Netzwerktechnologien
3 VO

Dr. Ivan Gojmerac
ivan.gojmerac@univie.ac.at

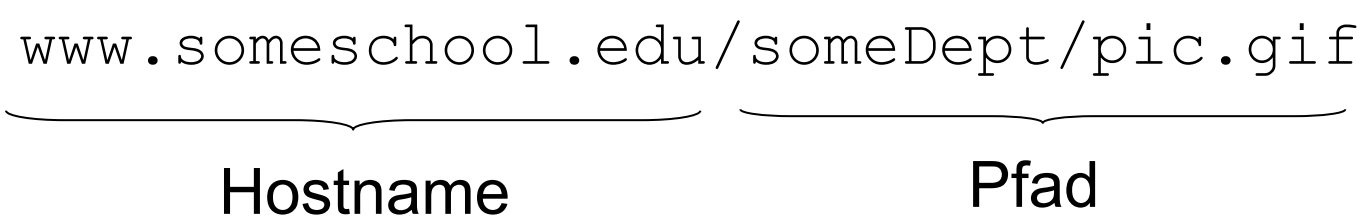
3. Vorlesungseinheit, 20. März 2013

Bachelorstudium Medieninformatik
SS 2013

2.2 Das Web und HTTP

2.2 Das Web und HTTP

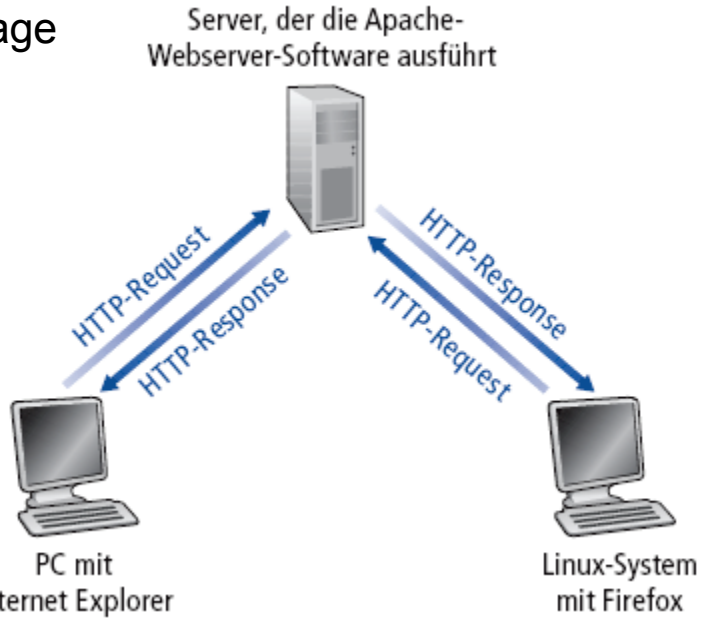
- Eine Webseite besteht aus Objekten
  - Objekte können sein: HTML-Dateien, JPEG-Bilder, Java-Applets, Audiodateien, usw.
- Eine Webseite hat eine Basis-HTML-Datei, die mehrere referenzierte Objekte beinhalten kann
- Jedes Objekt kann durch eine URL (Uniform Resource Locator) adressiert werden
  - Beispiel für eine URL:



2.2.1 Überblick über HTTP

HTTP (= HyperText Transfer Protocol)

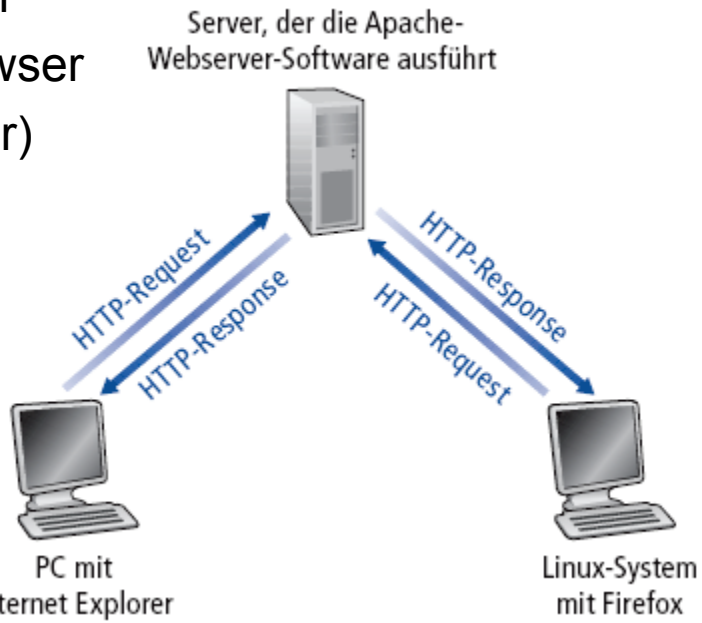
- **Das** Anwendungsprotokoll des Web
- Client/Server-Modell
  - Client: Browser, der Objekte anfragt, erhält und anzeigt
  - Server: Webserver verschickt Objekte auf Anfrage
- Definiert in
  - HTTP 1.0: RFC 1945
  - HTTP 1.1: RFC 2068



2.2.1 Überblick über HTTP

HTTP verwendet TCP:

1. Client baut mit der Socket-API eine TCP-Verbindung zum Server auf
2. Server wartet auf Port 80
3. Server nimmt die TCP-Verbindung des Clients an
4. HTTP-Nachrichten (Protokollnachrichten der Anwendungsschicht) werden zwischen Browser (HTTP-Client) und Webserver (HTTP-Server) ausgetauscht
5. Die TCP-Verbindung wird geschlossen



→ HTTP ist “zustandslos”

– Server merkt sich keine Informationen über frühere Anfragen von Clients

2.2.2 Nichtpersistente und persistente Verbindungen

Nichtpersistentes HTTP

- Maximal ein Objekt wird über eine TCP-Verbindung übertragen

→ HTTP/1.0 verwendet nichtpersistentes HTTP

Persistentes HTTP

- Mehrere Objekte können über eine TCP-Verbindung übertragen werden

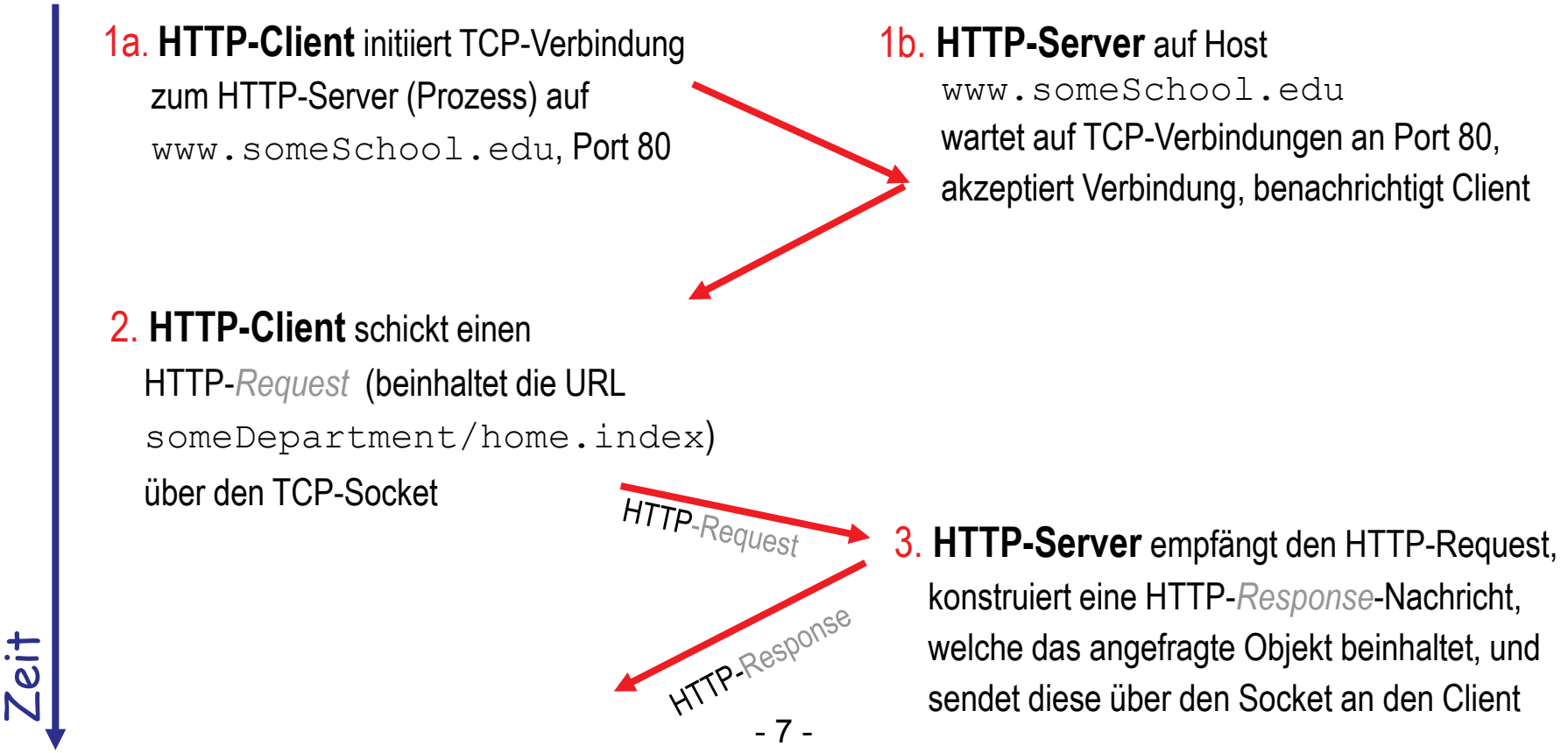
→ HTTP/1.1 verwendet standardmäßig persistentes HTTP

2.2.2 Bsp: Nichtpersistentes HTTP

Es soll folgende URL geladen werden:

`www.someSchool.edu/someDepartment/home.index`

(Die URL beinhaltet Text und Referenzen auf 10 JPEG-Bilder)



2.2.2 Nichtpersistentes HTTP

4. **HTTP-Server** fordert TCP auf die TCP-Verbindung zu schließen *(aber TCP beendet die Verbindung erst, wenn es sicher weiß, daß der Client die Response-Nachricht erhalten hat.)*

5. **HTTP-Client** empfängt die *Response*-Nachricht, schließt die TCP-Verbindung und stellt fest, dass zehn JPEG-Objekte referenziert werden.

6. Schritte 1 bis 5 werden für jedes der zehn JPEG-Objekte wiederholt, dann kann die Seite vollständig angezeigt werden

2.2.2 Nichtpersistentes HTTP

Verzögerung

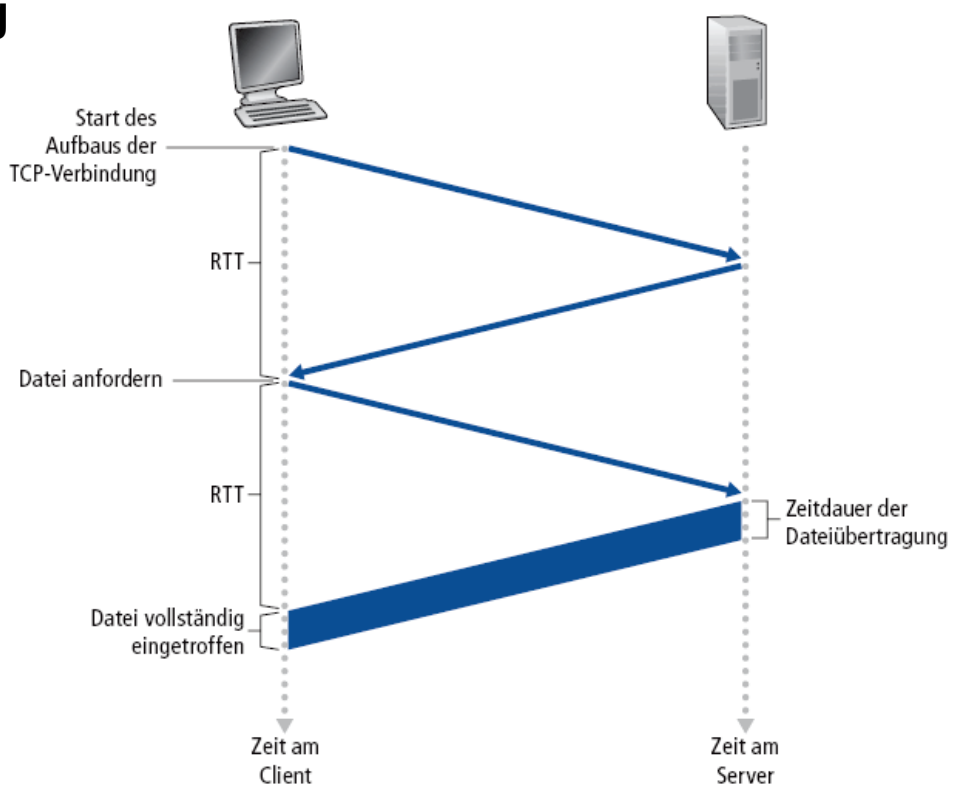
1 RTT für den TCP-Verbindungsaufbau

+1 RTT für den HTTP-Request, bis das erste Byte der HTTP-Response beim Client ist

+ Zeit für das Übertragen der Daten auf der Leitung

= 2 RTT + Übertragungsverzögerung

→ Definition **RTT** (Round Trip Time): Zeitspanne die benötigt wird um ein Paket vom Client zum Server und zurück zu schicken.



