, der anzeigt, dass die Anforderung vom Server nicht verstanden wurde
  - **404 Not Found**: Das angeforderte Dokument existiert nicht auf diesem Server
  - **505 HTTP Version Not Supported**: Die angeforderte HTTP-Protokollversion wird vom Server nicht unterstützt

Einen HTTP-Clienten simulieren

- 1. Mit „**telnet <host> 80**“ die Verbingung zum <host> aufbauen
- 2. GET-Request simulieren durch die Eingabe auf der Kommandozeile
  - **GET /<Pfad-der-Webseite> HTTP/1.1**
  - **Host: <host>**
  - **(Eingabetaste nochmals drücken)**
- 3. Die Antwort des Servers erscheint auf dem Bildschirm
- Video aus Coursera: http://1drv.ms/1FQTsFr
  - Internet B - Technology/B09 - Application Layer:
  - Von 10:50 bis ca. 16:10 (min:sec)

Webseiten Umleiten

- Wie funktioniert das Umleiten der Webseiten?
  - Z.B. URL-Abkürzungen wie goo.gl/stNNx5 => Lange URL

Client-side:

- Anweisung im Header-Teil eines html-Dokumentes:
  - <meta http-equiv="refresh" content="0; url=http://example.com/" />
- Client (Browser) interpretiert das html-Dokument and lädt einfach die neue Webseite, hier example.com
- Das HTTP-Protokoll weiß nichts davon!

Webseiten Umleiten

- Wie funktioniert das Umleiten der Webseiten?
  - Z.B. URL-Abkürzungen wie goo.gl/stNNx5 => Lange URL
- Server-side:
- 1. Hinweis in einem Konfigurationsfile des Web-Servers (bei Apache: .htaccess)
  - redirect 301 /index.html http://www.example.org/index.html
- 2. Der Server antwortet mit einem HTTP status code, z.B. 301 Moved Permanently, 303 See Other, 307 ..

