



Institut für Verteilte Systeme
Universität Ulm



universität
uulm



Kapitel B: Anwendungsschicht

Vernetzte Systeme | CS3160.000 | Wintersemester 2024/2025

Benjamin Erb | Frank Kargl

Institut für Verteilte Systeme
Universität Ulm



Überblick

B. Anwendungsschicht

- B.1 Grundlagen der Netzerkanwendungen ([▶](#))
- B.2 HTTP und das Web ([▶](#))
- B.3 E-Mail ([▶](#))
- B.4 DNS ([▶](#))
- B.5 P2P ([▶](#))

B.1 Grundlagen der Netzwerkanwendungen

Netzwerkanwendungen im Internet

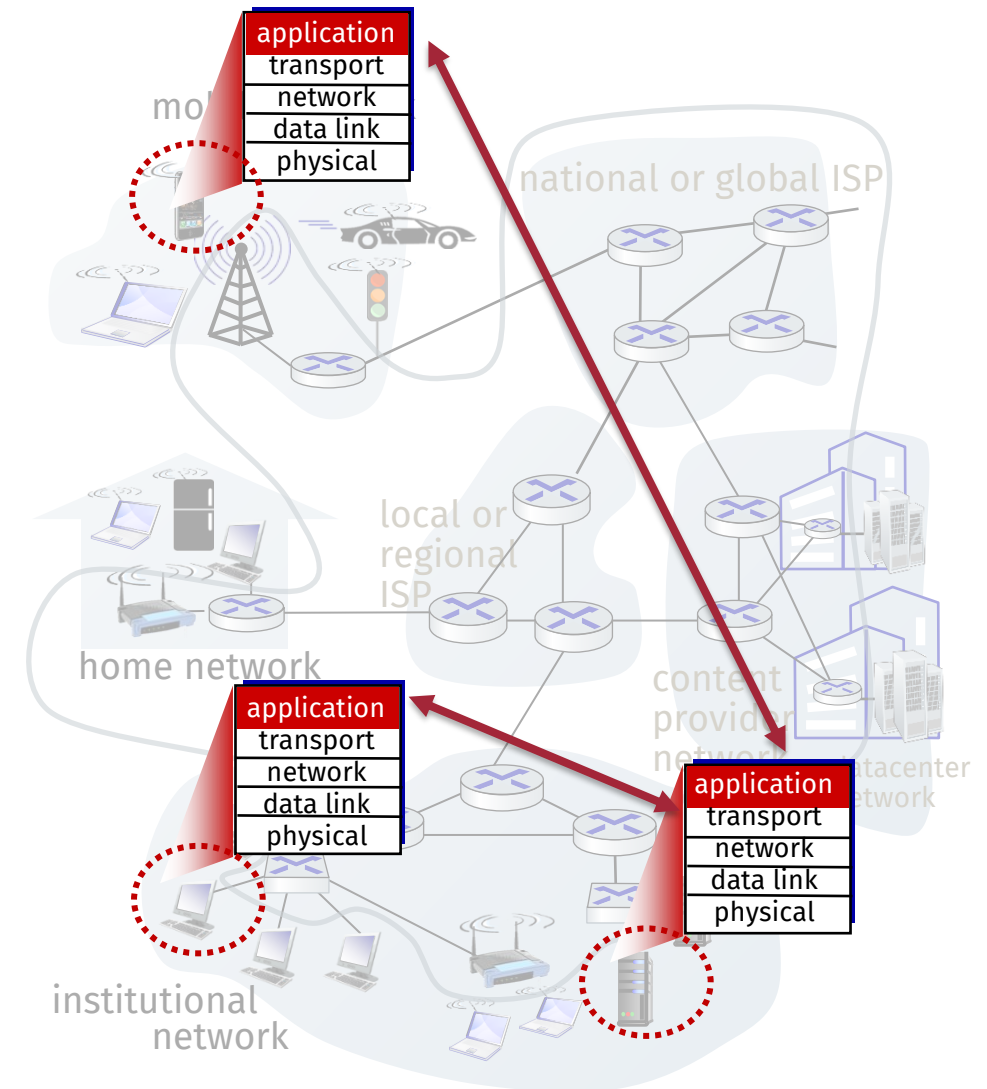
Typische Netzwerkanwendungen

- E-Mail
- World Wide Web
- Instant Messaging
 - z.B. WhatsApp, WeChat
- Remote Login
- P2P-Anwendungen
 - z.B. File Sharing
- Multiplayer-Spiele
 - z.B. Massively Multiplayer Online Games (MMOs)
- Video-Streaming
 - z.B. YouTube, Netflix
- Voice over IP (VoIP)
 - z.B. Skype, TeamSpeak, Discord
- Echtzeit-Videokonferenzen
 - z.B. Zoom, MS Teams, Skype
- Social Networking
- etc.

Erstellung von Netzwerkanwendungen

Erstellen von Programmen, die...

- auf unterschiedlichen Endsystemen laufen
- über das Netzwerk kommunizieren
 - typischerweise über eine API bzw. Service Access Point (SAP) mit der Transportschicht
 - Beispiel: Webbrowser kommuniziert über TCP-Sockets mit dem Webserver
- auf bestimmter Architektur basieren
 - Client-Server vs. Peer-to-Peer (P2P)
- unabhängig vom Kernnetz entwickelt werden
 - Komponenten im Kernnetz sind unabhängig von Netzanwendungen
- schnell entwickelt und verteilt werden können



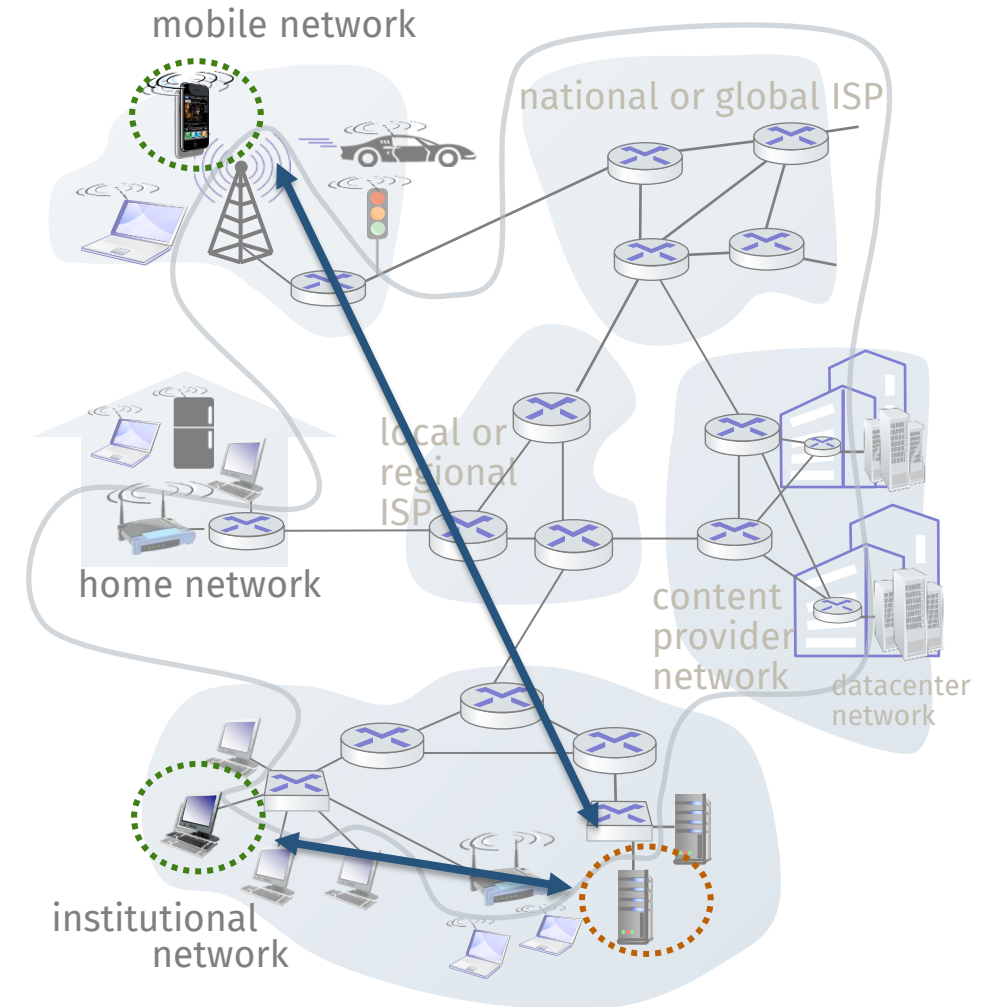
Architekturen für Netzerkanwendungen: Client-Server

Server

- Host dauerhaft erreichbar
 - bietet Dienst an
 - *always-on*
 - permanente IP-Adresse bzw DNS-Name
 - oft in Rechenzentrum oder data center

Clients

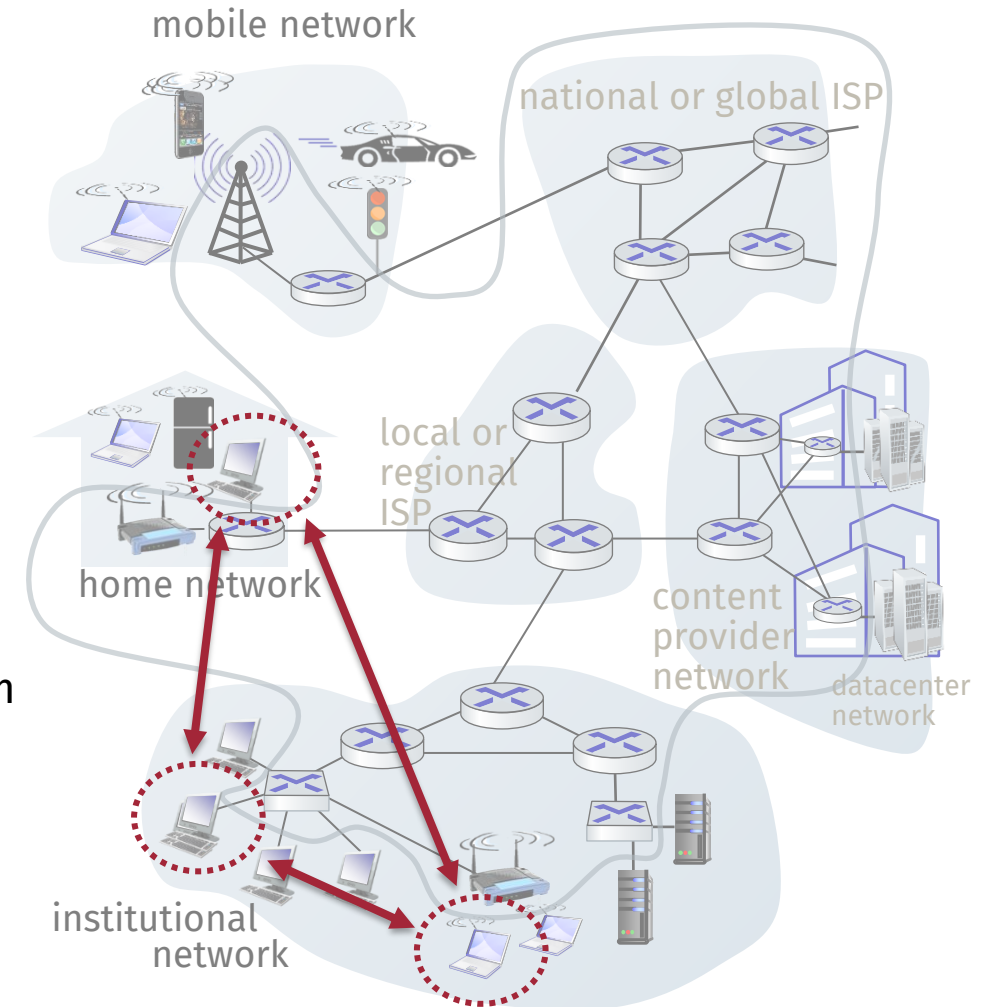
- initiieren Verbindung zum Server
- manchmal nur zweitweise online
- haben oft dynamische, wechselnde IP-Adressen
- kommunizieren nicht direkt miteinander
- nutzen Dienst vom Server



Architekturen für Netzwerkanwendungen: Peer-to-Peer

P2P-Architektur

- kein ständig verfügbarer Server
 - direkte Kommunikation beliebiger Hosts (**Peers**)
 - Peers
 - nutzen Dienste anderer Peers
 - erbringen Dienste für andere Peers
 - nur zeitweise verbunden (wechselnde IP-Adressen)
 - Selbstskalierbarkeit
 - Steigerung der Leistungsfähigkeit des Dienstes mit jedem weiteren Peer
- ➔ komplexes Management



Anwendungsprotokolle (1)

Spezifikation von Anwendungsprotokollen

- Typ der Nachricht
 - z.B. Request oder Response
- Nachrichtensyntax
 - Felder und Begrenzungen
- Nachrichtensemantik
 - Bedeutung der Informationen in Feldern
- Regeln für das Verhalten von Beteiligten
 - Wann senden Prozesse Nachrichten?
 - Wie reagieren sie auf eingehende Nachrichten?

