

Folien zur Veranstaltung Rechnernetze in der AI Wintersemester 2018 (Teil 3)

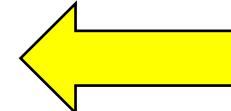
Prof. Dr. Franz Korf
Franz.Korf@haw-hamburg.de

Basierend auf der RN Vorlesung von M. Hübner

Kapitel 3: Anwendungsschicht

Gliederung

- Prinzipien von Protokollen der Anwendungsschicht
- Socket-Programmierung
- Electronic Mail (SMTP)
- Das World Wide Web (HTTP)
- Domain Name System (DNS)
- Zusammenfassung



Textbuch zu diesem Kapitel: J. Kurose & K. Ross: Computernetzwerke – Der Top-Down-Ansatz, Kapitel 2

**Folien und Abbildung teilweise aus:
J. Kurose & K. Ross: Computernetzwerke – Der Top-Down-Ansatz**

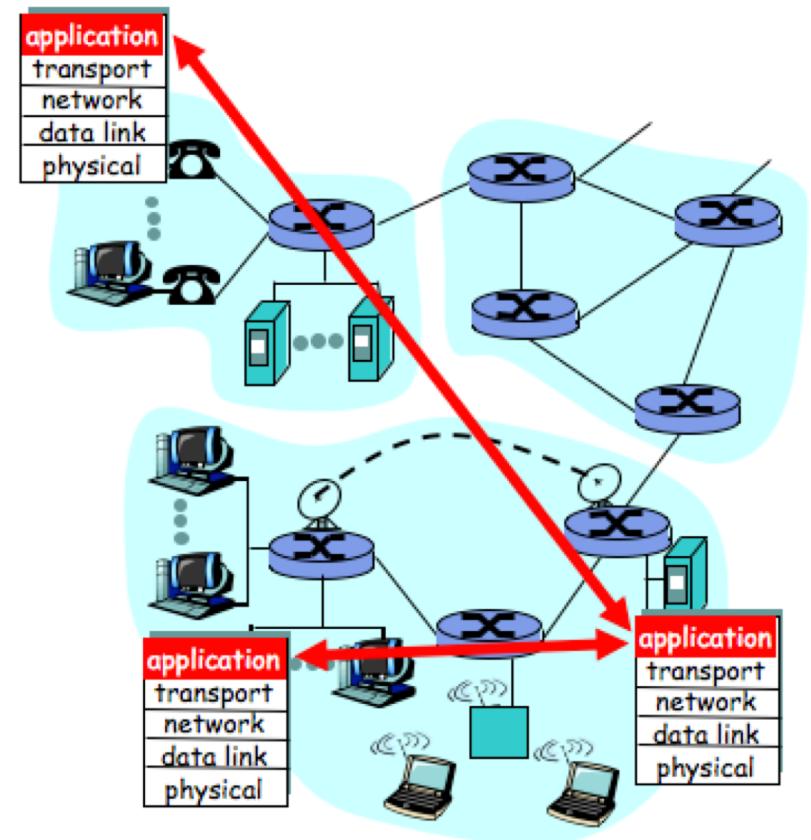
Anwendungen und Anwendungsschicht-Protokolle

Anwendungen:

- kommunizierende, verteilte Prozesse
- laufen auf Endgeräten im Benutzermodus
- tauschen Nachrichten aus, um eine verteilte Anwendung zu implementieren
(→ “verteiltes System”)

Anwendungsschicht-Protokolle

- ein Teil der Anwendung
- definieren Nachrichtenformate & Aktionen
- benutzen Kommunikationsdienste der Transportschicht (UDP, TCP)



Einordnung der Begriffe

- Ein **Prozess**: Programm, das auf einem Host läuft
- Innerhalb desselben Hosts kommunizieren 2 Prozesse in der Regel über **Interprozess-Kommunikation** (definiert durch das BS)
- Prozesse auf unterschiedlichen Hosts kommunizieren über ein **Anwendungsschicht-Protokoll**
- **User Agent**: Prozess, der mit “darüberliegendem” Benutzer und “darunterliegendem” Netzwerk kommuniziert.
Dabei implementiert er ein Anwendungsschicht-Protokoll
- Beispiele für User Agents:
 - Web: Browser
 - E-mail: Mail reader
 - Streaming audio/video: Media player

Anwendungsprotokolls

- **Typische Arten von Messages:** Request, Response
- **Syntax einer Message:** Felder der Message und ihre Darstellung
- **Semantik einer Message:** Bedeutung der Felder
- **Regeln** legen fest, wie wann welche Message von wem gesendet werden
- **Offene Protokolle:**
 - Werden in der Regel in RFCs spezifiziert
 - Unterstützen Kompatibilität und Interoperabilität
 - z.B.: HTTP, SMTP
- Proprietäre Protokolle :
 - z.B.: Skype

Das Client-Server Prinzip

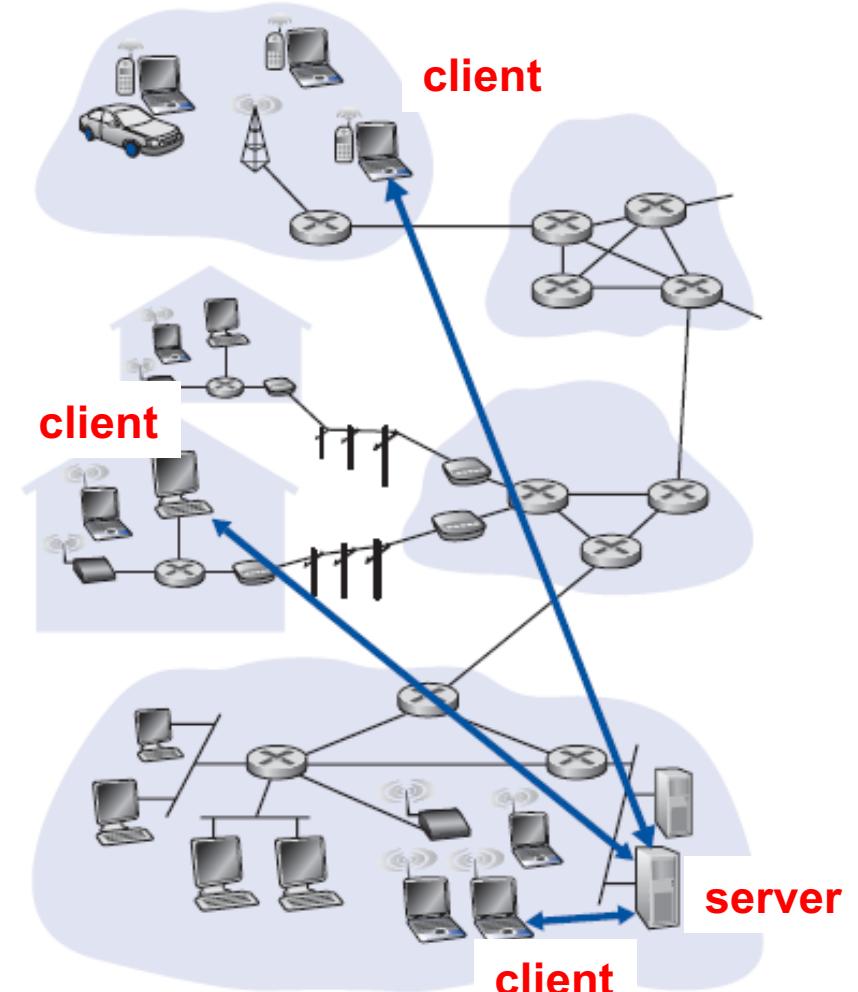
Typische Netzwerkanwendungen bestehend aus zwei Teilen

Server

- Stellt einen Dienst zur Verfügung
- z.B.: Web-Server sendet die gewünschte Web-Seite, Mail-Server liefert E-Mail, ...

Client

- verlangt vom Server eine Dienstleistung
- beginnt mit Kontaktaufnahme zum Server ("spricht zuerst")
- z.B.: Web: Client im Browser; E-mail: im Mail-Reader. ...



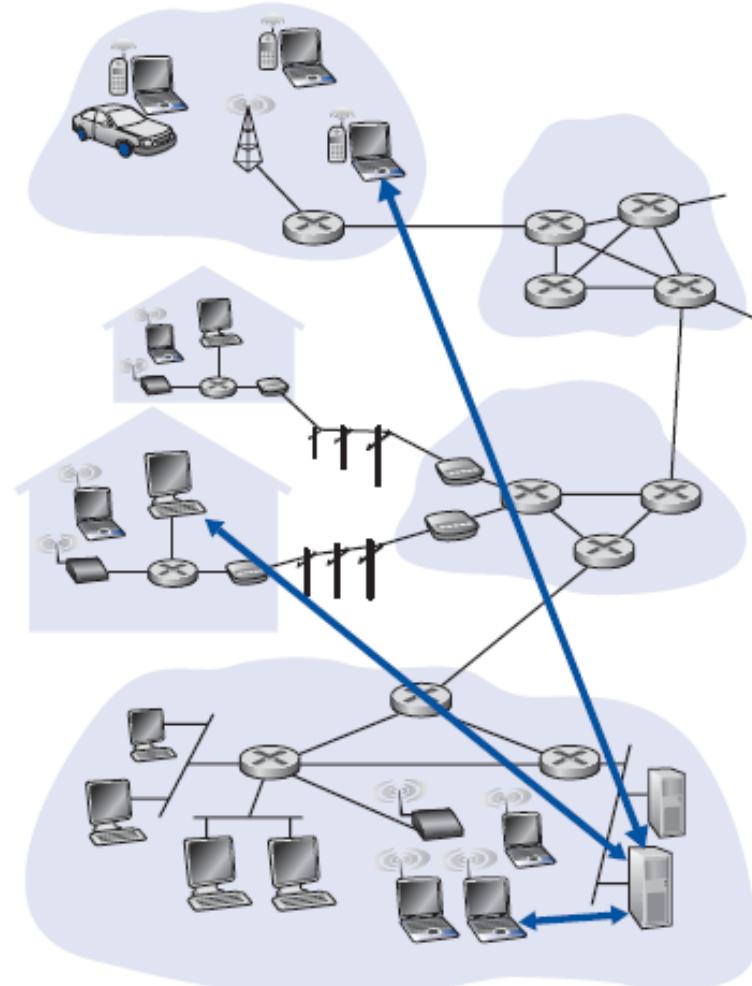
Client-Server-Architekturmerkmale

Server

- Immer eingeschaltet
- Feste IP-Adresse
- Serverfarmen, um zu skalieren

Client

- Kommunizieren mit Servern
- Sporadisch angeschlossen
- Können dynamische IP-Adressen haben
- Kommunizieren nicht direkt miteinander



Peer-to-Peer (P2P) Architekturmerkmale

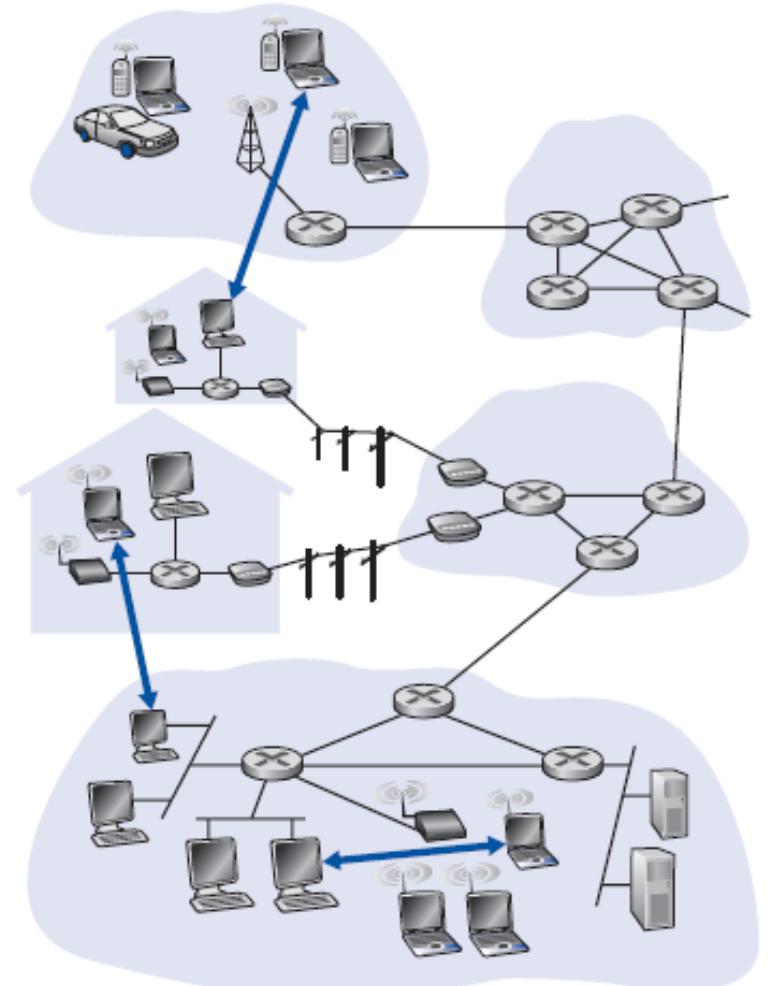
Ansatz

- Keine Server (die immer laufen)
- Beliebige Endsysteme ("Peers") kommunizieren direkt miteinander
- Peers
 - bieten anderen Peers Dienste an
 - Fordern Dienste von anderen Peers an
- self scalability: Peers bringen neue Services ein
- Peers sind nur sporadisch angeschlossen und wechseln ihre IP-Adresse

Beispiel: Gnutella

Eigenschaften

- Gut skalierbar da verteilt
- Schwer zu warten und zu kontrollieren



Kombination von Client-Server und P2P

Skype

- P2P-Anwendung für Voice-over-IP
- Zentraler Server: Adresse des Kommunikationspartners finden
- Verbindung zwischen Clients: direkt (nicht über einen Server)

Instant Messaging

- Chat zwischen zwei Benutzern: P2P
- Zentralisierte Dienste: Erkennen von Anwesenheit, Zustand, Aufenthaltsort eines Anwenders
- Benutzer registriert seine IP-Adresse beim Server, sobald er sich mit dem Netz verbündet
- Benutzer fragt beim Server nach Informationen über seine Freunde und Bekannten

Anwendungsschicht: Realisation der verteilten Interprozess-Kommunikation

API: Application Programming Interface:

- Definiert eine Schnittstelle zwischen der Anwendungsschicht und der Transport-Schicht ("Service Access Point")
- **Internet API: Socket**
Zwei Prozesse kommunizieren durch Senden von Daten ins Socket und Lesen von Daten aus dem Socket

Wie identifiziert ein Anwendungsprozess den Partnerprozess auf dem anderen Rechner?

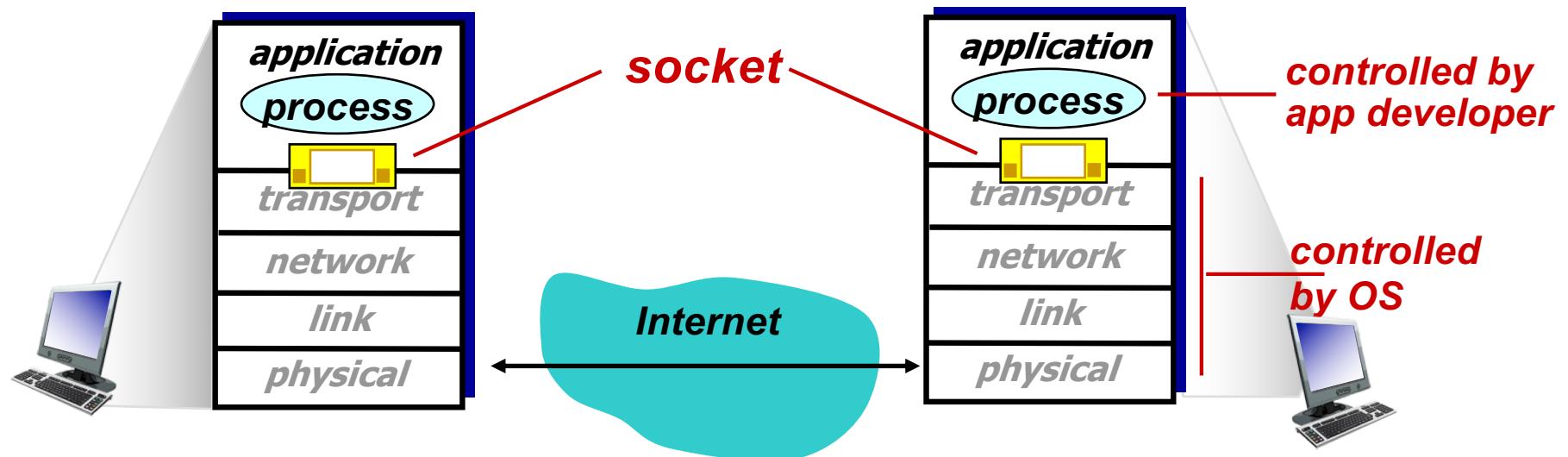
- IP-Adresse des entfernten Rechners (weltweit eindeutig)
- Portnummer: Information für den empfangenden Rechner, an welchen lokalen Prozess die Nachricht weitergeleitet werden soll.

Beispiel:

- Sende HTTP message an gaia.cs.umass.edu web server:
IP address: 128.119.245.12 port number: 80

Interprozesskommunikation über Sockets

- Prozesse senden und empfangen Messages über Sockets
- Sockets bilden die Türen zum Netzwerk
 - Ein Prozess schiebt eine Message durch ein Socket – eine Tür ins Netzwerk
 - Der sende Prozess verlässt sich auf die Infrastruktur auf der anderen Seite der Tür – die Message wird an das Socket – die Tür des empfangenden Prozesses geliefert.



Welchen Transportdienst brauchen Anwendungen?

Zuverlässigkeit der Datenübertragung (data integrity)

- Einige Anwendungen (z.B. Audio) können einige Verluste tolerieren
- Andere Anwendungen (z.B.: Dateitransfer, telnet) brauchen 100% zuverlässigen Datentransfer

Zeitanforderungen

- Einige Anwendungen (z.B. Internet Telefonie, interaktive Spiele) brauchen möglichst geringe Verzögerungen, um effektiv zu sein

Bandbreite (Übertragungsrate)

- Einige Anwendungen (z.B.: Multimedia) brauchen garantierte Datenraten, um effektiv zu sein.
- Einige Anwendungen ("elastisch") nehmen jede Übertragungsrate, die sie bekommen.