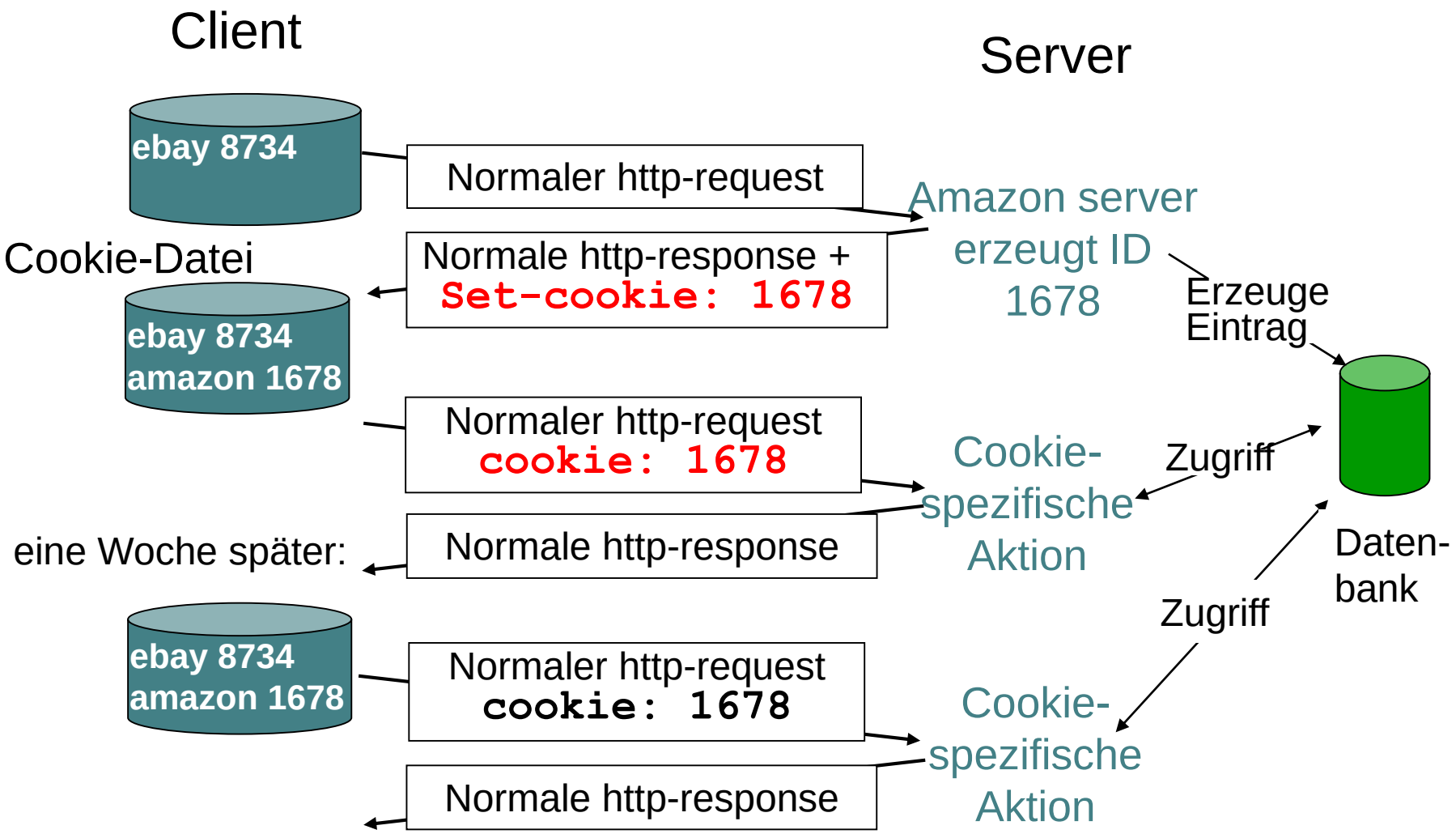
Client request	Server response
GET /index.php HTTP/1.1 Host: www.example.org	HTTP/1.1 301 Moved Permanently Location: http://www.example.org/index.asp

Benutzer-Zustand via Cookies

- Sie erlauben dem Server, die Clients wiederzuerkennen
- Vier Komponenten
  1) **Cookie Header-Zeile** in HTTP-*Response*-Nachricht
  2) **Cookie Header-Zeile** in HTTP-*Request*-Nachricht
  3) **Cookie-Datei** auf der Client-Seite, verwaltet vom Browser
  4) **Datenbank** auf der Webserver-Seite

- Ein Benutzer besucht zum ersten Mal eine Webseite …
- Wenn der erste HTTP-Request beim Server ankommt, erzeugt Server:
  - Eindeutige **ID** des Benutzers
  - Eintrag (mit dieser ID) in der eigenen Datenbank
- Dann: Bei nächster HTTP-Response-Nachricht wird eine Headerzeile “Set-Cookie” mit ID gesendet
  - Der Client sendet dann diese ID mit allen weiteren HTTP-Request-Nachrichten