Offene Protokolle

- offene Spezifikationen von Anwendungen
- Internet: meist definiert in RFCs
- Sicherstellung von Interoperabilität
- Beispiele: HTTP, SMTP

Proprietäre Protokolle

- nicht-öffentliche Spezifikationen
- oft für kommerzielle Anwendungen im Internet
- Beispiele: Skype-Protokoll, Online-Spiele

Anwendungsprotokolle (2)

Wichtige Aspekte von Anwendungsprotokollen

- ASCII-basierte vs. binäre Protokolle
- Control- vs. (Nutz-)Daten-Nachrichten
- zentralisierte vs. dezentrale Architektur
- zustandlos vs. zustandsbehaftet
- Transport: zuverlässig vs. unzuverlässig
- Komplexität im Edge Netzwerk
- Core Netzwerk: KISS Prinzip
 - keep it simple and stupid

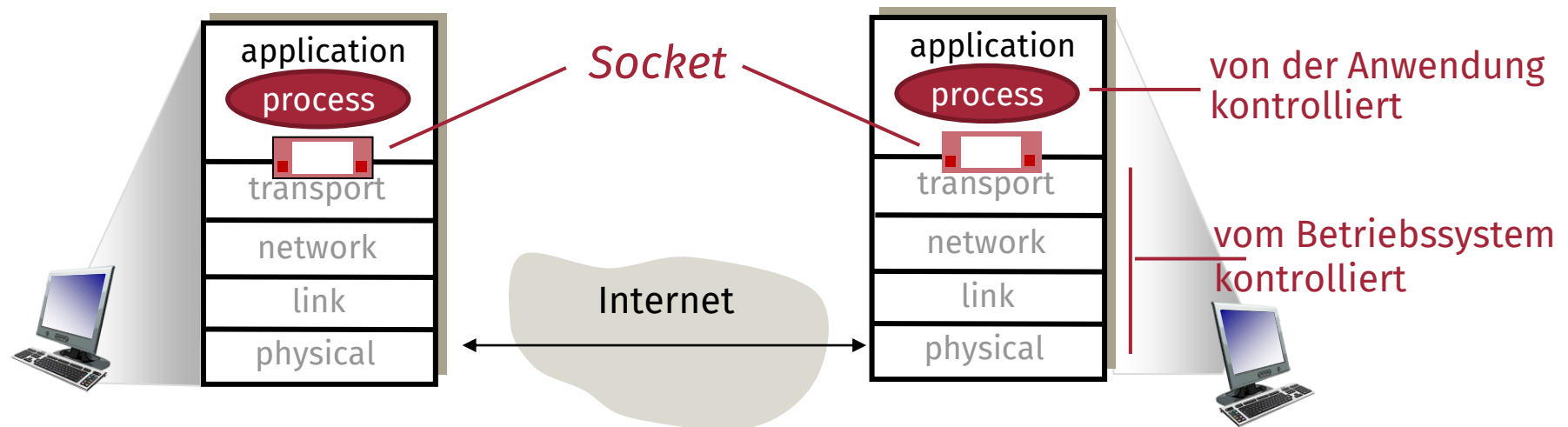
Häufige Merkmale

- Request/Reply-Nachrichtenaustausch
 - Client fragt Informationen oder Dienst an
 - Server antwortet mit Daten und/oder Status
- Nachrichtenformate
 - Header: Informationen über Daten im Body
 - Body: eigentliche Daten

Sockets

Kommunikation zwischen zwei Prozessen (laufende Programme)

- auf selbem Host: Inter-Prozess Kommunikation (z.B. shared memory)
- auf unterschiedlichen Hosts: Austausch von Nachrichten
- Prozesse senden und empfangen Nachrichten über Sockets
- Socket ist Schnittstelle zum Betriebssystem (ähnlich zu file descriptor)



Adressierung von Sockets und Prozessen

- Prozesse benötigen Identifikator zur Adressierung in Nachrichten
- Host-Interface hat eindeutige 32-bit oder 128-bit IP-Adresse
- Frage: Reicht die IP-Adresse eines Interfaces aus, um einen Prozess zu adressieren?
→ *Nein. Auf einem Host können viele Prozesse laufen.*
- Identifikator beinhaltet IP-Adresse und Port-Nummer des jeweiligen Prozesses
- well-known Port-Nummern
 - HTTP: 80
 - E-Mail (SMTP): 25
- Adressierung des Webserver `www.uni-ulm.de`
 - IP-Adresse: 134.60.1.22
 - Port-Nummer: `tcp/80`

Dienste der Transportschicht für die Anwendungsschicht

Datenintegrität

- Zuverlässigkeit des Datentransports
 - gesendete Daten $\hat{=}$ empfangene Daten
- ➔ Tolerierung von Paketveränderungen und Paketverlusten?
 - z.B. E-Mail vs. Audio-Streaming

Quality of Service

- Latenz zwischen Endpunkten beim Transport von Anwendungsdaten
- ➔ Notwendigkeit niedriger Latenzen?
 - z.B. Web vs. Internet-Telefonie

Durchsatz

- Durchsatz bei der Übertragung von Anwendungsdaten
- ➔ Notwendigkeit eines Mindestdurchsatzes?
 - z.B. Video-Streaming vs. File Transfer

Security

- u.a. Vertraulichkeit und Unveränderlichkeit (Integrität) der Anwendungsdaten

Anforderungen an die Transportschicht

Anwendung	Datenverlust	Durchsatz	Delay kritisch?
File Transfer	nicht tolerierbar	elastisch	nein
E-Mail	nicht tolerierbar	elastisch	nein
Web-Surfen	nicht tolerierbar	elastisch	nein
Audio/Video (Livestream)	tolerierbar	Audio: 5 kbit/s – 1Mbit/s Video: 10 kbit/s – 25Mbit/s	ja, oft < 100ms
Audio/Video (gepuffert)	tolerierbar	(siehe oben)	ja (einige Sekunden)
interaktive Online-Spiele	tolerierbar	kbit/s oder mehr	ja, oft < 100ms
Text-Messaging	nicht tolerierbar	elastisch	ja/nein

Dienste von TCP und UDP

TCP

- **zuverlässiger Transport** zwischen sendendem und empfangendem Prozess
 - **Flusskontrolle** (flow control): Sender überlastet Empfänger nicht
 - **Lastkontrolle** (congestion control): Sender wird gedrosselt, wenn Netzwerk überlastet
 - **verbindungsorientiert**: expliziter Verbindungsaufbau zwischen Client und Server
- ◆ **fehlend**: QoS, Delay, minimaler Durchsatz, Security

UDP

- **unzuverlässiger Datentransport** zwischen sendendem und empfangendem Prozess
 - **verbindungslos**: Client und Server tauschen Nachrichten aus
- ◆ **fehlend**: Zuverlässigkeit, Flusskontrolle, Lastkontrolle, Durchsatz, Security
- (Frage: Warum gibt es UDP?)

Sicherheit bei TCP

TCP & UDP

- jeweils keine Verschlüsselung
- ◆ Übertragung sensitive Daten (Passwörter, Kreditkartennummern, etc.) im Klartext im Internet

TLS (früher SSL)

- verschlüsselte und integritätsgeschützte TCP-Verbindungen
- Authentisierung der Kommunikationsendpunkte (mindestens Server, Client nur optional)
- Realisierung im Session Layer oder Application Layer
- Implementierung in Form von TLS-Bibliotheken (intern Verwendung von TCP-Sockets)
- TLS Socket API
 - Klartextdaten, die an TLS Socket gesendet werden, werden verschlüsselt über das Internet übertragen
 - mehr dazu in Kapitel H

Transportprotokolle bei Anwendungen im Internet

Anwendung	Anwendungsprotokoll	Transportprotokoll
E-Mail	SMTP [RFC2821]	TCP
Remote Terminal	Telnet [RFC854]	TCP
World Wide Web	HTTP/1.1 [RFC2616]	TCP
File Transfer	FTP [RFC959]	TCP
Multimedia-Streaming	HTTP (z.B. YouTube), RTP [RFC1889]	TCP oder UDP
Internet-Telefonie	SIP/RTP, proprietär	TCP oder UDP

B.2 HTTP und das Web

WWW und HTTP

Überblick

- Webseiten bestehen aus 1...n Ressourcen
 - HTML-Dokumente
 - Bilder (JPEG, PNG, etc.), JavaScript-Dateien, CSS-Dateien, Multimediateien etc.
- Webseite besteht aus HTML-Dokument und darin referenzierten Ressourcen
- eindeutige Identifikation von Ressourcen durch Uniform Resource Locator (URL)

`http://www.uni-ulm.de/in/vs/index.html`

Protokoll

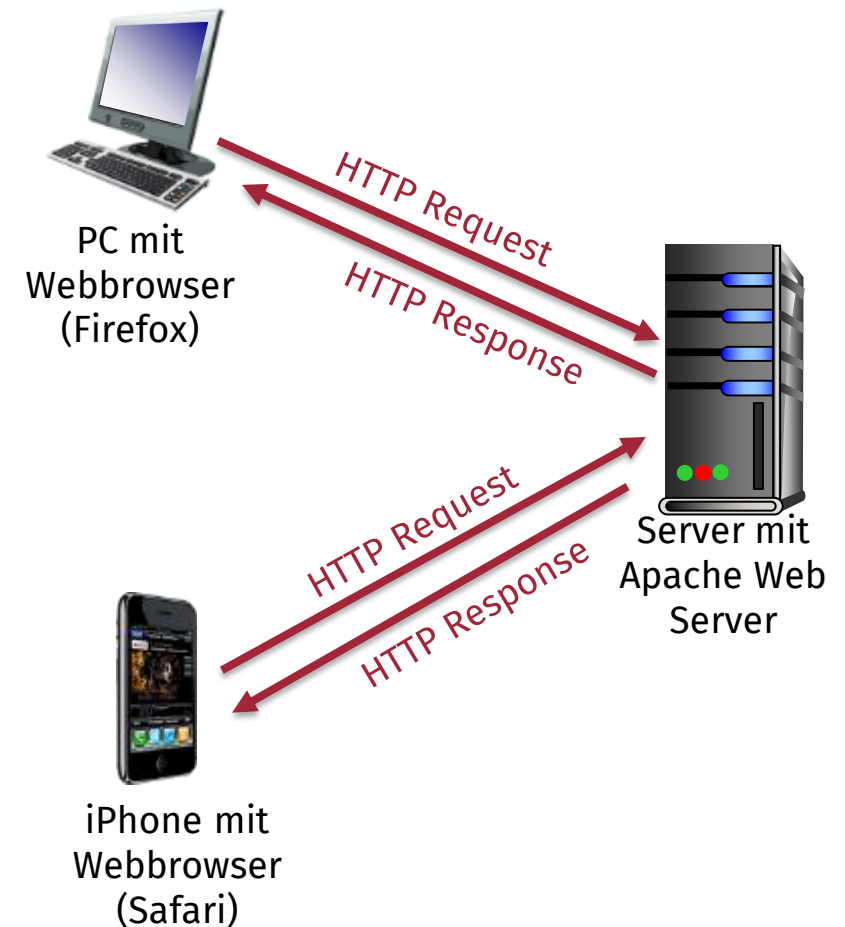
Hostname (DNS)

Pfad

HTTP/1.1: Überblick (1)

Hypertext Transfer Protocol (HTTP)

- Anwendungsprotokoll des World Wide Web
- Client/Server-Modell
- Client (meist Webbrowser)
 - senden (HTTP-)Requests, empfangen Ressourcen
 - meist auch: Rendern des Ressourcen
- Server (Webserver)
 - beantworten HTTP-Requests mit HTTP-Responses
 - stellen statische und dynamische Ressourcen bereit
 - statische Ressourcen: z.B. Dateien auf Webserver
 - dynamische Ressourcen: serverseitiges Programm erstellt Ressource zur Laufzeit (z.B. mit Datenbank)