Security

Anforderungen gängiger Anwendungen

Anwendung	Datenverlust	Bandbreite	Zeitanforderungen
File Transfer	nicht tolerierbar	elastisch	nein
E-Mail	nicht tolerierbar	elastisch	nein
Web Seiten	nicht tolerierbar	elastisch	nein
Echtzeit Audio/Video	tolerierbar	Audio: 5Kb/s-1Mb/s Video: 10Kb/s-5Mb/s	ja, < 100 ms
Streaming Audio/Video	tolerierbar	wie oben	ja, wenige s
Interaktive Spiele	tolerierbar	wenige Kb/s	ja, < 100 ms
Instant Messaging	nicht tolerierbar	elastisch	ja und nein

Transporte des Internets: TCP & UDP

TCP (Transmission Control Protocol)

- Zuverlässige, reihenfolge-erhaltende Übertragung eines Byte-Stroms
- Flusskontrolle (Überlastungsvermeidung des Empfängers)
- Überlastkontrolle (Überlastungsvermeidung des Netzwerks)
- Verbindungsaufbau

UDP (User Datagram Protocol)

- Unzuverlässiger Datentransfer einzelner Datenpakete
- geringer Overhead
- Nicht geboten: Verbindungsaufbau, Flusskontrolle, Überlastkontrolle

Weder TCP noch UDP bieten Garantien über Verzögerung und Übertragungskapazität

Einsatz der Transportprotokolle

Anwendung	Anwendungsprotokoll	Transportprotokoll
e-mail	SMTP [RFC 821]	TCP
remote terminal access	TELNET[RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP or RTP[RFC 3550]	TCP or UDP
remote file server	NFS [RFC 3530]	TCP or UDP
IP Telefonie	SIP[RFC 3261] + RTP	TCP or UDP

TCP & Security

Weder TCP noch UDP

- Unterstützen Security Funktionalitäten wie Verschlüsselung
- Z.B. werden Passwörter im Klartext übertragen

SSL (Secure Sockets Layer) und sein Nachfolger **TLS (Transport Layer Security)** bieten

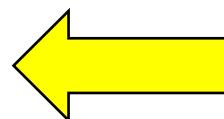
- Verschlüsselung
- End-Punkt Authentifizierung
- Daten Integrity

Integration von SSL

- Anwendungen benutzen SSL Libraries, die auf TCP aufsetzen
- Security ist in die Anwendung integriert, kein Transport Dienst

Kapitel 3: Anwendungsschicht

Gliederung

- Prinzipien von Protokollen der Anwendungsschicht
- Socket-Programmierung 
- Electronic Mail (SMTP)
- Das World Wide Web (HTTP)
- Domain Name System (DNS)
- Zusammenfassung

Vorbemerkungen

Inhalt des Abschnitts:

- Entwicklung von Client/Server Applikationen, die über Sockets kommunizieren

Socket API

- Eingeführt im BSD4.1 UNIX, 1981
- Durch unzählige Anwendungen genutzt
- Client/Server-Paradigma
- Zwei Arten des Transportdienstes über die Socket API:
 - Unzuverlässiger Datagrammdienst (→ UDP)
 - Zuverlässiger Bytestrom (→ TCP)

TCP Socket Programmierung

Aus Sicht der Anwendung:

- TCP stellt zuverlässige, reihenfolge-erhaltende Byte-Übertragung (“pipe”) zwischen Client und Server bereit

Grundlegende Schritte:

- Der Client muss den Server kontaktieren
 - Der Server-Prozess muss als Erster laufen
 - Der Server muss ein Socket für dem Client-Kontakt erzeugt haben

TCP Socket Programmierung (Fortsetzung)

Grundlegende Schritte:

- Client kontaktiert den Server durch:
 - Erzeugung eines TCP- Sockets
 - Spezifizierung der IP-Adresse + Port-Nummer des Server- Prozesses
- Nach Erzeugung des Client-Socket: Client baut Verbindung zum Server auf (3-Wege-Handshake)
- Wenn durch den Client kontaktiert, erzeugt der Server einen neuen “Arbeits”-Socket, mit dem danach Server und Client kommunizieren
 - Erlaubt dem Server, gleichzeitig mit vielen Clients zu kommunizieren (durch fork / Threads)

TCP Socket Programmierung

Beteiligte Sockets

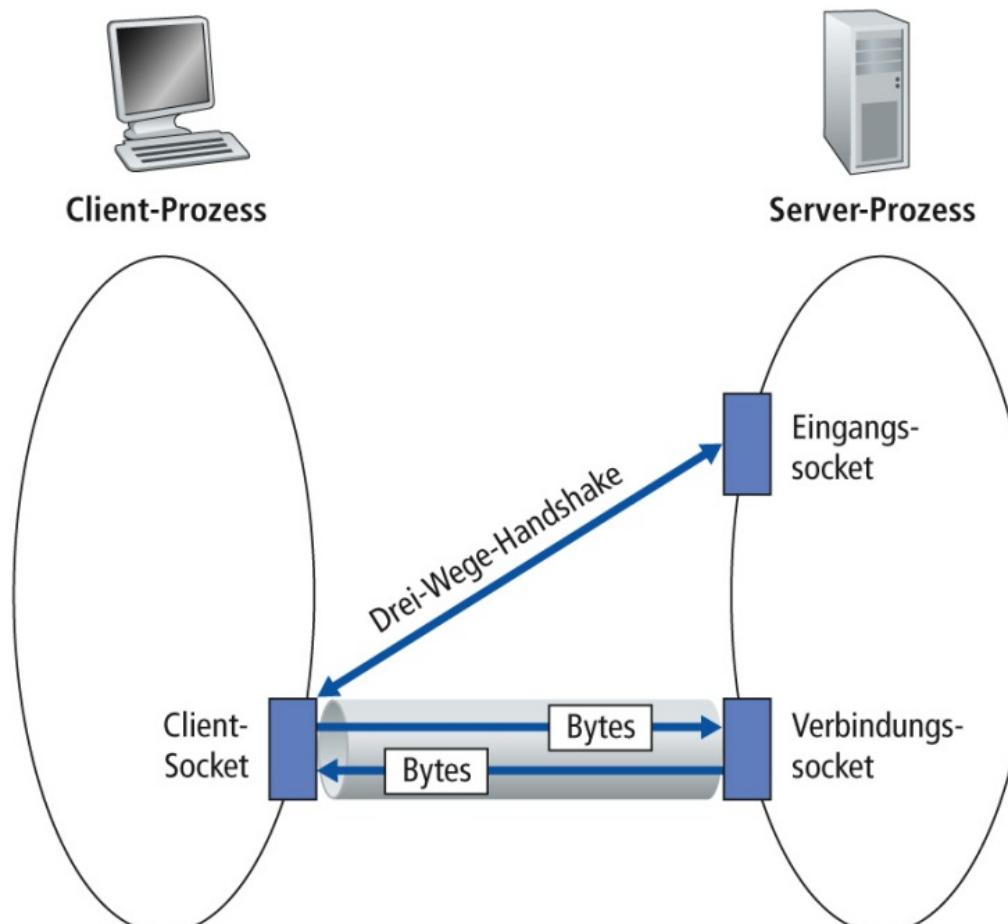


Abbildung 2.29: Der TCPServer-Prozess besitzt zwei Sockets.

TCP Socket Programmierung

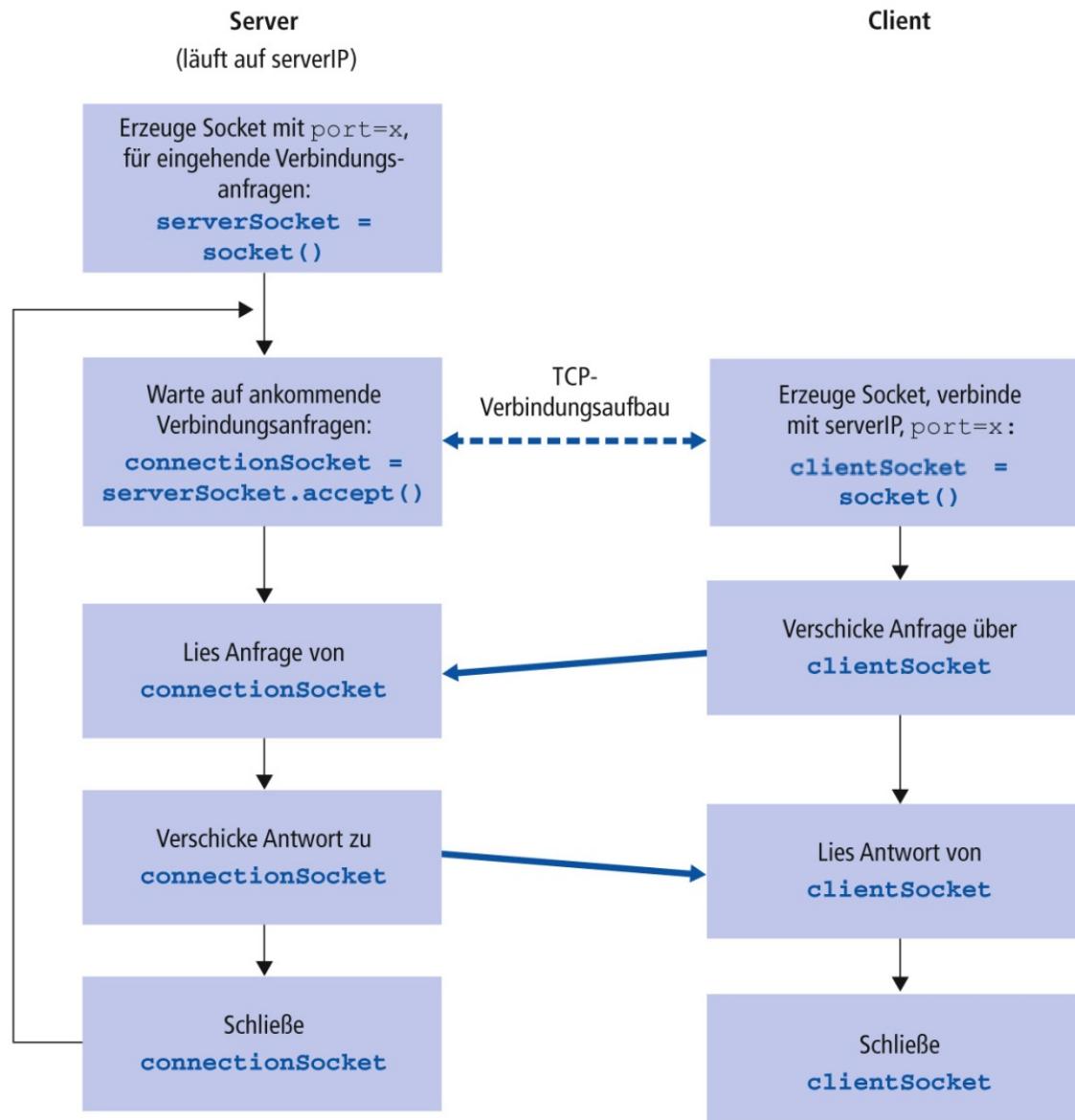


Abbildung 2.30: Die Client-Server-Anwendung auf Basis von TCP.

JAVA: Socket-Methoden (TCP)

Klasse **Socket** im Package **java.net**:

- Die Daten werden über einen Stream versendet / empfangen!

Konstruktor: **public Socket (String host, int port)**

- Creates a stream socket and connects it to the specified port number on the named host

Konstruktor: **public Socket (InetAddress address,int port)**

- Creates a stream socket and connects it to the specified port number at the specified IP address

public InputStream getInputStream()

- Returns an input stream for this socket.

public OutputStream getOutputStream()

- Returns an output stream for this socket.

public void close()

JAVA: ServerSocket-Methoden (TCP)

Klasse **ServerSocket** im Package **java.net**:

- Zusätzliche Methode für das "Lauschen"

Konstruktor: **public ServerSocket (int port)**

- Creates a server socket, bound to the specified port.

public Socket accept()

- Listens for a connection to be made to this socket and accepts it. The method blocks until a connection is made. A new Socket is created for that connection and returned.

public void close()

Beispiel zur TCP Socket Programmierung

Beispiel: Client-Server Anwendung mit JAVA:

- Client liest eine Zeile von standard input (**inFromUser stream**), sendet die Zeile zum Server via socket (**clientSocket.getOutputStream()**)
- Server liest Zeile vom socket
- Server konvertiert Zeile zu Großbuchstaben, sendet Zeile zurück zum Client
- Client liest modifizierte Zeile vom socket (**clientSocket.getInputStream()**) und gibt sie aus

Beispiel zur TCP Socket Programmierung

Server (running on hostid)

```
create socket,  
port=x, for  
incoming request:  
welcomeSocket =  
    ServerSocket(x)
```

```
wait for incoming  
connection request  
connectionSocket =  
    welcomeSocket.accept()
```

```
read request from  
connectionSocket.getInputStream()
```

```
write reply to  
connectionSocket.getOutputStream()
```

```
close  
connectionSocket.close()
```

Client

```
create socket,  
connect to hostid, port=x  
clientSocket =  
    Socket(hostid,x)
```

```
send request using  
clientSocket.getOutputStream()
```

```
read reply from  
clientSocket.getInputStream()  
close  
clientSocket.close()
```



UDP Socket Programmierung

Aus Sicht der Anwendung:

- UDP überträgt in einem unzuverlässigen Datentransfer eine Gruppe von Bytes (Datagramme) zwischen Client und Server
 - Keine Verbindung zwischen Client und Server
 - Unzuverlässiger Datentransfer: Vertauschung und Verlust von Segmenten möglich
- Verwaltungsaufwand:
 - + Weniger Verwaltungsoverhead für direkte Kommunikation
 - Der Client fügt zu jeder Nachricht die Ziel-IP-Adresse und Zielportnummer hinzu
 - Der Server muss die IP-Adresse und Portnummer des Clients aus jeder empfangenen Nachricht extrahieren, falls er ein Antwortpaket adressieren möchte.

UDP Socket Programmierung (Fortsetzung)

Grundlegende Schritte:

- Server erzeugt ein Socket, über das er auf Pakete wartet. – Danach kommen nur noch UDP Pakete über das Socket rein.
- Client erzeugt ein Socket, über das er den Server kontaktiert. - Danach geht es schon mit dem Senden los.

UPD Socket Programmierung

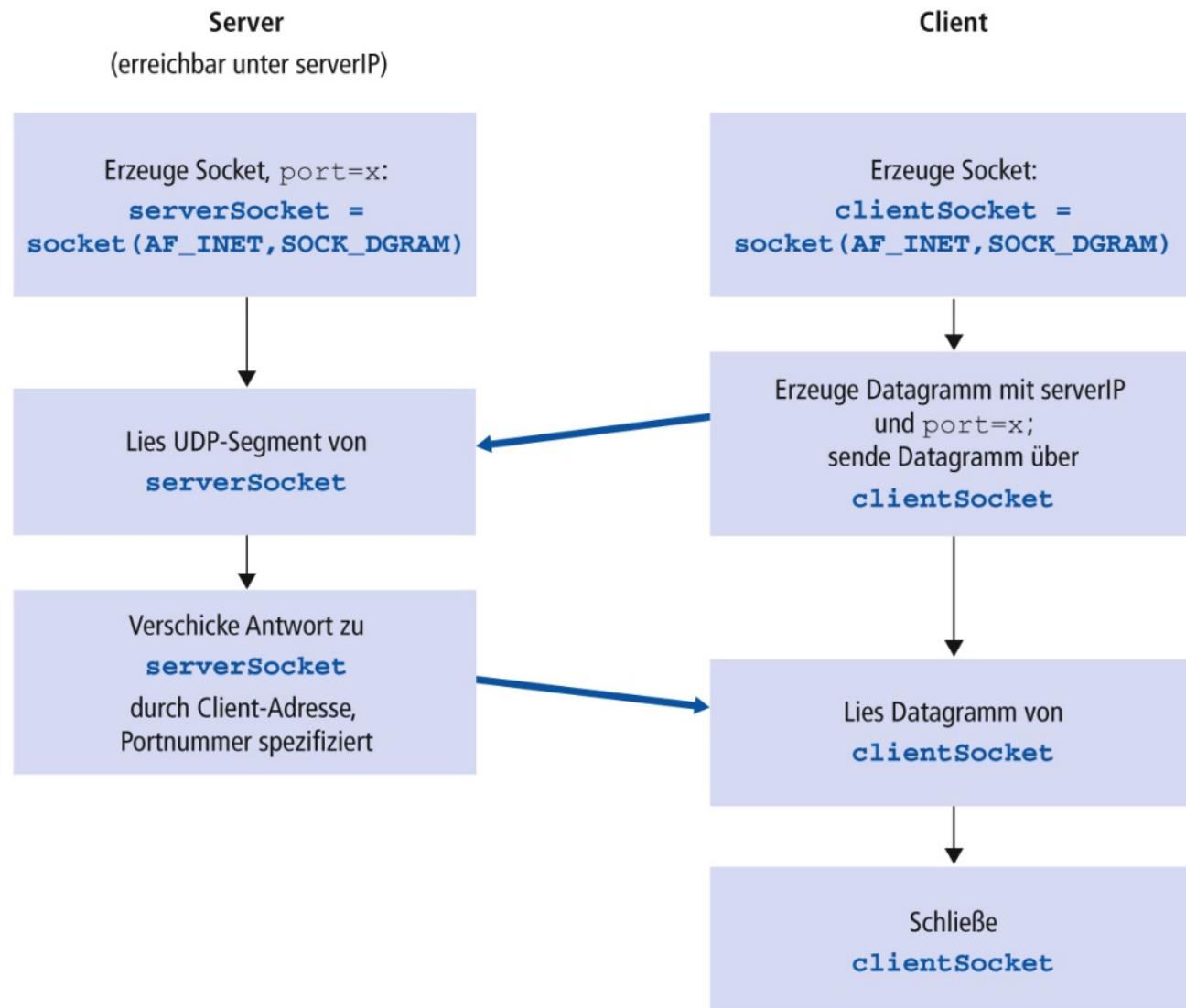


Abbildung 2.28: Die Client-Server-Anwendung auf Basis von UDP.

JAVA: Socket-Methoden (UDP)

Klasse **DatagramSocket** im Package **java.net**:

- Die Daten werden in ein Paket verpackt, das ALLE Informationen über den Sender und die Daten enthält!

Konstruktor: **public DatagramSocket ()**

- Creates a datagram socket and binds it to any available port (→ send)

Konstruktor: **public DatagramSocket (int port)**

- Creates a datagram socket and binds it to the specified port on the local host machine (→ receive)

public void send (DatagramPacket packet)

- Sends a datagram packet from this socket.

public void receive (DatagramPacket packet)

- Receives a datagram packet from this socket. This method blocks until a datagram is received.

public void close()

JAVA: Socket-Methoden (UDP)

Klasse **DatagramPacket** im Package **java.net**:

Konstruktor: **public DatagramPacket (byte[] buf, int length,**

InetAddress destAddress,int destPort)

- Creates a datagram packet for **sending** packets of length length to the specified port number on the specified host.

Konstruktor: **public DatagramPacket (byte[] buf, int length)**

- Creates a DatagramPacket for **receiving** packets of length length.

public InetAddress getAddress()

- Returns the IP address of the machine to which this datagram is being sent or from which the datagram was received.