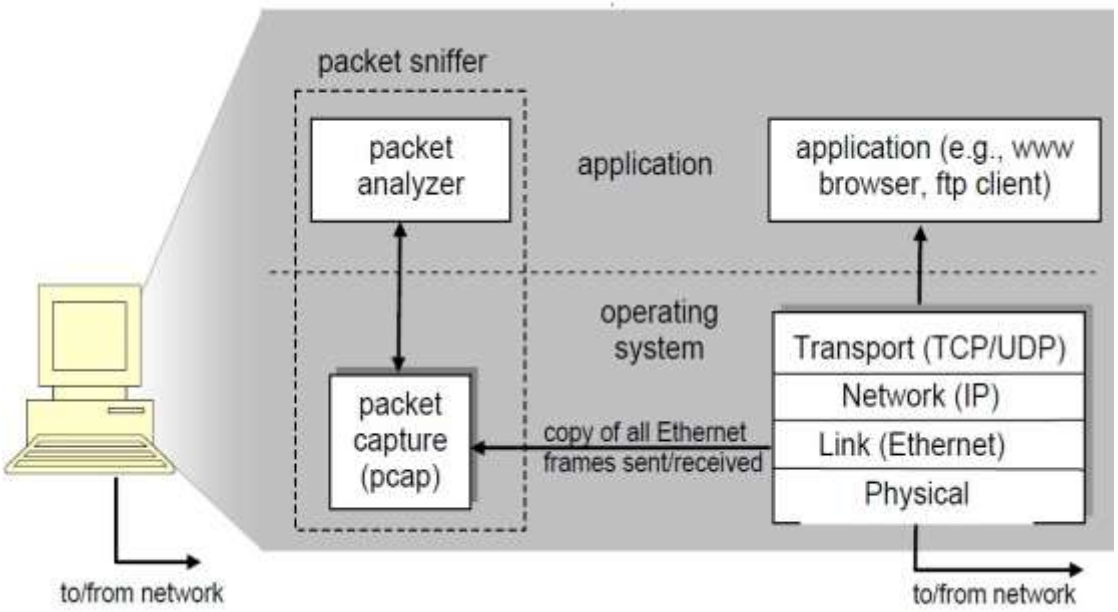
Benutzer-Zustand via Cookies /2



Wireshark Packet Sniffer

Wireshark

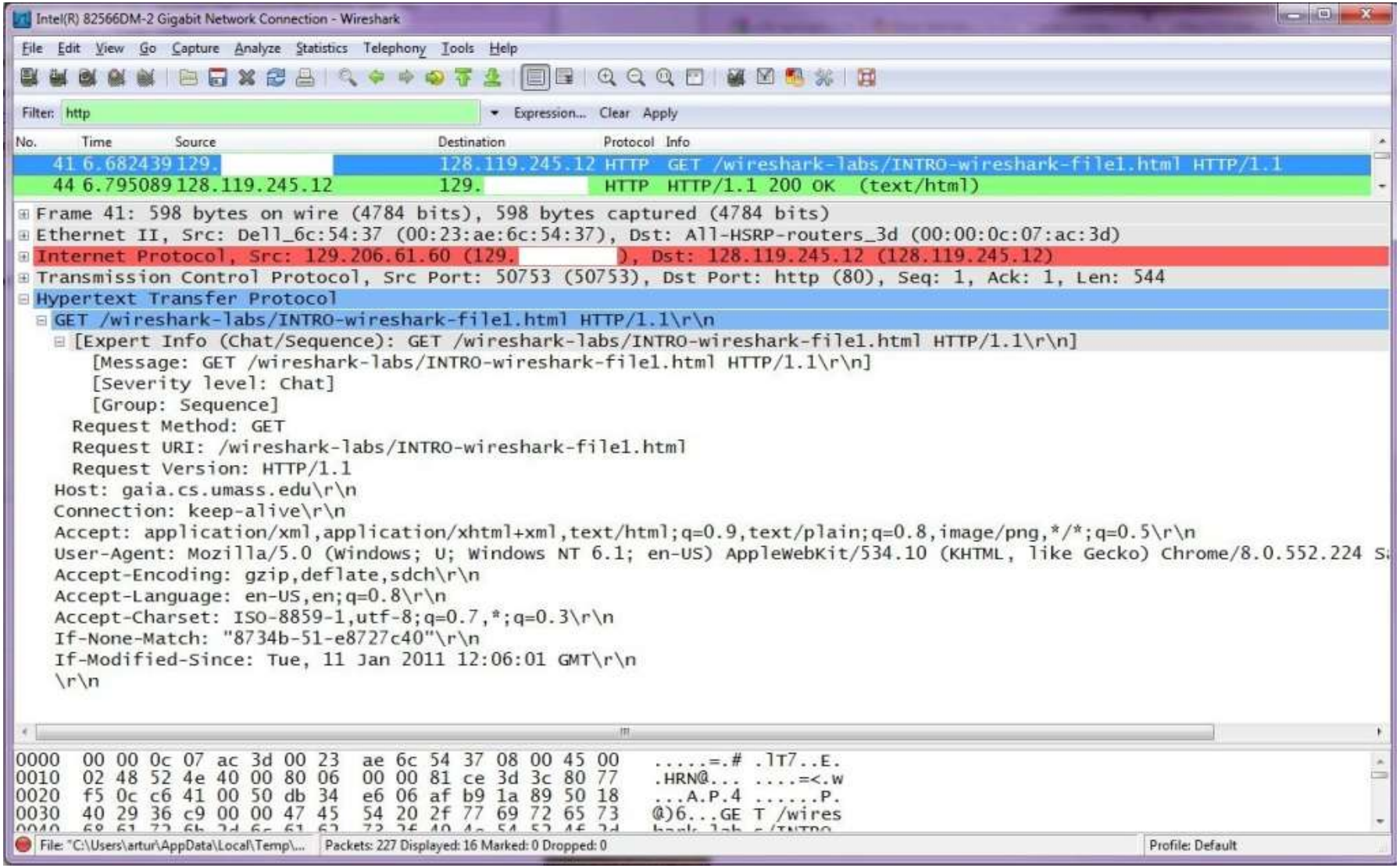
- Wireshark ist ein **Paket-Sniffer** (Paket-Schnüffler)
  - Kopiert passiv Nachrichten, die von einem Computer ausgesandt und empfangen werden
  - Kostenlos, läuft unter Linux, Windows, Mac
  - Download: http://www.wireshark.org/download.html
  - Achtung: Falls Sie Wireshark auf einem fremden (nicht eigenem) Computer installieren, überprüfen Sie zuerst, ob Sie das tun dürfen!