HTTP/1.1: Überblick (2)

Verwendung von TCP

- Client initiiert TCP-Verbindung zum Server
 - Port tcp/80
- Server akzeptiert TCP-Verbindungsaufbau vom Client
- Austausch von HTTP-Nachrichten (application-layer protocol messages) zwischen Browser (HTTP-Client) und Web Server (HTTP-Server)
- TCP-Verbindung wird geschlossen

Zustandslosigkeit bei HTTP

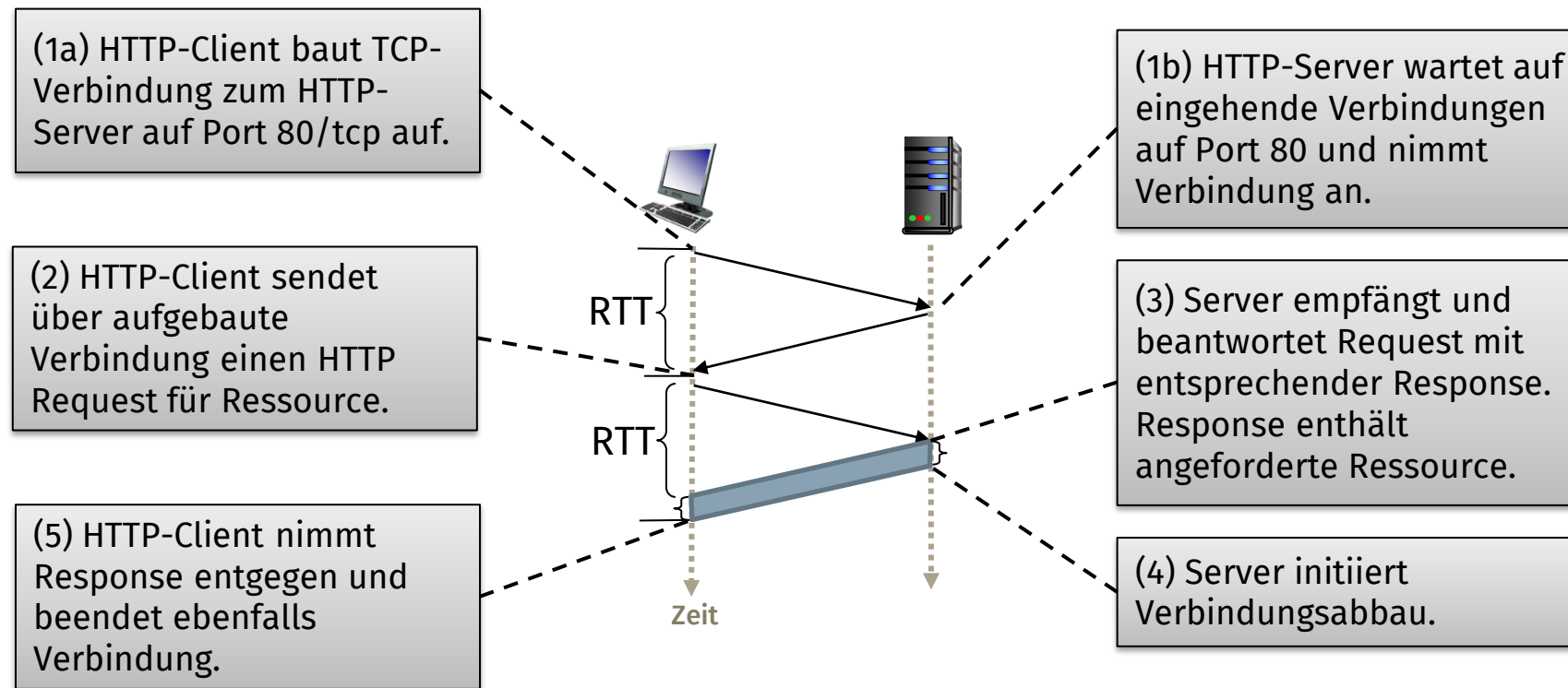
- HTTP ist zustandslos (*stateless*)
- Server behält keine Informationen über frühere Anfragen des Clients

Bemerkung

- allgemein höhere Komplexität bei zustandsbehafteten Protokollen
- Zustand früherer Anfragen muss (oft serverseitig) gespeichert und verwaltet werden
- bei Verbindungsabbruch oder Absturz: Gefahr inkonsistenter Zustände

Verbindungen bei HTTP (1)

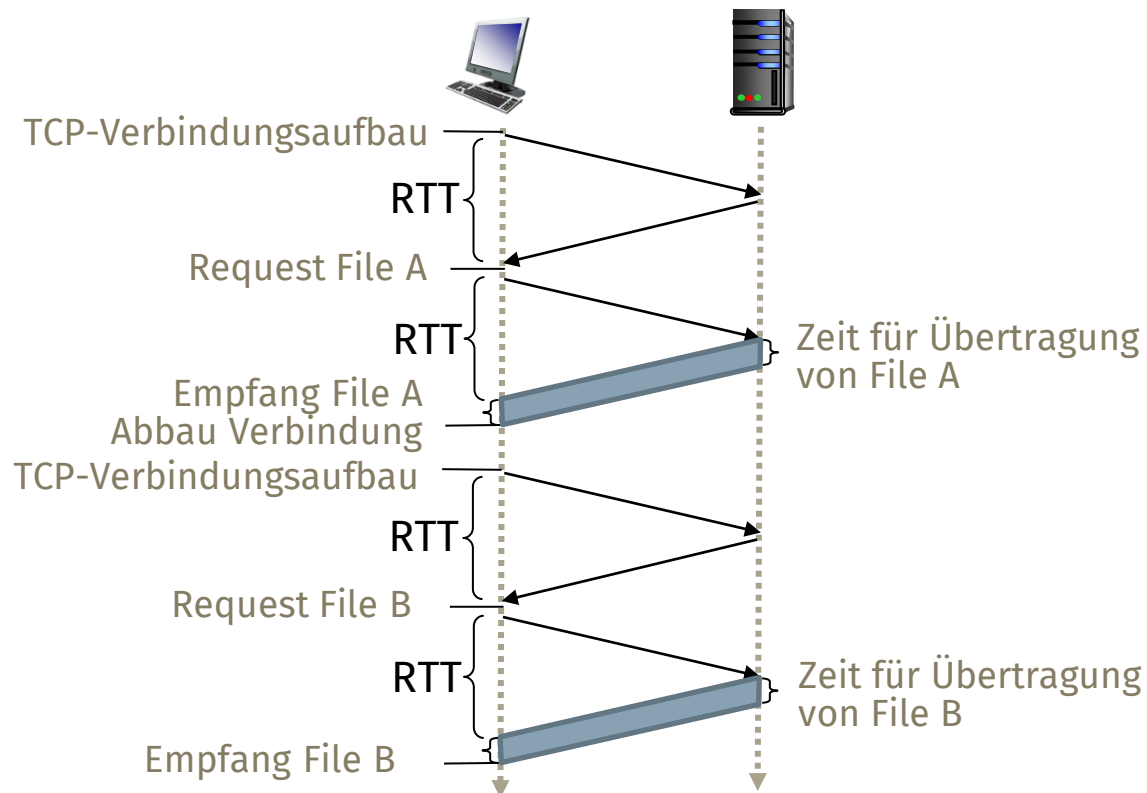
Detaillierter Ablauf einer Verbindung und Anfrage



Verbindungen bei HTTP (2)