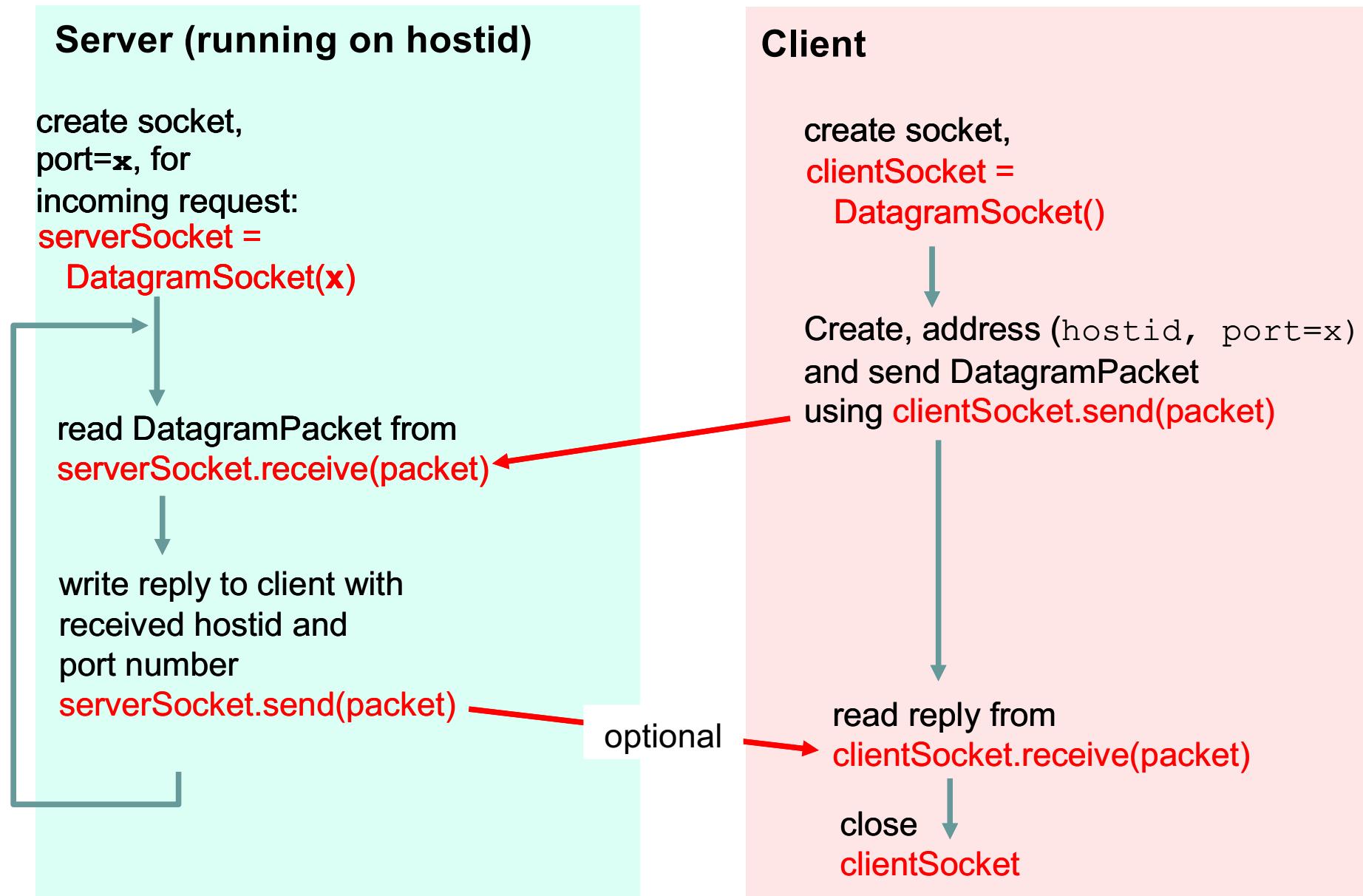
public int getPort()

- Returns the port number on the remote host to which this datagram is being sent or from which the datagram was received.

public byte[] getData()

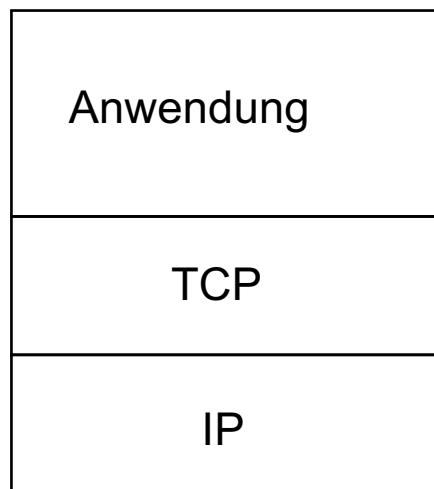
- Returns the data buffer

Beispiel zur UDP Socket Programmierung



JAVA: TLS (SSL) - Sockets

- TCP liefert keine Security Features
- Fatal für sicherheitskritische Anwendungen wie “Kaufen im Netz“ etc.
- Netscape hat 1995 mit SSL (Secure Socket Layer) angefangen (proprietär, Fokus http, nicht sicher), Weiterentwicklung bis zur Version 3.0
- Ab 1999 TLS (Transport Layer Security), durch IETF standardisiert.



“Normale” Anwendung



Anwendung mit TLS

**Hier: TLS als
Black Box**

JAVA: TLS (SSL) - Sockets

Klasse **SSLSocket** und **SSLSocketFactory** im Package **javax.net.ssl** :

- SSLSockets verwendbar wie (TCP-) Sockets
- Vor Weitergabe (an TCP) werden die Daten verschlüsselt und gegen Veränderung geschützt
- Erzeugung über SSLSocketFactory, da verschiedene Implementierung mit unterschiedlichen Sicherheitseinstellungen möglich sind

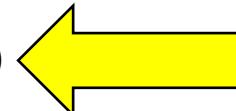
Beispiel:

```
SSLSocketFactory factory =  
    (SSLSocketFactory) SSLSocketFactory.getDefault();
```

```
SSLSocket clientSocket =  
    (SSLSocket)factory.createSocket(String host,int port);
```

Kapitel 3: Anwendungsschicht

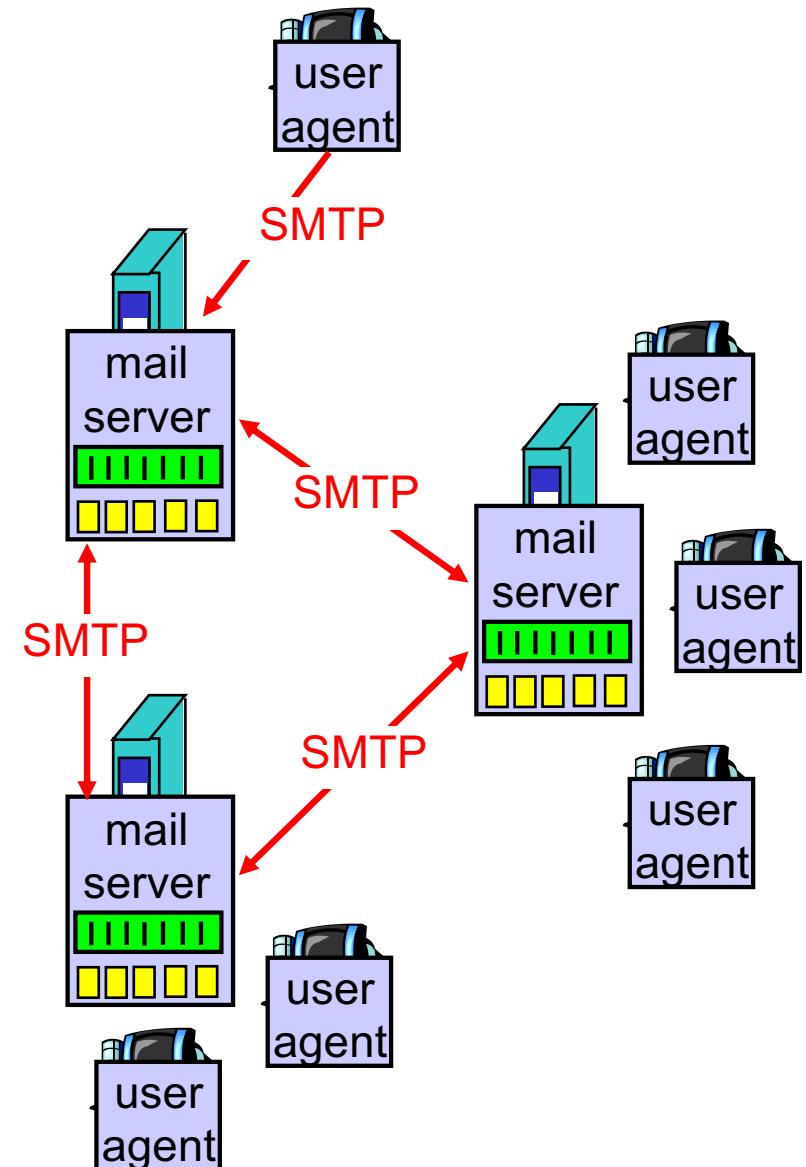
Gliederung

- Prinzipien von Protokollen der Anwendungsschicht
- Socket-Programmierung
- Electronic Mail (SMTP) 
- Das World Wide Web (HTTP)
- Domain Name System (DNS)
- Zusammenfassung

Grundlegende Elemente

User agent

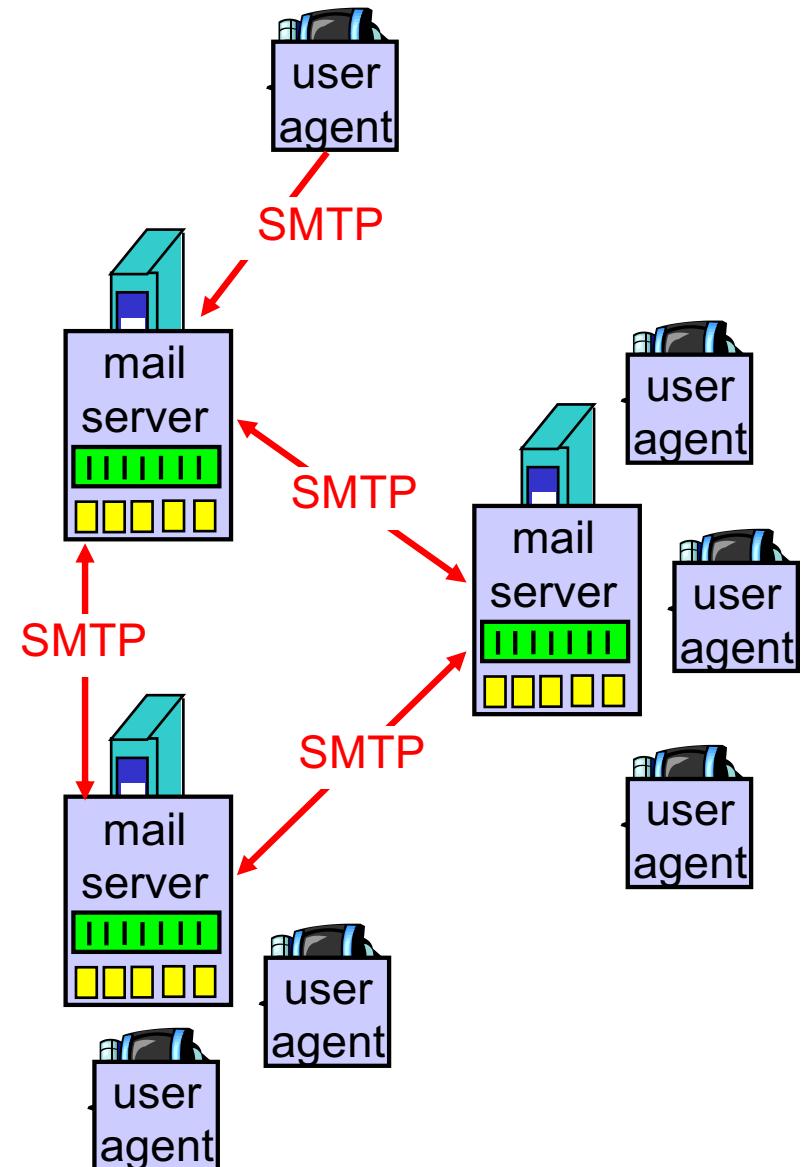
- “Mail Reader”-Programme
- Verfassen, Bearbeiten, Lesen, Speichern von Mails
z.B. Outlook, Thunderbird, ...
- Senden Nachrichten zu einem Mailserver (per SMTP oder http)
- Holen Mails vom Mailserver ab (POP3, IMAP oder http)
 - eingehende Nachrichten werden im Mailserver gespeichert
 - User Agent darf offline sein



Grundlegende Elemente

Mailserver

- **Mailbox:** enthält eingegangene Nachrichten für Benutzer
- **Message queue:** Warteschlange von zu sendenden Nachrichten
- Üblich: **Direkte Kommunikation** zwischen dem Server, der das Postfach des Senders verwaltet, und dem Server, der das Postfach des Empfängers verwaltet.
- Mail Server verwaltet die Mails: z.B.: Erneute Zustellung von Mails im Falle eines Fehlers.
- Mail Server sollte der Regel immer online sein.
- Kommunikation zwischen Servern mit SMTP

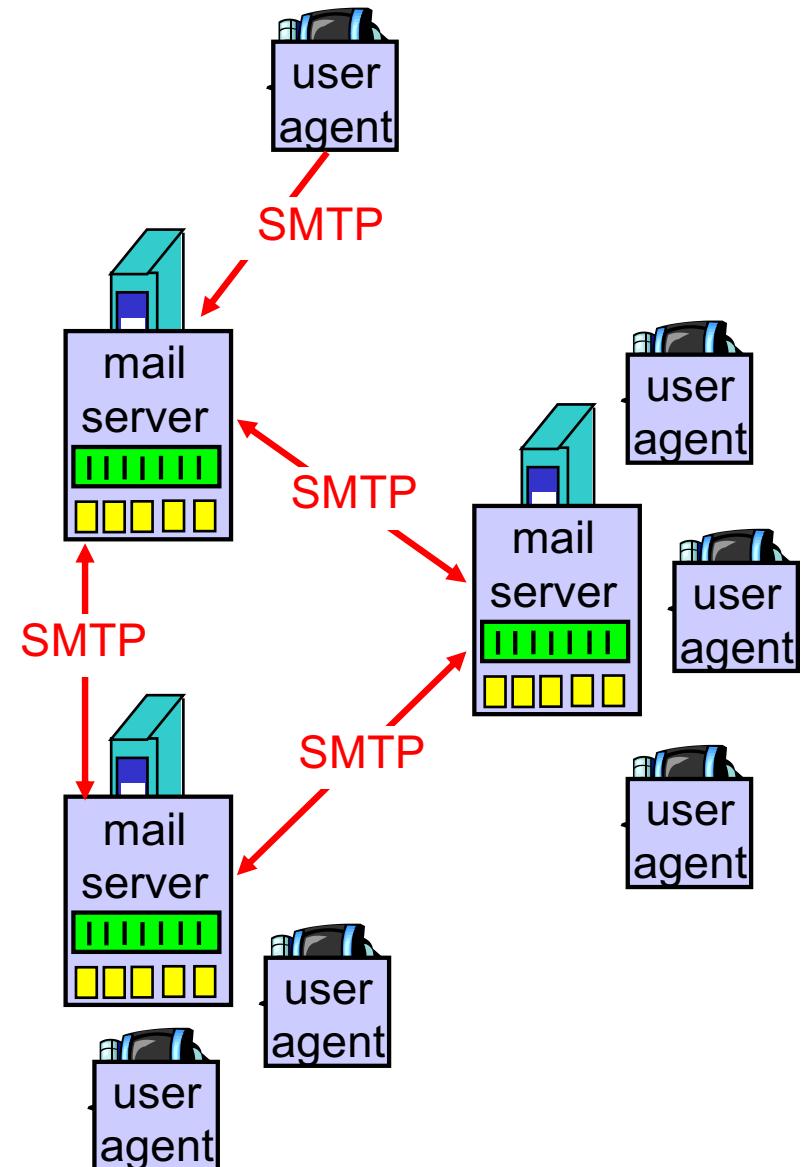


Grundlegende Elemente

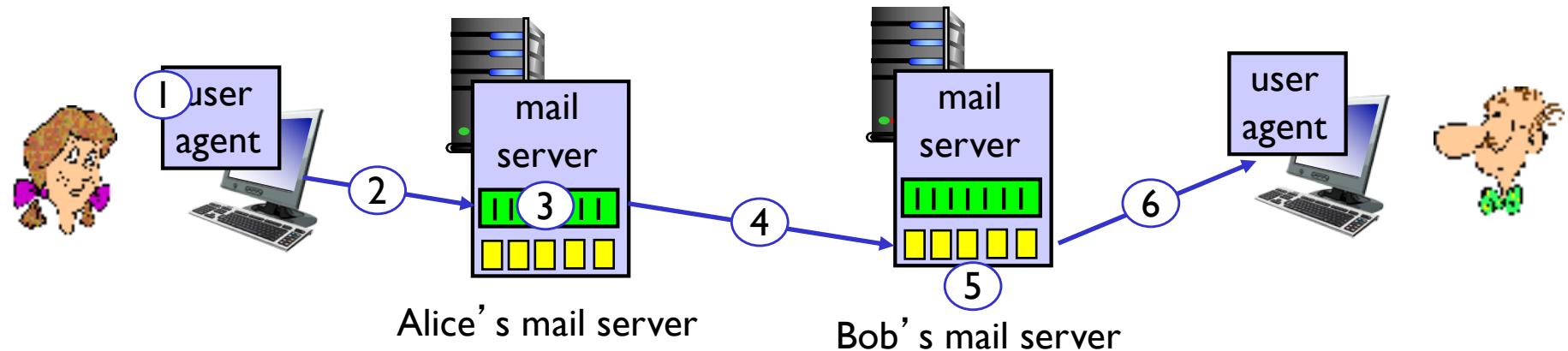
SMTP (Simple Mail Transfer Protocol)

- Einsatz:
 - Zwischen Mailservern zum Weiterleiten von Mails
 - Zwischen User Agent und Mailserver zum Senden von Mails
- Client Server Protokoll:
 - Client: Sender Mail Server
 - Server: Empfangender Mail Server
 - Push Protokoll: Protokoll zum senden von Daten zum Server¹
- Verwendet TCP

¹ Im Gegensatz zu Pull Protokollen

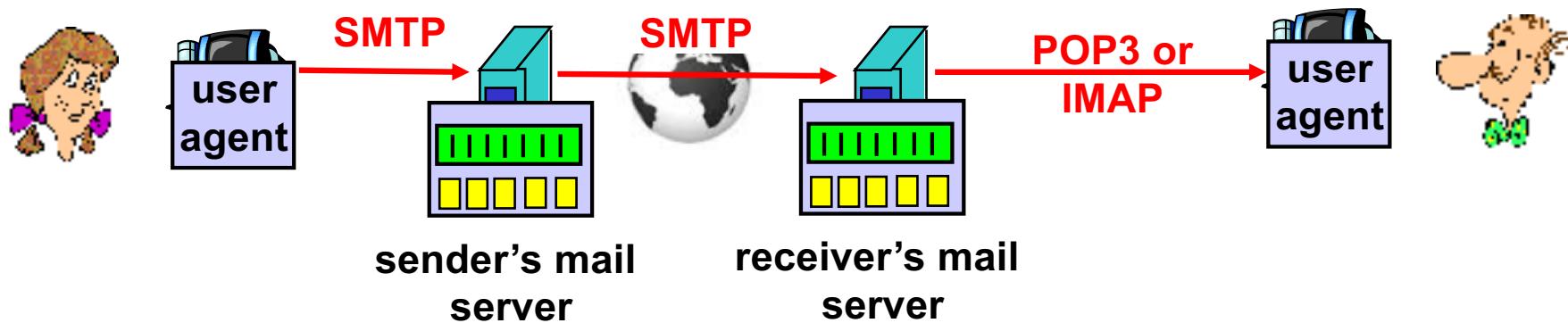


Die Reise einer Mail



1. Alice schreibt die Mail auf ihrem UA und gibt Bobs Mailadresse als Empfänger an.
2. Alice's UA sendet die Mail an Ihren Mailer Server, der sie in die Message Queue legt.
3. Der Clientteil des Mail Servers von Alice öffnet eine TCP Verbindung zum Serverteil des Mail Servers von Bob.
4. Der Clientteil des Mail Servers von Alice sendet die Mail via SMTP an den Serverteil des Mail Servers von Bob.
5. Bobs Mail Server legt die Mail on Bobs Postfach.
6. Wenn Bob seinen UA startet, liest er die Mail aus seinem Postfach aus.