2.2.2 Vorteile von persistentem HTTP

Probleme mit nichtpersistentem HTTP:

- 2 RTTs pro Objekt
- Aufwand im Betriebssystem für jede TCP-Verbindung
- Browser öffnen oft mehrere parallele TCP-Verbindungen, um die referenzierten Objekte zu laden

Persistentes HTTP

- Server lässt die Verbindung nach dem Senden der Antwort offen
- Nachfolgende HTTP-Nachrichten können über dieselbe Verbindung übertragen werden

2.2.2 Persistentes HTTP

Persistent ohne Pipelining:

- Client schickt neuen Request erst, nachdem die Antwort auf den vorangegangenen Request empfangen wurde
- 1 RTT für jedes referenzierte Objekt (ca. ½ Dauer von nichtpersistentem HTTP)

Persistent mit Pipelining:

- Standard in HTTP/1.1
- Client schickt Requests, sobald er die Referenz zu einem Objekt findet
- Idealerweise wird nur wenig mehr als 1 RTT für das Laden **aller** referenzierten Objekte benötigt

2.2.3 HTTP-Nachrichtenformat

Es gibt zwei Arten von HTTP-Nachrichten: *Request* und *Response*

HTTP-Request-Nachricht

- In ASCII (vom Menschen leicht zu lesen)

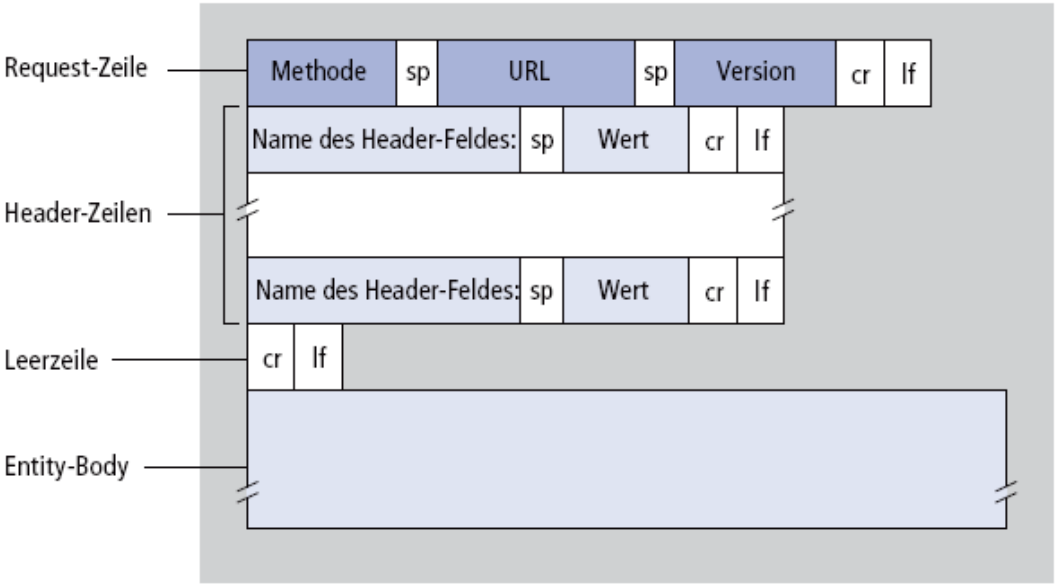
Request-Zeile aus Methodenfeld (z.B. GET, POST, HEAD commands), URL- und HTTP-Versionsfeld

Header-Zeilen

```
GET /somedir/page.html HTTP/1.1\r\n
Host: www.someschool.edu\r\n
User-agent: Mozilla/4.0\r\n
Connection: close\r\n
Accept-language:fr\r\n
\r\n
```

Zusätzlicher Wagenrücklauf + Zeilenvorschub (Leerzeile) zeigt das Ende des Headers an

2.2.3 Request-Nachricht Format

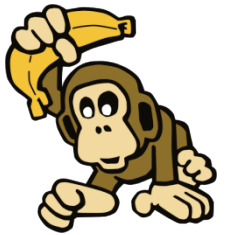


Der Entity-Body wird bei der GET-Methode nicht verwendet (voriges Beispiel), aber bei der POST-Methode um Daten zu versenden.

→ Bsp. Wenn ein Benutzer Suchbegriffe an eine Suchmaschine sendet.

Manche HTML-Formulare verwenden allerdings auch die GET-Methode um eingegebene Daten zu übermitteln indem sie diese in die angeforderte URL schreiben:

→ `www.somesite.com/animalsearch?monkey&banana`



2.2.3 Verfügbare Anweisungen bei HTTP Versionen

HTTP/1.0

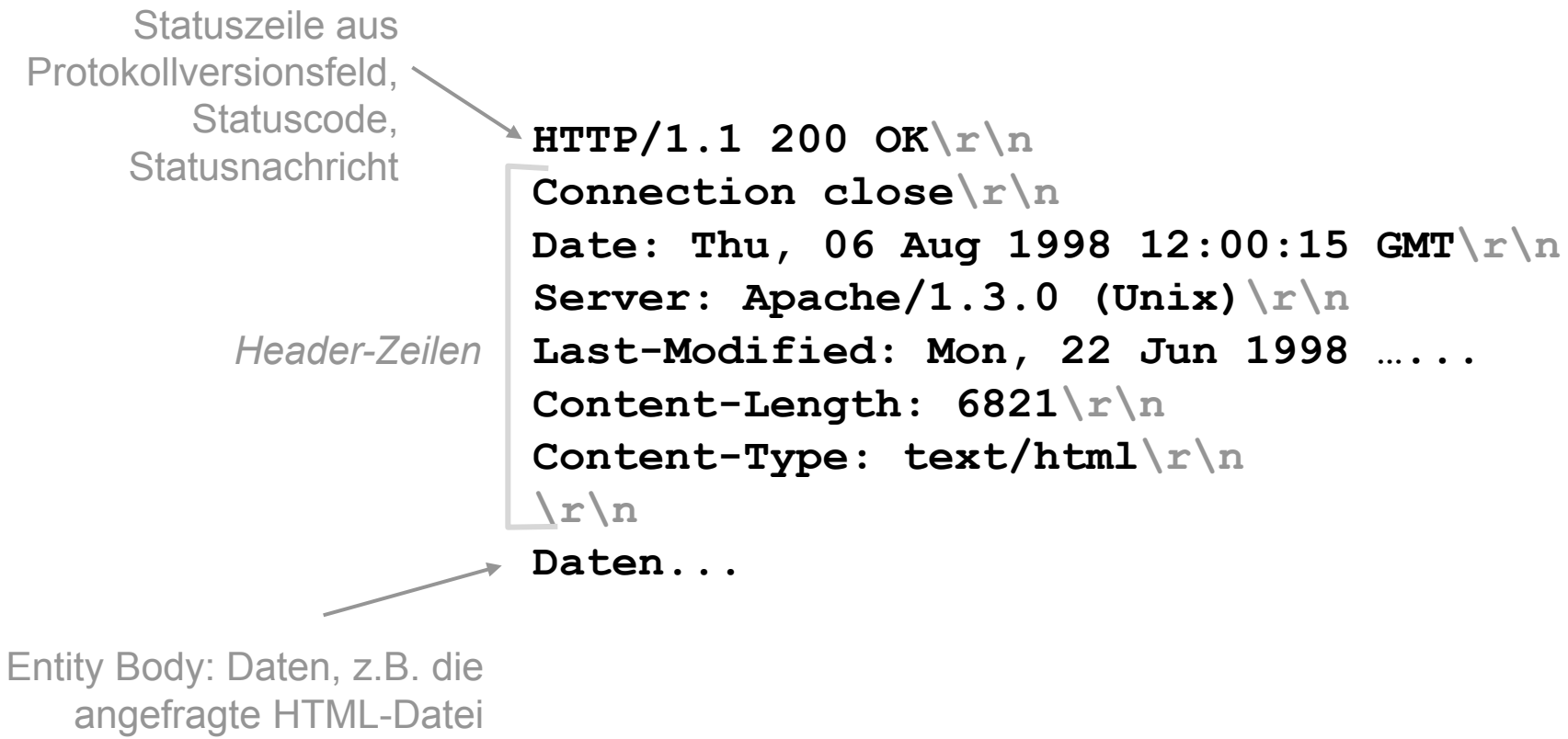
- GET
- POST
- HEAD
  Gibt dem Server die Anweisung nur die Kopfzeilen der Antwort (und nicht das Objekt) zu übertragen

HTTP/1.1

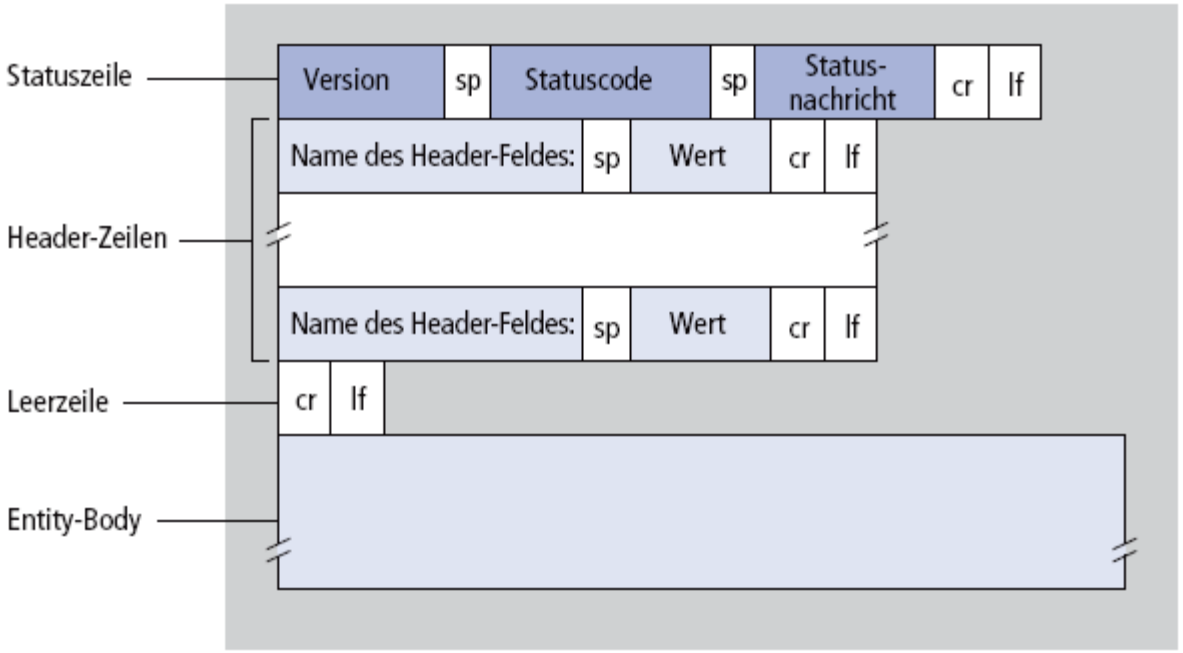
- GET
- POST
- HEAD
- PUT
  Lädt die im Datenteil enthaltene Datei an die durch eine URL bezeichnete Position hoch
- DELETE
  Löscht die durch eine URL angegebene Datei auf dem Server

2.2.3 HTTP-Nachrichtenformat

HTTP-Response-Nachricht



2.2.3 Response-Nachricht Format



Der Entity-Body ist das wichtigste Element der Nachricht - er enthält das (auf der vorherigen Folie als „Daten...“ symbolisierte) angeforderte Objekt.

2.2.3 Response-Nachricht Statuscodes

200 OK
- Request war erfolgreich, gewünschtes Objekt ist in der Antwort enthalten

301 Moved Permanently
- Gewünschtes Objekt wurde verschoben, neue URL ist in der Antwort enthalten

400 Bad Request
- Request-Nachricht wurde vom Server nicht verstanden

404 Not Found
- Gewünschtes Objekt wurde nicht gefunden

505 HTTP Version Not Supported

2.2.4 Benutzer-Server-Interaktion: Cookies

HTTP ist “zustandslos”

- Server merkt sich Informationen über frühere Anfragen von Clients nicht

Zum *Merken* benötigt man Cookies:

- Definiert in RFC 2965
- Werden clientseitig gespeichert
- Ermöglichen es Websites Benutzer wiederzuerkennen

Einsatz von Cookies z.B. für:

- Autorisierung
- Einkaufswagen
- Empfehlungen
- Sitzungszustand (z.B. bei Web-E-Mail)