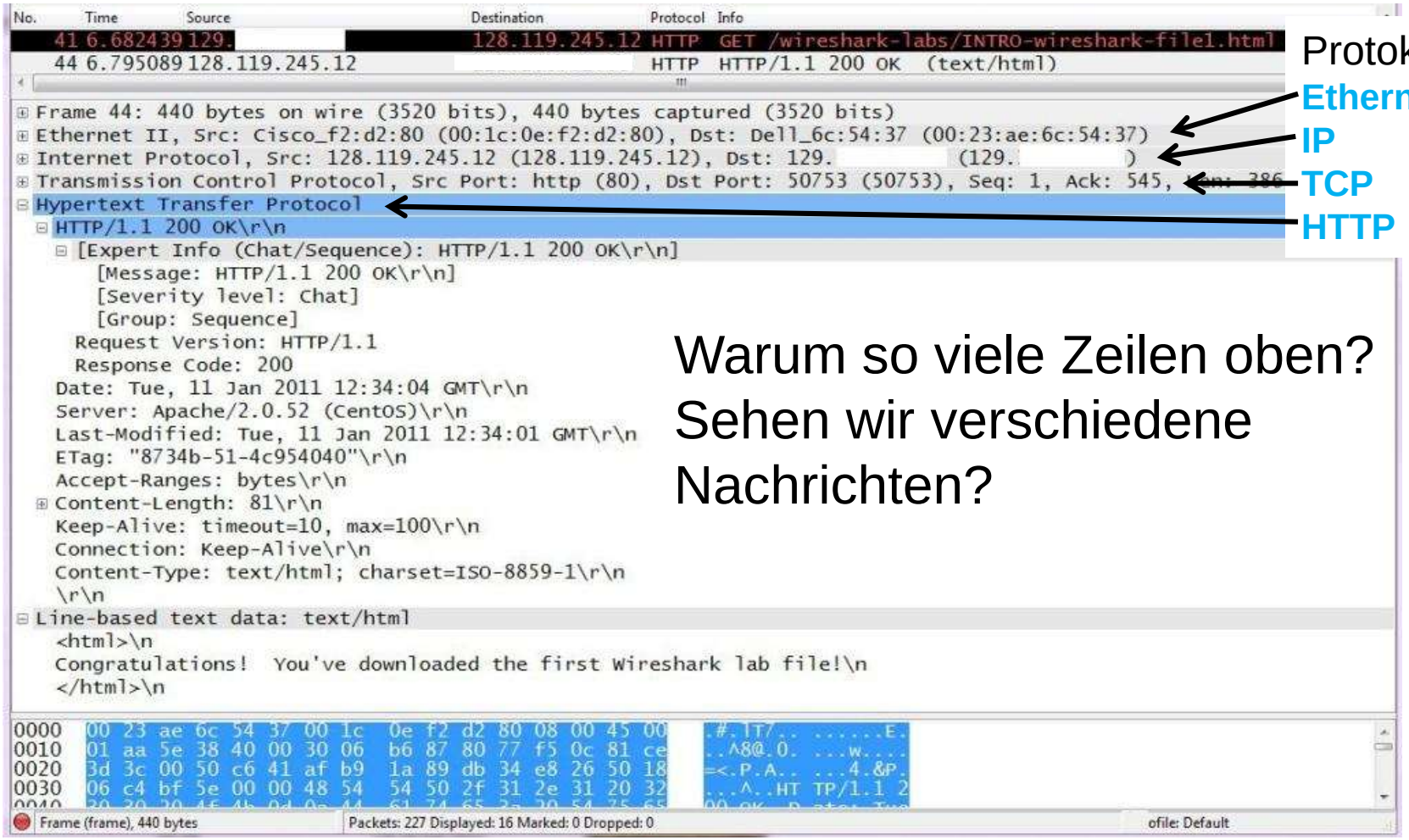
HTTP „Schnüffeln“ mit Wireshark (WS)