Protokolle „auf der Reise einer Mail“



- Sende Mail vom Client zum Server und von Quell-Server zum Ziel-Server via SMTP
- Mailzugriffs-Protokoll: Abholen vom Mailserver
 - POP: Post Office Protocol [POP3: RFC 1939]
Autorisation (User Agent ↔ Mailserver) und Download
 - IMAP: Internet Mail Access Protocol [RFC 1730]
größere Funktionalität (wesentlich komplexer)
Nachrichten bleiben i.d.R. auf dem Server
- HTTP: Webmail (*Outlook Web Access etc.*)

SMTP-Protokoll [RFC 821 / 5321]

- benutzt **TCP** für zuverlässigen Datentransfer über Port 25 (bei TLS (SSL)-Verschlüsselung Port 465 oder 587)
- direkte Verbindung von sendendem Mailserver zu empfangendem Mailserver
- 3 Phasen:
 - Handshaking
 - Transfer von Nachrichten
 - Schließen der Verbindung
- Befehl/Antwort – Schema
 - Befehle: ASCII Text
 - Antworten: Statuscode und Beschreibung
- Gesamte (!) Nachrichten müssen im 7-bit ASCII-Format vorliegen!
- Persistente TCP Verbindung: Mehrere Mails über die selbe Verbindung.

SMTP Beispiel-Sitzung

S: 220 hamburger.edu

S: Server

C: HELO crepes.fr

C: Client

S: 250 Hello crepes.fr, pleased to meet you

C: MAIL FROM: <alice@crepes.fr>

S: 250 alice@crepes.fr... Sender ok

C: RCPT TO: <bob@hamburger.edu>

S: 250 bob@hamburger.edu ... Recipient ok

C: DATA

S: 354 Enter mail,end with "." on a line by itself

C: Do you like ketchup?

C: How about pickles?

C: .

<CR><LF>.<CR><LF> zeigt das Nachrichtenende an!

S: 250 Message accepted for delivery

C: QUIT

S: 221 hamburger.edu closing connection

SMTP Sicherheit geht gegen 0

- Keine Gewährleistung von Geheimhaltung / Integritätssicherung
- Zusätzliche Sicherheitsmechanismen nötig:
 - **ESMTP** (Extended SMTP): Statt HELO wird EHLO verwendet
→ Aushandlung des Sicherheitsstandards
 - **Authentifikation**: Nur angemeldete Benutzer dürfen den Mailserver benutzen! Konzepte:
 - SMTP-After-POP: vor dem Senden Post abholen (Anmeldung!)
 - SMTP-Authentifikation: AUTH-Befehl mit Benutzername + Passwort [RFC 4954, 4616]
- **Verschlüsselung** der TCP-Verbindung:
 - **SMTPS**: Verschlüsselung mit TLS (SSL) über eigenen Port 465
 - **StartTLS**: Wechsel zu TLS (SSL) in aktueller TCP-Verbindung [RFC 3207]
 - Verwendet i.d.R. Port 587

Mail-Nachrichtenformat [RFC 822 / 2822]

SMTP RFC 821: Protokoll für den Austausch von E-Mail-Nachrichten

RFC 822: Definition eines Nachrichtenformats
(wichtig u.a. für User-Agenten)

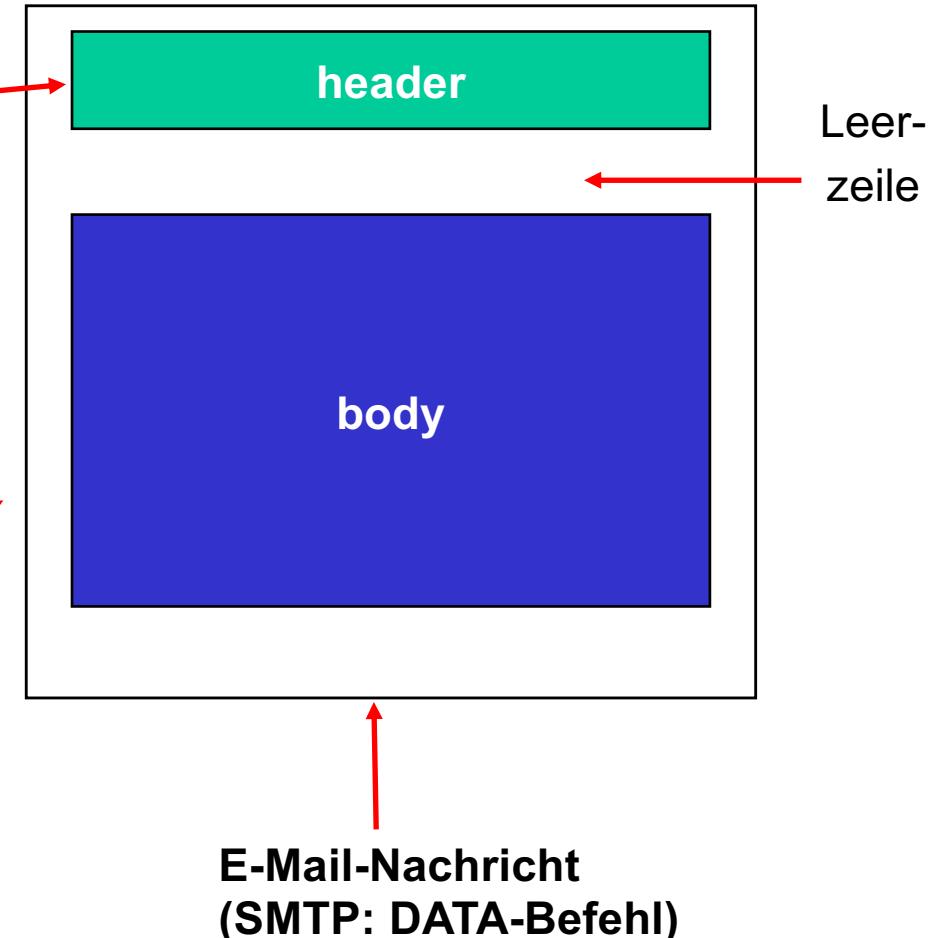
Header-Zeilen, z.B.

- From:
- To:
- cc:
- Subject:

sind keine SMTP-Befehle!

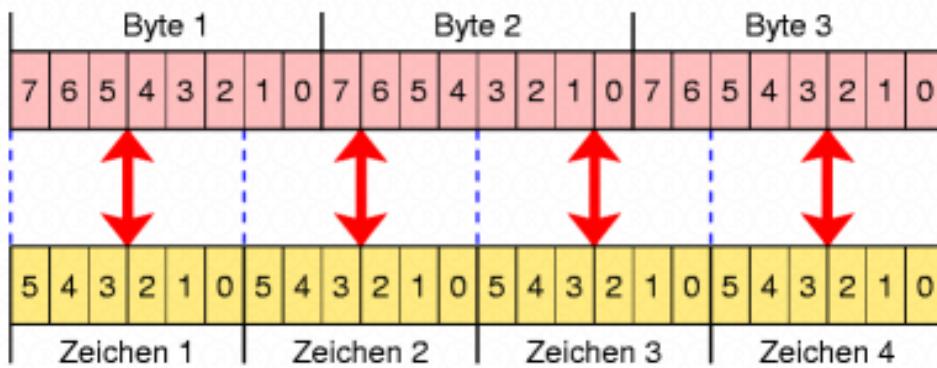
Rumpf (“body”)

- die eigentliche Nachricht



SMTP-Nachrichtencodierung

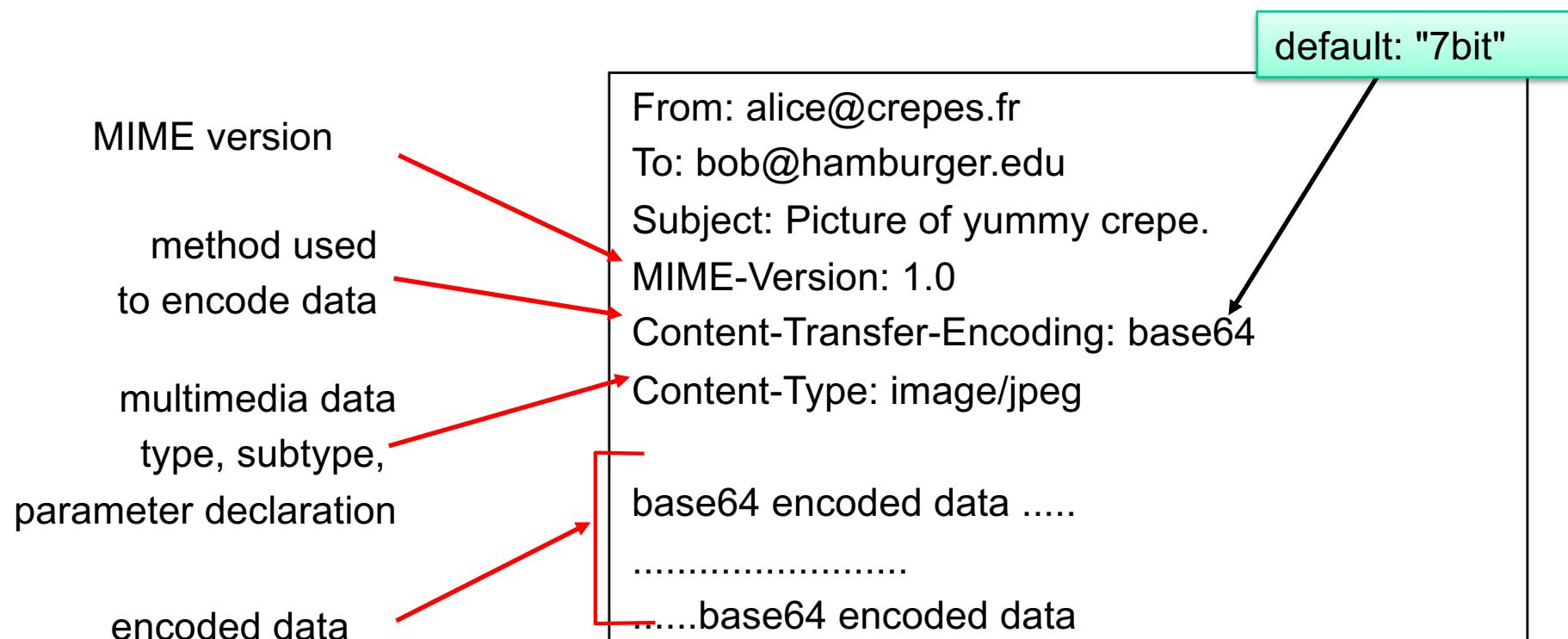
- SMTP – Nachrichten (header & body) müssen 7-bit ASCII-codiert sein!
- Problematisch: <CR><LF>. <CR><LF> im Nachrichtentext (*wieso?*)
→ Hinzufügen eines zusätzlichen ".", wenn eine DATA-Zeile zufällig mit "." beginnt
- Wie werden Umlaute, Sonderzeichen, Word-Dokumente, Bilder, Videos übertragen?
- Nachrichten werden nach 7-bit ASCII transformiert
 - übliche Codes: **base64**, quoted printable



- Teile 3 Byte in vier 6 Bit Einheiten
- Bilde die 64 Kombinationen auf A-Z a-z 0-9 + / ab
- Das = Zeichen wird beim Padding benutzt.

Nachrichtenformat: Multimedia-Erweiterungen (MIME)

- MIME: multimedia mail extension, RFC 2045 + 2046
- MIME liefert Zusatzinformationen über den Typ der Daten, die in der Mail übertragen werden.
- Zusätzliche Zeilen im Nachrichtenheader deklarieren den MIME-Typ des Inhalts.



MIME-Typen

Content-Typ: Type/Subtyp; Parameter

➤ Text

Beispiel-Subtypen: **plain**, **html**

➤ Image

Beispiel-Subtypen: **jpeg**, **gif**

➤ Audio

Beispiel-Subtypen: **basic**, **32kadpcm** (32 kbps)

➤ Video

Beispiel-Subtypen: **mpeg**, **quicktime**

➤ Application

Alle Daten, die in keine der übrigen Kategorien passen (spezielle Anwendungen)

Beispiel-Subtypen: **msword**, **octet-stream**

Multipart-Typ

From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

--98766789
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain

Dear Bob,
Please find a picture of a crepe.
--98766789

Content-Transfer-Encoding: base64
Content-Type: image/jpeg
Content-Disposition: attachment; filename=x.jpg

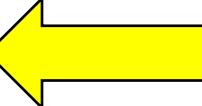
base64 encoded data
.....
.....base64 encoded data
--98766789--

Nachrichtenteil 1
(Typ: text/plain)

Nachrichtenteil 2
(Typ: image/jpeg)

Kapitel 3: Anwendungsschicht

Gliederung

- Prinzipien von Protokollen der Anwendungsschicht
- Socket-Programmierung
- Electronic Mail (SMTP)
- Das World Wide Web (HTTP) 
- Domain Name System (DNS)
- Zusammenfassung

Blick auf die Anwendung

- **http (Hypertext Transfer Protocol)**: Das Anwendungsschicht Protokoll über das verschiedene WEB Anwendungen laufen.
 - **HTTPS (HyperText Transfer Protocol Secure)**: http über TLS (s.o.)
- **Webseite** besteht aus einer Menge von Web-Objekten = Dateien (Text, Bilder, ...)
- Ein Objekt verweist auf weitere Objekte
- Referenzierung von Webseiten über **URL (Uniform Resource Locator)**
 - <Protokoll>://<Hostname>/<Pfadname>
- **HTML (Hypertext Markup Language)**: feste Sprachdefinition zur Beschreibung von Inhalt, Layout und Aktionen (eingebettete Programme)
 - HTML ist mit den Anforderungen der WEB Anwendungen gewachsen
- Das http Protokoll kann all das transportieren, was die Protokoll Regeln (z.B. Pull Protokoll) und das Message Format zulassen.
- Aber: Beim Protokoll Design muss man die Anwendung beachten.

Typische Anwendungen

- In den Anfangszeiten: Laden einer Statischen Seite von einem Server
- Heute

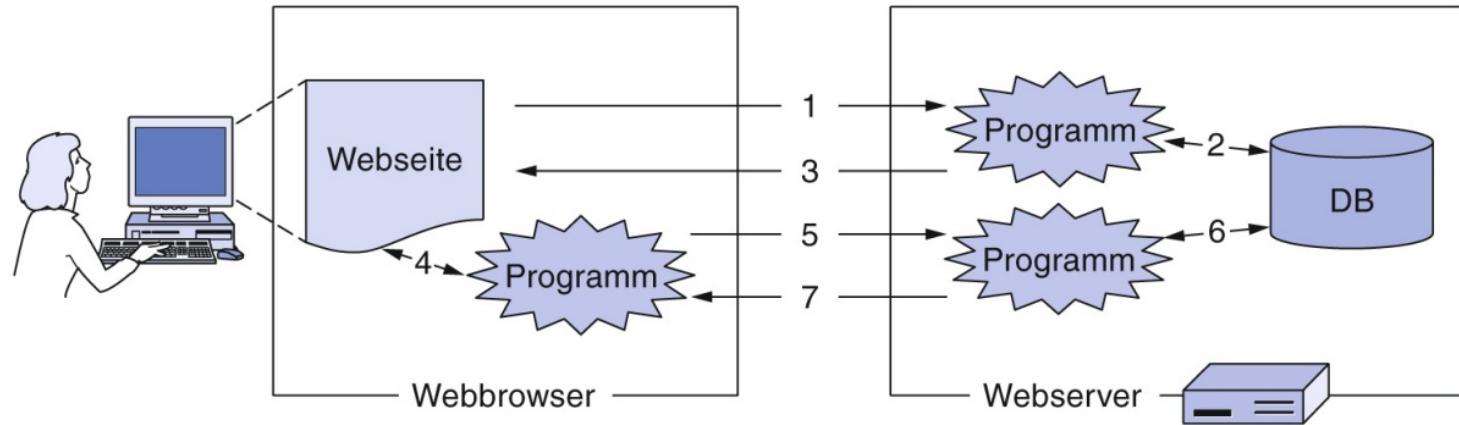


Abbildung 7.29: Dynamische Seiten.

- Anfrage startet ein Programm auf dem Server (1)
- Erzeugung der Seite (2) Übertragung der Seite an den Browser (3)
- Auf der Seite sind kleine Programme enthalten, die auf dem Client laufen (z.B.: Routenberechnung) (4)
- Fordere weitere Daten vom Server an (5).
- Weitere Daten werden auf dem Server berechnet (6) und zurückgeschickt (7)

Quelle: Tanenbaum & Wetherall: Computernetzwerke,
5. Auflage, Pearson Studium

Anforderungen an das Protokoll

- Was betrifft das http Protokoll

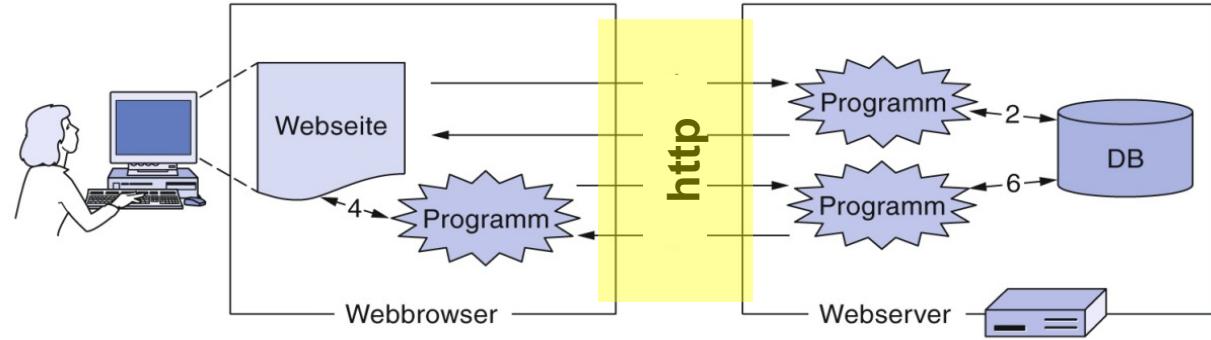


Abbildung 7.29: Dynamische Seiten.

- Das Protokoll muss Anwendungen dieser Art effizient unterstützen
- Skalierbarkeit
- Effiziente Übertragung von Objekten
- "Unnötige Übertragungen" einsparen
- Pull Protokoll
- Client Server Architektur
- Übertragung von Informationen (Formulardaten) an den Server
- Security
- ...