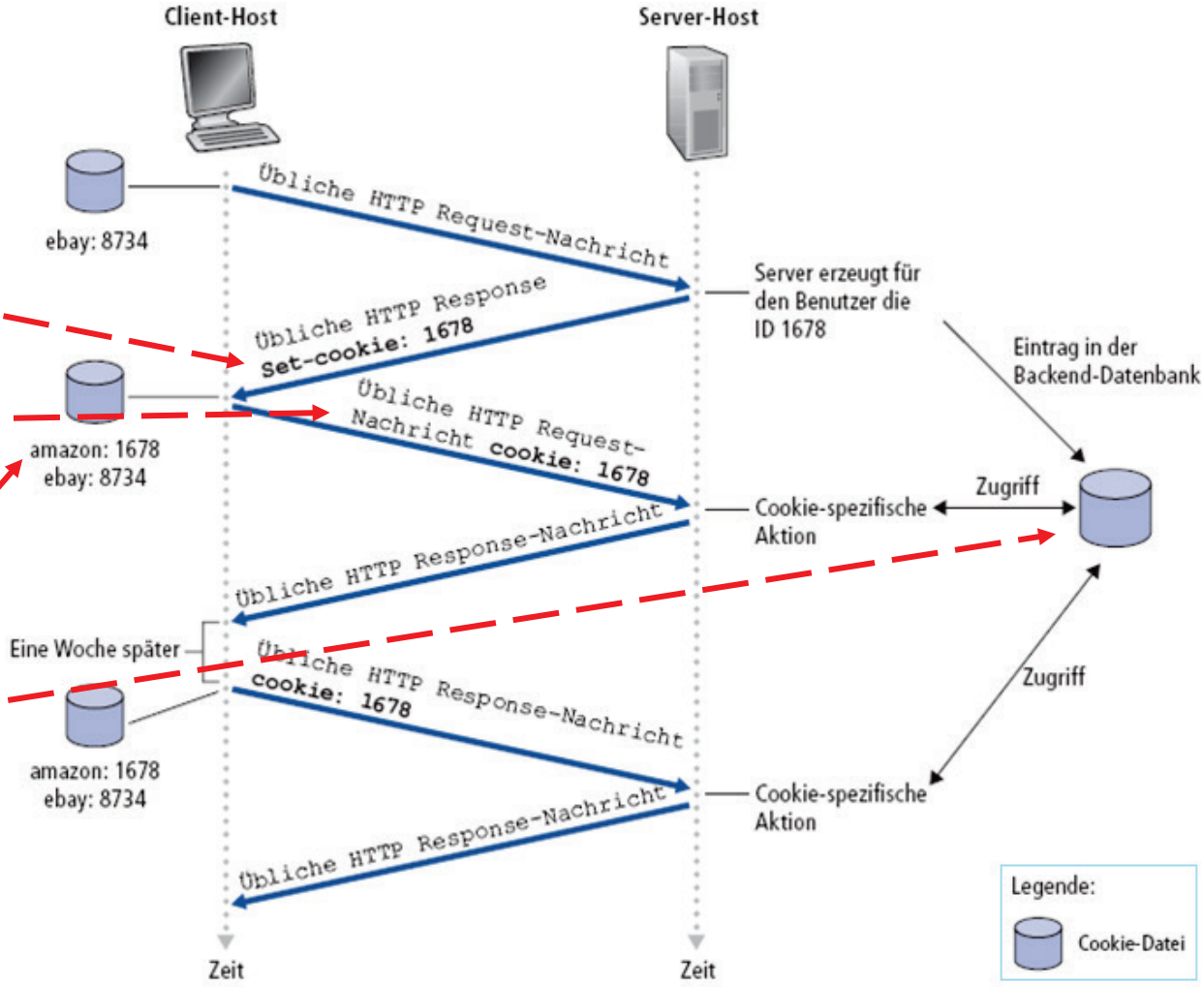
2.2.4 Benutzer-Server-Interaktion: Cookies

1) Cookie-Kopfzeile in der HTTP-Response-Nachricht

2) Cookie-Kopfzeile in der HTTP-Request-Nachricht

3) Cookie-Datei, die auf dem Rechner des Anwenders angelegt und vom Browser verwaltet wird

4) Backend-Datenbank auf dem Webserver



2.2.4 Benutzer-Server-Interaktion: Cookies

Cookies und Privatsphäre

Cookies ermöglichen es Websites, viel über den Anwender zu lernen:

- Formulareingaben (Name, E-Mail-Adresse)
- Besuchte Seiten

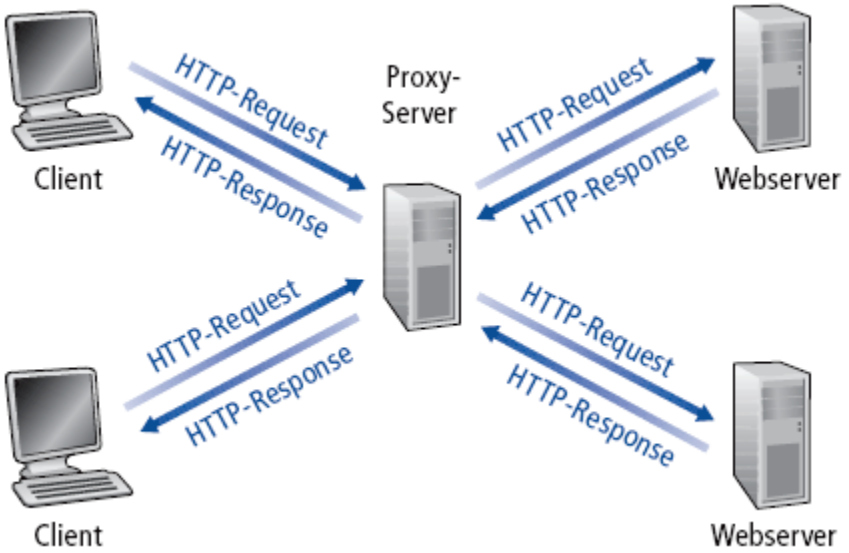
Alternativen um Zustand zu *merken*

- In den Endsystemen: Zustand wird im Protokoll auf dem Client oder Server gespeichert und für mehrere Transaktionen verwendet
- Cookies: HTTP-Nachrichten beinhalten den Zustand

2.2.5 Webcaching

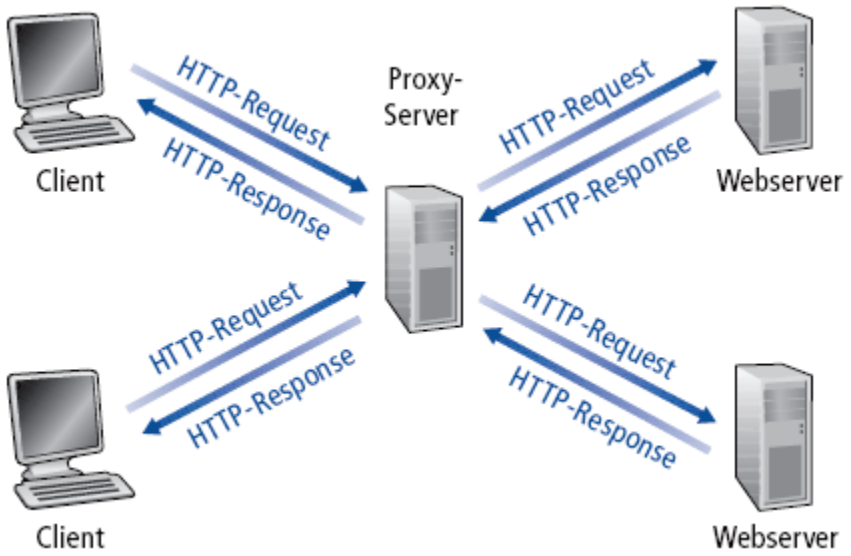
Webcache (auch Proxyserver genannt)

- Netzwerkentität, die im Namen des eigentlichen Webservers HTTP-Requests beantwortet.
- Hat seinen eigenen Plattenspeicher in dem er Kopien der vor kurzem angeforderten Objekte aufbewahrt.



2.2.5 Funktionsweise von Webcaching

1. Benutzer konfiguriert Browser so, daß HTTP-Requests zuerst an den Webcache gerichtet werden.
2. Browser stellt eine TCP-Verbindung zum Webcache her und sendet einen HTTP-Request für das gewünschte Objekt
3. Webcache überprüft ob Objekt im Cache:
   - *Falls vorhanden*:
     Cache gibt Objekt in einer HTTP-Response-Nachricht an Client-Browser zurück.
   - *Sonst*:
     Webcache öffnet TCP-Verbindung zum eigentlichen Server, fragt das Objekt an und leitet es dann zum Client weiter.



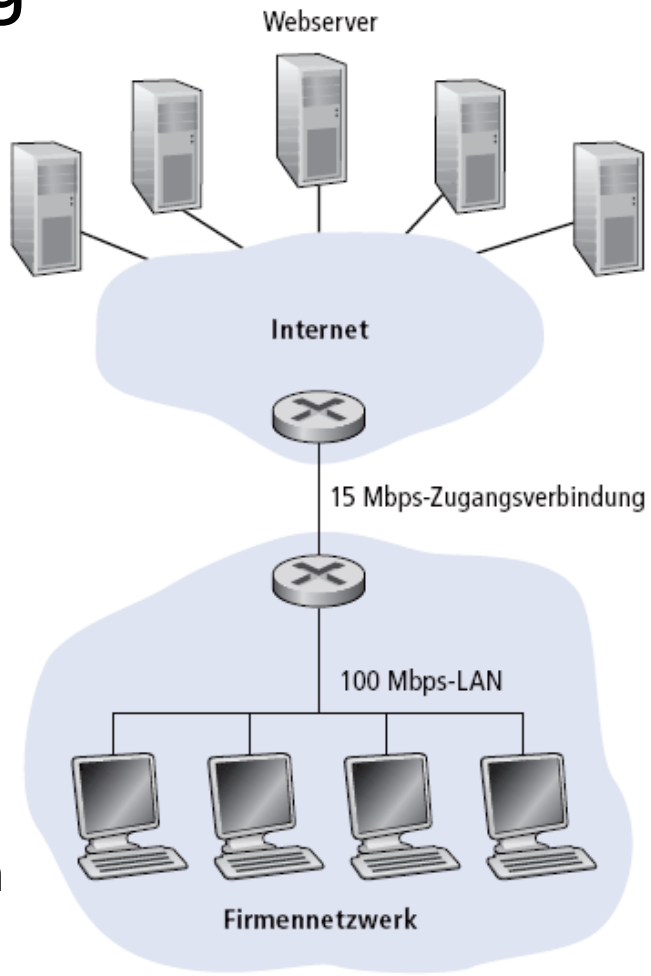
2.2.5 Vorteile von Webcaching

- Cache arbeitet als Client und als Server
- Explizit oder transparent
- Üblicherweise ist der Cache beim ISP installiert
  - z.B. Universität, Firma oder ISP für Privathaushalte

Vorteile:

- Verringert die Antwortzeit
  Oft ist eine höhere Bandbreite zwischen Client und Cache (der bei ISP läuft) verfügbar als zwischen Client und Webserver
- Verringert den Datenverkehr auf der Zugangsleitung eines Firmennetzwerkes
- Kostengünstig

2.2.5 Beispiel für Webcaching



Annahmen

Bandbreite der Zugangsleitung = 15 Mbps

Bandbreite des LAN = 100 Mbps

Ø Größe eines Objektes = 100.000 Bit

Ø Rate von Anfragen aller Webbrowser der Firma = 150/s

Verzögerung vom Router zum Server und zurück = 2 sec

Resultat

Auslastung des LAN = 15%

Auslastung der Zugangsleitung = **100%**

Verzögerung von Internet + Zugangsleitung + LAN =

2s + mehrere Sekunden, Minuten (!)+ Millisekunden

→ Wartezeit untragbar!

Offensichtlicher Lösungsansatz:
Bandbreite der Zugangsleitung erhöhen

2.2.5 Beispiel für Webcaching

Annahmen

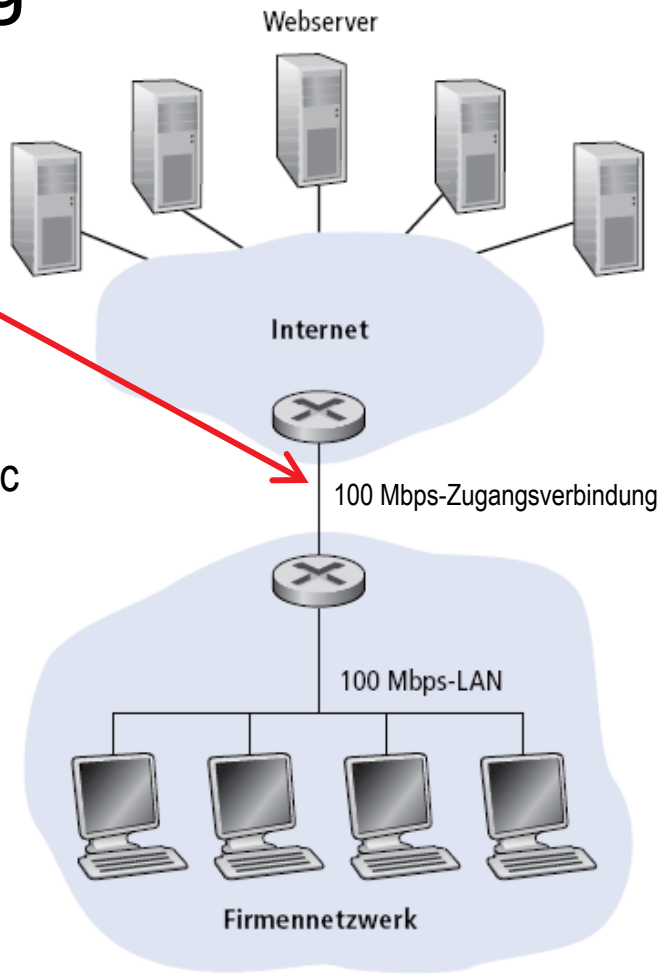
Bandbreite der Zugangsleitung jetzt = **100 Mbps**

Bandbreite des LAN = 100 Mbps

Ø Größe eines Objektes = 100.000 Bit

Ø Rate von Anfragen aller Webbrowser der Firma = 150/s

Verzögerung v. Router d. Firma zum Server und zurück = 2 sec



Resultat

Auslastung des LAN = 15%

Auslastung der Zugangsleitung = **15%**

Verzögerung von Internet + Zugangsleitung + LAN =

2s + **Millisekunden** + Millisekunden

ABER: Bandbreite der Zugangsleitung erhöhen ist oft sehr teuer!