- Browser starten, WS starten
- Pakete aufzeichnen
  - Bei „Start capture on interface:“ auf ihre aktive Netzwerkkarte klicken
  - Im Browser eine Adresse eingeben, z.B.
    - http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html
    - Warten auf das Laden der Webseite
  - Mit „Capture/Stop“ oder Ctrl+E das Aufzeichnen beenden
- Analyse in WireShark
  - In WireShark in der Zeile „Filter“ eingeben: „http“
  - Auf das Paket mit „/wireshark-labs/INTRO-wireshark-file1.html“ klicken
    - Unten die Details bei „Hypertext …“ expandieren
  - Ähnliches für das Paket mit „200 OK (text/html)“

Die HTTP-Request-Nachricht



Die HTTP-Response-Nachricht



Protokolle:
Ethernet II
IP
TCP
HTTP

Warum so viele Zeilen oben?
Sehen wir verschiedene Nachrichten?

Zusammenfassung

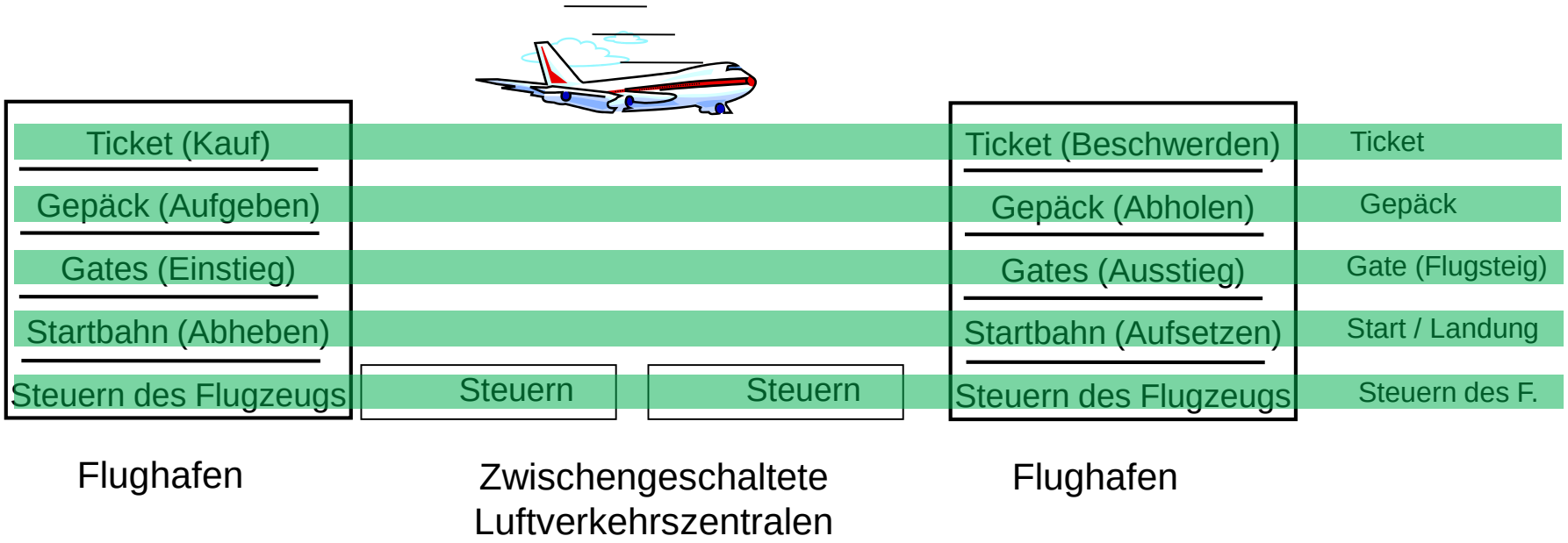
- Schichtmodell der Protokollstacks
- Grundlagen von Netzwerkanwendungen
- HTTP und das Web

- Quellen:
  - Kurose Kapitel 1 (Abschnitt 1.5)
  - Kurose / Ross – Wireshark_INTRO (Link)
  - Kurose / Ross Kapitel 2, Abschnitte 2.4, 2.7, 2.8

Danke.

Zusätzliche Folien

Analogie - Horizontale Schichten bei der Luftfahrt



- Fluggesellschaftsfunktionalität - in **Schichten**
- Jede Schicht bietet dabei zusammen mit den darunter liegenden Schichten eine bestimmte Funktionalität an
  - Einen sogenannten **Dienst**

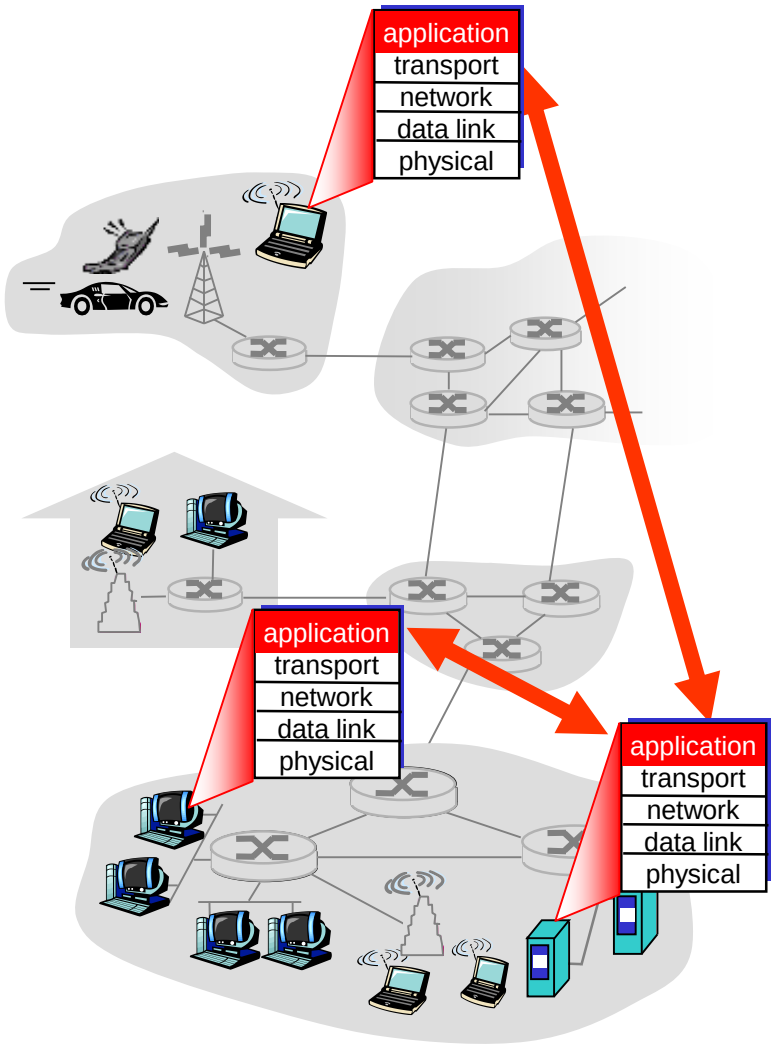
Warum Schichtenarchitektur?

