Netzwerktechnologien
3 VO

Univ.-Prof. Dr. Helmut Hlavacs
helmut.hlavacs@univie.ac.at

Dr. Ivan Gojmerac
gojmerac@ftw.at

Bachelorstudium Medieninformatik
SS 2012

Kapitel 2 – Anwendungsschicht

2.1 Grundlagen der Netzwerkanwendungen
2.2 Das Web und HTTP
2.3 Dateitransfer: FTP
2.4 E-Mail im Internet
2.5 DNS – der Verzeichnisdienst des Internets
2.6 Peer-to-Peer-Anwendungen
2.7 Socket-Programmierung mit TCP
2.8 Socket-Programmierung mit UDP

2.1 Grundlagen der Netzwerkanwendungen

Entwicklung von Netzanwendungen

Programme, die

- auf mehreren (verschiedenen) Endsystemen laufen
- über das Netzwerk kommunizieren
  - Beispiel: Die Software eines Webservers kommuniziert mit dem Browser

Kaum Software für das Innere des Netzwerkes

- Im Inneren des Netzwerkes werden keine Anwendungen ausgeführt
- Die Konzentration auf Endsysteme erlaubt eine schnelle Entwicklung und Verbreitung der Software

Beispiele für Netzanwendungen:

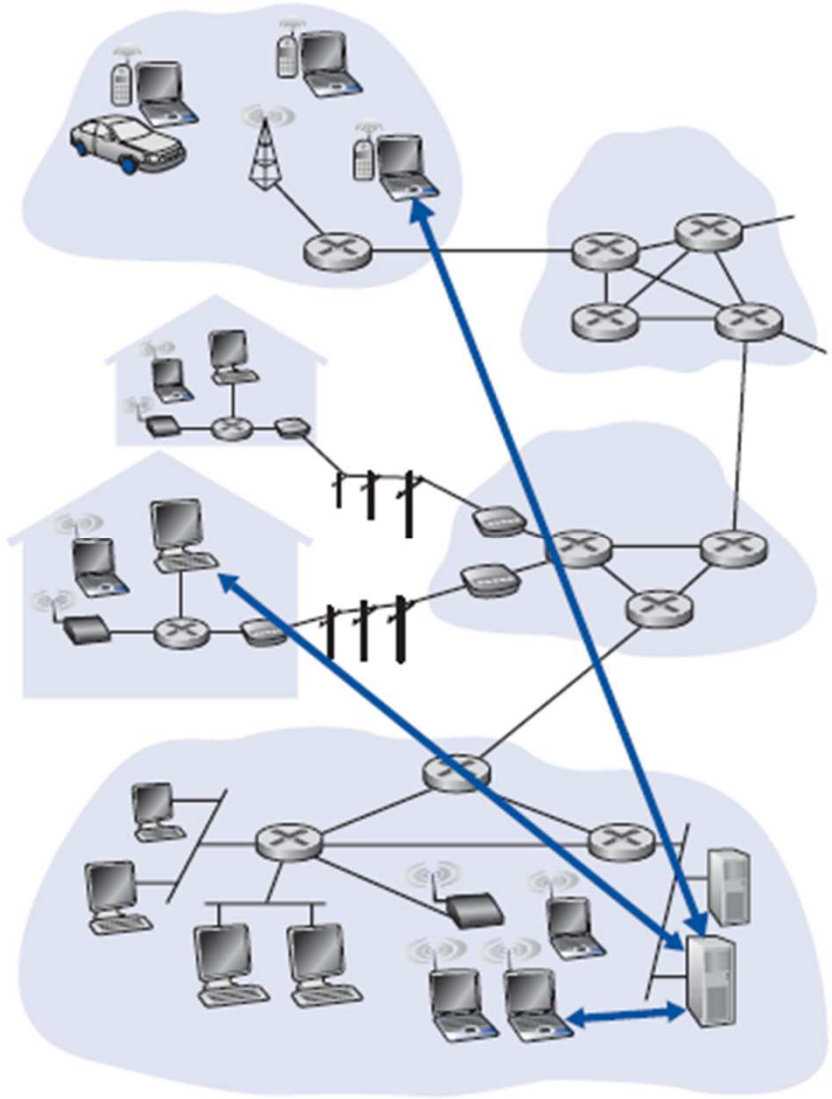
- E-Mail
- Web
- Instant Messaging
- Terminalfernzugriff
- P2P-Filesharing
- Netzwerkspiele
- Streaming von Videoclips
- Voice over IP (VoIP)
- Videokonferenzen
- Grid Computing