→ Anderer Lösungsansatz: Webcaching

2.2.5 Beispiel für Webcaching

Annahmen

Bisherige Annahmen bleiben gleich, aber Bandbreite der Zugangsleitung wieder nur = 15 Mbps

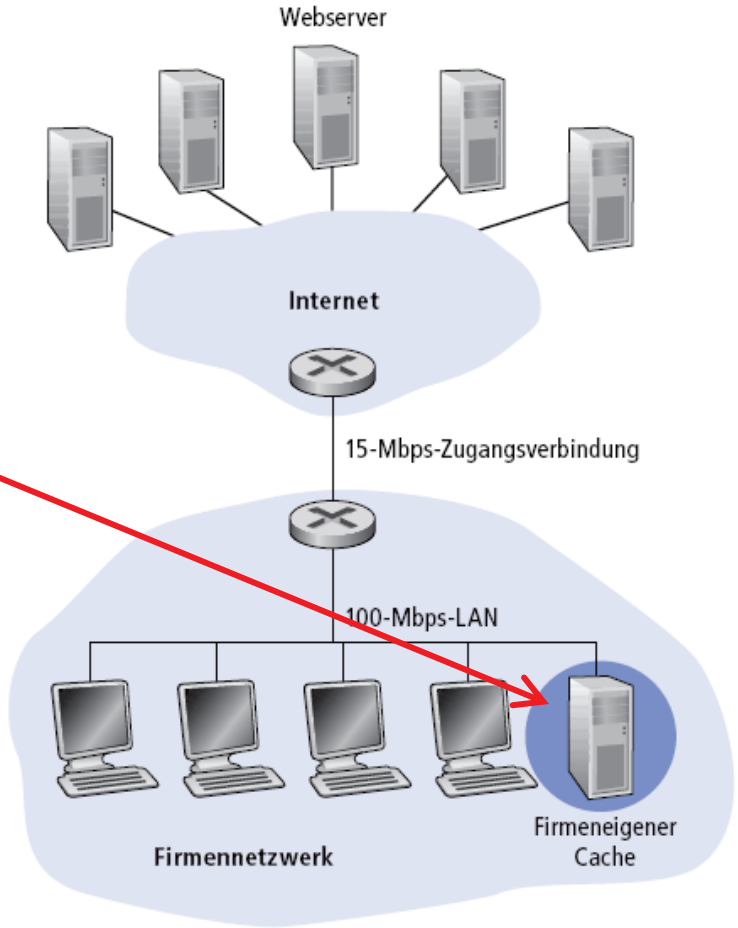
Diesmal Lösungsansatz **Web-Cache!**

Angenommene Cache-Trefferrate = 0,4

Resultat

Anfragen die **nahezu sofort** aus dem **Cache** beantwortet werden = **40%**

Anfragen die weiterhin von Webservern beantwortet werden = 60%



2.2.5 Beispiel für Webcaching

Resultat

Anfragen die **nahezu sofort** aus dem **Cache** beantwortet werden = **40%**

Anfragen die weiterhin von Webservern beantwortet werden = 60%

Auslastung der Zugangsleitung nur noch = **60%**

Verzögerungen auf der Zugangsleitung **verringert** (~ bei 10 msec)

Verzögerung von Internet + Zugangsleitung + LAN =

0,6 * 2s + 0,4 * **Millisekunden** + Millisekunden **< 1,4s**

0,6 da **60%** von Webservern beantwortet

0,4 da **40%** Im Cache gefunden

→ Erhebliche Verbesserung

→ Potentiell kostengünstiger als die Erhöhung der Bandbreite!

2.2.6 Bedingtes GET

Conditional GET (Bedingtes GET)

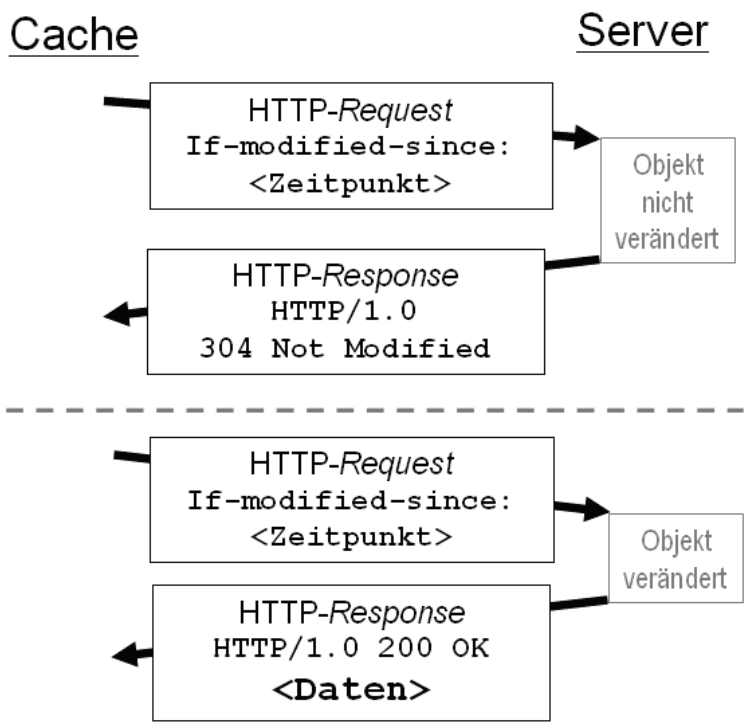
Ein Mechanismus von HTTP mit dem ein Cache beim Server sicherstellen kann, daß die Kopien in seinem Speicher nicht veraltet sind.

Eine HTTP-Request-Nachricht ist eine Conditional-GET-Nachricht, wenn sie enthält:

- Die GET-Methode und
- Die Header-Zeile `If-Modified-Since:`

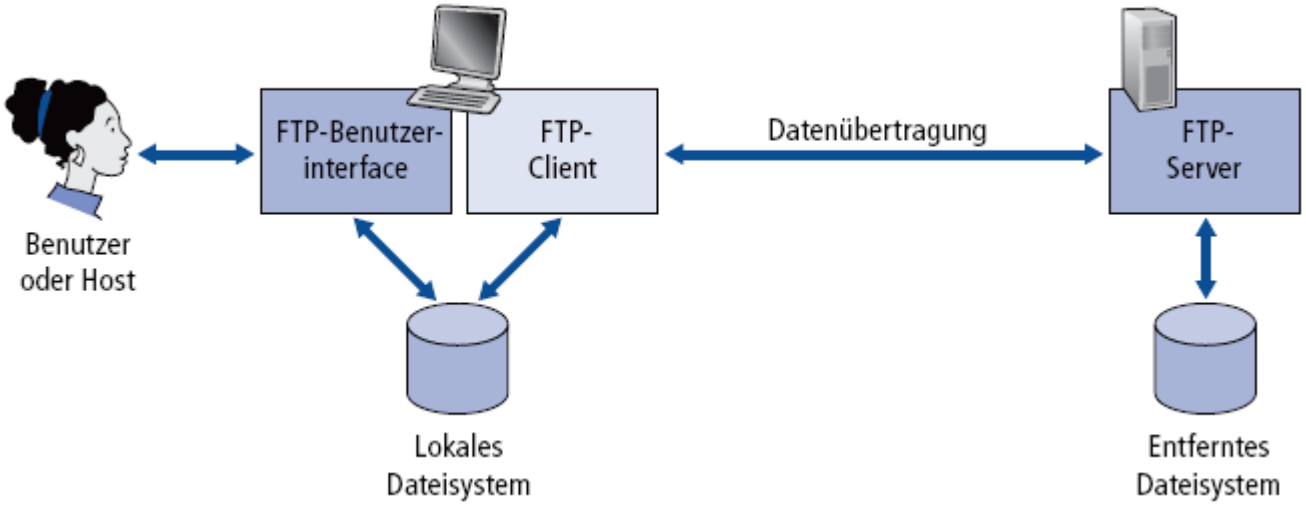
- **Cache:** gibt Änderungsdatum der gespeicherten Version im HTTP-Request an
- **Server:** HTTP-Response enthält kein Objekt, wenn die Version im Cache aktuell ist:

  Code: **`HTTP/1.0 304 Not Modified`**



2.3 Dateitransfer: FTP

2.3 Dateitransfer: FTP



- Protokoll zum Übertragen einer Datei von/zu einem entfernten Rechner.
- Client/Server-Modell
  - *Client:* Seite, die den Transfer initiiert (vom oder zum entfernten Rechner)
  - *Server:* entfernter Rechner
- Definiert in RFC 959
- FTP-Server verwenden TCP Port 21

2.3 Dateitransfer: FTP

1. FTP-Client kontaktiert FTP-Server auf Port 21, verwendet TCP als Transportprotokoll
2. Client autorisiert sich über die Kontrollverbindung
3. Client betrachtet das entfernte Verzeichnis indem er Kommandos über die Kontrollverbindung schickt
4. Jedes Mal wenn der Server ein Kommando für eine Dateiübertragung empfängt öffnet er eine neue TCP-Datenverbindung zum Client
5. Nach der Übertragung einer Datei schließt der Server die Verbindung

2.3 Dateitransfer: FTP

Out-of-Band Übermittlung der Steuerinformationen

FTP verwendet eine Kontrollverbindung seperat zum Datenkanal

FTP-Server speichern Informationen zu einem jeden Benutzer (Gegensatz zu HTTP):

- Zugehörige Kontrollverbindungen
- Aktuelles Verzeichnis auf dem entfernten Host in dem der Benutzer navigiert

→ Die Gesamtanzahl von Sitzungen, die gleichzeitig verwaltet werden können, wird dadurch eingeschränkt!

2.3.1 FTP-Befehle und -Antworten

Kommandos:

Werden als ASCII-Text über die Kontrollverbindung übermittelt

- `USER` *`username`*
- `PASS` *`password`*
- `LIST` - Gibt eine Liste der Dateien im aktuellen Verzeichnis zurück
- `RETR` *`filename`* - Lädt eine entfernte Datei auf den lokalen Rechner
- `STOR` *`filename`* - Überträgt eine lokale Datei auf den entfernten Rechner

Antworten:

Statuscodes und Erläuterungen (wie bei HTTP)

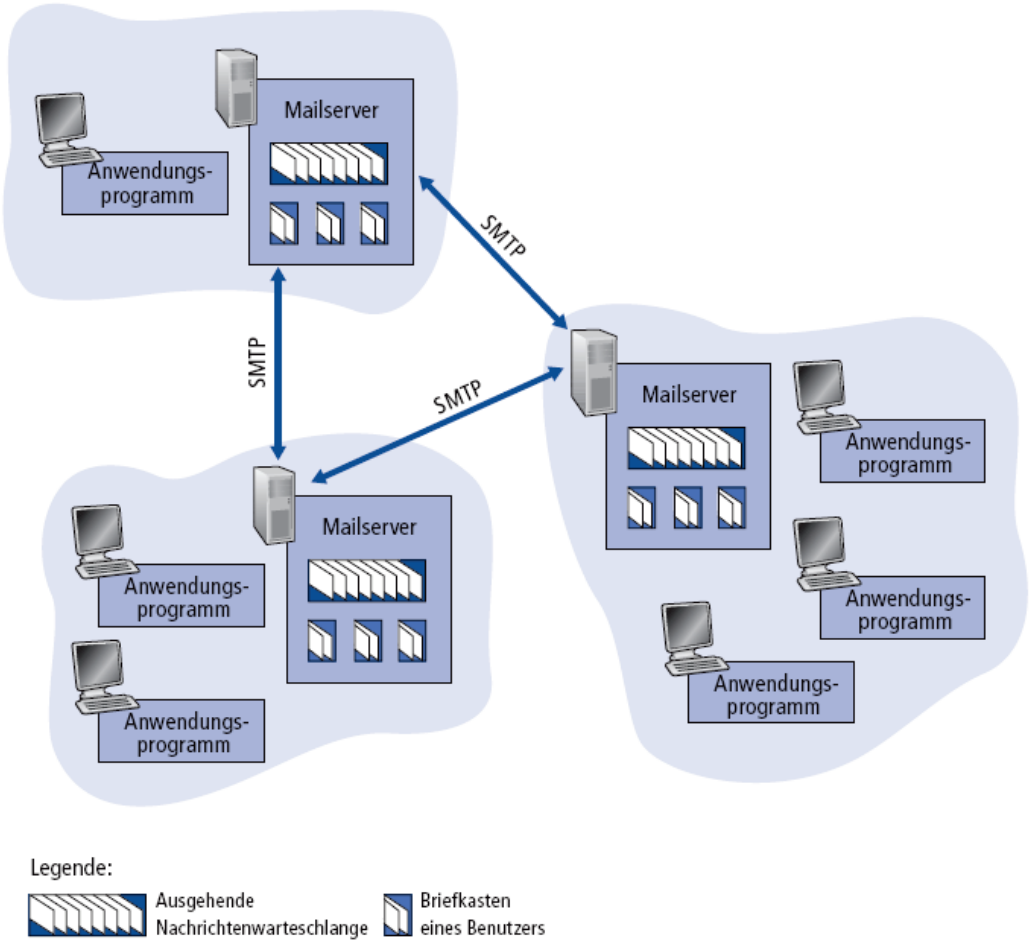
- `331 Username OK, password required`
- `125 data connection already open; transfer starting`
- `425 Can't open data connection`
- `452 Error writing file`

2.4 E-Mail im Internet

2.4 E-Mail im Internet

E-Mail besteht aus drei Hauptbestandteilen:

1. Anwendungsprogramm
2. Mailserver
3. Übertragungprotokoll: SMTP



2.4 E-Mail im Internet

1. Anwendungsprogramm ("Mail Reader"):

- Funktion: Erstellen, Editieren, Lesen von E-Mail-Nachrichten
  - z.B. Microsoft Outlook, Mozilla Thunderbird

2. Mailserver:

- Die Mailbox enthält die eingehenden Nachrichten eines Benutzers
- Die Warteschlange für ausgehende Nachrichten enthält die noch zu sendenden E-Mail-Nachrichten

3. Übertragungsprotokoll SMTP (Simple Mail Transfer Protocol):

- Wird verwendet, um Nachrichten zwischen Mailservern auszutauschen
  - Client: Sendender Mailserver (Wichtig: Anwendungsprogramme sind zugleich auch SMTP-Clients, es sei denn, dass proprietäre Protokolle wie Microsoft Exchange benutzt werden!)
  - Server: Empfangender Mailserver

2.4.1 SMTP

- Definiert in RFC 2821
- **Zuverlässiger Transport:**
  E-Mail-Nachrichten werden vom Client zum Server mit TCP (Port 25) übermittelt
- **Direkter Transport der Nachrichten:**
  Von den Mailservern der Absender zu den Mailservern der Empfänger ohne Zwischenlagerung
- **Verwendet persistente Verbindungen:**
  Bei mehreren Nachrichten mit gleichem Sender und Empfänger können alle über dieselbe TCP-Verbindung übertragen werden
- Nachrichten (sowohl Header als auch Daten) müssen in 7-Bit-ASCII kodiert sein
  → Veraltete Beschränkung (früher wegen knapper Übertragungskapazität)