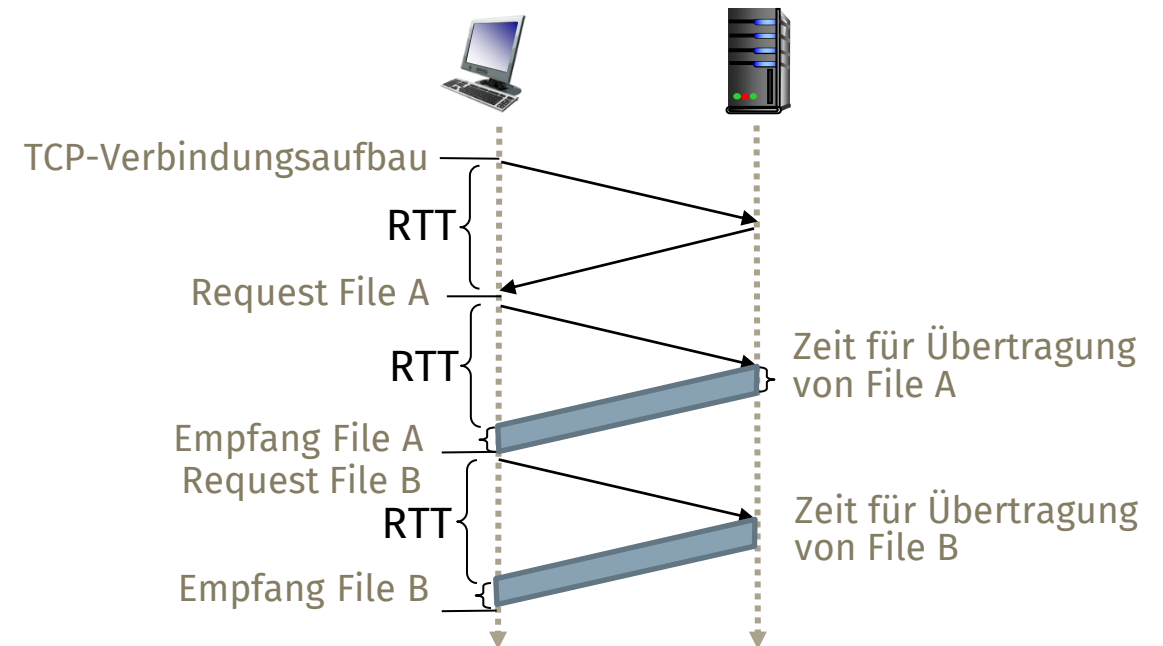
Nicht-persistente HTTP-Verbindung

- maximal eine Ressource je TCP-Verbindung
- Verbindungsende nach Übertragung

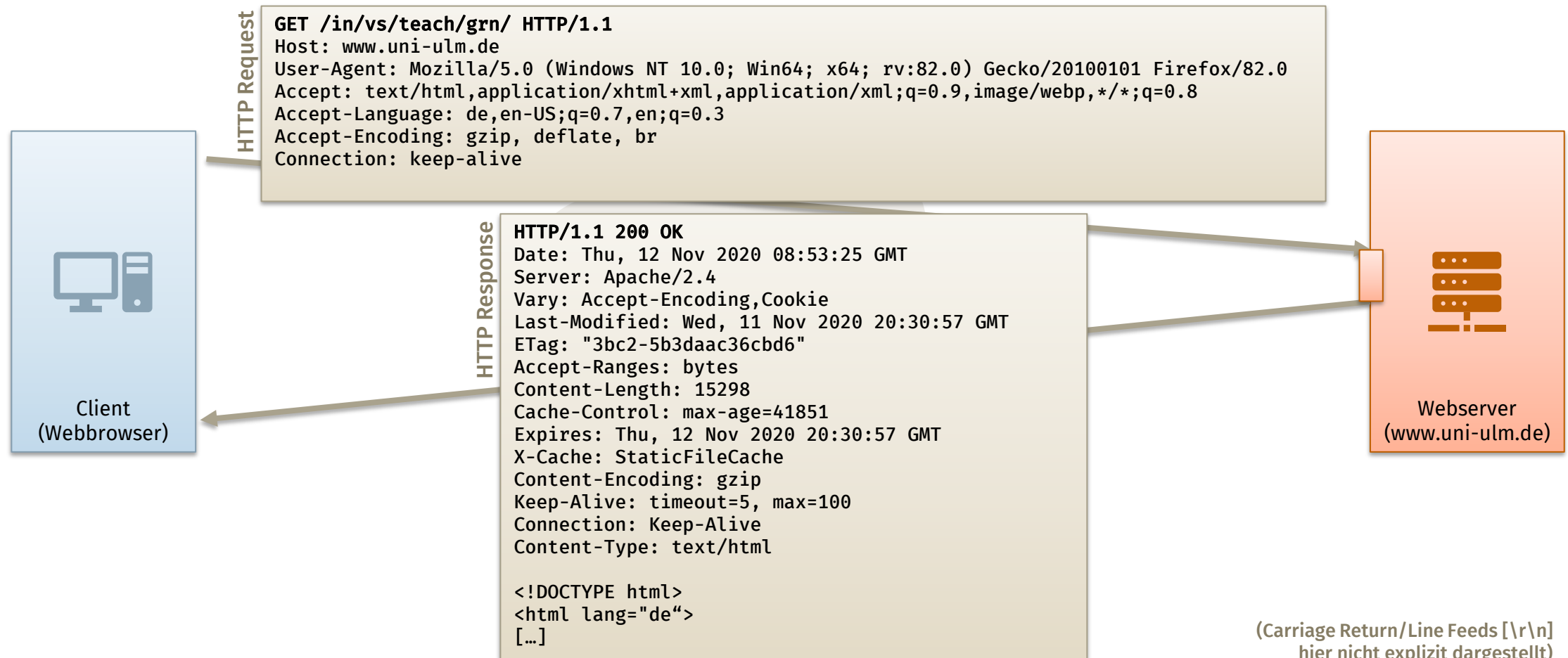


Persistente HTTP-Verbindung

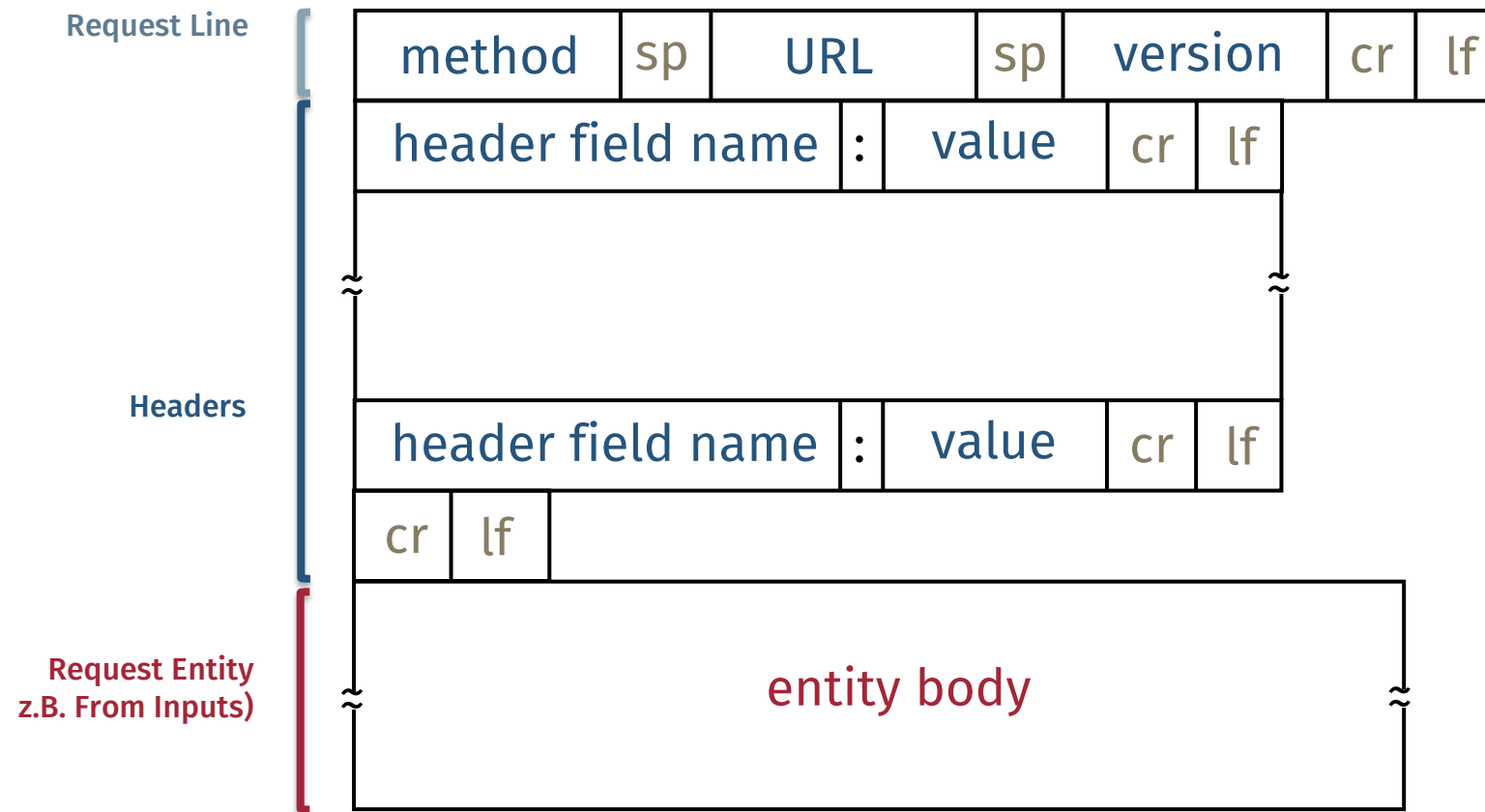
- Verwendung einer TCP-Verbindung für die sequentielle Übertragung von Ressourcen



Nachrichtenformat bei HTTP



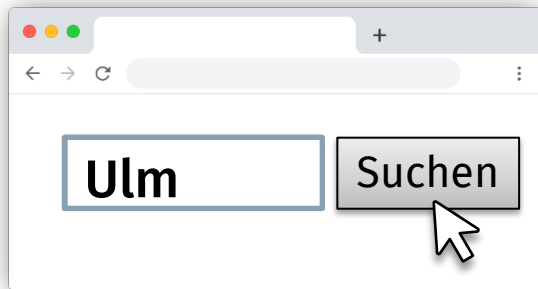
Nachrichtenformat: HTTP Request



HTTP Methoden

- GET
 - (idempotenter) Abruf
- POST
 - (nicht-idempotenter) Abruf
 - Client-Daten in Request Body
- HEAD
 - wie GET, aber nur mit Header in Response, jedoch ohne Body
- PUT (ab HTTP/1.1)
 - Upload einer Ressource an die spezifizierte URL
- DELETE (ab HTTP/1.1)
 - Löschen einer Ressource auf Server
- OPTIONS

Client-Daten bei Requests: GET vs. POST Methode



Request mit GET-Methode

```
GET /search?query=Ulm HTTP/1.1  
Host: website.com
```

Request mit POST-Methode

```
POST /search HTTP/1.1  
Host: website.com  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 9  
  
query=Ulm
```

Nachrichtenformat: HTTP Response

Protokoll Status Code Reason Phrase

Response Line
(Statuszeile)

HTTP/1.1 200 OK\r\n

Headers

Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n

Feld-Name

Content-Type: text/html; charset=ISO-8859-1\r\n

Feld-Wert

\r\n

Ressource Entity
(Response Body,
z.B. HTML-Datei)

<html xmlns='http://www.w3.org/1999/xhtml'>
<head> [...]

Beispiele für Response Status Codes

- 1xx: Informational
- 2XX: Success
 - 200 OK
 - Request erfolgreich, Ressource im Body
- 3XX: Redirection
 - 301 Moved Permanently
 - Ressource dauerhaft verschoben, neue URL in Location Header
- 4XX: Client Error
 - 404 Not Found
 - angefragte Ressource existiert nicht
- 5XX: Server Error
 - 505 HTTP Version Not Supported
 - unbekannte HTTP Version

Ausblick: Neuere HTTP-Protokollversionen

HTTP/2 und HTTP/3

- verschiedene Weiterentwicklungen in den letzten Jahren
- binäre HTTP-Varianten
- mehrere parallele (logische) Datenströme in einer TCP-Verbindung
- Änderungen im Zusammenhang mit der Transportschicht

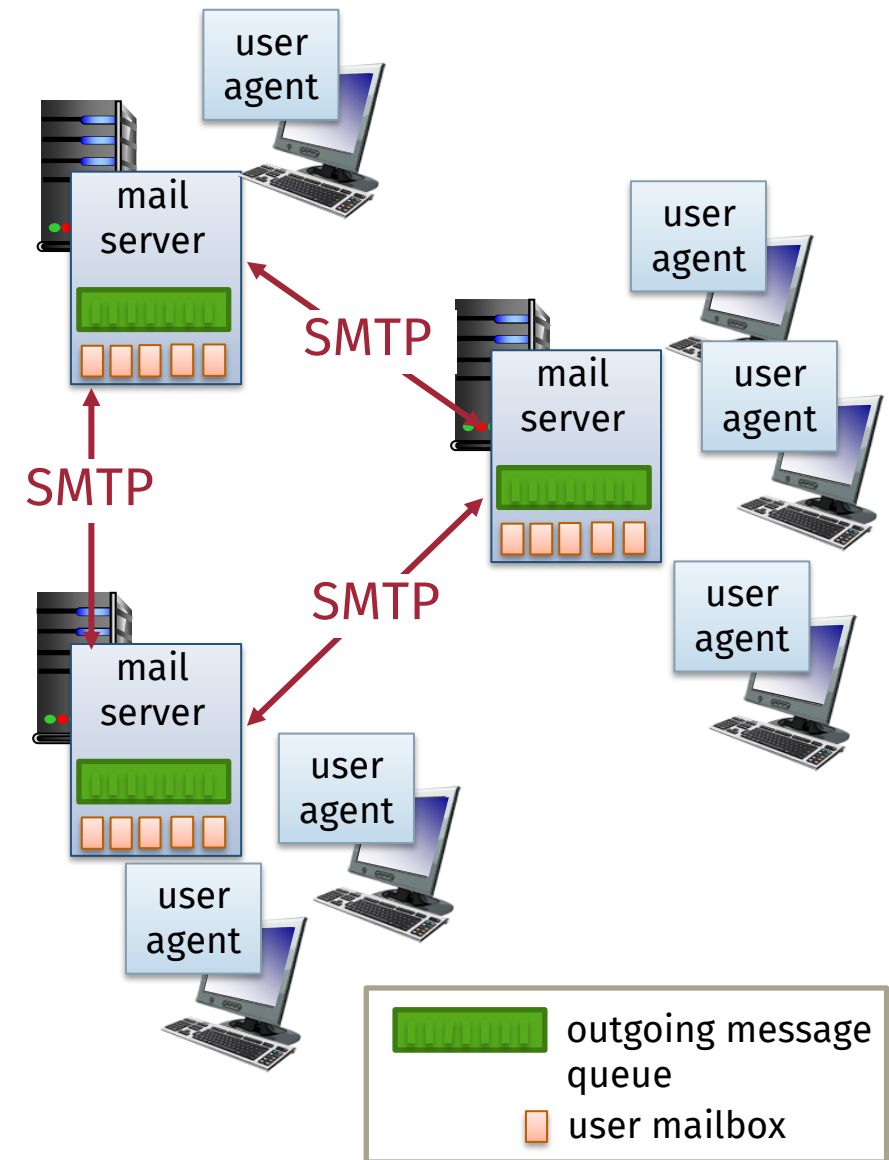
→ später mehr zu neueren HTTP-Versionen in Kapitel C

B.3 E-Mail

Electronic Mail (E-Mail)

Hauptbestandteile des Systems:

- User Agents
 - a.k.a. “Mail Client”
 - erstellen und lesen von Mails
 - z.B. Outlook, Thunderbird, iPhone Mail Client, etc.
- Mail Server
 - Mailbox beinhaltet eingehende Mail Nachrichten der User
 - SMTP-Protokoll zum Austausch von Mail Nachrichten zwischen Servern (und Abschicken von Nachrichten vom User Agent)
 - Abrufen von Nachrichten vom Server über andere Protokolle (siehe später)
- Simple Mail Transfer Protocol (SMTP)