- Modularisierung **erleichtert die Instandhaltung und Aktualisierung** des Systems
  - Änderung einer Schicht ist unsichtbar (transparent) für den Rest des Systems
- **Vereinfacht das Analysieren und Entwerfen von Netzwerken**
  - Explizite Struktur ermöglicht die Identifikation der Bestandteile, die Beziehungen werden herausgestellt
- Potentielle Nachteile der Schichtenarchitektur?

- Duplizierte Funktionalität einer S. in einer anderen
- Die Funktionalität einer Schicht kann Informationen benötigen, die nur in einer anderen Schicht vorliegen

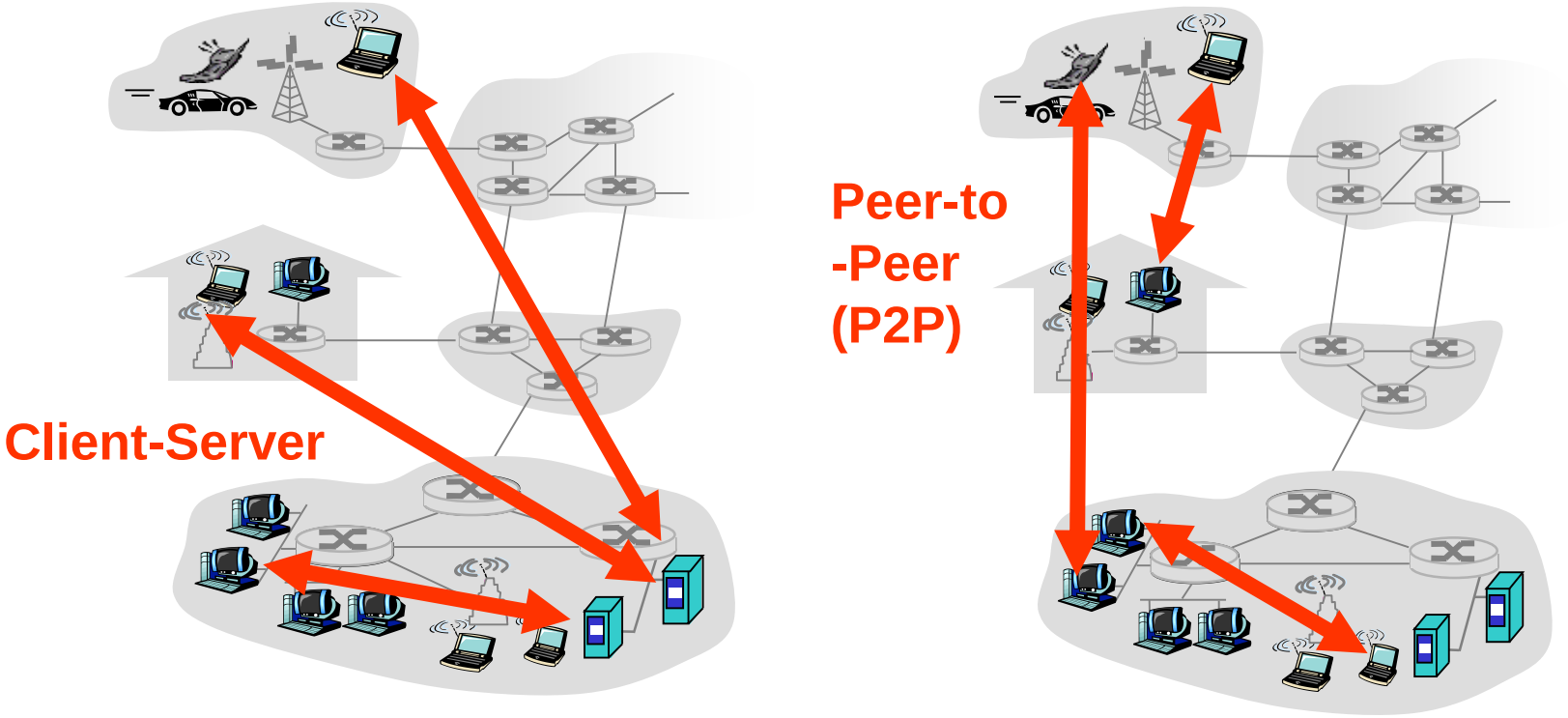
Beispiele von Netzwerkanwendungen?