2.1.1 Architektur von Netzwerkanwendungen

Client-Server-Architektur

Server
- Bearbeitet Anfragen von Clients
- Immer eingeschaltet
- Feste IP-Adresse
- Serverfarmen, um zu skalieren

Clients
- Kommunizieren mit Servern
- Permanent oder nur manchmal online
- Können dynamische IP-Adressen haben
- Kommunizieren nicht direkt miteinander



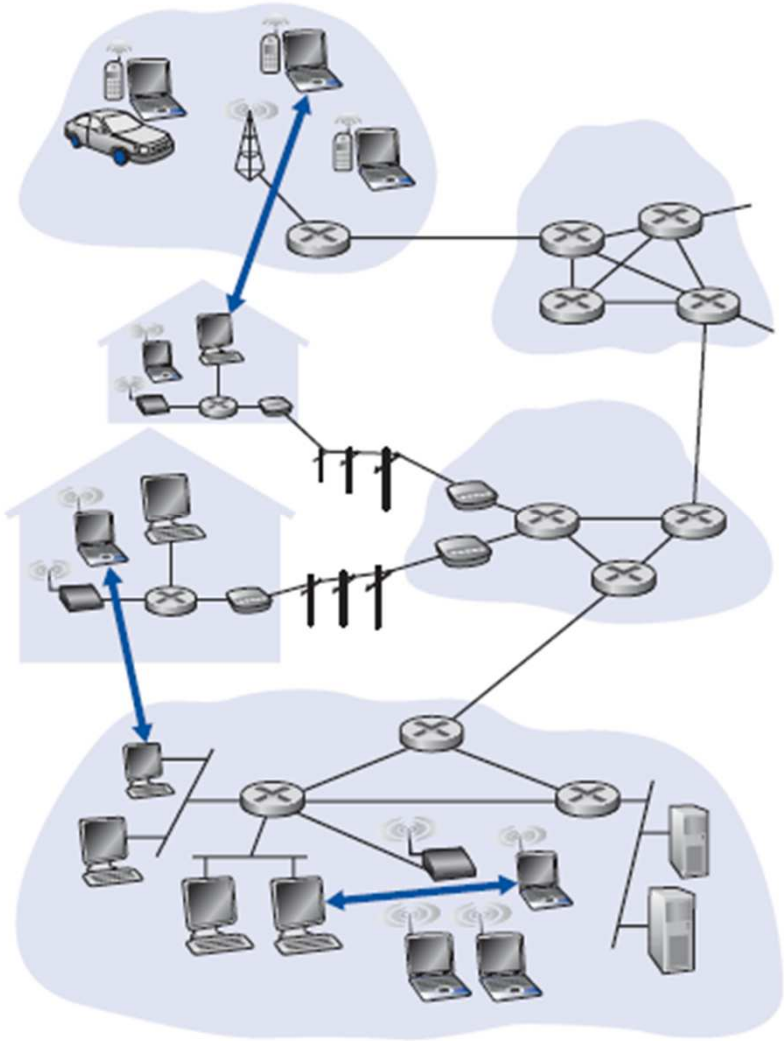
Beispiele für bekannte Anwendungen mit Client-Server-Architektur:

- Das Web
- FTP
- Telnet
- E-Mail

2.1.1 Architektur von Netzwerkanwendungen

Reine Peer-to-Peer-Architektur (P2P)

- **Keine** Server (kostengünstig)
- Beliebige Endsysteme kommunizieren direkt miteinander
- Peers sind nur sporadisch angeschlossen und wechseln ihre IP-Adresse
- Selbstskalierbarkeit (!), aber schwer zu warten und zu kontrollieren!