Quelle: Tanenbaum & Wetherall:
Computernetzwerke, 5. Auflage, Pearson Studium

HTTP (Hypertext Transfer Protokoll)

Grundlage: Client-/Server-Prinzip

- **Client:** "Browser": Interpretation und Anzeige von Web-Objekten
- **Server:** Web-Server: sendet Web-Objekte als Antwort auf eine Anfrage

HTTP/1.0: RFC 1945

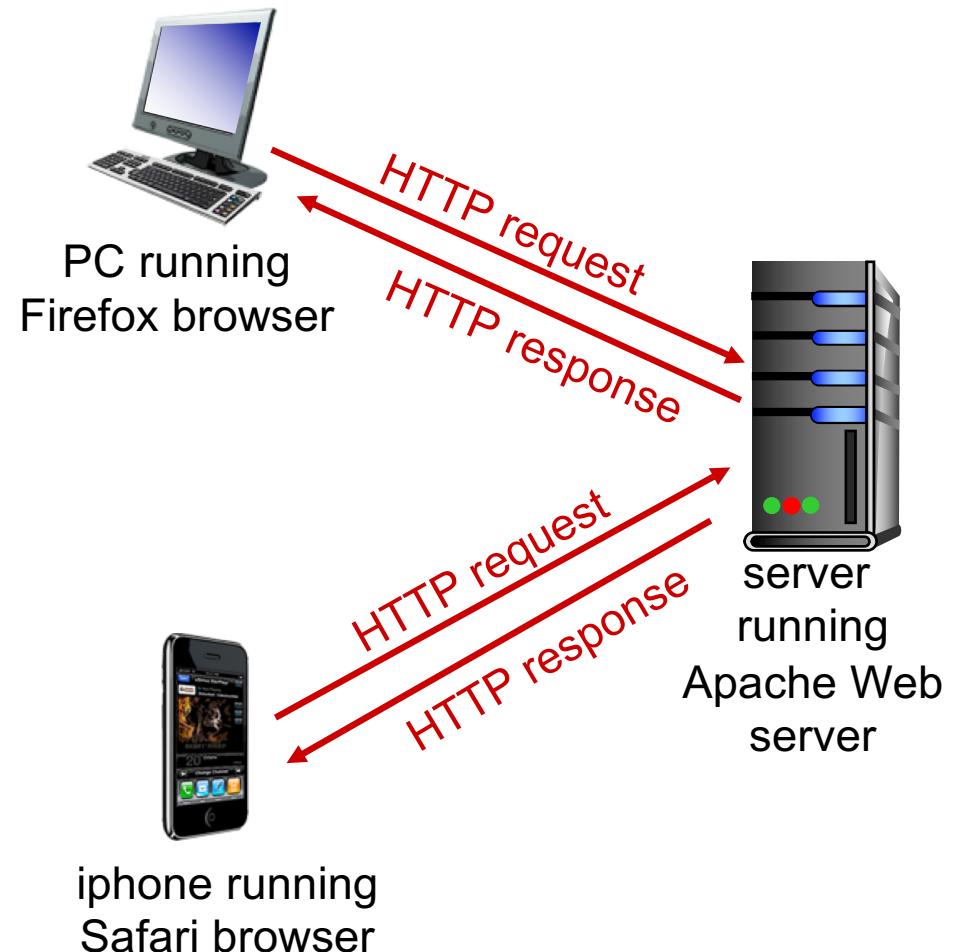
- nur noch selten verwendet

HTTP/1.1: RFC 2616

- aktueller Standard

HTTP/2.0: RFC 7540

- (Mai 2015!), hauptsächlich Performance-Optimierungen bzgl. HTTP/1.1



Welchen Einfluss haben
Unterschiedliche Endgeräte
auf das http Protokoll?

Eigenschaften von http

- Client Server Architektur
- HTTP benutzt TCP als Transport-Protokoll
 - Der Client ermittelt aus der übergebenen URL den Rechnernamen des Servers und erfragt über DNS die zugehörige IP-Adresse
 - Der Client initiiert eine TCP-Verbindung zum Server mit Portnummer 80 (erzeugt Socket) [*bei TLS (SSL)-Verschlüsselung Port 443*]
 - Der Server, akzeptiert die TCP-Verbindung des Client
 - HTTP-Nachrichten (Anwendungsschicht-Messages) werden zwischen Browser und Web Server ausgetauscht (ASCII-Klartext)
 - Anschließend kann die TCP-Verbindung geschlossen werden.
- **Zustandsloses Protokoll**
 - Der Server speichert keine Informationen über vergangene Anfragen

Diskussion zustandsloses Protokoll

- **Zustandslosigkeit** - die Eigenschaft eines Protokolls (System), wenn mehrere Anfragen grundsätzlich als voneinander unabhängige Transaktionen behandelbar sind.
- Kein Bezug zu früheren Anfragen.
Der Server muss keine Daten – keinen Zustand – zu einer Anfrage speichern – die Vergangenheit muss nicht protokolliert und verwaltet werden.
- Tritt beim Server oder Client ein Fehler auf (Absturz), ist dies für das Protokoll nicht relevant – es können keine inkonsistente Zustände auftreten.
- Eine Anfrage muss alle Informationen, die zur Beantwortung der Anfrage notwendig sind, enthalten.
- Basis für leistungsstarke und effiziente Server Implementierungen.

Gegenstück: **Zustandsbehaftung**

Beispiel HTTP 1.0 - nicht persistent

Benutzer gibt URL ein

- <http://www.someSchool.edu/someDepartment/index.html>
- 11 Objekte: Ein Text Objekt mit 10 Referenzen auf jpeg-Bildern

1a. HTTP-Client initiiert TCP-Verbindung

zum HTTP-Server
(-Prozess) auf Rechner
www.someSchool.edu
(Port 80 ist default)

1b. HTTP-Server auf Rechner
www.someSchool.edu wartet auf TCP-
Verbindungsanfragen über Port 80,
"akzeptiert" die Verbindung und
erzeugt Socket für den Client.

2. HTTP-Client sendet HTTP-
Anfragenachricht (enthält URL-Infos)
in das TCP-Verbindungssocket

3. HTTP-Server erhält Anfragenachricht,
erzeugt eine **Antwortnachricht** (enthält
das angefragte Web-Objekt
someDepartment/index.html) und
sendet die Nachricht in das Socket für
den Client.

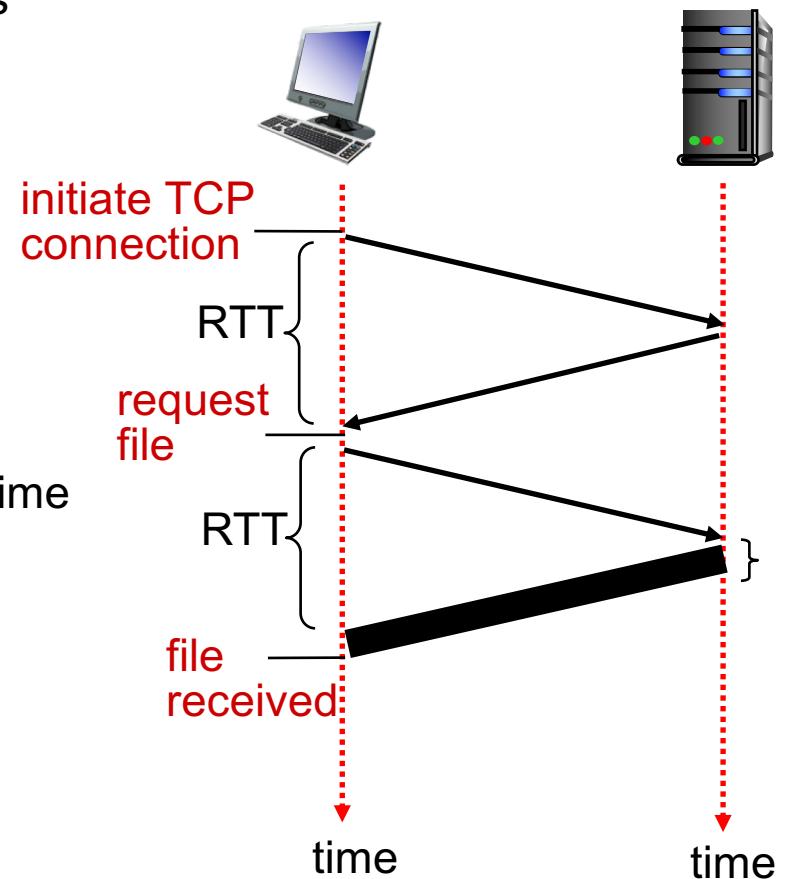
Zeit

Beispiel HTTP 1.0 - nicht persistent (Fortsetzung)

-
- Zeit ↓
1. HTTP-Client sendet die Anfrage nach index.html.
 2. HTTP-Server sendet die Anfrage nach index.html.
 3. HTTP-Server sendet die Anfrage nach index.html.
 4. HTTP-Server schließt die TCP-Verbindung.
 5. HTTP-Client empfängt die Antwortnachricht (enthält den Inhalt der Datei index.html) zeigt index.html an, analysiert die html-Datei, findet 10 referenzierte jpeg-Objekte
 6. Schritte 1-5 werden wiederholt für jedes der 10 jpeg-Bilder

Diskussion nicht persistente http Verbindungen

- Für die Übertragung eines Objektes wird eine eigene TCP Verbindung aufgebaut.
- Round Trip Time RTT: Die Zeit, die ein kleines Paket für die Übertragung zum Server und zurück braucht.
- HTTP/1.0: nach jedem Anfrage / Antworttausch wird die TCP-Verbindung geschlossen!
- Zeitaufwand $2 \times \text{RTT} + \text{ca. file transmission time}$
 - TCP-Verbindungsauftbau : $1 \times \text{RTT}$
 - Anfrage / Antwortübermittlung:
 $1 \times \text{RTT} + \text{ca. file transmission time}$
- TCP-”Slow Start”-Phase trifft jedes Objekt!
- Diskussion: Welche Delay Parameter tragen zur RTT bei?



Persistente http Verbindungen

- default ab HTTP/1.1 mehrere Anfrage / Antwort – Zyklen über dieselbe TCP-Verbindung möglich
- **Pipelining:** Der Client kann Anfragen über die TCP-Verbindung sofort senden, ohne auf die Antwort einer früheren Anfrage zu warten (allerdings: feste Reihenfolge der Antworten!)

Quelle: Tanenbaum & Wetherall:
Computernetzwerke, 5. Auflage, Pearson Studium

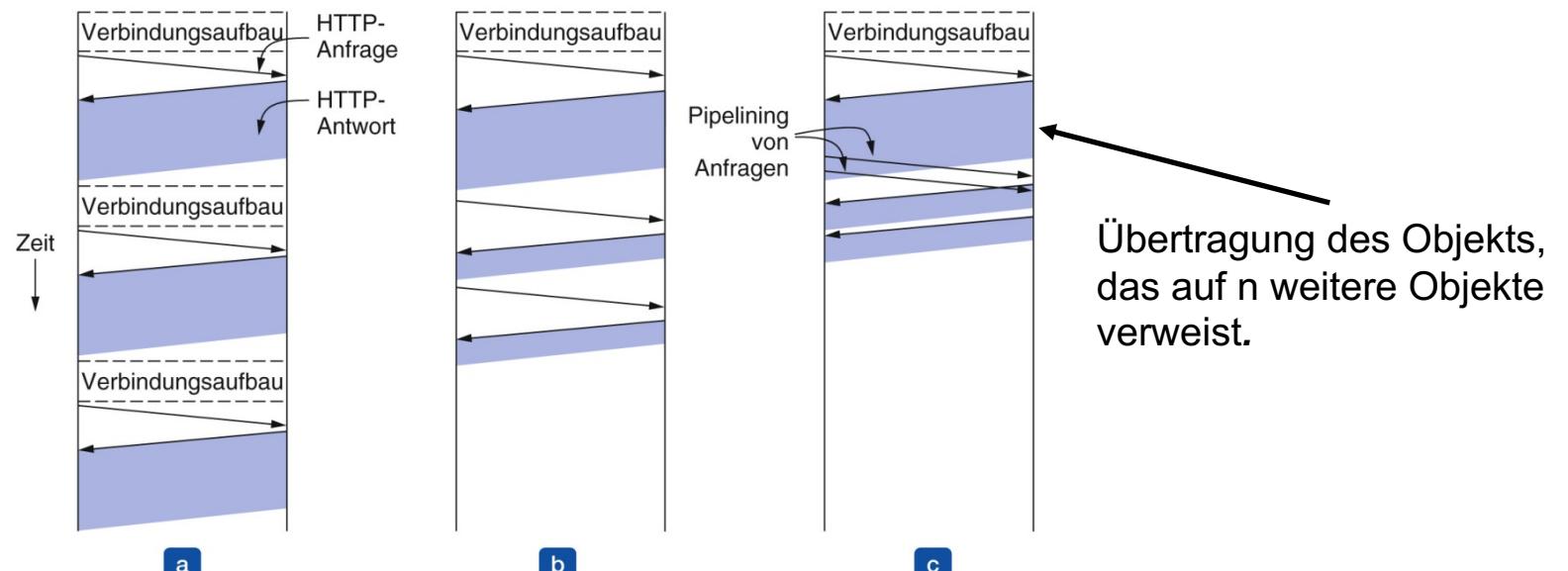


Abbildung 7.36: HTTP mit a) mehreren Verbindungen und sequentiellen Anfragen, b) mit einer dauerhaften Verbindung und sequentiellen Anfragen und c) mit einer dauerhaften Verbindung und Anfragen in Pipelines.

Persistente http Verbindungen (Fortsetzung)

- HTTP 1.1: Mehrere parallele TCP Verbindungen
- HTTP/2.0: mit mehreren unabhängigen "Streams" innerhalb derselben TCP-Verbindung (Vermeidung des "Head-of-Line-Blocking"-Problems, d.h. kein Warten auf vorherige lange Antworten nötig)
- Wann schließt der Server die TCP Verbindung zu einem Client?
 - Timeout (typische Größe: 60 s keine Anfrage, dann TCP Verbindungen beenden)
 - Bei vielen TCP Verbindungen zum Server wird Timeout verkleinert.

http Request Messages

- Sendet der Client an den Server

request line
(GET, POST,
HEAD commands)

header
lines

carriage return,
line feed at start
of line indicates
end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

carriage return character

line-feed character

- ASCII (human-readable format)
- telnet oder openssl: Protokoll von Hand abfahren

Format http Request Messages

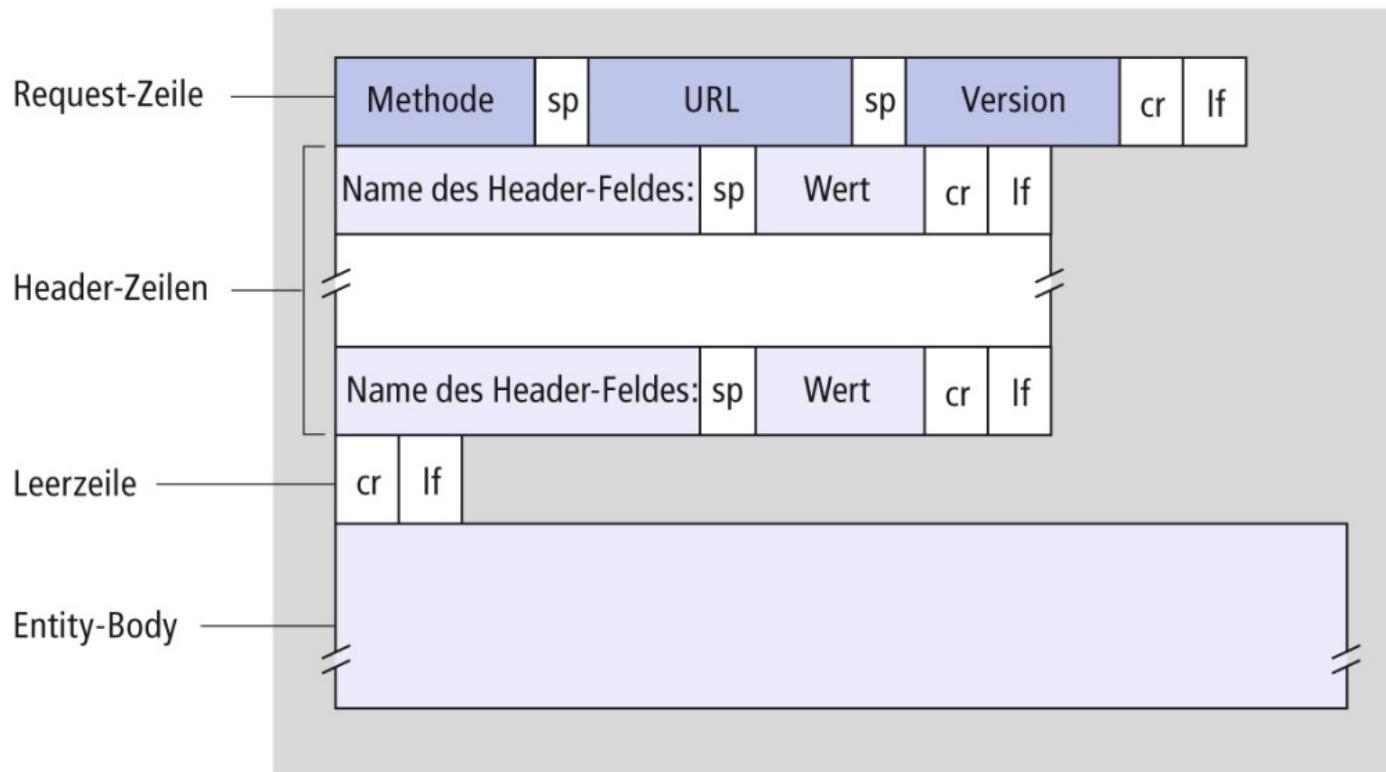


Abbildung 2.8: Allgemeines Format einer HTTP-Request-Nachricht.

- HTTP 1.0: method = GET|HEAD|POST
- HTTP 1.1: method = GET|HEAD|POST|PUT|DELETE|TRACE|CONNECT

Request Messages

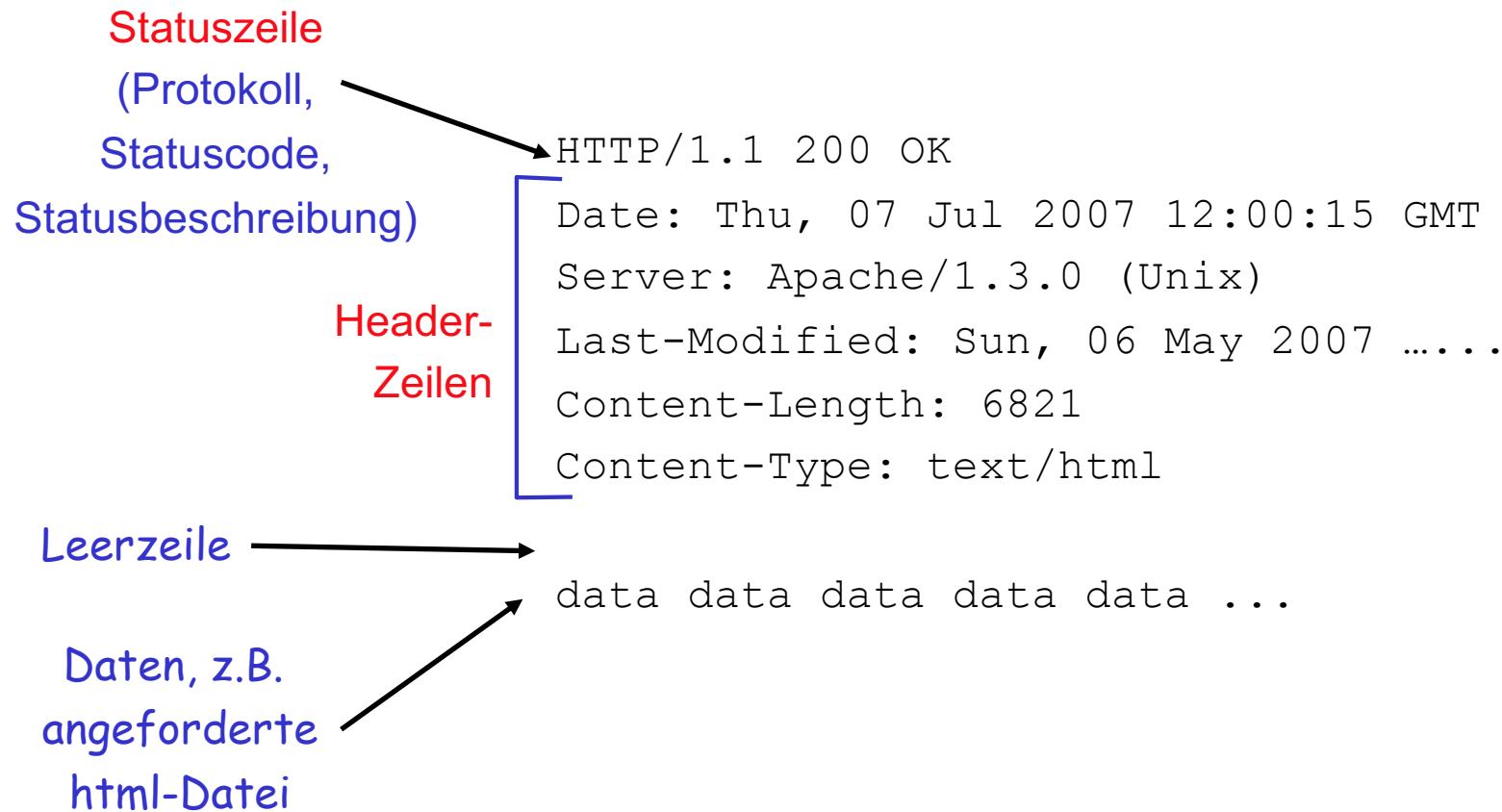
Methode	Beschreibung
http 1.0	GET Lesen einer Webseite
	HEAD Lesen des Headers einer Webseite
	POST Anhängen an eine Webseite
	PUT Speichern einer Webseite
	DELETE Entfernen einer Webseite
	TRACE Echo-Wiedergabe für die eingehende Anfrage
	CONNECT Verbindung über einen Proxy
	OPTIONS Abfrage bestimmter Optionen zu einer Seite

Abbildung 7.37: In HTTP integrierte Anfragemethoden.