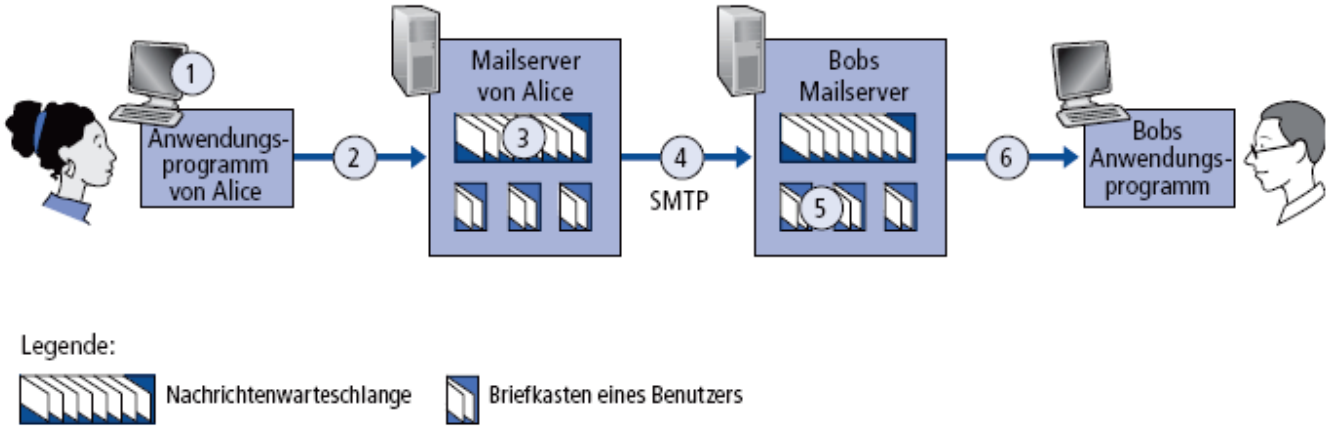
2.4.1 SMTP

- **Drei Phasen des Mail-Versands:** Analog zu einer Unterhaltung
  - Handshaking (Begrüßung)
  - Transfer of Messages (Austausch von Informationen)
  - Closure (Verabschiedung)

- Interaktion basiert auf dem Austausch von Befehlen (*Commands*) und Antworten (*Responses*)
  - *Command*: ASCII-Text
  - *Response*: Statuscode und Bezeichnung

- Ein SMTP-Server verwendet `CRLF.CRLF` (`CR` für Wagenrücklauf / carriage return, `LF` für Zeilenvorschub / line feed), entsprechend einer Zeile, die nur einen Punkt enthält, um das Ende einer Nachricht zu signalisieren

2.4.1 SMTP Beispiel



1. Alice verwendet ihr Anwendungsprogramm, um eine Nachricht an `bob@someschool.edu` zu erstellen
2. Alices Anwendungsprogramm versendet die Nachricht an ihren Mail-Server; Nachricht wird in der Warteschlange gespeichert
3. Alices Mailserver öffnet als Client eine TCP-Verbindung zu Bobs Mailserver
4. SMTP-Client versendet die Nachricht von Alice über die TCP-Verbindung
5. Bobs Mailserver empfängt die Nachricht von Alices Mailserver und speichert diese in Bobs Mailbox
6. Bob verwendet (irgendwann) sein Anwendungsprogramm und liest die Nachricht

2.4.1 Beispiel für eine SMTP Sitzung

```
S: 220 hamburger.edu
C: HELO crepes.fr
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: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

← Eigentlich `CRLF.CRLF` um Nachrichtenende zu signalisieren

2.4.2 Vergleich SMTP und HTTP

- Protokolltyp:
  - HTTP: Pull-Protokoll (Protokoll zum Herunterladen)
    → TCP-Verbindung wird von dem Host aufgebaut, der die Datei erhalten will
  - SMTP: Push-Protokoll (Protokoll zum Senden von Daten)
    → TCP-Verbindung wird von dem Mailserver aufgebaut, der die Datei senden will
- Beide interagieren mittels ASCII-Befehl/Antwort-Paaren sowie Statuscodes
- Kodierung:
  - HTTP: Keine besonderen Einschränkungen
  - SMTP: Überträgt Header und Daten in 7Bit-ASCII-Format. Sonderzeichen und Binärdaten (z.B. eine Bilddatei) müssen extra in 7Bit-ASCII codiert werden
- Umgang mit Dokumenten mit Medienobjekten:
  - HTTP: Jedes Objekt ist in einer eigenen Antwortnachricht gekapselt
  - SMTP: Mehrere Objekte können in einer Mail-Nachricht (multipart msg) versendet werden

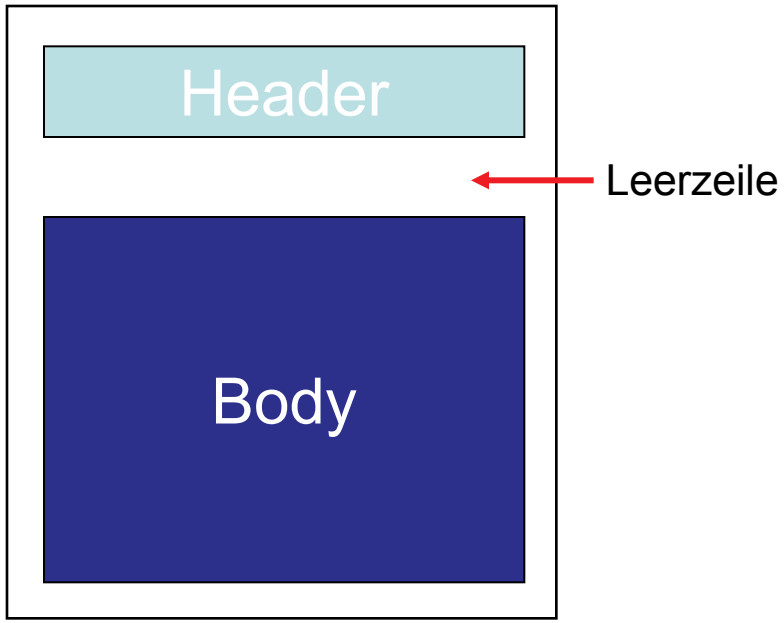
2.4.3 Mail-Nachrichtenformate

RFC 822: Standard für Textnachrichten

- Header-Zeilen, z.B.
  - To:
  - From:
  - Subject:

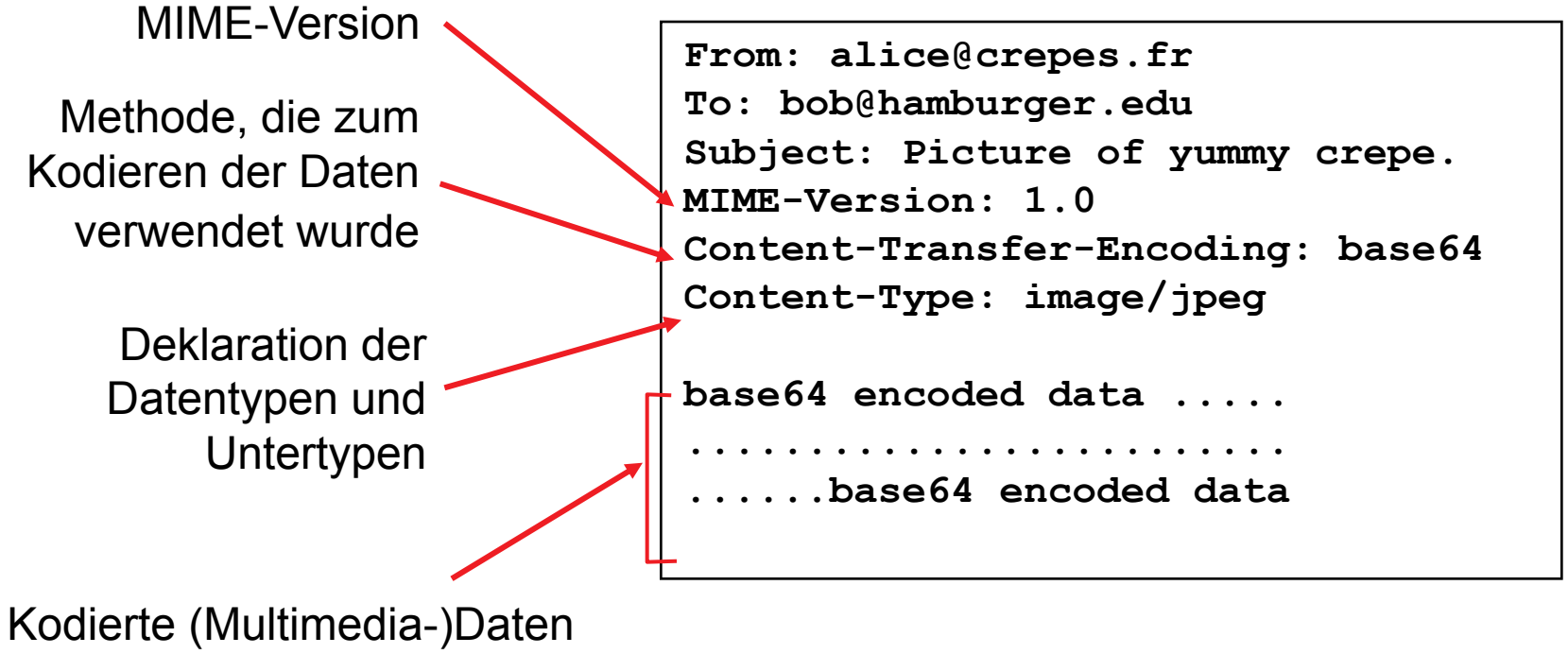
  ***Keine** SMTP-Befehle!*

- Body
  - Die eigentliche Nachricht in ASCII



2.4.3 MIME - Erweiterung des Nachrichtenformates für Nicht-ASCII-Daten

- **MIME** (Multimedia Mail Extension), definiert in RFC 2045 und RFC 2056
- Zusätzliche Zeilen im Header deklarieren den MIME-Typ des Inhaltes



2.4.3 Datentypen in MIME

Text

Beispiele für Subtypen:

- **`plain`**
- **`html`**

Audio

Beispiele für Subtypen:

- **`basic`** (8-bit mu-law encoded),
- **`32kadpcm`** (32 kbps coding)

Bilder

Beispiele für Subtypen:

- **`jpeg`**
- **`gif`**
- **`png`**

Video

Beispiele für Subtypen:

- **`mpeg`**
- **`quicktime`**

Anwendungen

Daten müssen von der Anwendung vor der Wiedergabe interpretiert werden.

Beispiele für Subtypen:

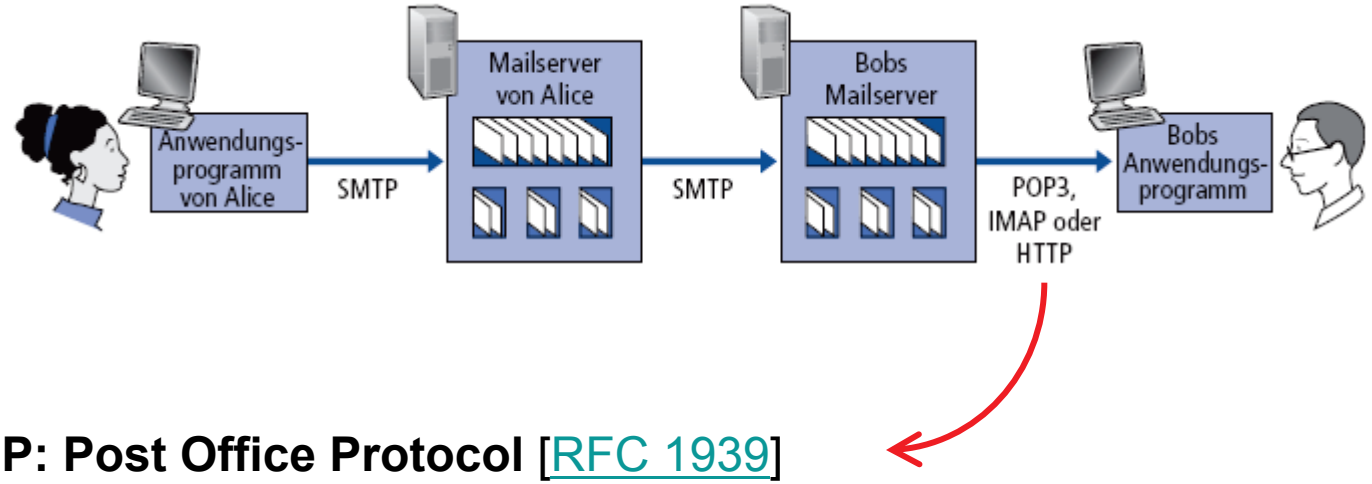
- **`msword`**
- **`octet-stream`**

2.4.4 Mail-Zugriffsprotokolle

- Das Senden einer E-Mail ist ein zweistufiger Prozess:
  - Alices Anwendungsprogramm auf ihrem Computer sendet die Nachricht zuerst per SMTP an Alices Mailserver
  - Alices Mailserver empfängt die Nachricht und versucht in regelmäßigen Abständen die Nachricht per SMTP an Bobs Mailserver zu übermitteln, bis er Erfolg hat. Bobs Mailserver speichert dann die Nachricht in Bobs Postfach

- Will Bob mit seinem Anwendungsprogramm (Client) die Nachricht von Bobs Mailserver (Server) empfangen, kann er dazu kein SMTP verwenden, da das Abrufen der Nachricht eine Pull-Operation ist.
- Beliebte Mail-Zugriffsprotokolle (Pull-Protokolle): **POP3**, **IMAP** und **HTTP**

2.4.4 Mail-Zugriffsprotokolle



- **POP: Post Office Protocol** [RFC 1939]
  Autorisierung (Anwendung ⟷ Server) und Zugriff/Download
- **IMAP: Internet Mail Access Protocol** [RFC 1730]
  Größere Funktionalität (deutlich komplexer)
  Manipulation der auf dem Server gespeicherten Nachrichten
- **HTTP:**
  z.B. bei Hotmail, Yahoo!Mail etc.

2.4.4 POP3

Autorisierungsphase:

- Befehle des Clients:
  - **`user`**: Benutzername
  - **`pass`**: Passwort
- Antworten des Servers:
  - **`+OK`**
  - **`-ERR`**

Transaktionsphase:

- **`list`**: Nachrichten auflisten
- **`retr`**: Nachrichten herunterladen
- **`dele`**: Löschen von Nachrichten
- **`Quit`**: **Ende**

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