Beispiele für bekannte Anwendungen mit P2P-Architektur:

- File-Distribution (z.B. BitTorrent)
- Filesharing (z.B. eMule, LimeWire)
- IPTV (z.B. PPLive)

2.1.1 Architektur von Netzwerkanwendungen

Kombination von Client-Server und P2P

Skype - Voice-over-IP Anwendung

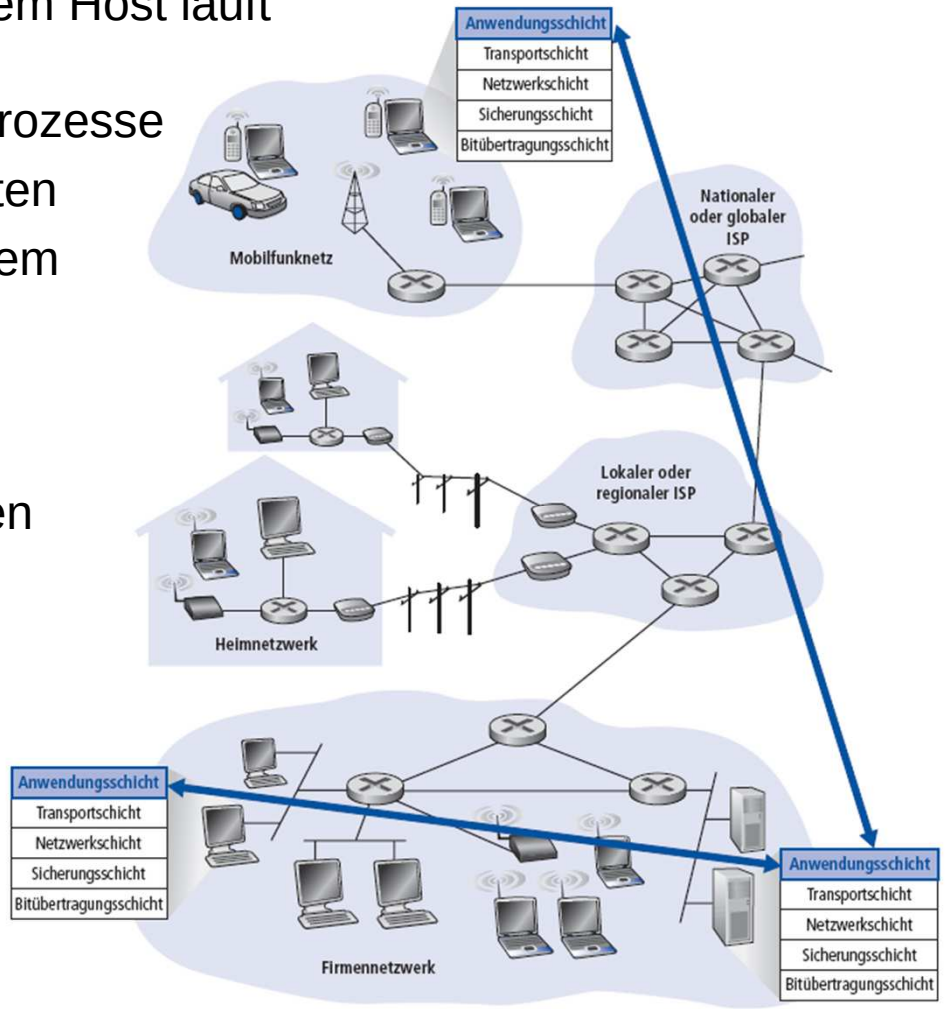
- Zentraler Server: Adresse des Kommunikationspartners finden
- Verbindung zwischen Clients: direkt (P2P)