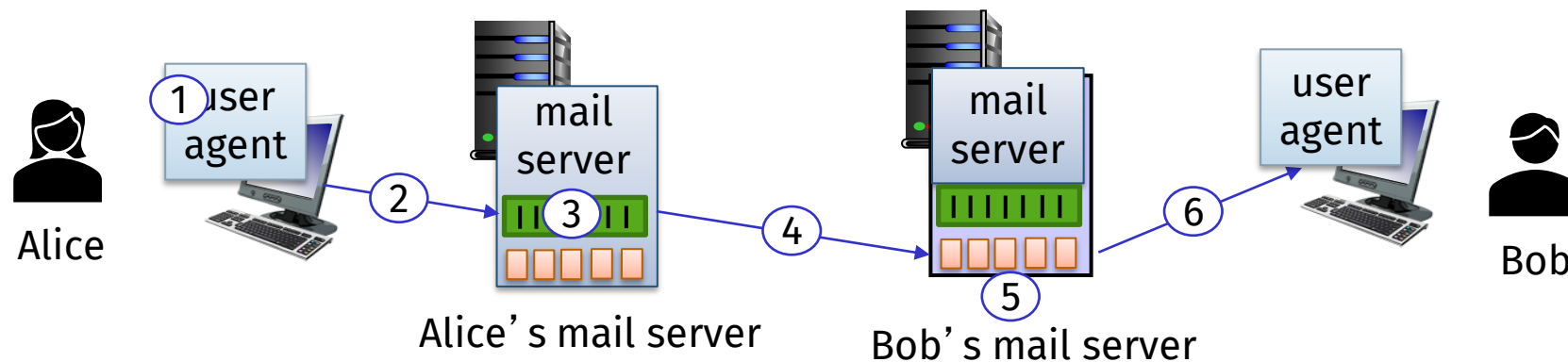
Electronic Mail: SMTP [RFC 2821]

Simple Mail Transfer Protocol

- TCP-basiert für zuverlässigen Nachrichtentransport vom Client zum Server
- well-known Port: tcp/25
- direkte Übertragung vom sendenden Server zum empfangenden Server
 - früher auch häufiges Store-and-Forward
- drei Protokollphasen
 - Handshake (Hello)
 - Nachrichtentransfer
 - Beenden der Verbindung
- Command/Response Protokoll (wie HTTP, FTP)
 - ASCII-basiert
 - Response: Status Code und Text
 - Nachrichten müssen in 7-bit ASCII vorliegen → MIME encoding

Szenario: Alice sendet E-Mail an Bob

1. Alice erstellt E-Mail im UA, adressiert an bob@somewhere.de
2. User Agent von Alice sendet die Nachricht an ihren Mail Server, Nachricht wird dort in Queue gespeichert.
3. Alice's Mailserver öffnet TCP-Verbindung zu Bob's Mailserver.
4. SMTP-Client sendet E-Mail Nachricht über die TCP-Verbindung.
5. Bob's Mailserver speichert E-Mail Nachricht in Bob's Mailbox.
6. Bob startet User Agent und ruft die Nachricht ab.



Beispiel für SMTP Protokoll

```
$ ncat mail.uni-ulm.de 25
S: 220 mail.uni-ulm.de ESMTP Sendmail 8.14.6/8.14.2; Sat, 12 Oct 2013 12:27:02 +0200 (CEST)
C: HELO kargl.net
S: 250 mail.uni-ulm.de Hello [130.255.105.88], pleased to meet you
C: MAIL FROM: <frank@kargl.net>
S: 250 2.1.0 <frank@kargl.net>... Sender ok
C: RCPT TO: <frank.kargl@uni-ulm.de>
S: 250 2.1.5 <frank.kargl@uni-ulm.de>... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Subject: Time for lunch
C:
C: Let's have a break!!!
C: .
S: 250 2.0.0 r9CAR23T011885 Message accepted for delivery
C: QUIT
S: 221 2.0.0 mail.uni-ulm.de closing connection
```


Eigenschaften von SMTP

Protokoll-Merkmale von SMTP

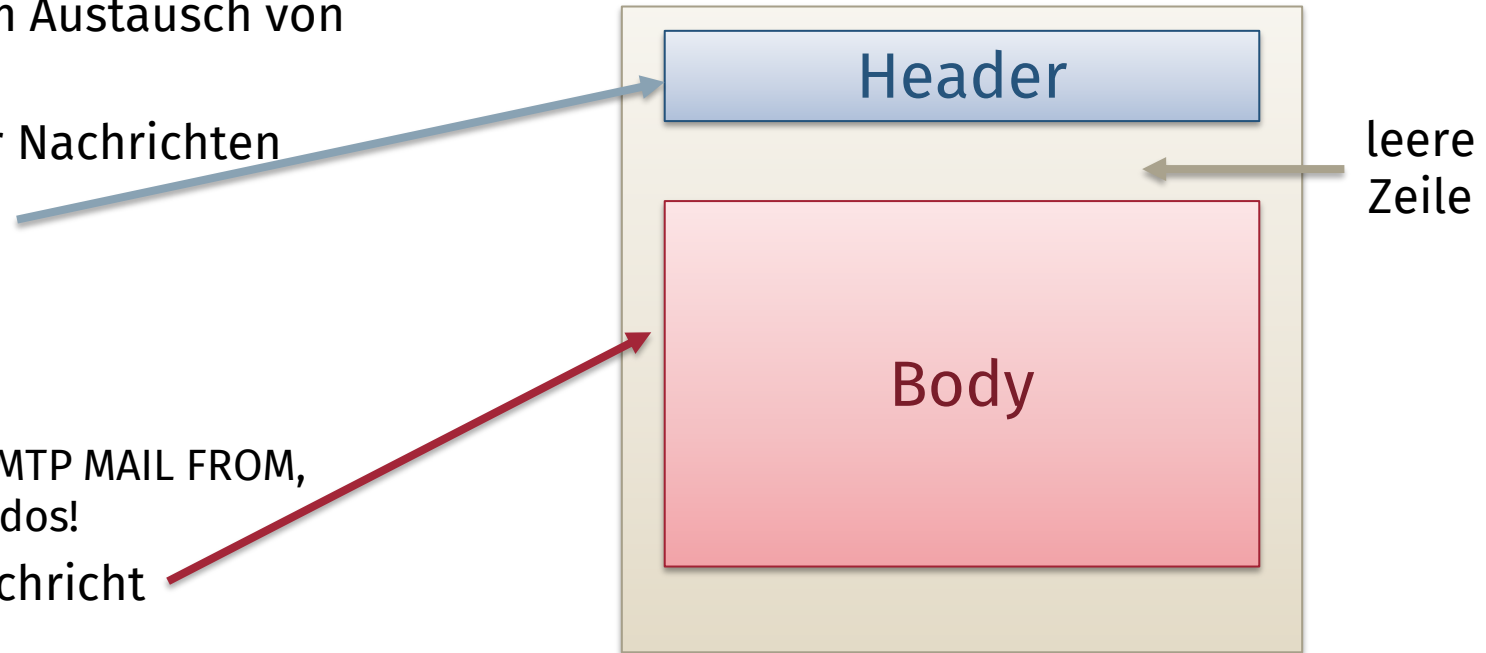
- Verwendung persistenter Verbindung
 - mehrere E-Mails in einer Verbindung möglich
- SMTP verlangt, dass die Nachricht (Header & Body) in 7-bit ASCII vorliegen
 - MIME Encoding
- `\r\n.\r\n` beendet Nachricht

Vergleich mit HTTP

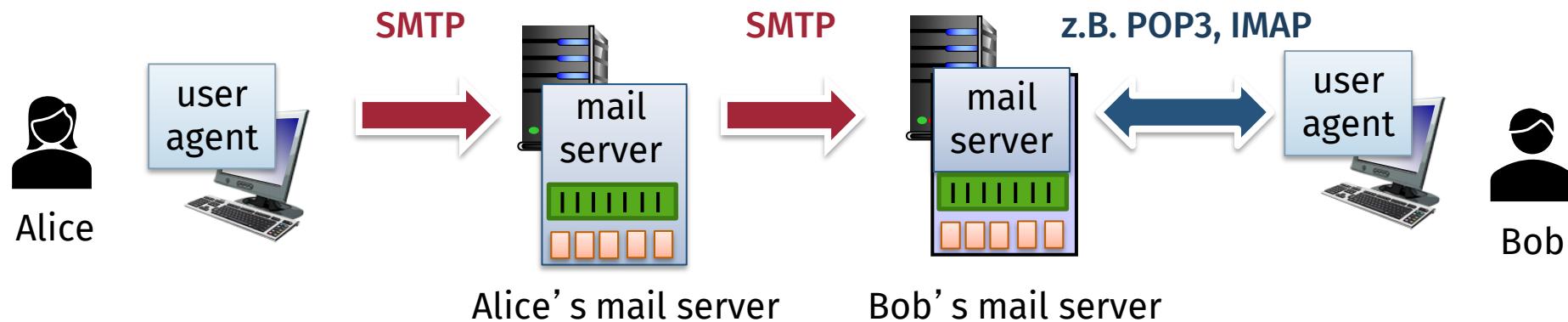
- Interaktionsmodell
 - HTTP: pull Modell
 - SMTP: push Modell
- Übertragung mehrerer Ressourcen
 - HTTP: jedes Objekt in eigener Response Nachricht
 - SMTP: erlaubt mit MIME auch multipart Nachrichten
- ASCII Command/Response Protokoll, Status Codes

Mail-Nachrichtenformat

- SMTP: Protokoll zum Austausch von Nachrichten
- RFC 822: Aufbau der Nachrichten
- Header-Zeilen, z.B.
 - To:
 - From:
 - Subject:
 - unabhängig von SMTP MAIL FROM, RCPT TO: Kommandos!
- Body: eigentlich Nachricht
 - nur ASCII Zeichen



Mail Access Protocols



SMTP

- Zustellung bis zum Server des Empfängers
- dort: Speicherung in „Inbox“


Mail Access Protokoll (Mail-Abholung aus Inbox)

- POP3: Post Office Protocol [RFC 1939]
 - Authentisierung/Autorisierung, Download von Nachrichten
- IMAP: Internet Mail Access Protocol [RFC 1730]
 - mehr Features als POP3, Emails verbleiben auf Server, Mailfolder
- alternativ: Abruf über Web-Frontend (HTTP)
 - z.B. gmx.de, web.de, gmail, etc.

POP3-Protokoll

■ Authorization Phase


- Client-Kommandos
 - **user:** Benutzername
 - **pass:** Passwort (Klartext!)
- Server-Antworten
 - +OK
 - -ERR



```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully
    logged on
```

■ Transaktionsphase

- Client-Kommandos
- **list:** vorhandene Nachrichten auflisten
- **retr:** Nachrichten abrufen
- **dele:** Nachrichten löschen
- **quit**



```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

IMAP-Protokoll

- Nachrichten verbleiben normalerweise auf Server
- Organisation von Nachrichten in Foldern durch User
- Verwaltung von Zuständen von Nachrichten
 - Zuordnung zu Foldern
 - Metadaten
 - z.B. gelesen, markiert