```
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
```

S: Server
C: Client

2.4.4 POP3 und IMAP

POP3 Modi:

- "*Download-and-Delete*"-Modus (Beispiel auf voriger Folie)
  Andere E-Mail-Clients haben nach Abruf keine Möglichkeit mehr die Mails zu lesen
- "*Download-and-Keep*"-Modus
  Ermöglicht den reinen Lesezugriff auf Nachrichten; auch andere Clients haben Zugriff
- POP3 ist zustandslos zwischen einzelnen Sitzungen

IMAP:

- Alle Nachrichten bleiben an einem Ort: auf dem Server
- Nachrichten können auf dem Server in Ordnern verwaltet werden
- IMAP bewahrt den Zustand zwischen einzelnen Sitzungen:
  - Namen von Ordnern und Zuordnung von Nachrichtennummer und Ordnername bleiben erhalten

2.5 DNS

2.5 DNS

Bei Menschen gibt es viele verschiedene Identifikationsmechanismen.

- z.B. Name, Ausweisnummer

Internet-Hosts und Router werden auch über bestimmte Mechanismen identifiziert:

- IP-Adresse (32 Bit) – für die Adressierung in Paketen
- "Name", z.B. **`www.univie.ac.at`** – von Menschen verwendet

Frage: *Wie findet die Abbildung zwischen IP-Adressen und Namen statt?*

2.5 DNS

Frage: *Wie findet die Abbildung zwischen IP-Adressen und Namen statt?*

→ Domain Name System (DNS)

- *Verteilte Datenbank*, implementiert eine Hierarchie von *Nameservern*
- *Protokoll der Anwendungsschicht*, wird von Hosts verwendet um Namen *aufzulösen* (Abbildung zwischen Adresse und Name)
  - Zentrale Internetfunktion, implementiert als Protokoll der Anwendungsschicht

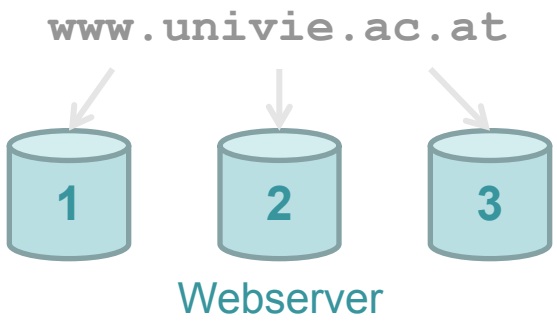
www.univie.ac.at

131.130.70.8

2.5.1 Von DNS erbrachte Dienste

DNS-Dienste:

- Übersetzung von Hostnamen in IP-Adressen
- Aliasnamen für Hosts
  - Kanonische Namen und Aliasnamen
- Aliasnamen für Mailserver
- Lastausgleich (Loadbalancing)
  - Replizierte Webserver: Mehrere IP-Adressen von einem kanonischen Namen



2.5.2 Arbeitsweise von DNS

Warum ist DNS nicht zentralisiert?

- Robustheit gegenüber Fehlern und Angriffen
  - Wenn ein zentraler DNS-Server zusammenbrechen würde, dann auch das ganze Internet!
- Datenverkehrsaufkommen
  - Wenn es nur einen zentralen DNS-Server gäbe müsste dieser *alle* DNS-Anfragen beantworten.
- Große „Distanz“ zur zentralisierten Datenbank
  - Ein einzelner DNS-Server kann nicht in der Nähe aller anfragenden Clients sein. Anfragen von weit entfernten Orten mussten über möglicherweise langsame und überlastete Leitungen übertragen werden, was zu spürbaren Verzögerungen führen könnte.

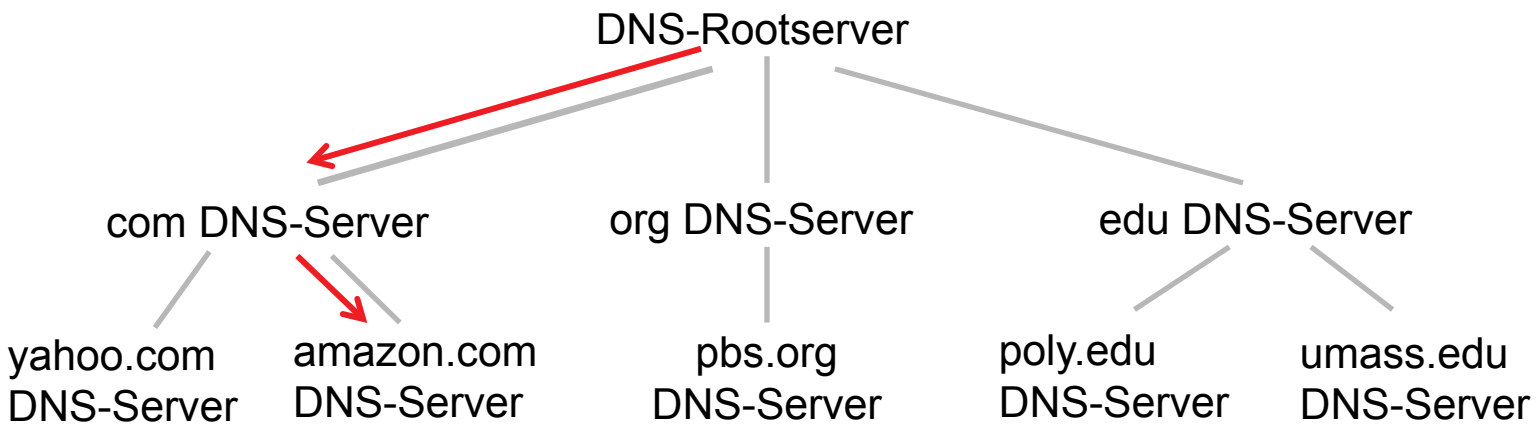
2.5.2 Arbeitsweise von DNS

Warum ist DNS nicht zentralisiert?

- Wartung
  - Ein einziger DNS-Server müsste Datensätze für alle Internethosts beinhalten. Daher wäre diese zentralisierte Datenbank nicht nur riesig, sondern sie müsste auch häufig aktualisiert werden um jeden neuen Host zu enthalten.

→ Fazit: Eine zentralisierte Datenbank in einem einzelnen DNS-Server ist einfach nicht skalierbar. DNS kann als **verteilte, hierarchische Datenbank** im Internet betrachtet werden.

2.5.2 Arbeitsweise von DNS



Client sucht die IP-Adresse von www.amazon.com:

1. Client fragt seinen lokalen DNS-Server
2. Dieser fragt einen DNS-Rootserver, um den DNS-Server für `com` zu finden
3. Danach fragt er den `com`-DNS-Server, um den `amazon.com`-DNS-Server zu finden
4. Dann wird der `amazon.com`-DNS-Server gefragt, um die IP-Adresse zu `www.amazon.com` zu erhalten

2.5 Hierarchie der DNS-Server

DNS-Rootserver

- 13 DNS-Rootserver weltweit im Internet
- Jeder dieser DNS-Rootserver ist eigentlich ein Cluster replizierter Server
- Kennt die Adressen der Top-Level-Domain (com, net, org, de, uk, …) Server
- Gibt diese Informationen (Adressen) bei Anfragen an die lokalen Nameserver weiter