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 verbindet
- Benutzer fragt beim Server nach Informationen über seine Freunde und Bekannten

2.1.2 Kommunikation zwischen Prozessen

- Prozess: Programm, welches auf einem Host läuft
- Innerhalb eines Hosts können zwei Prozesse mit Inter-Prozess-Kommunikation Daten austauschen (durch das Betriebssystem unterstützt)
- Prozesse auf verschiedenen Hosts kommunizieren, indem sie Nachrichten über ein Netzwerk austauschen



Die Abbildung veranschaulicht, wie Prozesse miteinander mittels der Anwendungsschicht des fünfschichtigen Internet-Protokollstapels kommunizieren.

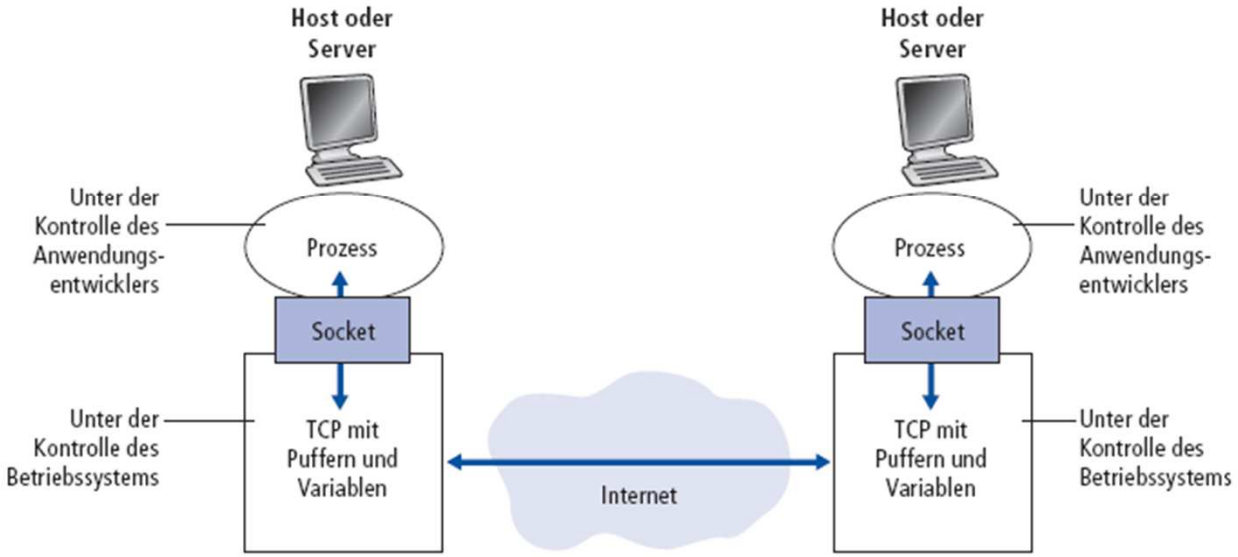
2.1.2 Client- und Server-Prozesse

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

- Client-Prozess: definiert als Prozess, der die Kommunikation beginnt
- Server-Prozess: definiert als Prozess, der darauf wartet, kontaktiert zu werden

- P2P: Rollen wechseln

2.1.2 Sockets



- Prozesse senden/empfangen Nachrichten über einen **Socket**
- **Schnittstelle** zwischen der Anwendungsschicht und der Transportschicht = Anwendungsprogrammierschnittstelle (API)
- Analogie Tür zu einem Haus (entspricht Socket zu einem Prozess):
  - Der sendende Prozess schiebt die Nachrichten durch seine Tür raus
  - Der sendende Prozess verlässt sich auf die Transportinfrastruktur auf der anderen Seite der Tür, um die Nachricht zum Socket des empfangenden Prozesses zu bringen
  - Sobald die Nachricht beim Zielhost ankommt, tritt sie durch die Tür des empfangenden Prozesses, der danach auf die Nachricht reagiert