```
$ nc localhost 143
* OK Dovecot ready.
a001 login fkargl secret_password
a001 OK Logged in.
a002 select inbox
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft $NotJunk NotJunk JunkRecorded
$Junk $Forwarded Forwarded $MailFlagBit0 $MailFlagBit1 $MailFlagBit2)
...
* 317 EXISTS
* 0 RECENT
...
a002 OK [READ-WRITE] Select completed.
a003 fetch 12 body[header]
* 12 FETCH (BODY[HEADER] {884}
Return-Path: <ieee_return8@mmsend10.com>
X-Original-To: frank@kargl.net
Delivered-To: fkargl@kargl.net
...
Date: Thu, 25 Jul 2013 12:23:53 -0400
From: IEEE CommSoc <CommunicationsSociety@comsoc.org>
To: <frank@kargl.net>
Subject: Get Connected - IEEE ComSoc Global Network
a003 OK Fetch completed.
a004 store 12 +flags \deleted
* 12 FETCH (FLAGS (\Deleted \Seen))
a004 OK Store completed.
a005 logout
* BYE Logging out
a005 OK Logout completed.
```

B.4 DNS

Domain Name System (DNS)

Identifikatoren für Menschen

- Name, Ausweisnummer, Steuernummer

Internet Hosts & Routing

- IP Adresse (32 Bit)
 - u.a. Verwendung für Routing in Netzwerkschicht
- Hostname (z.B. `www.uni-ulm.de`)
 - wird von Menschen verwendet
 - z.B. als Teil von URLs („Internetadressen“)

Domain Name System:

- verteilte Datenbank
 - hierarchische Topologie
 - Strukturierung durch viele DNS Nameserver
- *Anwendungsprotokoll* zur Kommunikation zwischen DNS Nameserver und DNS-Clients
 - DNS Protokoll: zentrale Internet-Funktionalität *als Anwendung* implementiert
 - kein eigentlicher Bestandteil von TCP/IP
 - KISS Prinzip: Komplexität nur am Rand (Edge)

Frage: Wie findet die Übersetzung statt?
Namen ↔ IP-Adressen

DNS: Dienste & Struktur

Dienste von DNS

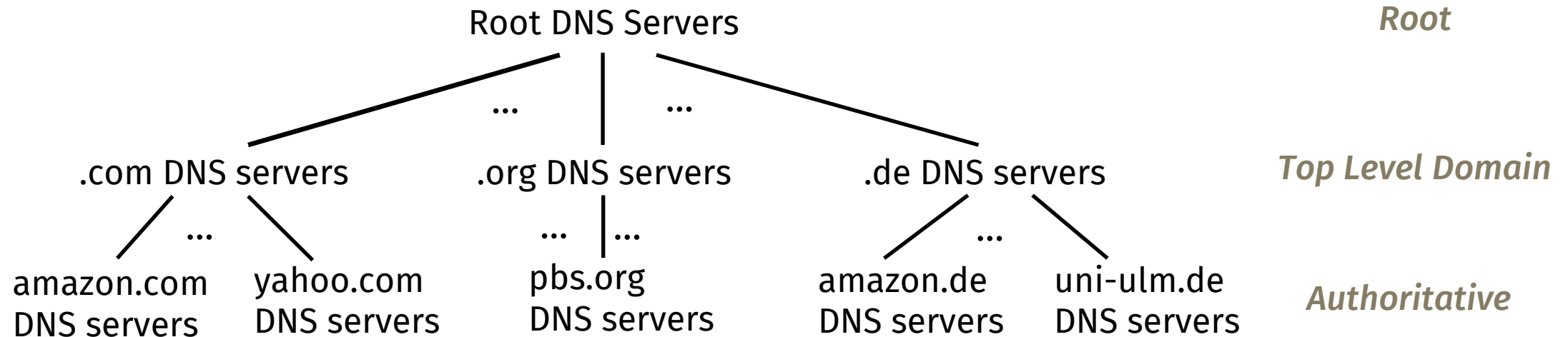
- Auflösung von Hostnamen & IP-Adressen
 - IP address translation: (Hostname => IP)
 - reverse translation: (IP => Hostname)
- Host Aliase (alternative Namen)
- zuständige Mail Server
- Lastverteilung
 - ein Name zeigt auf viele IP-Adressen (z.B. Web-Server Farm)
 - Aufteilung von neu eingehenden Anwendungsverbindungen auf eine IP-Adresse aus Pool
- weitere Dienste/Auflösungen

Warum kein zentrales DNS?

- ◆ *single point of failure*
- ◆ Datenvolumen
- ◆ Wartung und Zuverlässigkeit
- ◆ verteiltes Management der Daten

*Zentralisierte Lösungen
skalieren nicht gut in
verteilten Systemen!*

DNS: Eine verteilte, hierarchische Datenbank



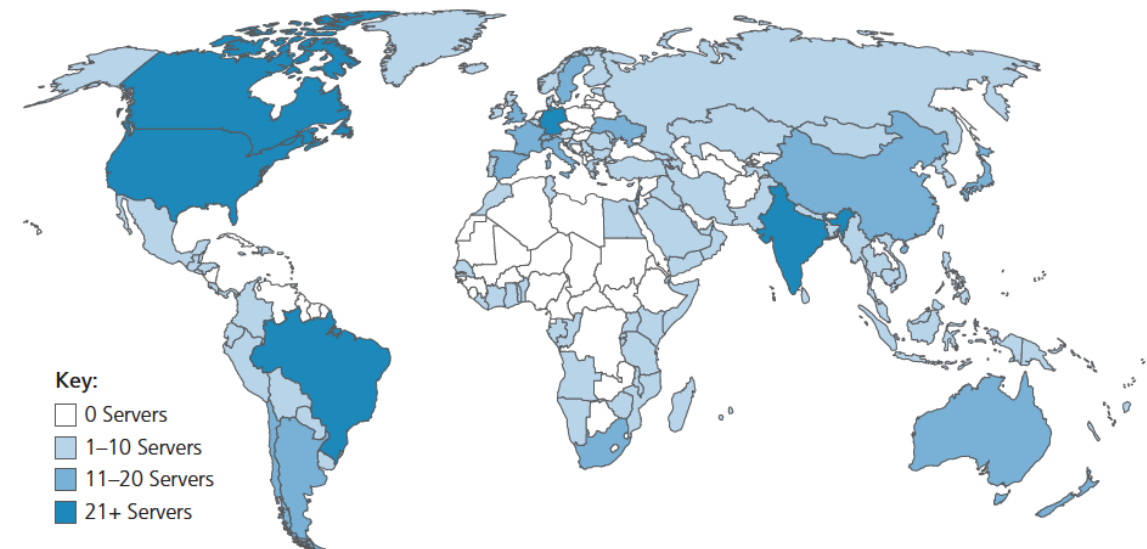
Ungefährer Ablauf

- Client möchte `www.uni-ulm.de` zugreifen
- Client erhält vom Root DNS Server de TLD DNS Server
- Client erhält vom de TLD DNS Server den `uni-ulm.de` DNS Server
- Client fragt `uni-ulm.de` DNS Server nach IP-Adresse von `www.uni-ulm.de`

DNS Root Nameserver

- Kontaktierung durch andere DNS-Server falls Top-Level-Domain (TLD) nicht direkt auflösbar
 - passiert extrem selten (Caching von Ergebnissen)
- essenziell wichtig für das Internet
 - vermutlich Zusammenbruch des Internets beim Ausfall aller DNS Root-Server
 - Verwaltung durch die **Internet Corporation for Assigned Names and Numbers (ICANN)**

13 logische Root Name Servers, weltweit repliziert (meist anycast Instanzen mit dutzenden Standorten weltweit; alleine 200 Server in den USA)



Top-level Domain Server und autoritative DNS Server

Top-level Domain (TLD) Server

- verantwortlich für Top-level Domains