2.5 Hierarchie der DNS-Server

- Top-level domain (TLD) Server
  - Verantwortlich für `com`, `org`, `net`, `edu`, `gov` etc. sowie für alle Länder-Domains, z.B. `at`, `de`, `uk`, `fr`, `ca`, `jp`
  - Network Solutions ist verantwortlich für den `com`-TLD-Server
  - Educause hat die Verantwortung für den `edu`-TLD-Server
- Authoritative DNS-Server
  - DNS-Server einer Organisation, der eine autorisierte Abbildung der Namen dieser Organisation auf IP-Adressen anbietet
  - Verwaltet von der entsprechenden Organisation oder einem Service Provider

2.5.2 Hierarchie der DNS-Server

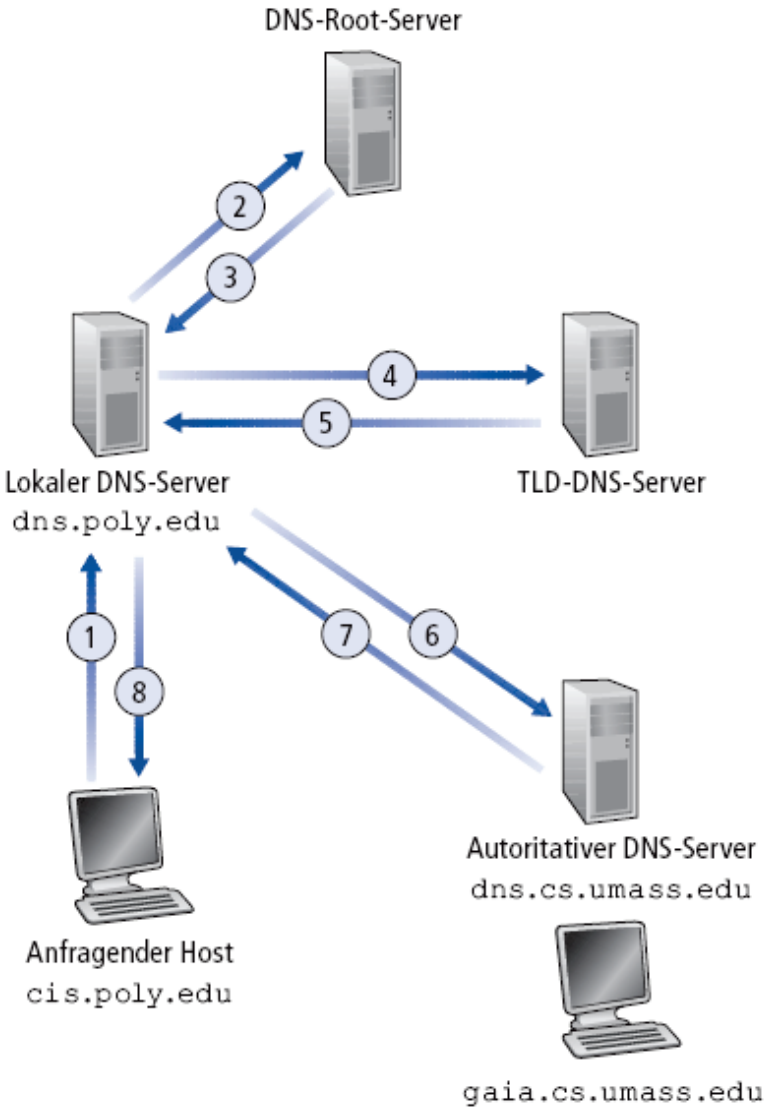
Lokale DNS Server

- Gehören **nicht** strikt zur Hierarchie der DNS-Server
- Jeder ISP (z.B. Firmen, Universität, ISP für Privatkunden) besitzt einen lokalen Nameserver
- Werden auch “Default-Nameserver” genannt
- Agieren als Proxy
- Wenn ein Host eine DNS-Anfrage startet, dann schickt er diese an seinen lokalen Nameserver
  - Dieser kümmert sich um die Anfrage so lange, bis eine endgültige Antwort vorliegt
  - Dazu kontaktiert er bei Bedarf Root-Nameserver, TLD-Nameserver und autoritative Nameserver
  - Dann schickt er die Antwort an den Host zurück

2.5.2 Namensauflösung mit DNS

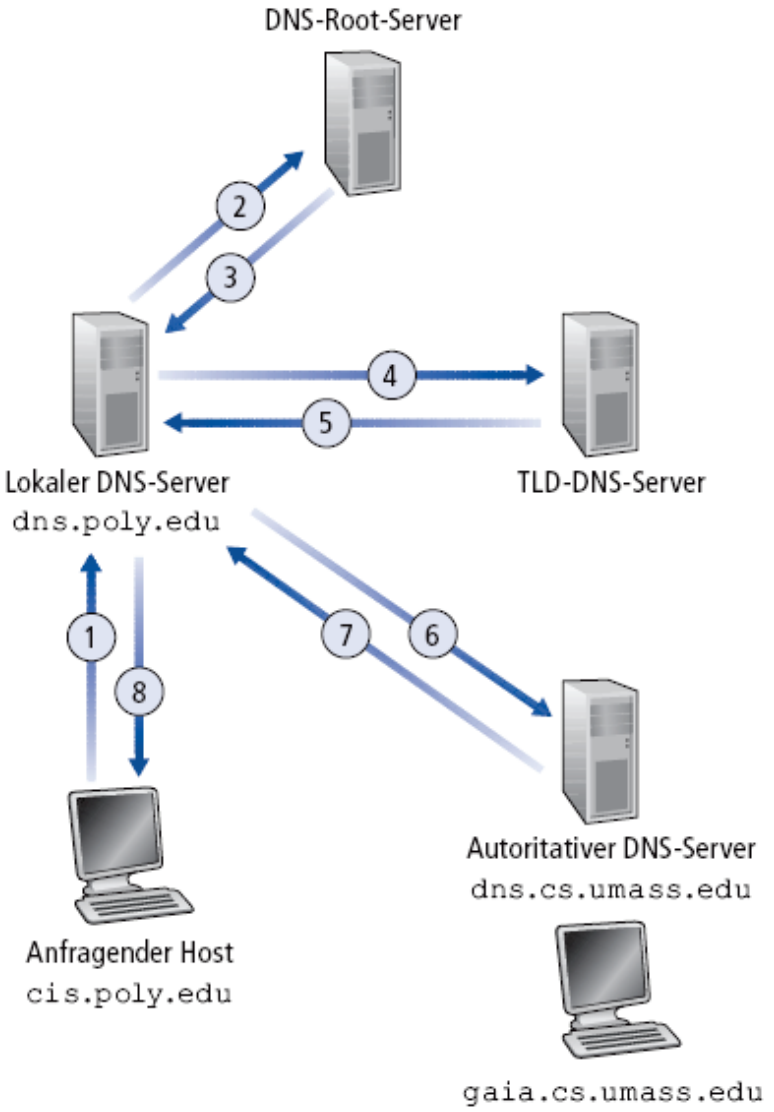
Der Host `cis.poly.edu` fragt nach der IP-Adresse von `gaia.cs.umass.edu`:

1. **Der Host** sendet eine Anfrage mit dem zu übersetzenden Hostnamen `gaia.cs.umass.edu` an seinen lokalen DNS-Server.
2. **Der lokale DNS-Server** leitet die Anfrage an einen DNS-Rootserver weiter.
3. **Der DNS-Rootserver** reagiert auf den Teil `edu` und gibt eine Liste von IP-Adressen von TLD-Servern für `edu` zurück.



2.5.2 Namensauflösung mit DNS

4. **Der lokale DNS-Server** sendet dann die Anfrage erneut an einen der TLD-Server.
5. **Der TLD-Server** reagiert auf den Teil `umass.edu` und gibt die IP-Adresse des autoritativen DNS-Servers zurück.
6. **Der lokale DNS-Server** sendet dann die Anfrage erneut an den autoritativen DNS-Server.
7. **Der autoritative DNS-Server** antwortet mit der IP-Adresse für `gaia.cs.umass.edu`.

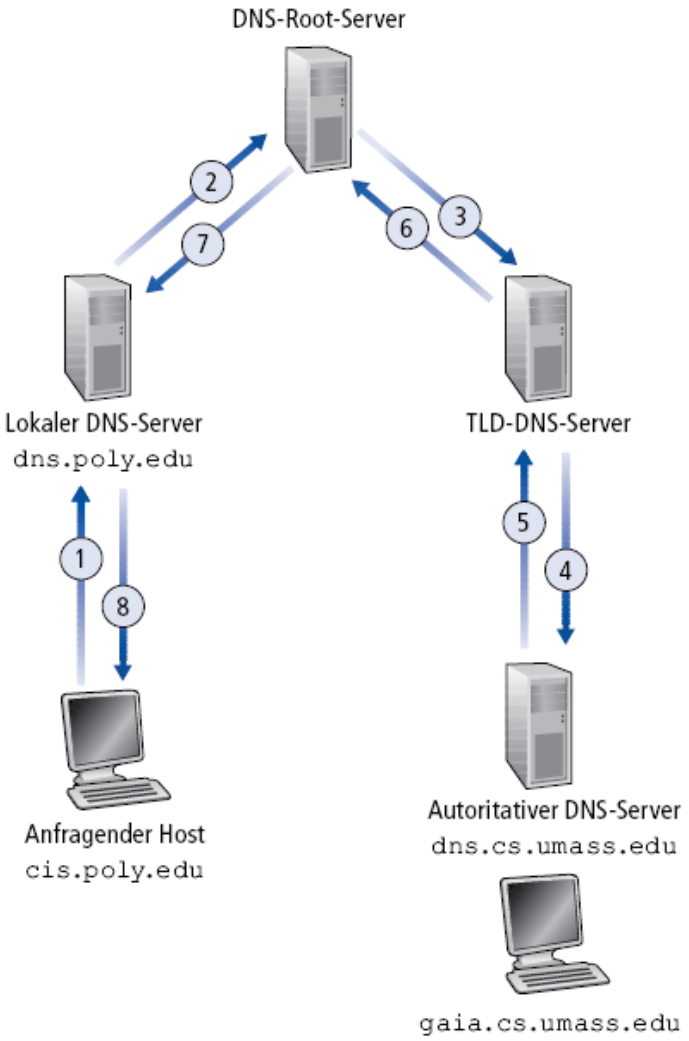


2.5.2 Namensauflösung mit DNS

Zwei Arten der Anfragen:

- **Iterative Anfragen**
  Angesprochene Server in der Hierarchie antworten mit einem Verweis auf andere Server
  "Ich kenne den Namen nicht, frag diesen Server …"

- **Rekursive Anfragen**
  Wenn der gefragte Server nicht direkt eine Antwort an den fragenden Host zurückgibt, sondern die Anfrage weiterleitet handelt es sich um eine rekursive Anfrage
  (→ **siehe Abb. rechts**).
  - Zusätzliche Belastung!
  - Root-Nameserver erlauben dies häufig nicht, andere Nameserver dagegen schon!



2.5.2 DNS Caching

- Sobald ein Nameserver eine Abbildung zur Namensauflösung kennenlernt, merkt er sich diese in einem Cache
  - Die Adressen der TLD-Server werden üblicherweise von den lokalen Nameservern gecacht (Root-Nameserver werden eher selten angesprochen)

  → Verbessert Leistungen hinsichtlich Verzögerungen

  → Reduziert Anzahl der zur Auflösung benötigten DNS-Nachrichten

  - Die Einträge im Cache werden nach einer vorgegebenen Zeit wieder gelöscht
- Mechanismen zur Pflege von Cache-Einträgen und zur Benachrichtigung bei Änderungen werden derzeit von der IETF entwickelt
  - Definiert in RFC 2136

2.5.3 DNS Resource Records

Resource Records (=RR, Ressourcendatensätze)

- Ein RR ist ein Viertupel mit den Feldern (`Name, Wert, Typ, TTL`)
- DNS-Server speichern ihre Informationen in Form von RR
- Jede DNS-Antwortnachricht beinhaltet einen oder mehrere RR
- RR werden definiert in RFC 1034 und RFC 1035

2.5.3 DNS Resource Records

RR-Format: `( Name, Wert, Typ, TTL )`

TTL – Bestimmt, wann eine Ressource aus einem Cache entfernt werden sollte.