2.1.2 Addressierung von Prozessen

- Um eine Nachricht empfangen zu können, muss ein Prozess identifiziert werden können

- Prozesse werden durch die 32Bit lange IP-Adresse des Hosts auf dem sie laufen UND eine Portnummer identifiziert
  - Beispiel-Portnummern:
    - HTTP-Server: 80
    - E-Mail-Server: 25

2.1.2 Anwendungsschicht

Anwendungsprotokolle bestimmen:

- Arten von Nachrichten, Syntax der Nachrichten, Semantik der Nachrichten, Regeln für das Senden von und Antworten auf Nachrichten

Öffentlich verfügbare Protokolle:

- Definiert in RFCs
- Ermöglichen Interoperabilität
- z.B. HTTP, SMTP

Proprietäre Protokolle:

- z.B. Skype

- Brauchen: **Transportprotokoll**

2.1.3 Transportdienste für Anwendungen

Wahl des Transportdienstes nach 3 Kriterien:

- Datenverlust
  - Einige Anwendungen können Datenverlust tolerieren (z.B. Audioübertragungen)
  - Andere Anwendungen benötigen einen absolut zuverlässigen Datentransfer (z.B. Dateitransfer)
- Bandbreite
  - Einige Anwendungen (z.B. Multimedia-Streaming) brauchen eine Mindestbandbreite, um zu funktionieren (in-elastisch, BB-empfindlich)
  - Andere Anwendungen verwenden einfach die verfügbare Bandbreite (bandbreitenelastische Anwendungen)
- Zeitanforderungen
  - Einige Anwendungen (z.B. Internettelefonie oder Netzwerkspiele) tolerieren nur eine sehr geringe Verzögerung
- Sicherheit

2.1.3 Beispiele für Anforderungen von Anwendungen

Anwendung	Datenverlust	Bandbreite	Echtzeit
Dateitransfer	Kein Verlust	Elastisch	Nein
E-Mail	Kein Verlust	Elastisch	Nein
Web	Kein Verlust	Elastisch (wenige Kbps)	Nein
Internettelefonie/ Bildkonferenz	Toleriert Verluste	Audio: wenige Kbps bis 1 Mbps Video: 10 Kbps bis 5 Mbps	Ja: einige Hundert ms
Gespeichertes Audio/Video	Toleriert Verluste	Wie oben	Ja: wenige Sekunden
Interaktive Spiele	Toleriert Verluste	Wenige Kbps bis 10 Kbps	Ja: einige Hundert ms
Instant Messaging	Kein Verlust	Elastisch	Ja und nein

2.1.4 Dienste der Transportprotokolle

TCP-Dienst:

- Verbindungsorientierung: Herstellen einer Verbindung zwischen Client und Server
- Zuverlässiger Transport zwischen sendendem und empfangendem Prozess
- Überlastkontrolle: Bremsen des Senders, wenn das Netzwerk überlastet ist

- Nicht: Zeit- und Bandbreitengarantien, Verschlüsselung

UDP-Dienst:

- Unzuverlässiger Transport von Daten zwischen Sender und Empfänger

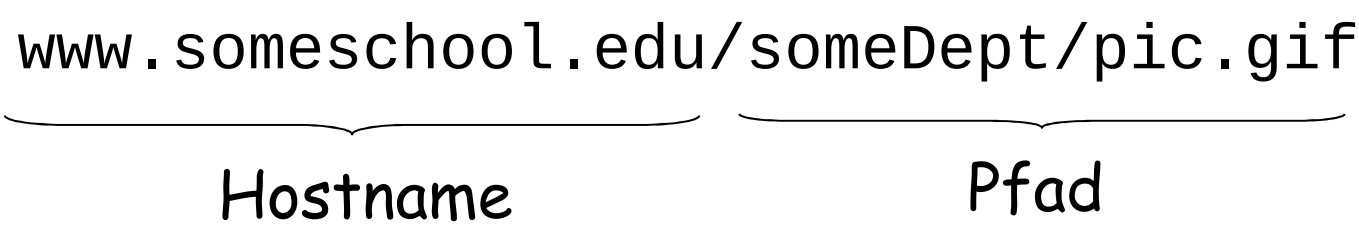
- Nicht: Verbindungsorientierung, Zuverlässigkeit, Überlastkontrolle, Zeit- oder Bandbreitengarantien, Verschlüsselung

2.1.4 Beispiele für Anwendungsschicht- und Transportprotokolle im Internet

Anwendung	Anwendungsschichtprotokoll	Zugrunde liegendes Transportprotokoll
E-Mail-Dienst	SMTP [RFC 2821]	TCP
Remote-Terminalzugang	Telnet [RFC 854]	TCP
World Wide Web	HTTP [RFC 2616]	TCP
Dateitransfer	FTP [RFC 959]	TCP
Multimedia-Streaming	HTTP (z. B. YouTube), RTP	TCP oder UDP
Internettelefonie	SIP, RTP oder proprietär (z. B. Skype)	Normalerweise UDP

2.2 Web und HTTP

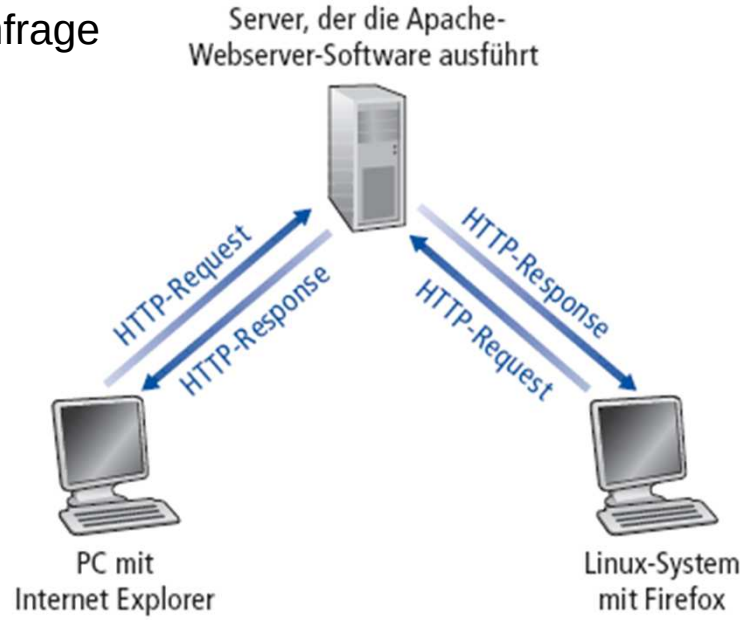
- Eine Webseite besteht aus Objekten
  - Objekte können sein: HTML-Dateien, JPEG-Bilder, Java-Applets, Audiodateien …
- Eine Webseite hat eine Basis-HTML-Datei, die mehrere referenzierte Objekte beinhalten kann
- Jedes Objekt kann durch eine URL (Uniform Resource Locator) adressiert werden
  - Beispiel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