 - **Country Code Domains** (de, uk, fr, ca, jp, tv, ...)
 - **Generic TLDs** (com, org, net, edu, aero, jobs, museums, ...)
- Betrieb durch Firmen (z.B. Network Solutions) oder sonstigen Organisationen (z.B. DENIC e.G.)

Autoritative DNS Server

- Nameserver im Besitz von Originaldaten einer Domäne
- Verwaltung einer Zone (Bereich des DNS Namensraums)
 - z.B. uni-ulm.de (abzüglich mathematik.uni-ulm.de etc.)
 - ≠ Domäne (.de Domain beinhaltet mehrere Zonen)
- Primary Nameserver: Verwaltung der Originaldaten
- Secondary Nameserver: Haltung von Kopien aus Redundanzgründen

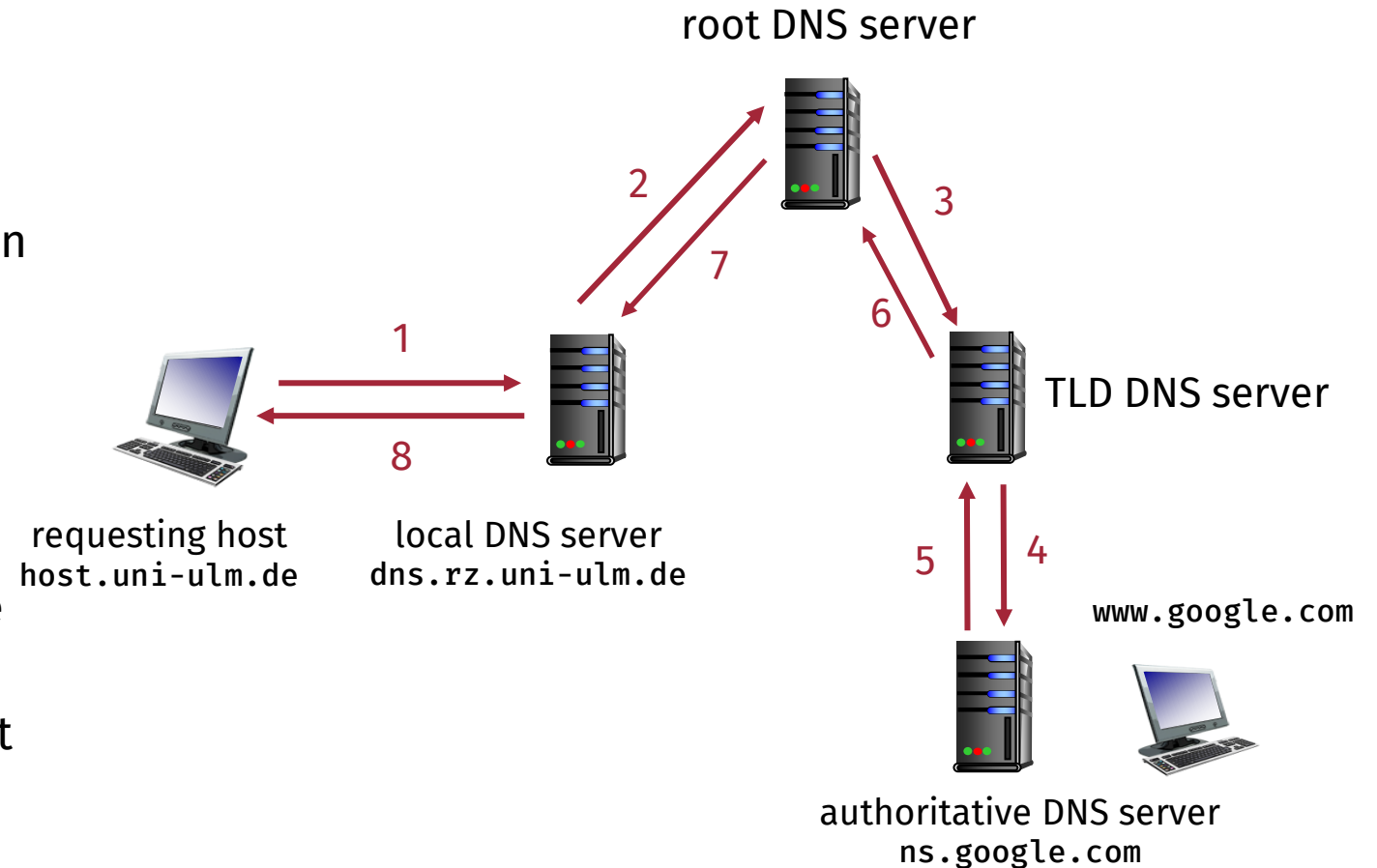
Local Resolving DNS Nameserver

- nicht zwingend Bestandteil der DNS-Hierarchie
 - kann aber auch gleichzeitig autoritativ für eine Zone sein
- ISPs betreiben Resolving Nameserver für ihre Kunden
 - „default name server“ wird z.B. per DHCP zugewiesen
- Hosts senden Anfragen in der Regel an **Resolving Nameserver**
 - verantwortlich für die weitere Auflösung der Anfrage (kontaktiert evtl. weitere Nameserver)
 - speichert Informationen von früheren Anfragen (Caching)

DNS-Namensauflösung (*rekursiv*)

Recursive Query

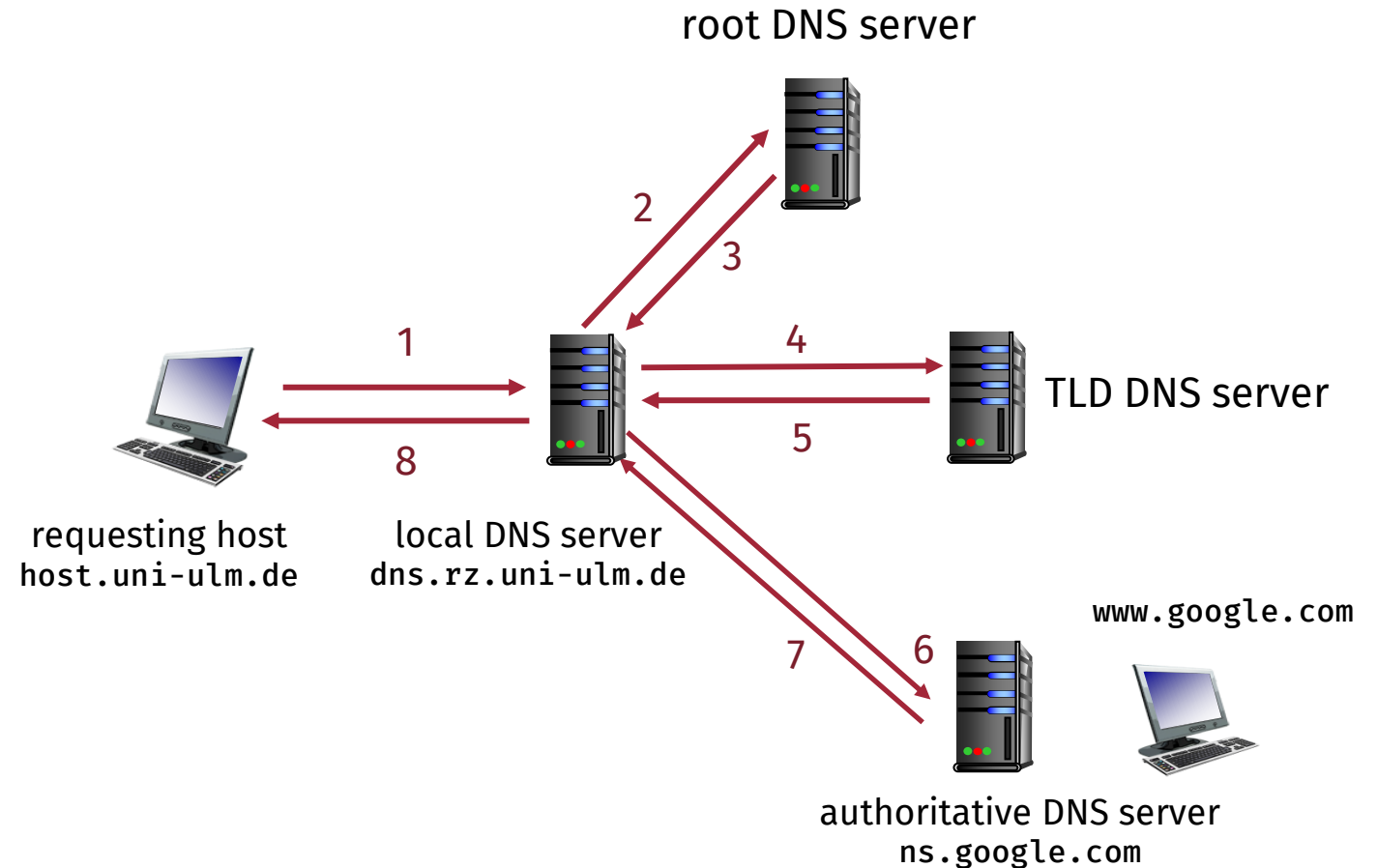
- Anfrage von `host.uni-ulm.de` für Hostname „`www.google.com`“
- angefragter Nameserver löst Namen vollständig auf
- ◆ **Nachteil**
 - hohe Last auf Nameservern nahe der Wurzel (Root & TLD)
 - Root- und TLD-Nameserver verweigern deshalb meist rekursive Auflösungen
- in der Praxis: *nur* local resolver löst Namen rekursiv auf



DNS-Namensauflösung (*iterativ*)

Iterative Query

- Anfrage von `host.uni-ulm.de` für Hostname „`www.google.com`“
- jeder Server liefert Verweis auf nächsten Server, solange bis autoritativer Nameserver erreicht ist



DNS: Caching, Aktualisieren von Records

Zwischenspeicherung früherer Anfragen (Caching)

- Nameserver speichern Informationen aus früheren Anfragen zwischen
 - direkte Beantwortung von bekannten Anfragen ohne vollständige Query
- vor allem zur Entlastung von Root- und TLD-Nameservern
- Antworten auf Basis von Informationen im Cache möglicherweise veraltet
 - *Expiration*: nach einiger Zeit (Time-To-Live; TTL) Löschung von Einträgen im Cache
 - *TTL*: Kompromiss zwischen Effizienz und Aktualität
- Strategie im Alltag: rechtzeitig vor wichtigen Änderungen TTL-Werte reduzieren
 - z.B. vor Umzug eines Host(namen) auf andere IP

DNS Records

Resource Records (RRs)

- DNS als verteilte Datenbank für RRs

Type=A

- *Name*: Hostname
- *Value*: IP-Adresse

Type=NS

- *Name*: Domain (z.B. uni-ulm.de)
- *Value*: Hostname des autoritativen Nameservers
- mehrere Rekordeinträge für gleichen Namen möglich

RR Format: (name, value, type, ttl)

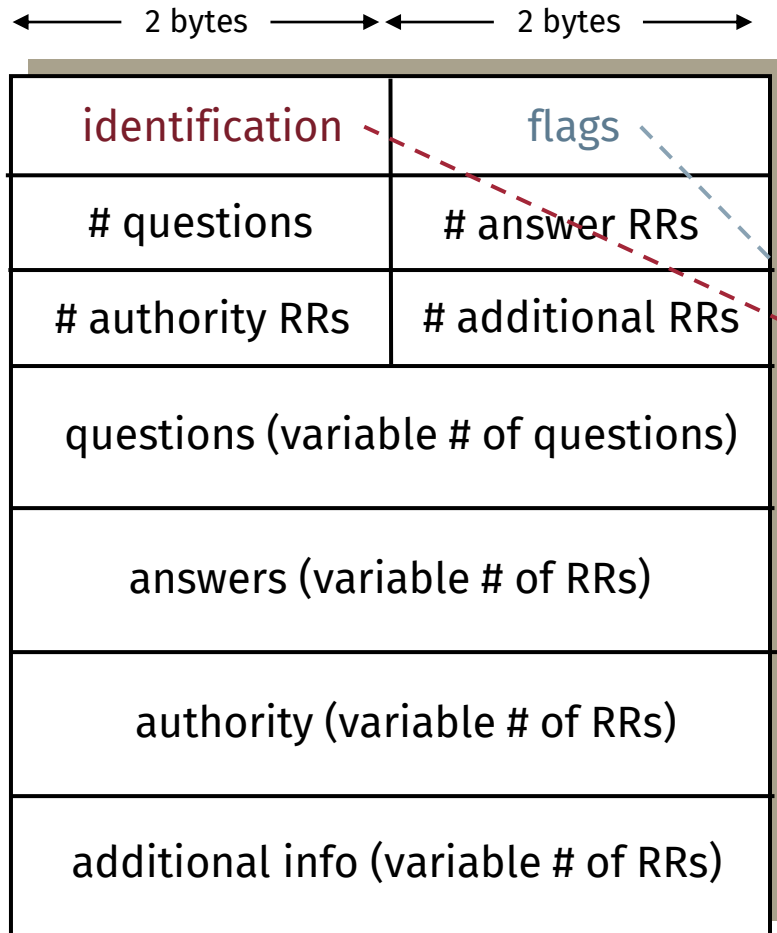
Type=CNAME

- *Name*: Alias
- *Value*: „canonical“ Name (echter Name)
- z.B. `www-vs.informatik.uni-ulm.de`
→ `voyager.informatik.uni-ulm.de`

Type=MX

- *Name*: Host- oder Domainname
- *Value*: Name des zuständigen Mailservers
- z.B. `uni-ulm.de` → `smtp.uni-ulm.de`

DNS Protokoll: Nachrichten (1)



DNS-Queries und DNS-Replies

- UDP oder TCP
- binäres Protokoll
- gleiches Nachrichtenformat (Query und Reply)

Message Header

- **Identification**: 16 Bit # für die Zuordnung von Query und Reply
- **Flags**
 - Query oder Reply