Name und **Wert** hängen vom Typ ab:

- Typ=A
  `name` ist der Hostname
  `value` ist die IP-Adresse
- Typ=NS
  `name` ist eine Domain (z.B. `foo.com`)
  `value` ist der Hostname des autoritativen Nameservers für diese Domain
- Typ=MX
  `value` ist der Name des Mailservers für die Domain **`name`**
- Typ=CNAME
  `name` ist ein Alias für einen kanonischen Namen: `www.ibm.com` ist ein Alias für `servereast.backup2.ibm.com`
  `value` ist der kanonische Name

2.5.3 DNS-Nachrichtenformat

DNS-Anfragen (Query) und DNS-Antwortnachrichten (Reply) haben dasselbe Format.

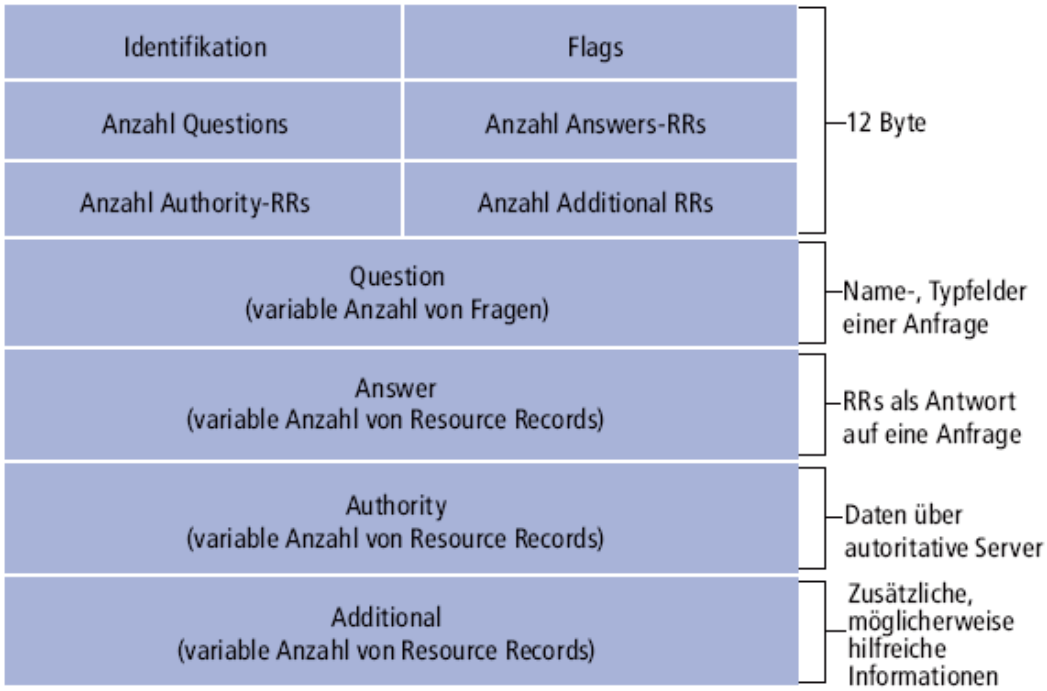
Header-Felder

- Identification: 16-Bit-ID, ermöglicht die Zuordnung einer eingehenden Reply-Nachicht zur vorangegangenen Query-Nachricht (ID ist bei beiden Nachrichten gleich).

- 1Bit-Flags:
  - query/reply
  - recursion desired
  - recursion available
  - reply is authoritative

- Vier Anzahl-Felder:

Diese Felder zeigen, wie häufig die einzelnen Datenabschnitte, die dem Header folgen, auftreten.



2.5.3 DNS-Nachrichtenformat

Body-Felder

- Question-Abschnitt: Enthält Informationen über die gestellte Anfrage.
  - Namensfeld – Enthält den angefragten Namen
  - Typfeld – Spezifiziert die Art der Frage

- Answer-Abschnitt:

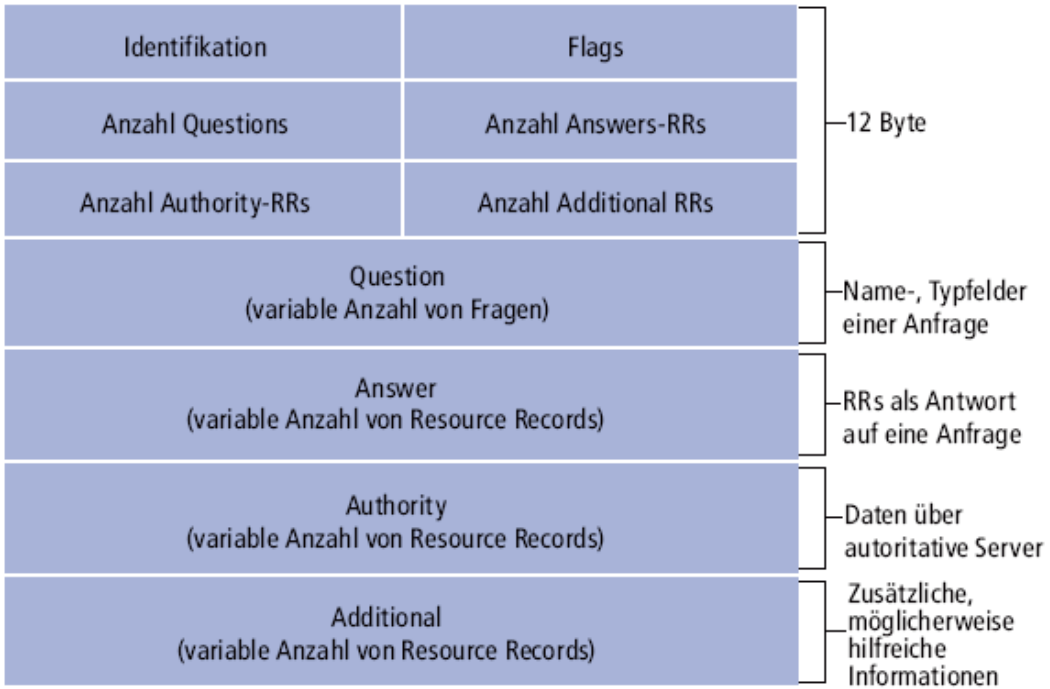
Enthält die RR für den Namen der ursprünglich angefragt wurde.

- Authority-Abschnitt:

Enthält RR zur Identifikation von autoritativen Servern für die Antworten.

- Additional-Abschnitt:

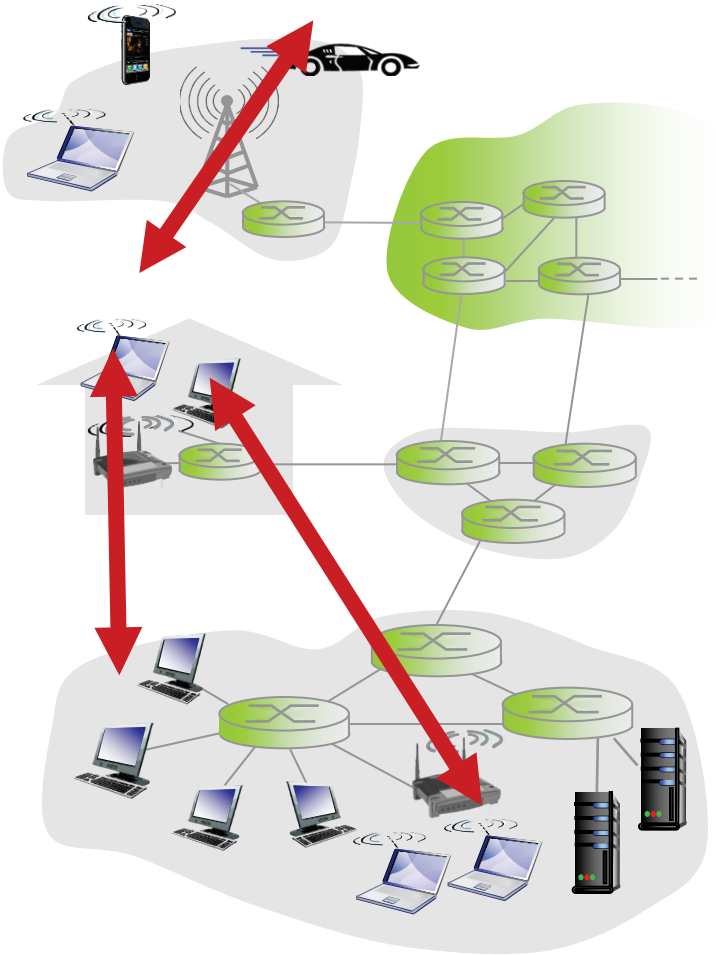
Enthält je nach Typ der Frage sonstige hilfreiche Datensätze zur Auflösung des angefragten Namens.



2.6 Peer-to-Peer-Anwendungen

2.6.1 Peer-to-Peer-Anwendungen

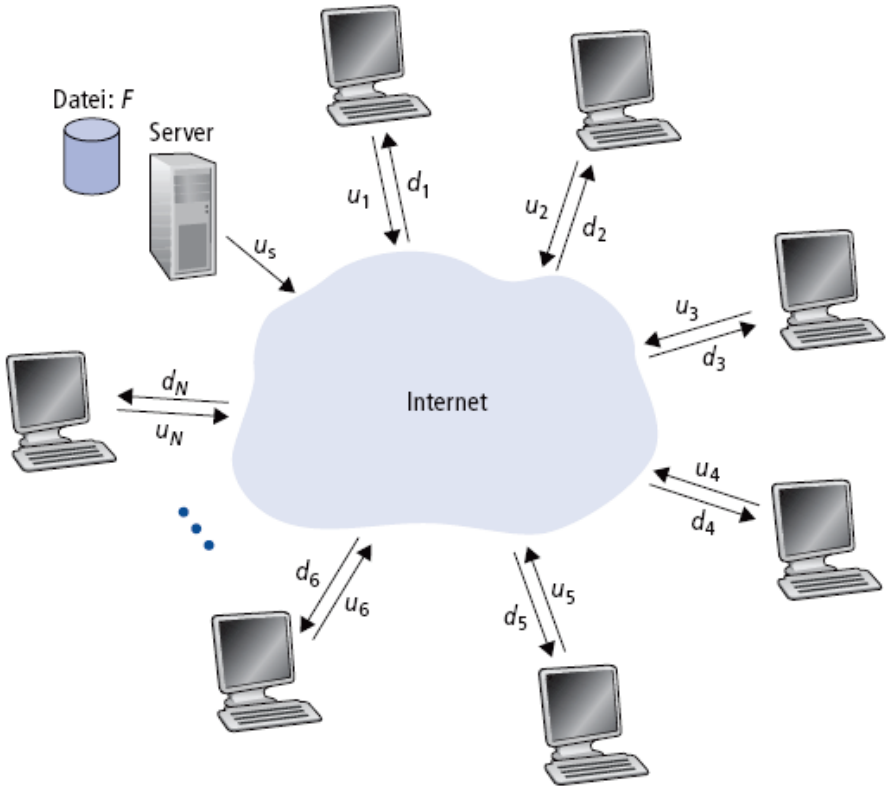
- Kein zentraler Server vorhanden
- Arbiträre Endsysteme kommunizieren direkt
- Die Peers (gleichwertige Entitäten) sind mit Unterbrechungen mit dem Netz verbunden und wechseln ihre IP Adressen



- Beispiele:
  - File Sharing (BitTorrent)
  - Streaming (KanKan)
  - VoIP (Skype)

2.6.1 Client-Server- vs. P2P-Architektur

Frage: Wie lange dauert es bei den beiden unterschiedlichen Architekturen eine Datei der Größe F, die zu Anfang nur auf einem Server liegt, an N andere Computer zu verteilen?



u_s: Bandbreite vom Server in das Netz

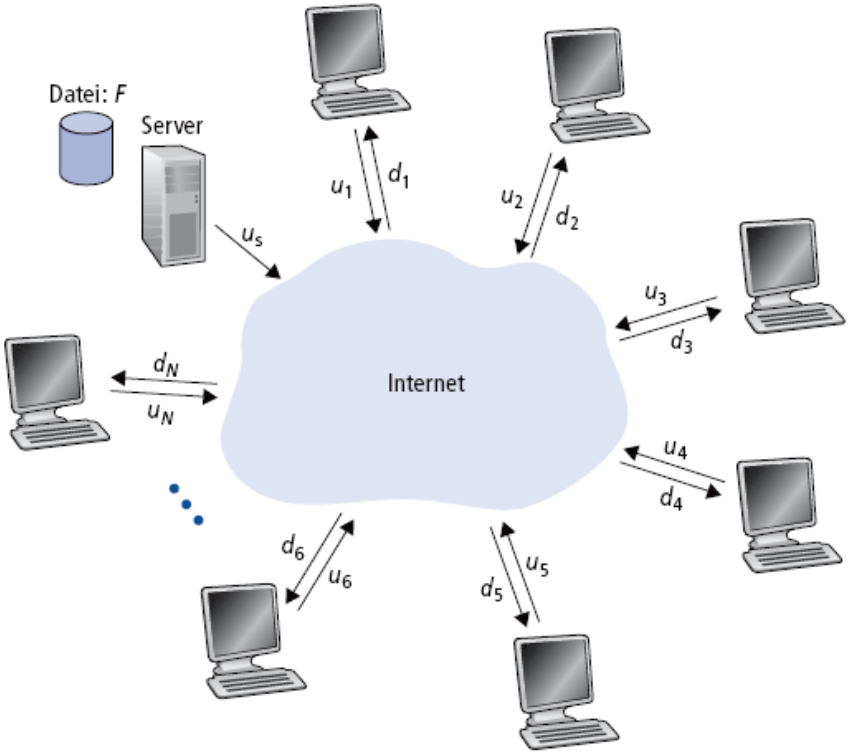
u_i: Bandbreite von Client/Peer *i* in das Netz

d_i: Bandbreite zu Client/Peer *i* aus dem Netz

2.6.1 Client-Server- vs. P2P-Architektur

Client-Server Architektur:

- Server sendet N Kopien parallel:
  → *Zeit: $N \cdot F/u_s$*
- Client i benötigt F/d_i Sekunden für den Download



Zeit um die Datei an *N* Computer mittels Client-Server-Architektur zu übertragen:

$$d_{cs} = \max \{ N \cdot F/u_s, F/\min(d_i) \}$$

Wächst für große N linear mit N!

u_s: Bandbreite vom Server in das Netz

u_i: Bandbreite von Client/Peer i in das Netz

d_i: Bandbreite zu Client/Peer i aus dem Netz

2.6.1 Client-Server- vs. P2P-Architektur

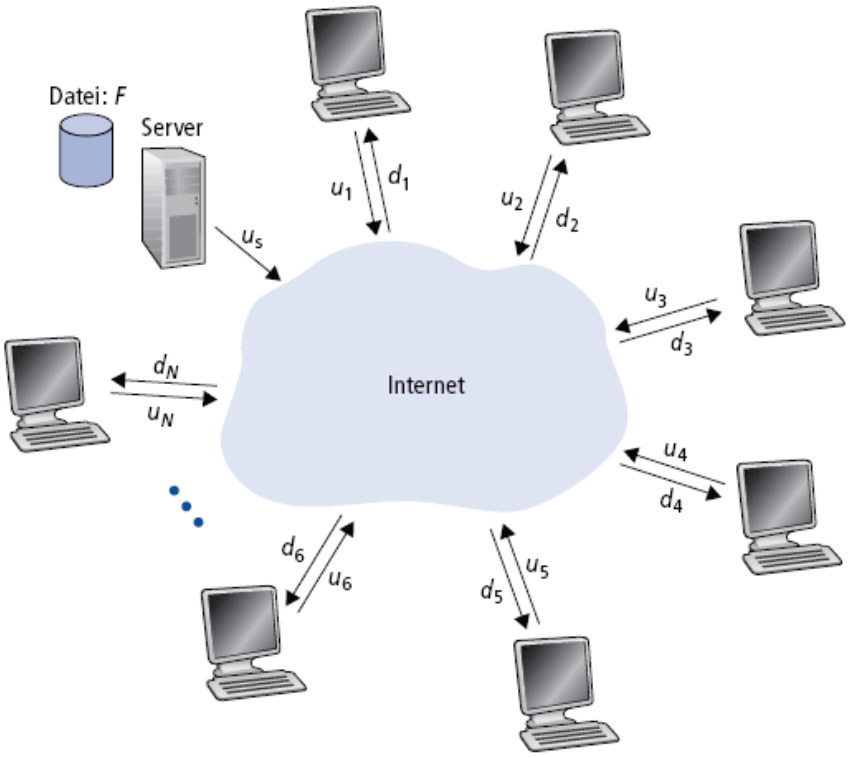
Peer-to-Peer Architektur:

- Server muss eine Kopie senden: F/u_s
- Client i braucht F/d_i Sekunden für den Download
- $N \cdot F$ Bits müssen insgesamt heruntergeladen werden
- Höchstmögliche Datenrate ins Netz:

$$u_s + \sum_{i=1,N} u_i$$

Zeit um die Datei an *N* Computer mittels Client-Server-Architektur zu übertragen:

$$d_{p2p} = \max \{ F/u_s, F/\min(d_i) , N \cdot F/(u_s + \sum_{i=1,N} u_i) \}$$

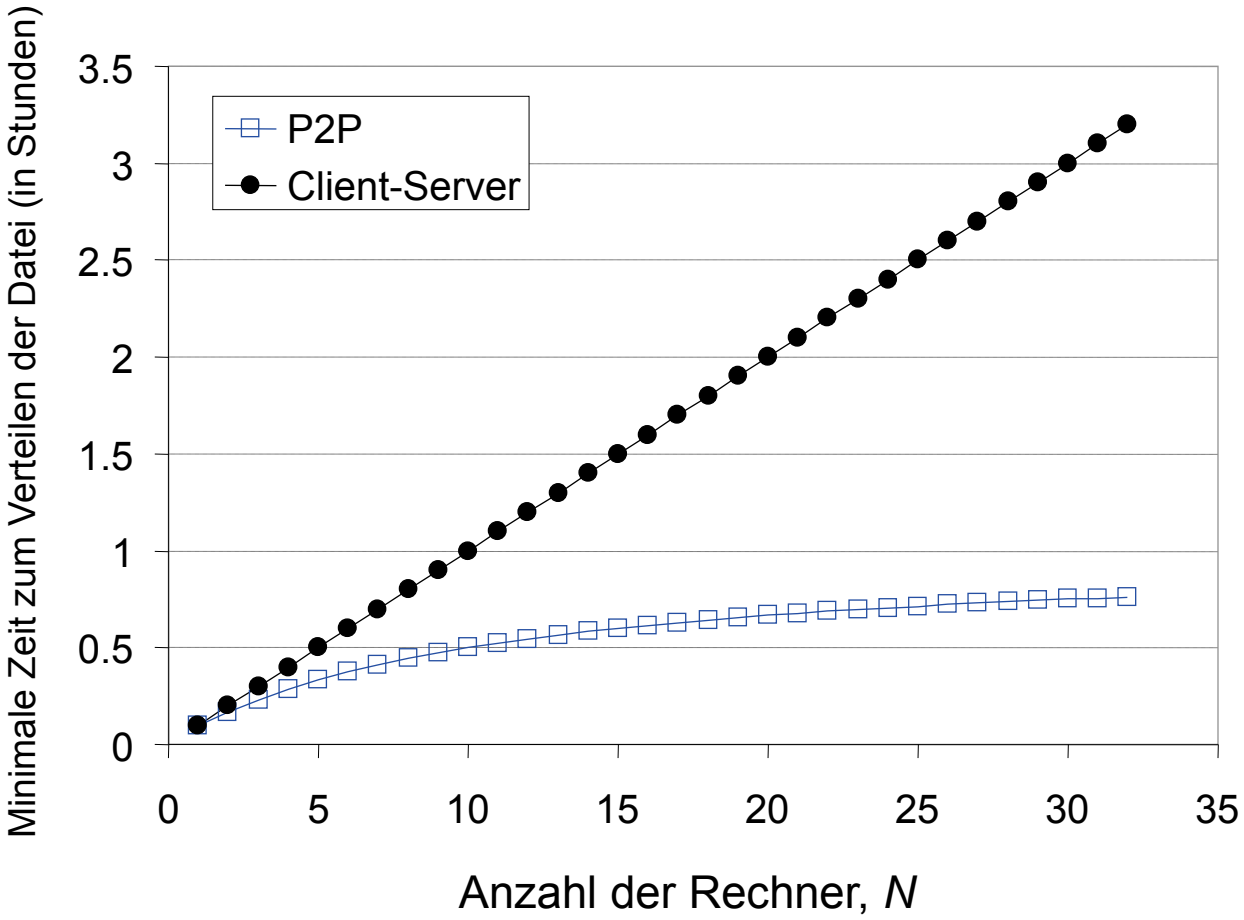


u_s: Bandbreite vom Server in das Netz

u_i: Bandbreite von Client/Peer i in das Netz

d_i: Bandbreite zu Client/Peer i aus dem Netz

2.6.1 Client-Server- vs. P2P-Architektur



Parameter:

- F/u_i = 1 Stunde
- $u_s = 10\ u_i$
- $D_{min} \geq u_s$