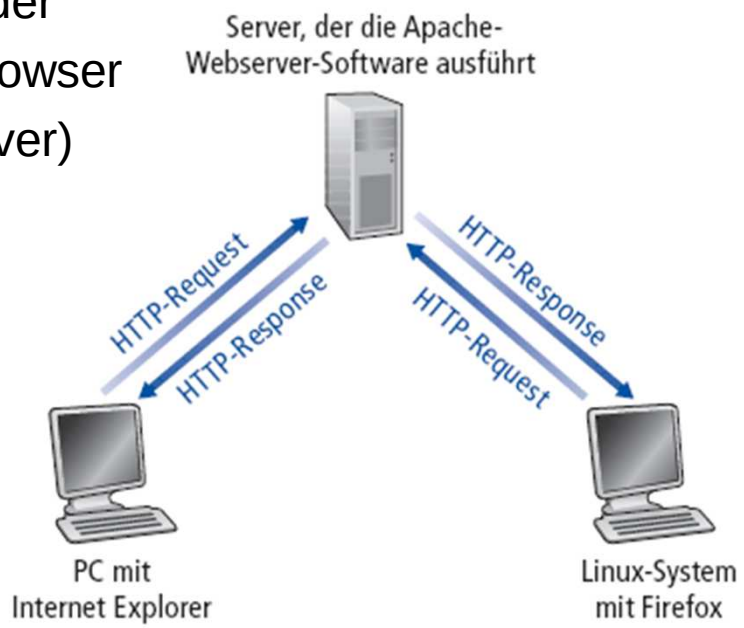
2.2.1 Überblick über HTTP

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

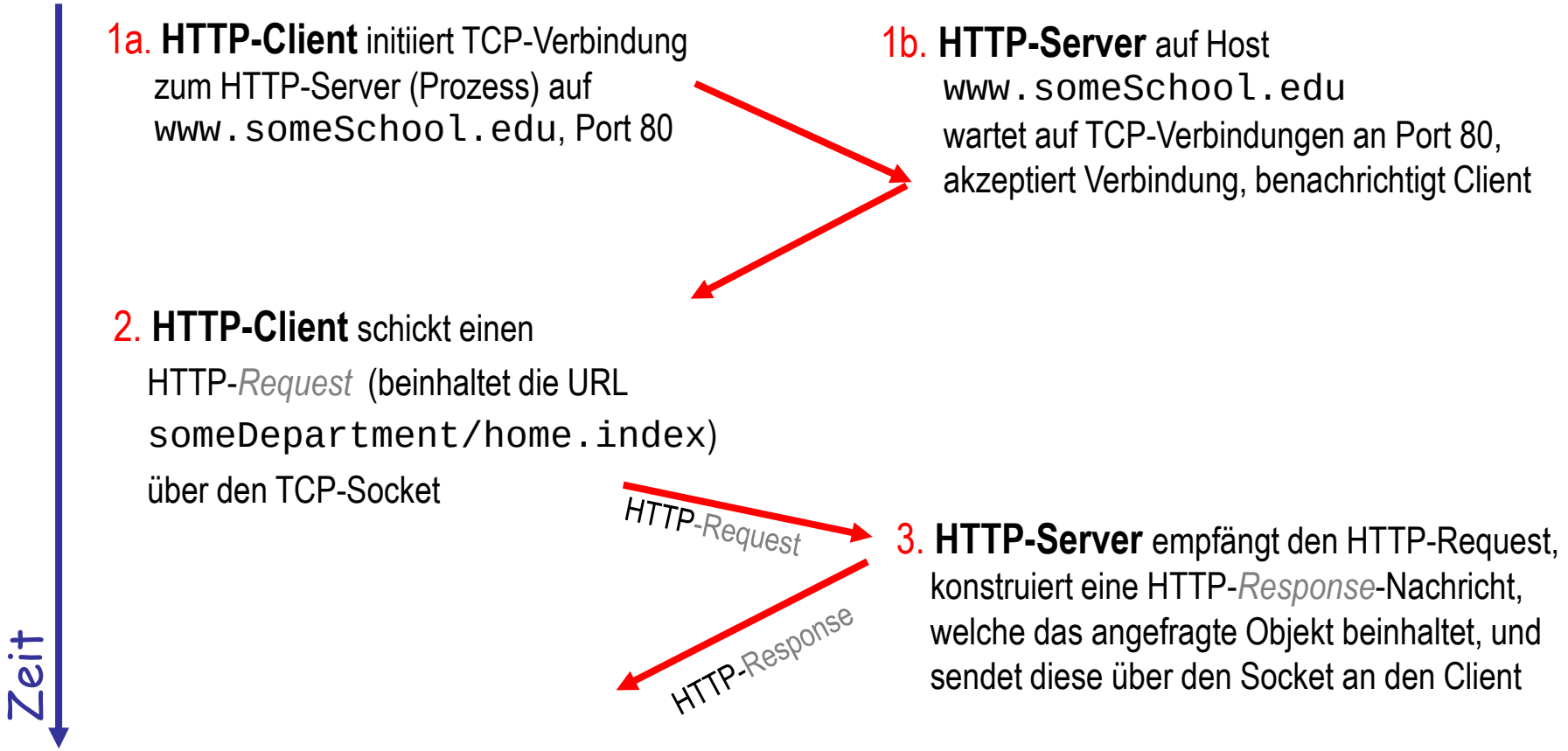
→ HTTP/1.1 verwendet standardmäßig persistentes HTTP