- Alternative zur POST Methode: GET Methode + Parameter in der URL
- https://www.ebay-kleinanzeigen.de/s-suchanfrage.html?keywords=Fahrrad+&categoryId=&locationStr=21020&locationId=&radius=0&sortingField=SORTING_DATE&adType=&posterType=&pageNum=1&action=find&maxPrice=&minPrice=

Quelle: Tanenbaum & Wetherall:
Computernetzwerke, 5. Auflage, Pearson Studium

http Response Messages



Format http Response Messages

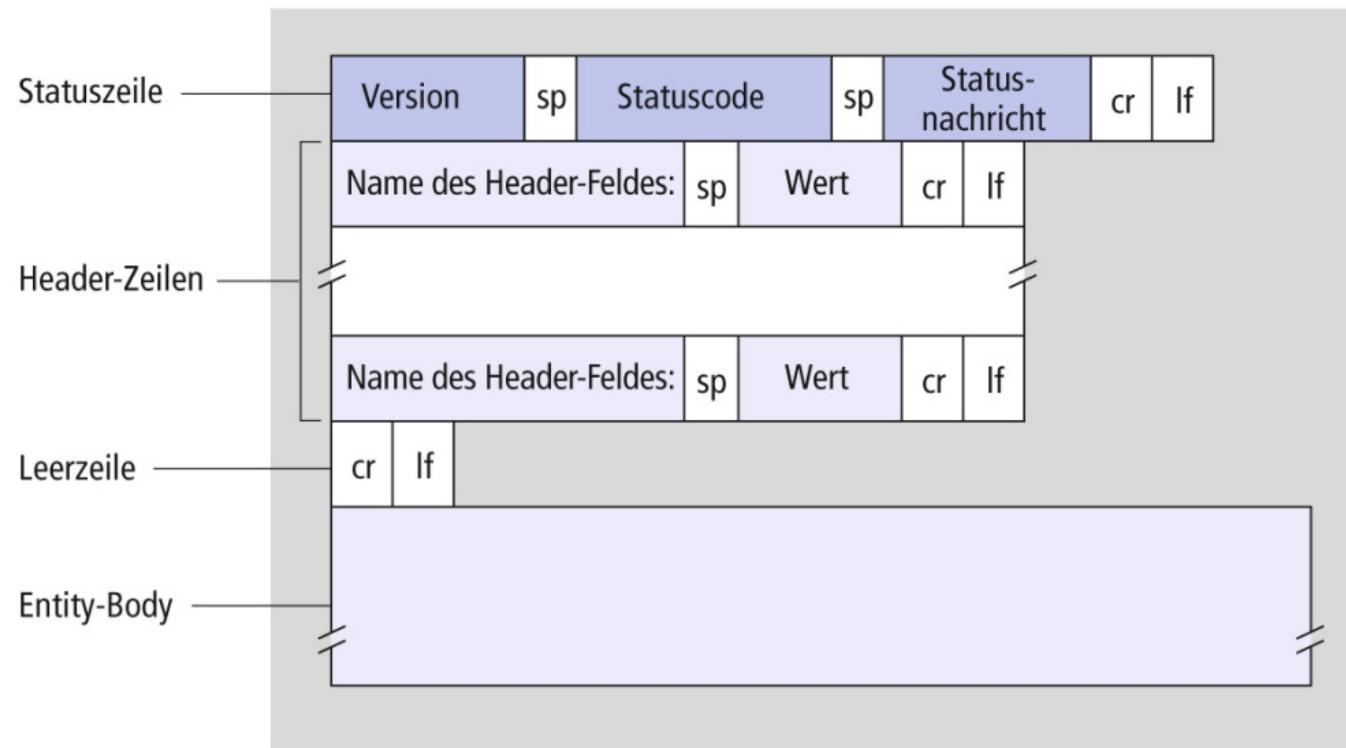


Abbildung 2.9: Allgemeines Format einer HTTP-Response-Nachricht.

HTTP Antwort - Statuscodes

Beispiele

200 OK

- Anfrage war erfolgreich, Objektdaten kommen im Datenbereich der Antwortnachricht

301 Moved Permanently

- Angefragtes Objekt permanent verschoben; die neue URL wird in der Headerzeile *location:* mitgeteilt

400 Bad Request

- Anfragenachricht konnte vom Server nicht interpretiert werden

404 Not Found

- Angeforderte Datei auf dem Server nicht vorhanden

505 HTTP Version Not Supported

- Die verwendete HTTP-Version wird vom Server nicht unterstützt

Typische http Header Zeilen

Header	Type	Contents
User-Agent	Request	Information about the browser and its platform
Accept	Request	The type of pages the client can handle
Accept-Charset	Request	The character sets that are acceptable to the client
Accept-Encoding	Request	The page encodings the client can handle
Accept-Language	Request	The natural languages the client can handle
Host	Request	The server's DNS name
Authorization	Request	A list of the client's credentials
Cookie	Request	Sends a previously set cookie back to the server
Date	Both	Date and time the message was sent
Upgrade	Both	The protocol the sender wants to switch to
Server	Response	Information about the server
Content-Encoding	Response	How the content is encoded (e.g., gzip)
Content-Language	Response	The natural language used in the page
Content-Length	Response	The page's length in bytes
Content-Type	Response	The page's MIME type
Last-Modified	Response	Time and date the page was last changed
Location	Response	A command to the client to send its request elsewhere
Accept-Ranges	Response	The server will accept byte range requests
Set-Cookie	Response	The server wants the client to save a cookie

Kleiner Versuch

```
C:> nc www.informatik.haw-hamburg.de 80
```

Anfrage

```
GET /index.html HTTP/1.0 <CR> <CR>
```

```
HTTP/1.1 200 OK
```

Antwort

```
Date: Wed, 02 Sep 2015 13:29:46 GMT
```

```
Server: Apache/2.2.3 (Linux/SUSE)
```

```
Last-Modified: Fri, 28 Jan 2011 09:45:37 GMT
```

```
Accept-Ranges: bytes
```

```
Content-Length: 6372
```

```
Connection: close
```

```
Content-Type: text/html
```

```
<html>
```

```
... Inhalt
```

```
</html>
```

Tools zum Herstellen einer interaktiven TCP-

Verbindung:

- netcat / nc (Linux / Windows)
- telnet (Linux / Windows) [veraltet]
- socat (nur Linux) [komplex]

Benutzer-Identifizierung: Cookies [RFC 2109]

- http zustandsloses Protokoll => Relation eines Zustandskonzepts über Cookies:
 - Über Cookies speichern die Endpunkte des Protokolls einen Zustand über mehrere Transaktionen hinweg.
 - Cookies werden in den http Header Zeilen transportiert.
- Über Cookies wird der Benutzer identifiziert.
- Cookies werden auf dem Client des Benutzers gespeichert.
- Die vier Komponenten des Cookie Konzepts:
 1. Server erzeugt Identifizierungsstring, z.B. ID1678453
 2. Server sendet "Cookie" in der Antwortnachricht zum Client. Set-cookie:ID1678453
 3. Client speichert Cookie in lokaler Datei und gibt den Wert bei späteren Anfragen an den Server mit: cookie:ID1678453
 4. Server speichert auf Basis des Cookies Informationen über den Anwender in einer Datenbank.

Beispiel

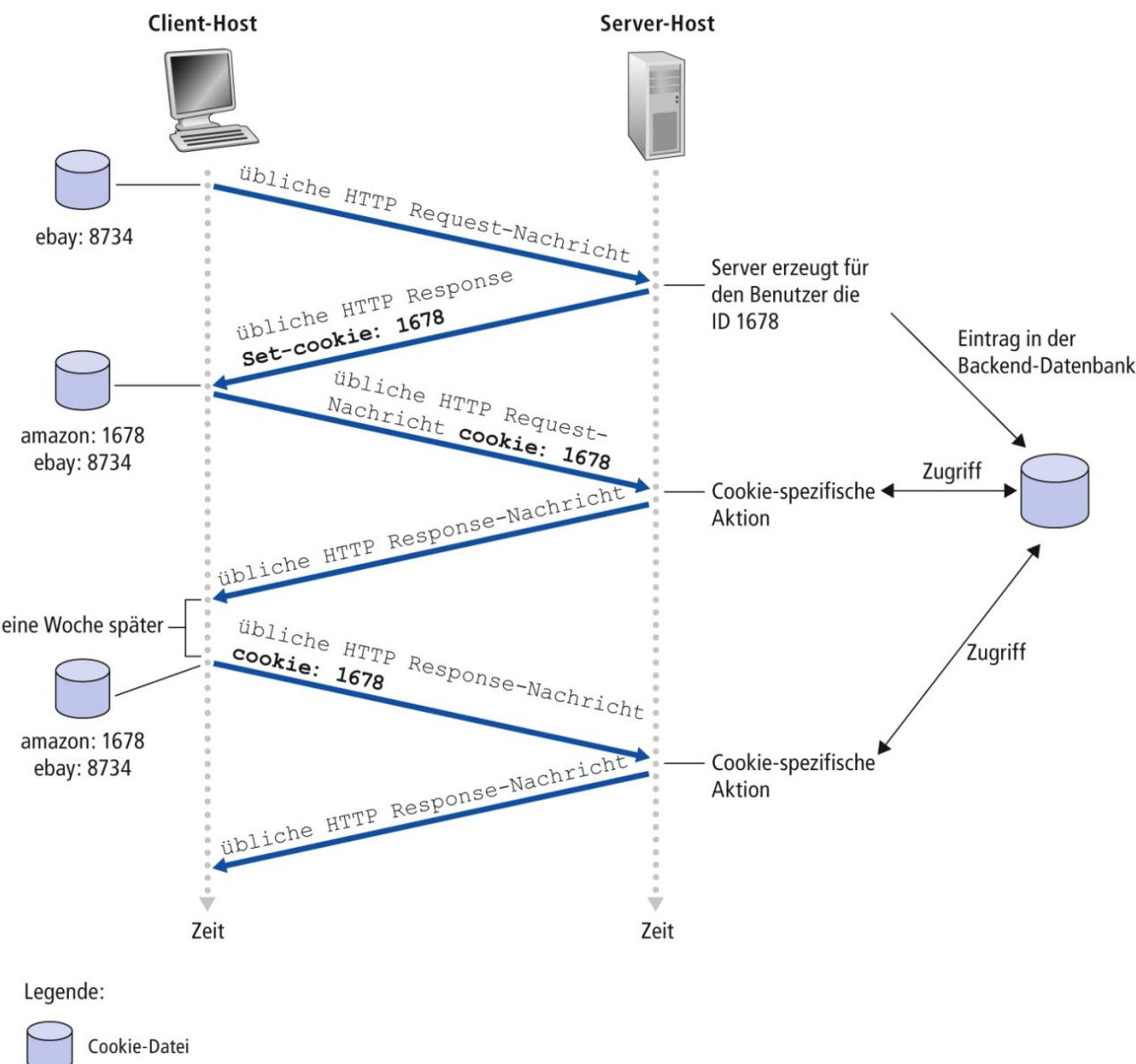


Abbildung 2.10: Speichern von Zuständen mithilfe von Cookies.

Caching von Objekten

Ansatz

- Greifen auf einen WEB Server über einen Cache zu.
 - Wenn eine aktuelle Version der Seite im Cache ist, dann wird diese aus dem Cache genommen.

Zwei Cache Typen

- Caching auf Client Seite
Der Cache liegt in einem lokalen Verzeichnis des Client Rechners.
- Web-Cache
Der Cache liegt auf einem schnell angebundenen Rechner im Netz

Wann ist ein Objekt im Cache aktuell?

Zwei Ansätze

- Expires Header Zeile
 - Der Server schickt eine Objekt mit einem Expires Header Zeile
 - Zusammen mit den Expires Datum wird das Objekt im Cache gespeichert und bei erneuten Anfrage verwendet, bis das Expires Datum erreicht ist.
- http Protokoll: Conditional GET (GET Methode + If-Modified-Since Header)
 - Server schickt zu einem Objekt die Header Zeile Last-Modified mit. Diese wird mit dem Objekt im Cache gespeichert.
 - Bei erneuten Zugriff auf das Objekt schickt der Cache eine GET Anfrage mit If-Modified-Since Header Zeile. Sie enthält das im Cache gespeicherte Last-Modified Datum.
 - Der Server antwortet mit einem leeren Body, wenn das Objekt im Cache noch aktuell ist.
Ansonsten wird die neue Objekt geschickt und im Cache gespeichert.

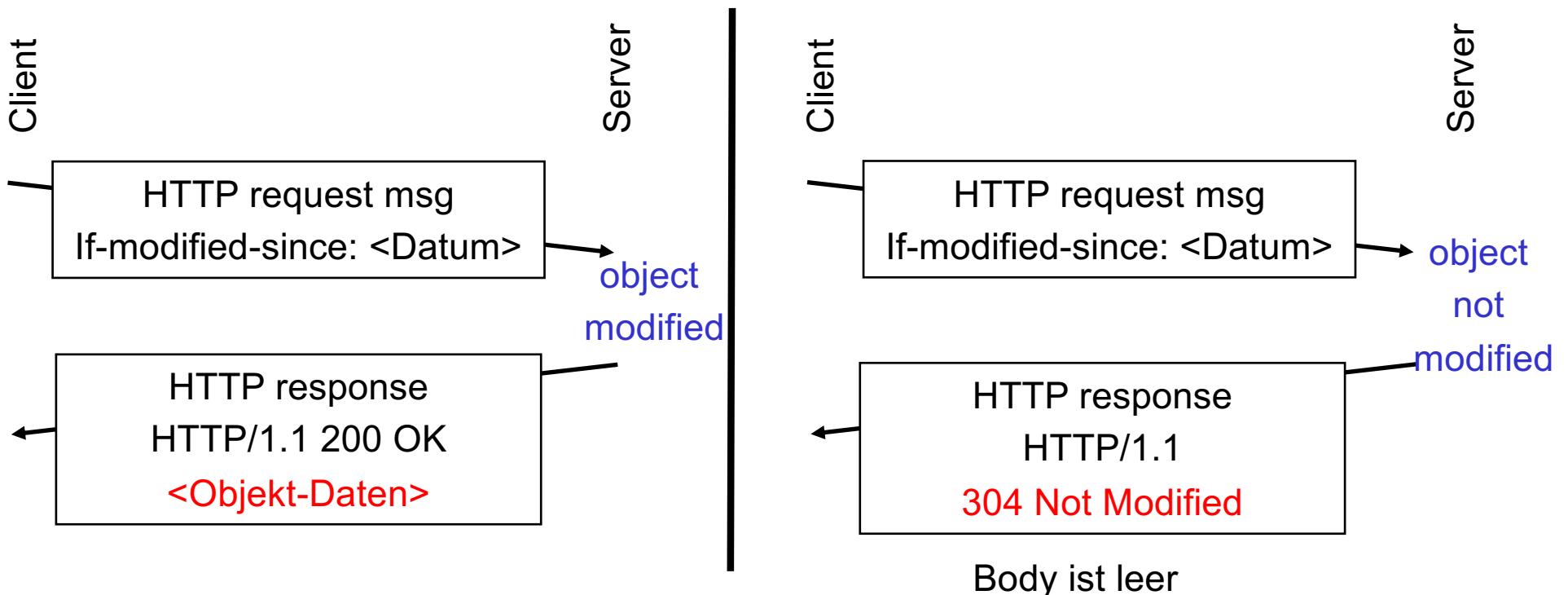
Caching auf Client-Seite

Seite liegt im Cache

Fall 1: Expires Datum existiert und liegt in der Zukunft

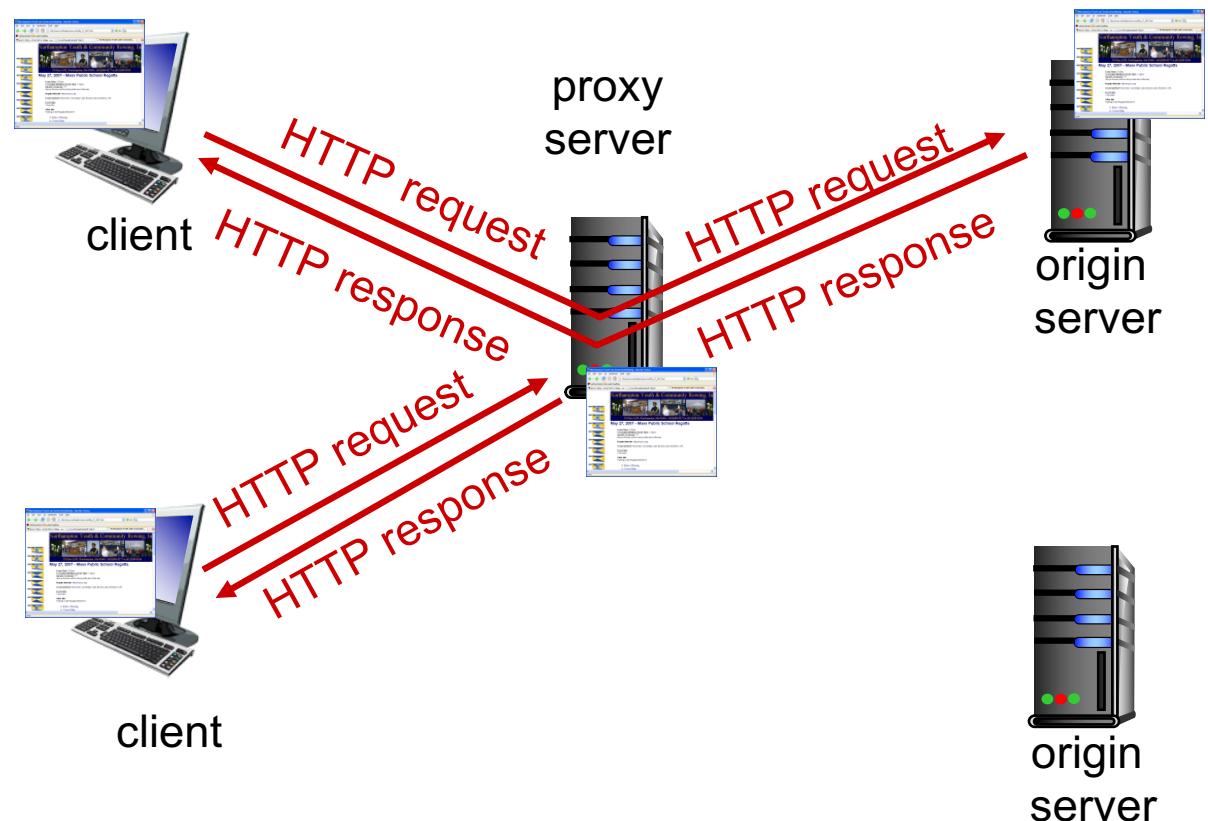
- Seite wird direkt aus dem Cache geholt – kein Netzwerzkzugriff