 - Recursion desired
 - Recursion available
 - Reply is authoritative

DNS Protokoll: Nachrichten (2)

← 2 bytes → ← 2 bytes →

identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

DNS-Queries und DNS-Replies

- UDP oder TCP
- binäres Protokoll
- gleiches Nachrichtenformat (Query und Reply)

Weitere Nachrichtenfelder

- Name & Typ für Queries
- volle RRs als Antwort auf Queries
- RRs von autoritativen NS
- weitere „hilfreiche“ RRs

Angriffe auf DNS

DDoS Angriffe

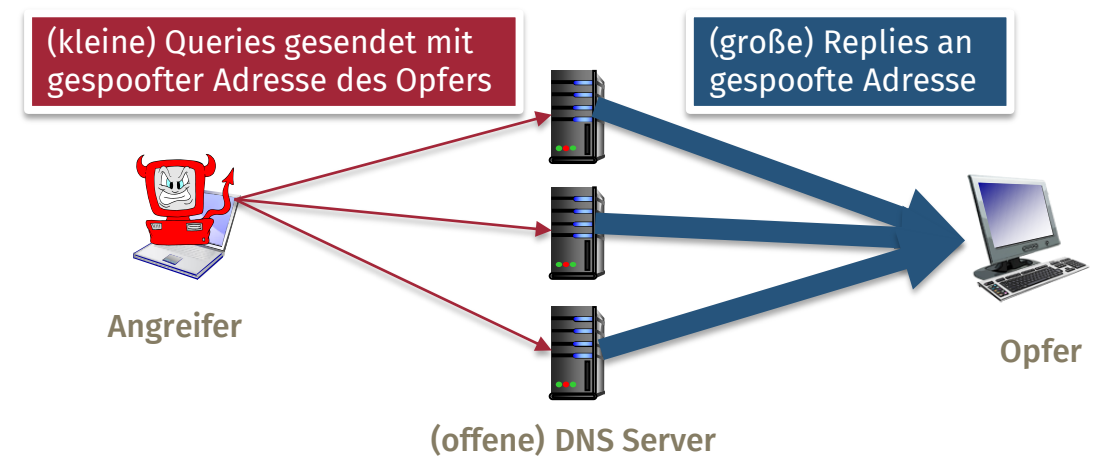
- Überlastung von Root oder TLD-Server
- heute kaum noch erfolgreich
 - Traffic Filter
 - Root/TLD-Nameserver vielfach gecached

DNS Redirection Angriffe

- *Man-in-the-Middle*
 - Angreifer fängt Datenverkehr ab und gibt sich als DNS-Server aus
- *DNS Cache Poisoning*
 - Caches in Middleboxes werden mit gefälschten Informationen „vergiftet“

DNS Amplification Angriffe

- DNS-Server werden für DDoS missbraucht
- Angreifer sendet (verhältnismäßig kleine) Queries mit gefälschten Absenderadressen (Angriffsziel) an DNS-Server
- DNS-Server schicken (deutlich größere) Antworten an Angriffsziel

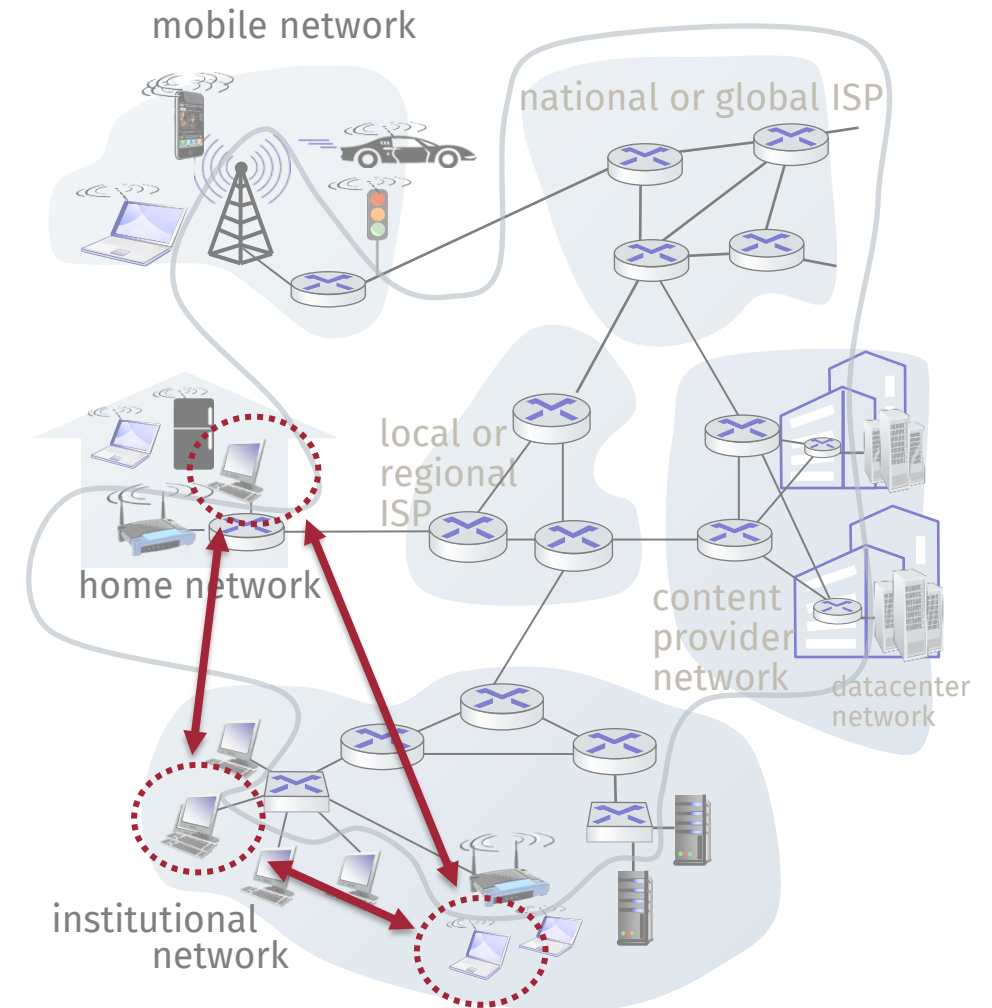


B.5 P2P

P2P-Architekturen

P2P-Architektur

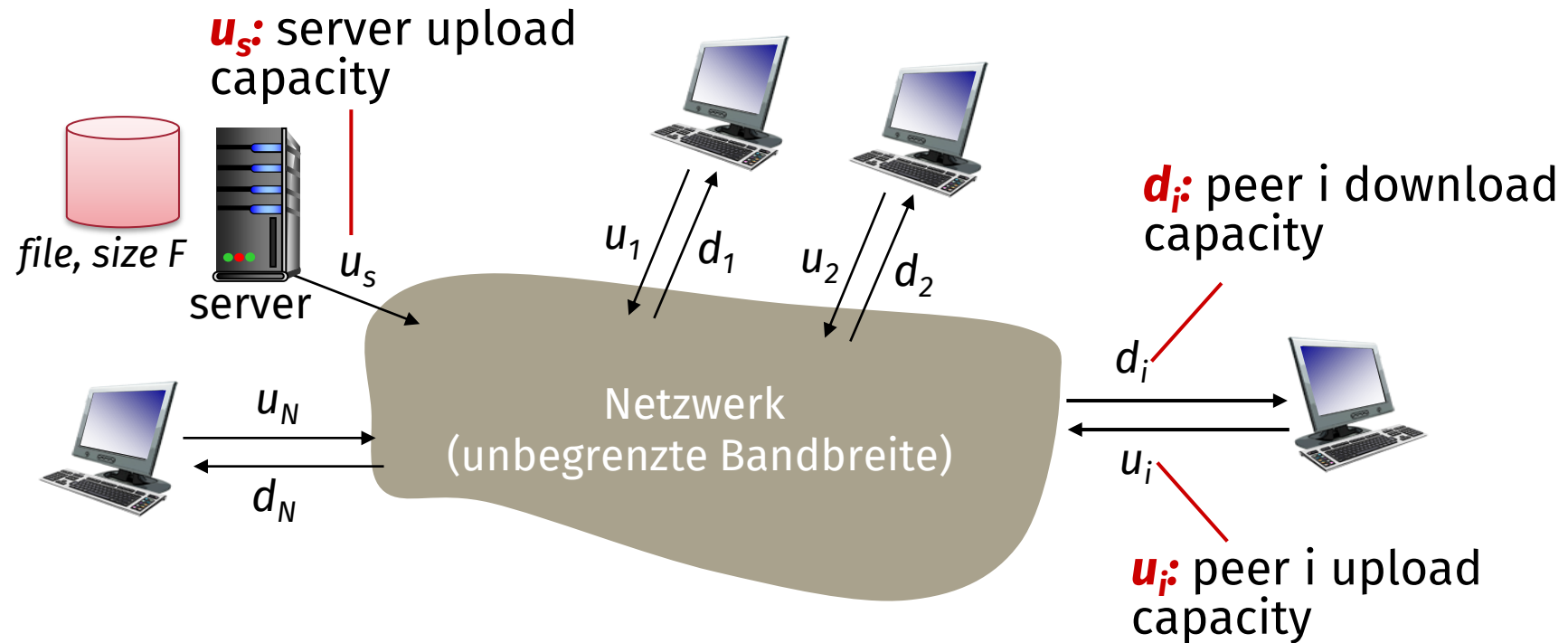
- kein ständig verfügbarer Server
- direkte Kommunikation beliebiger Hosts (Peers)
- Peers nur sporadisch online
- Beispiele für P2P-Anwendungen
 - File Sharing (z.B. BitTorrent)
 - Streaming (z.B. P2PTV)
 - VoIP (z.B. älteres Skype-Protokoll)



File Sharing: Client-Server vs. P2P

Frage: Wie lange dauert die Übertragung eines Files (Größe F) von einem Fileserver an N Peers?

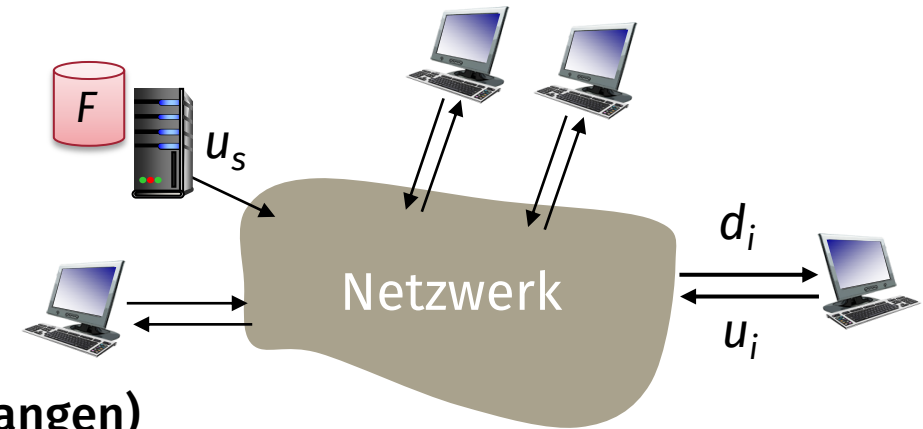
- Upload/Download der Clients ist limitierende Ressource



Verteilungszeit: Client-Server

Server-Übertragung

- sequentielles Senden (upload) von N File-Kopien
- Zeit für eine Kopie: F/u_s
- Zeit für N Kopien: NF/u_s



Client Übertragung (jeder Client muss eine Kopie empfangen)

- d_{\min} = minimale Client Download Rate
- minimale Client Download Zeit: F/d_{\min}

Zeit zur Übertragung
von F an N Clients
(Client-Server)

$$D_{c-s} \geq \max\{NF/u_s, F/d_{\min}\}$$

wächst linear mit N

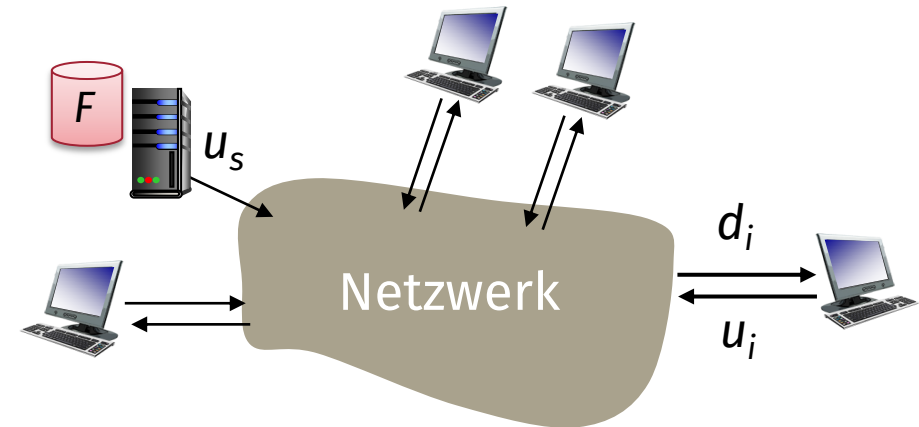
Verteilungszeit: Peer-to-Peer

Server-Übertragung (mindestens eine Kopie)

- Zeit für eine Kopie: F/u_s

Client

- jeder Client muss eine Kopie übertragen
- minimale Download-Zeit eines Clients: F/d_{\min}
- alle Clients zusammen müssen NF Bit übertragen



Maximale Upload-Rate (limitiert maximale Download-Rate): $u_s + \sum u_i$

wächst linear mit N ...

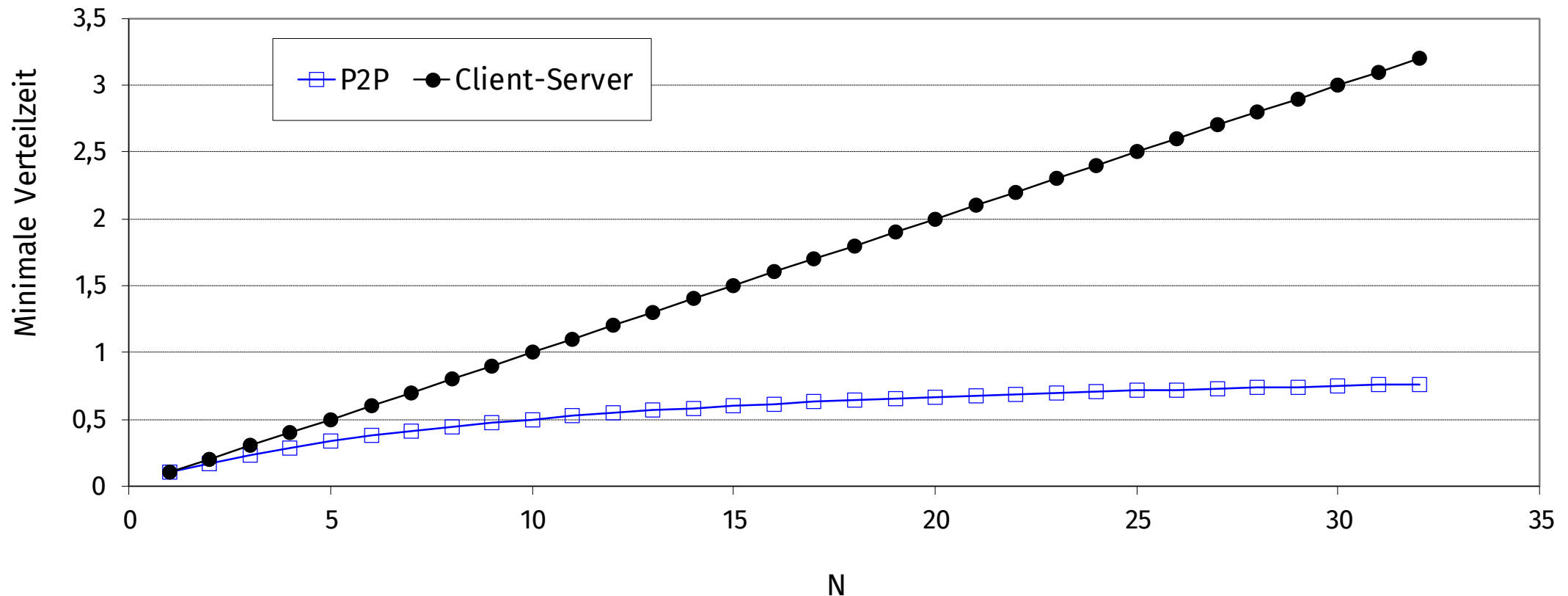
Zeit zur Übertragung
von F an N Clients
(Peer-2-Peer)

$$D_{P2P} \geq \max\{F/u_s, F/d_{\min}, NF/(u_s + \sum u_i)\}$$

... dieser Term auch, da jeder Peer mehr Upload Kapazität mitbringt

Verteilungszeit: Client-Server vs. Peer-to-Peer

Beispiel: Client Upload Rate = u , $F/u = 1$ hour, $u_s = 10u$, $d_{min} \geq u_s$



Literatur

B. Anwendungsschicht

■ B.1 Grundlagen der Netzerkennungen

→ Kurose & Ross (6. Aufl.): Kapitel 2.1

■ B.2 HTTP und das Web

→ Kurose & Ross (6. Aufl.): Kapitel 2.2

■ B.3 E-Mail

→ Kurose & Ross (6. Aufl.): Kapitel 2.4

■ B.4 DNS

→ Kurose & Ross (6. Aufl.): Kapitel 2.5

■ B.5 P2P

→ Kurose & Ross (6. Aufl.): Kapitel 2.6