- Email
- Web
- Chatting (Instant Messaging)
- Remote Login
- P2P Dateienaustausch
- Online Spiele
- Video-Streaming
- VoIP
- Soziale Netzwerke



Anwendungsarchitekturen

- Client-Server
  - Inklusive Rechenzentren und Cloud Computing
- Peer-to-Peer (P2P)
- Hybrides Modell: Client-Server und P2P



Welche Gütemerkmale bzw. Dienste braucht eine Anwendung?

- **Transfer ohne Datenverlust**
  - Einige Anwendungen (z.B. Audio) tolerieren gewissen Datenverlust, andere (z.B. ftp, ssh) brauchen 100% verlässlichen Transfer
- **Echtzeitfähigkeit**
  - Anwendungen wie VoIP, interaktive Spiele verlangen geringe Verzögerungen (**Latenzen**)
- **Hoher Durchsatz**
  - Anwendungen wie Video-Streaming, VoIP benötigen hohen Durchsatz
- **Sicherheit**
  - Anwendungen wie Homebanking, Email, … brauchen Verschlüsselung und Authentifizierung

Beispiele von Dienstanforderungen

Anwendung	Datenverlust	Durchsatz	Echtzeit
Dateitransfer	Kein Verlust	Elastisch	Nein
Email	Kein Verlust	Elastisch	Nein
Web-Browsing	Kein Verlust	Elastisch	Nein
Echtzeit Audio/Video	Tolerant	Audio: 5kbps-1Mbps Video:10kbps-5Mbps	Ja, 100’s msec
Interaktive Spiele	Tolerant	Wenige kbps	Ja, 100’s msec
Instant Messaging	Kein Verlust	Elastisch	Verschieden

Protokolle der Anwendungsschicht

- Anwendungsspezifisch; sie definieren u.a.
  - die **Art der** ausgetauschten **Nachrichten**, zum Beispiel Request-Nachrichten und Response-Nachrichten
  - die **Syntax der** verschiedenen **Nachrichtentypen**, also die Felder in der Nachricht, und ihre Kennzeichnungen
  - die **Semantik der Felder**, d.h. die Bedeutung der Information in den Feldern
  - **Regeln**, die bestimmen, **wann und wie ein Prozess Nachrichten sendet und auf Nachrichten reagiert**
- Protokolle sind oft durch RFCs festgelegt
  - Z.B. **HTTP** (Hypertext Transfer Protocol): RFC 2616
- Protokoll der Anwendungsschicht ist nur ein (kleiner) *Teil* einer Netzanwendung
  - Zu WWW gehören noch: HTML, Server+Browsersoftware