Fall 2: If-modified-since Datum wurde abgespeichert



Caching auf Server-Seite / Web-Cache / Proxy-Server

- **Proxy-Server:** Server, der falls möglich im Namen der Webserver antwortet.
- Beantwortung von Client-Anfragen durch “lokalen” Server
- Browser-Konfiguration:
“Proxy-Server verwenden”
- Client sendet alle HTTP-Anfragen an den Proxy Server
- Object im Web-Cache des Proxy Servers vorhanden:
liefere Web-Objekt zurück
- sonst: Proxy Server stellt Anfrage an Original-Server und leitet die Antwort an den Client weiter

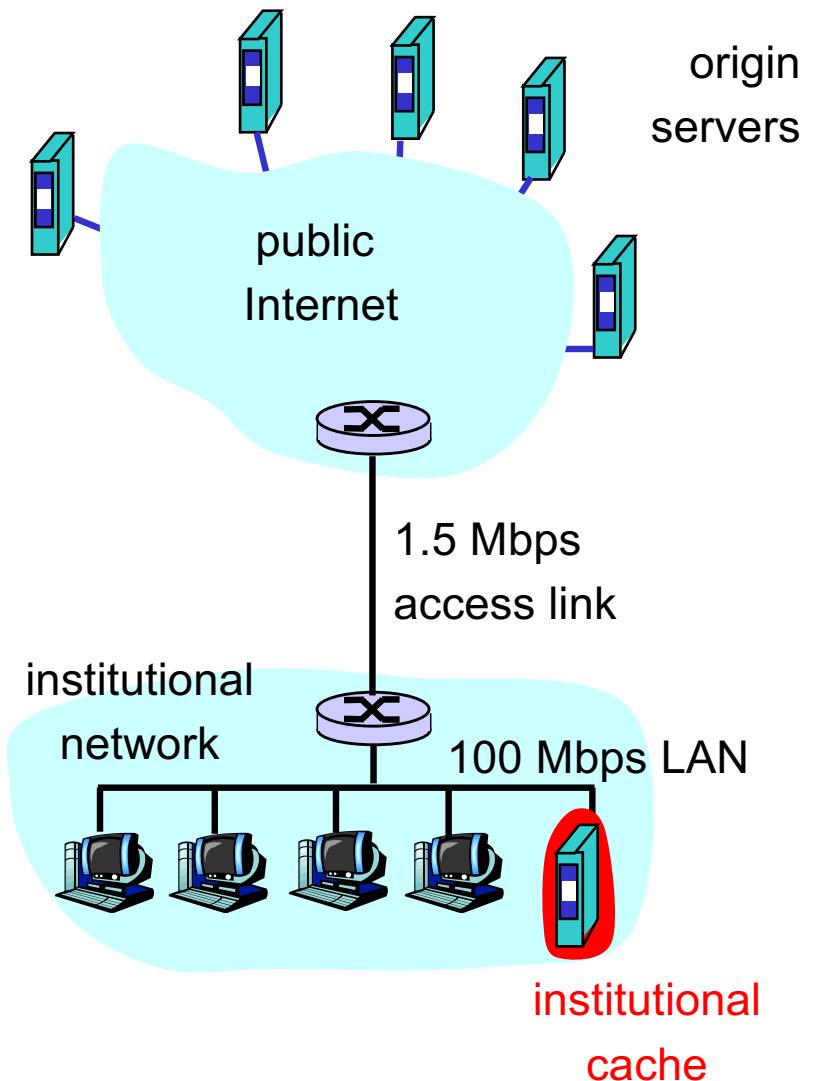


Vorteile des Caching durch Proxy-Server

- Voraussetzung: Proxy Server ist in der "Nähe" des Client (z.B. im selben Firmennetzwerk)
- geringere Antwortzeit
- weniger Verkehr ins Internet (Kosten und Verfügbarkeit der Verbindung zum ISP)
- Sicherheitsvorteile für geschlossene Netze (Zugriffsfilterung möglich) → *kommt später*

Kommentare

- Ein Proxy-Server ist Client und Server
- Einsatz: ISP , Firmen, Unis ...



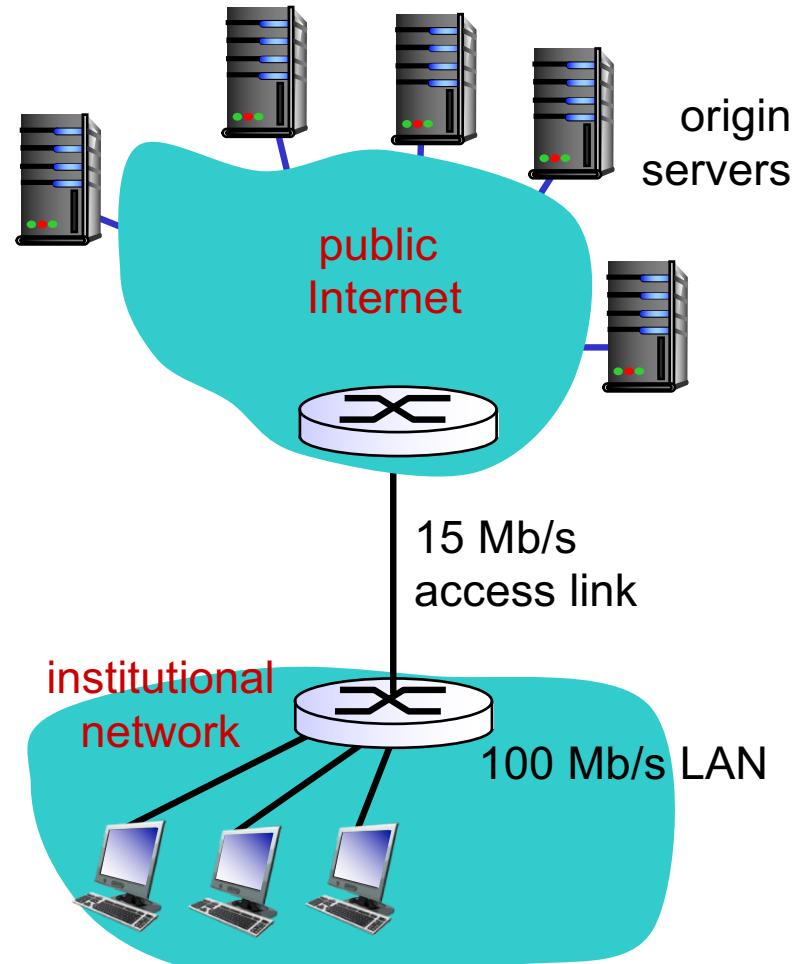
Übungsaufgabe

Situation

- Durchschnittliche Objektgröße: 1 Mbit
- Durchschnittliche Rate von Request: 15/sec
- RTT vom Firmen Router zu einem Server:
2 s

Aufgabe

- Berechnen Sie den Verkehrswert für das LAN und den Access Link der Firma.¹
- Interpretieren Sie die Verkehrswerte.



¹ Gehen Sie davon aus, dass im LAN auch mal alle Request über einen Link gehen können.

Übungsaufgabe

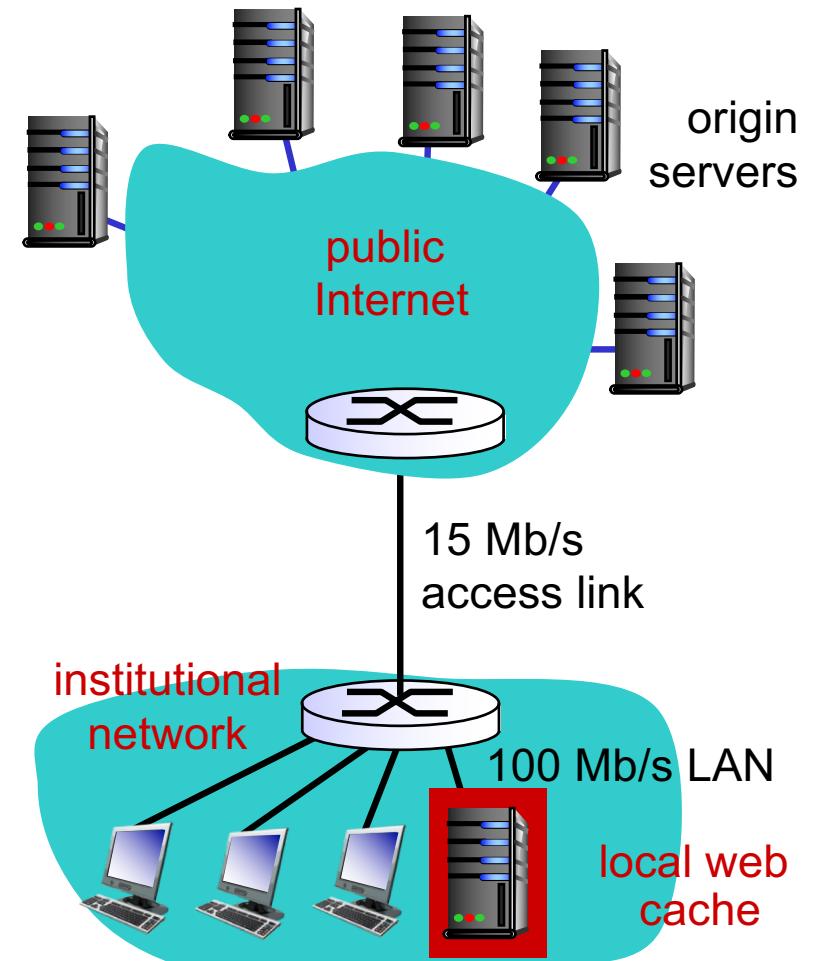
Lösung: Einsatz eines Proxy-Servers

Situation

- Trefferrate: 40%¹
- RTT Client – Proxy-Server: 0,01s

Aufgabe

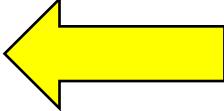
- Berechnen Sie den Verkehrswert für den Access Link der Firma.¹
- Berechnen Sie die durchschnittliche Latenz einer Anfrage.
- Vergleichen Sie den Latenz mit einer Netzwerkkonfiguration ohne Proxy Server und einem 100 Mb/s access link.



¹ Hier: Treffer verursacht kein Conditional GET

Kapitel 3: Anwendungsschicht

Gliederung

- Prinzipien von Protokollen der Anwendungsschicht
- Socket-Programmierung
- Electronic Mail (SMTP)
- Das World Wide Web (HTTP)
- Domain Name System (DNS) 
- Zusammenfassung

Problemstellung

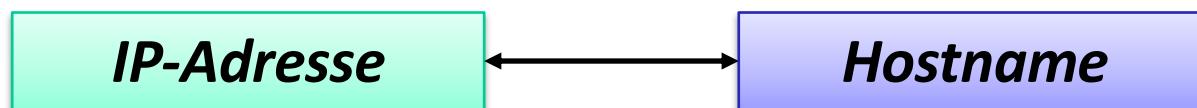
Identifizierung von Menschen:

- Name + Geburtsdatum oder Personalausweisnr. oder Sozialversicherungsnr,
- Abbildung durch Nachfragen und Meldeämter

Identifizierung Internet-Rechnern / Routern:

- **IP-Adresse** (32 bit / 128 bit): benutzt von Rechnern zur Adressierung von Datagrammen
 - weltweit eindeutig
- **Hostname** – benutzt von Menschen
 - (z.B. users.informatik.haw-hamburg.de oder www.wdrmaus.de)
 - weltweit eindeutig

Aufgabe: Wer verwaltet die Abbildung zwischen IP-Adresse und Hostnamen?



DNS: Domain Name System

DNS [RFC 1034/1035] ist ...

- eine verteilte Datenbank
 - implementiert durch eine Hierarchie von Name-Servern
- ein Anwendungsschicht-Protokoll
 - über das Hosts und Name-Server miteinander kommunizieren, um eine Abbildung von Hostnamen auf IP-Adressen zu erreichen.
- Client Server Paradigma

Kommentare

- DNS ist eine Kernfunktion für das Internet, die als Protokoll auf der Anwendungsschicht implementiert ist.
- DNS stellt einen Dienst für andere Anwendungsschicht-Protokolle zur Verfügung (ohne Benutzerschnittstelle), stellt also selbst keine Anwendung dar.
- DNS verwendet UDP via Port 53

DNS Dienste

Abbildung von Hostnamen auf IP Adressen

Host Aliasing

- Host mit einem komplizierten Hostname erhält eine oder mehrere Aliasnamen
- Kanonische Name und Aliasnamen

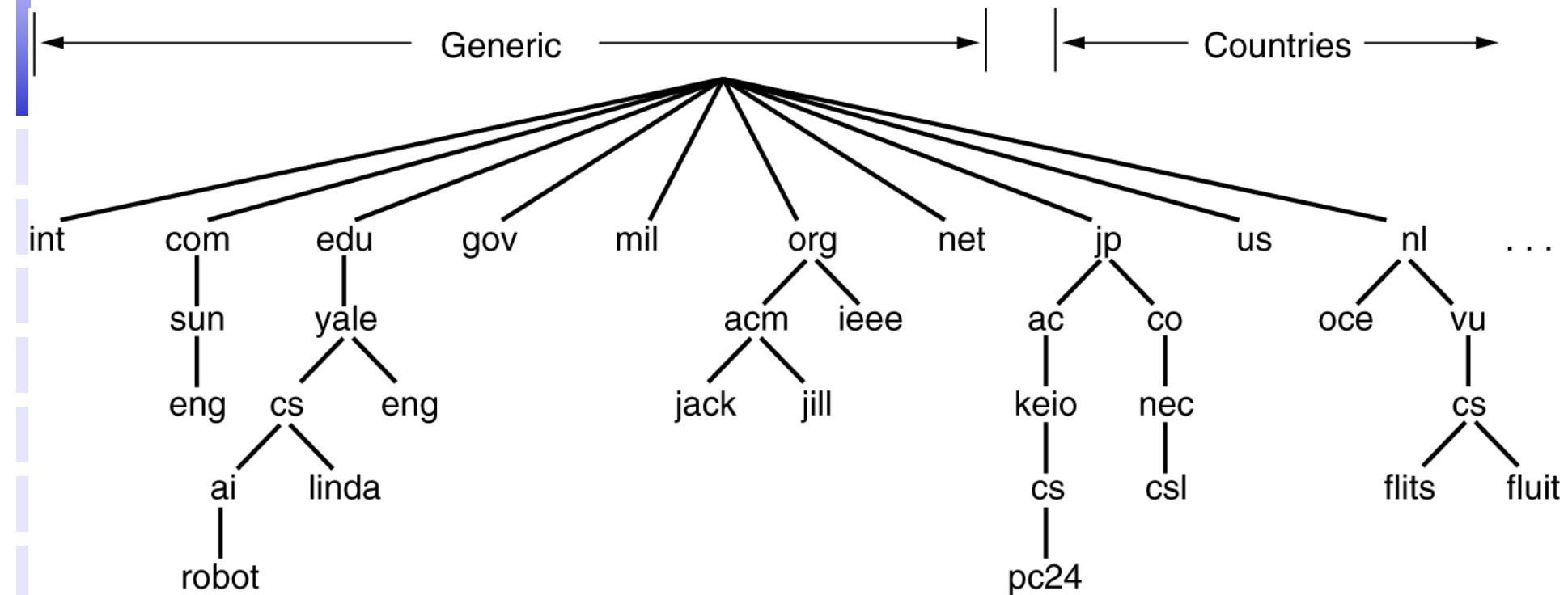
Mail-Server Aliasing

- Unterstützt die Verwendung von Alias Namen in E-Mail Adressen
- Web-Server und Mail-Server können den selben Aliasnamen haben

Lastverteilung

- Lastverteilung zwischen replizierten Servern
- DNS Server antworten bei einer Anfrage mit den IP Adressen aller replizierten Server (Reihenfolgen rotiert mit jeder Anfrage)

Namensraumstrukturierung: Domains und Subdomains



- über 200 Top-Level Domains
- Hostnamen (Rechnernamen) sind auf jeder Ebene möglich
- der gesamte Pfad gehört zum Namen
- Beispiele: *flits.cs.vu.nl.* oder *users.informatik.haw-hamburg.de.*

Warum kein zentraler DNS Server

Probleme

- Single point of failure
- Verkehrslast
- Entfernung der zentralen Datenbank
- Wartung / Aktualisierung
- Diese Architektur wäre nicht **skalierungsfähig!**

Stattdessen

- **DNS als verteilte hierarchische Datenbank**

DNS Server Hierarchie

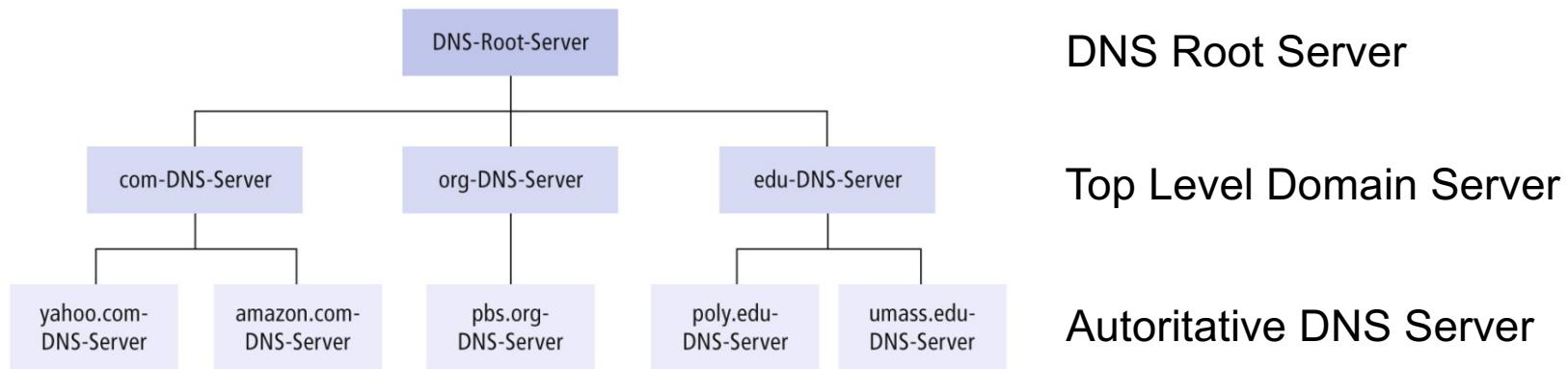


Abbildung 2.19: Teilausschnitt aus der Hierarchie der DNS-Server.

DNS Root Server

- Es gibt 13 DNS Root Server (Buchstaben A bis M)
- Jeder Server besteht nochmals aus einem Netzwerk an Replikationen

Top-Level-Domain Server (TLD)

- Sind für die Top Level Domains zuständig (de, fr, com, edu ...)

Autoritative DNS Server

- Pflicht: Jeder Organisation mit öffentlich zugänglichen Hosts zum so einen Server betreiben (plus Backup)

DNS Server Hierarchie (Fortsetzung)

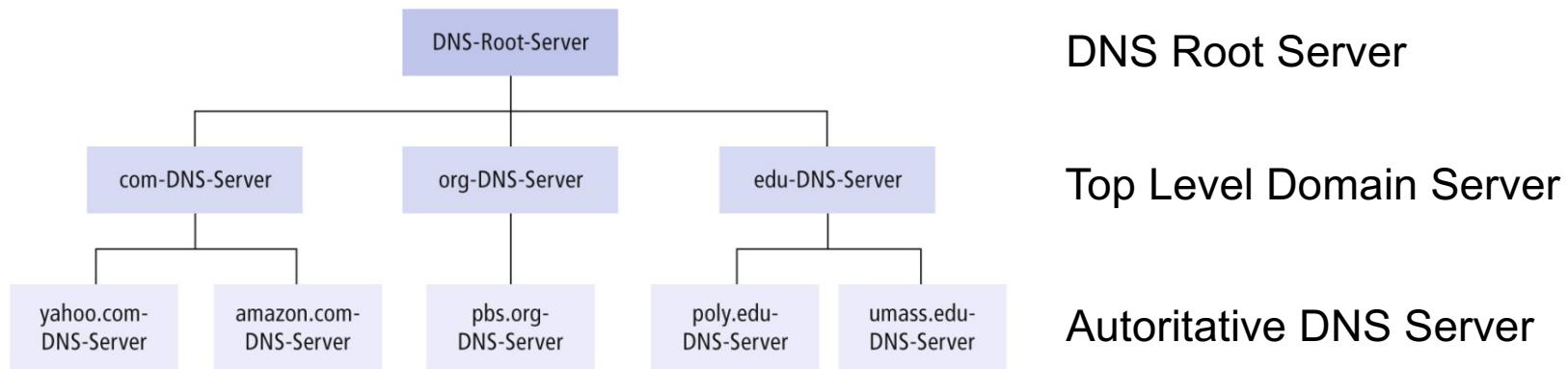


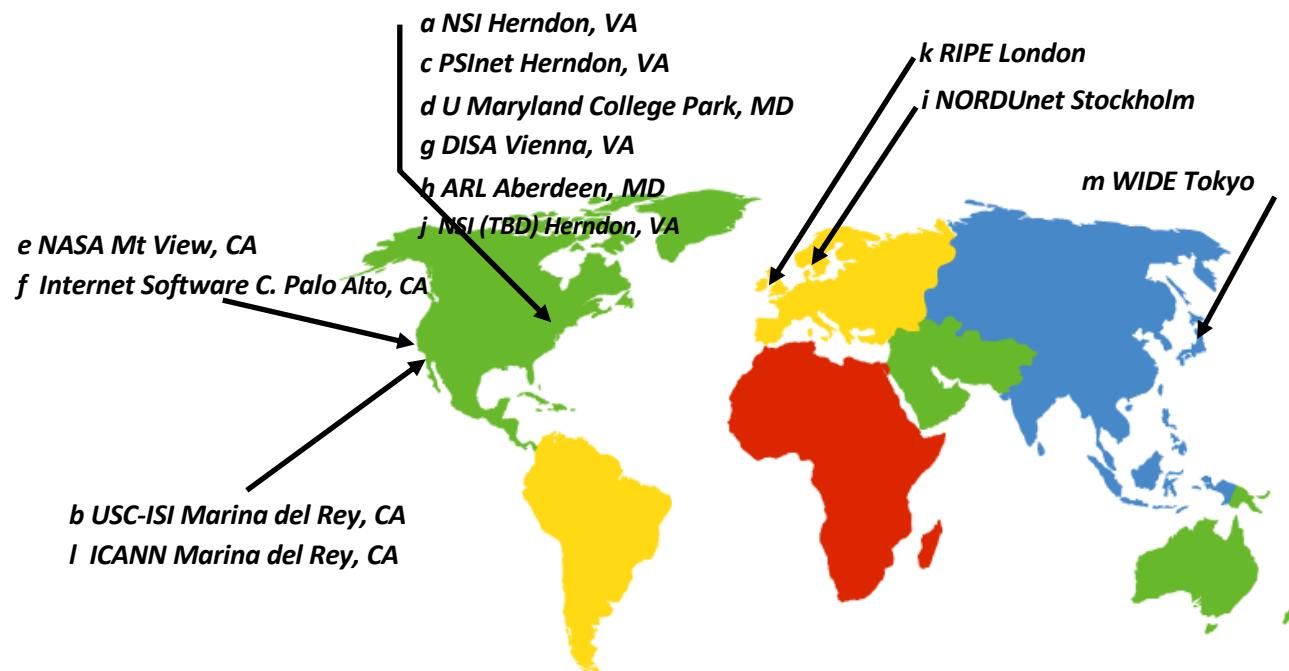
Abbildung 2.19: Teilausschnitt aus der Hierarchie der DNS-Server.

Lokaler DNS Server

- Gehören nicht zur Server Hierarchie, spielen aber eine wichtige Rolle
- Jeder ISP / jede Firma hat üblicherweise einen lokalen Name-Server ("DNS-Server")
- Alle DNS-Anfragen eines Hosts gehen zuerst zum lokalen Name-Server (IP-Adresse des lokalen DNS-Servers gehört zur IP-Konfiguration eines Hosts)
- Proxy Funktionalität: Leiten Anfragen an die Server Hierarchie
- Cache für Hostname – IP Adresse Abbildung

DNS Root Server

- ...wird von lokalen Name-Servers kontaktiert, die einen Hostnamen nicht auflösen können
- Weltweit gibt es 13 Root Name-Server (*jeweils mehrere Maschinen*)
 - <http://root-servers.org/>
 - <http://www.orsn.org/de/>
- IP-Adressen der Root-Name-Server sind durch eine spezielle Konfigurationsdatei allen anderen Name-Servern bekannt!

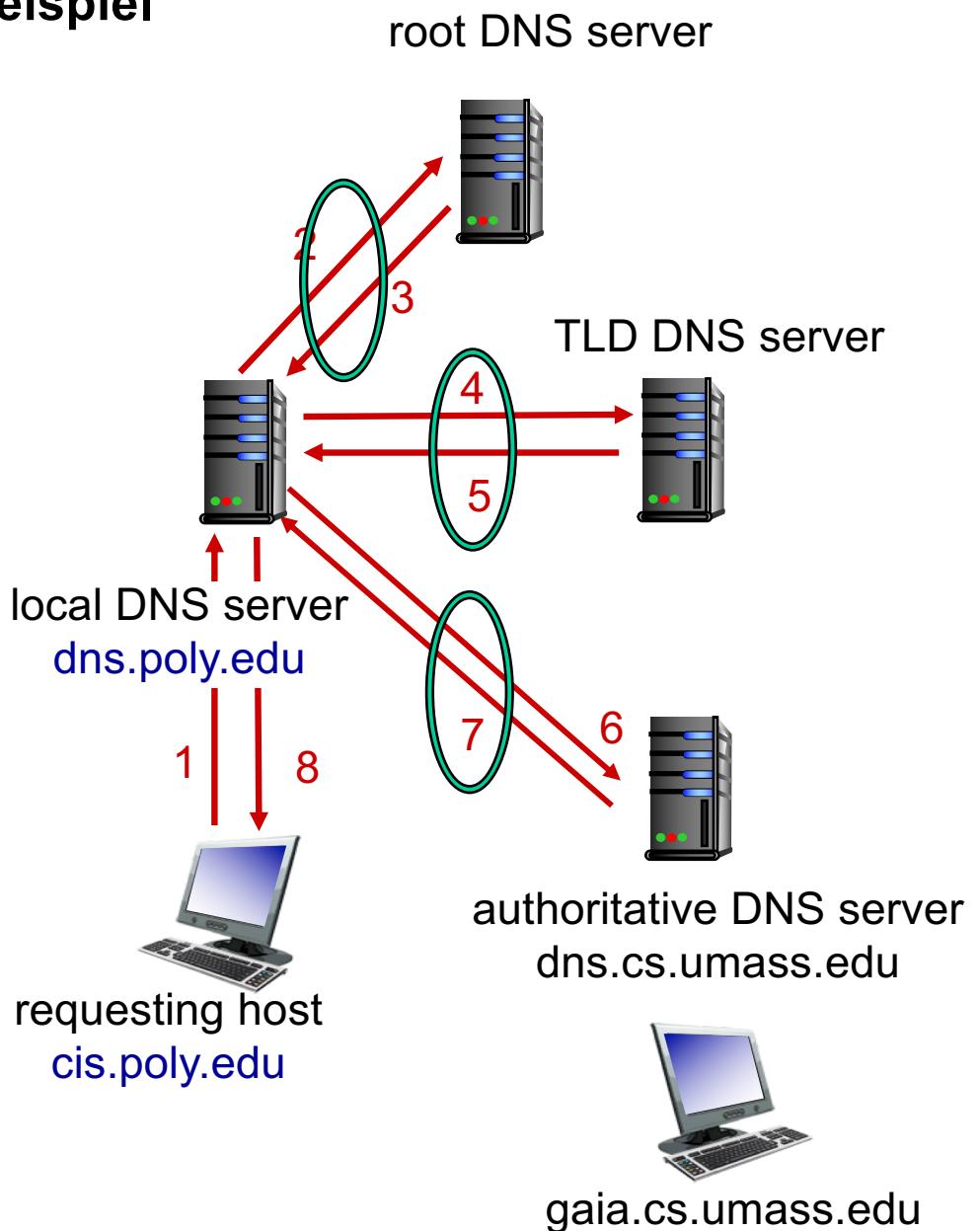


Beispiel

- Host **cis.poly.edu** benötigt die IP Adresse von **gaia.cs.umass.edu**
- Anfragen liefern eine **Liste** von IP Adressen zurück, die die Anfrage weiter bearbeiten können.
- DNS Caching reduziert die Anzahl der Anfragen.

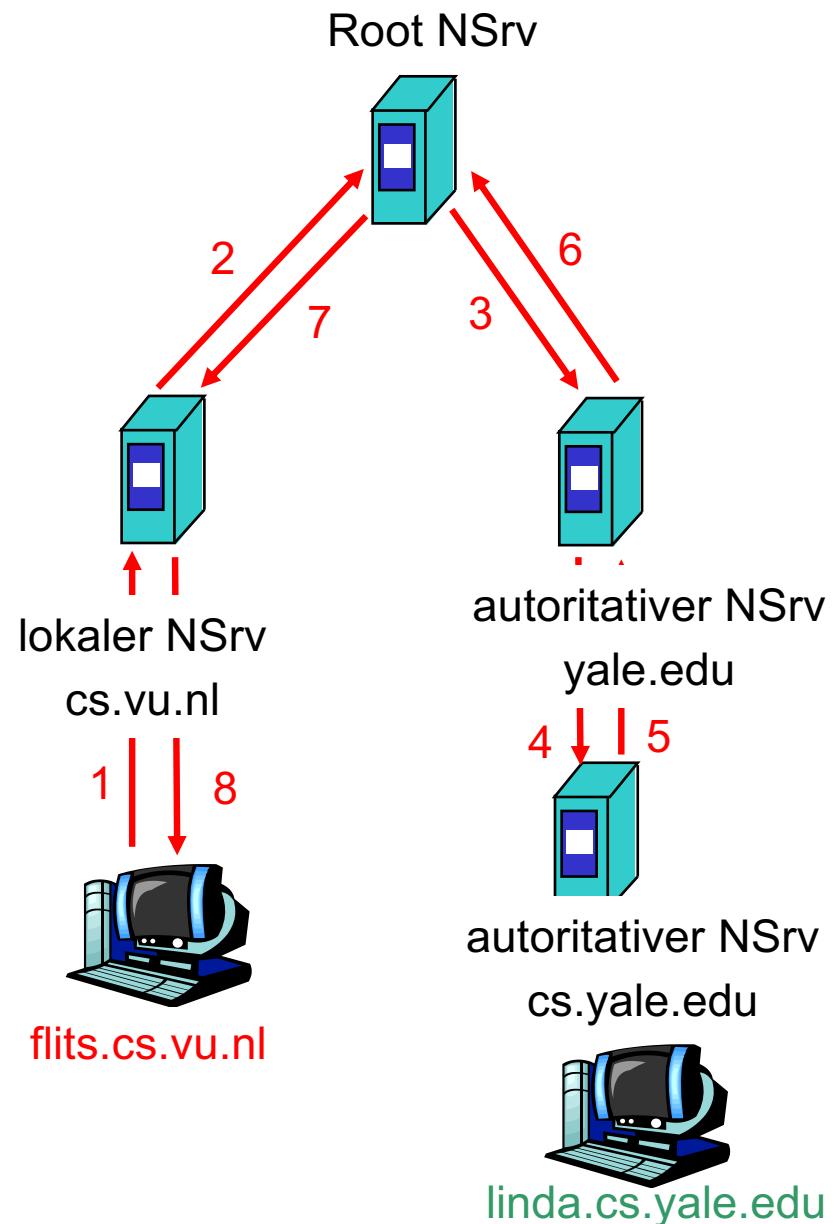


Iterative Anfragen



Beispiel Rekursive Anfragestrategie

- Host **flits.cs.vu.nl** benötigt IP-Adresse von **linda.cs.yale.edu**
- “Rekursive” Anfragestrategie
- Root-Nameserver sind in der Praxis meist auch authoritative NSrv für länder-unabhängige Top-Level-Domains (z.B. *.com*, *.edu*, *.org*)
- *yale.edu leitet die Anfrage rekursiv an cs.yale.edu weiter*



DNS Resource Records

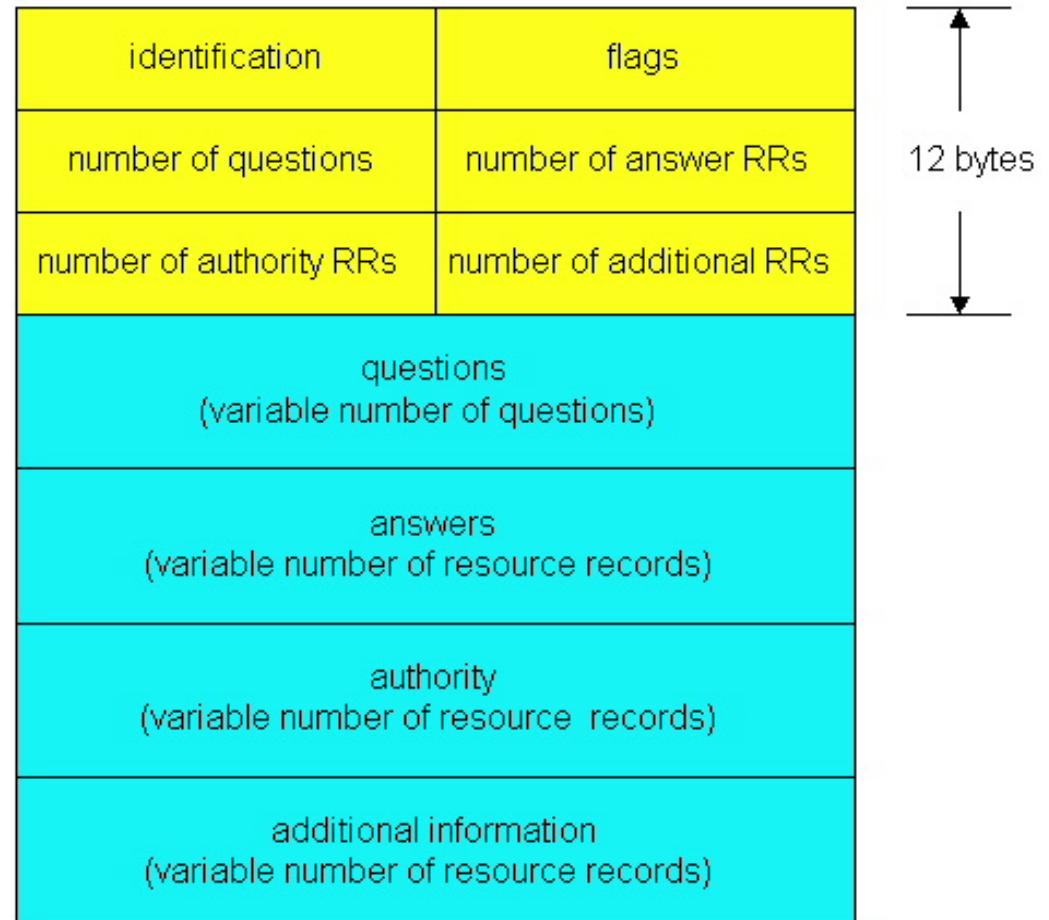
- Einträge in der verteilten Datenbank heißen **Resource Records (RR)**
- Für einen Host sind beliebig viele Einträge möglich
- **RR Format: (Name, TTL, Typ, Wert)**
 - TTL (Time To Live): Maximale Zeit, die ein lokaler Name-Server den Eintrag im **Cache** hält.
- Wichtigste Typen:
 - **A bzw AAAAA**: Name = Hostname, Wert = IP-Adresse (IPv4 bzw. IPv6)
 - **NS**: Name = Domain, Wert = IP-Adresse des autoritativen Name-Servers für diese Domain
 - **CNAME**: Name = Alias-Name, Wert = “realer” Name (“canonical”)
 - **MX**: Name = Servername in einer Mailadresse, Wert = zugeordneter “realer” Mailserver (-name)

DNS Caching

- Jeder DSN Server pflegt einen Cache
- Die Cache Einträge werden bis um TTL Zeitpunkt als gültig angesehen
- TLD Server sind oftmals in Lokalen Servern gecached
- Cache Einträge können ungültig sein, obwohl ihr TTL noch nicht erreicht ist.
 - Der entsprechende Host ist nicht sichtbar, bis TTL abgelaufen ist.
 - update/notify mechanisms proposed IETF standard RFC 2136

DNS-Protokoll: Nachrichtenformat

- Anfrage- und Antwortnachrichten: beide haben dasselbe Nachrichtenformat
- 12 Byte Header:
 - **identification:** 16 bit-Zahl zur Identifizierung einer Anfrage und zugehöriger Antwort
 - **Flags**
 - Anfrage oder Antwort?
 - Rekursion gewünscht?
 - Rekursion verfügbar?
 - Antwort ist autoritativ!



Tools: nslookup oder dig

Sicherheit des Domain Name Service (DNS): 😞

DOS Attacken über Botnet

- Ping Nachrichten (21.10.2002)
 - Paket Filter haben das Problem abgefangen
- Angriff über DNS Anfragen and TLD Server
 - Caching fängt die Folgen ab

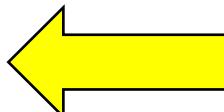
Man in the Middle Angriffe

- Angreifer sendet sinnlose Pakete an den Server, die er in seinem Cache einlagert.
- Angreifer sendet gefälschte Pakete an den Server, so dass Nutzer auf die Seite des Angreifers umgelenkt werden.

...

Kapitel 3: Anwendungsschicht

Gliederung

- Prinzipien von Protokollen der Anwendungsschicht
- Socket-Programmierung
- Electronic Mail (SMTP)
- Das World Wide Web (HTTP)
- Domain Name System (DNS)
- Zusammenfassung 

Zusammenfassung

- Dienstanforderungen von Applikationen: Zuverlässigkeit, Bandbreite, Verzögerung
- Client-Server Paradigma
- Anwendung der Internet-Transportdienstmodelle TCP, UDP
- Socket Programmierung
- Ausgewählte Protokolle: SMTP, HTTP, DNS
- Pull- und Push Protokolle
- Caching Strategien
- Nachrichtenformate
- Zustandslos / zustandsbehaftet
- ...