2.2.2 Bsp: Nichtpersistentes HTTP

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

Es soll folgende URL geladen werden:

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



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.

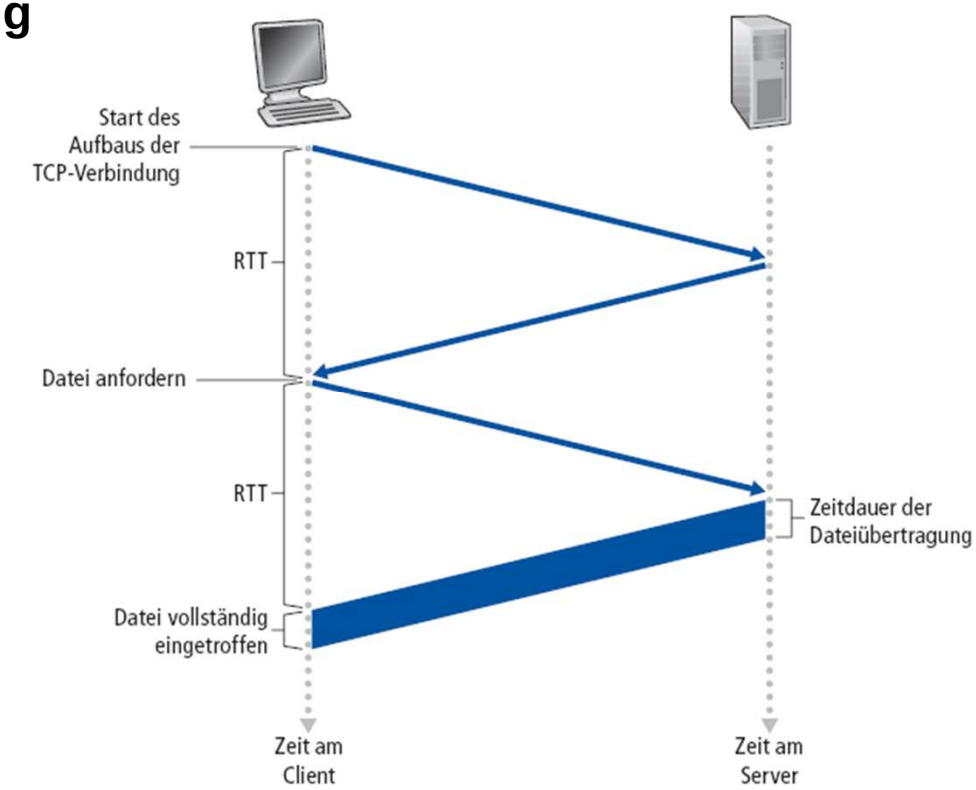
6. Schritte 1 bis 5 werden für jedes der zehn JPEG-Objekte wiederholt, dann kann die Seite vollständig angezeigt werden

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): Zeit, um ein kleines 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

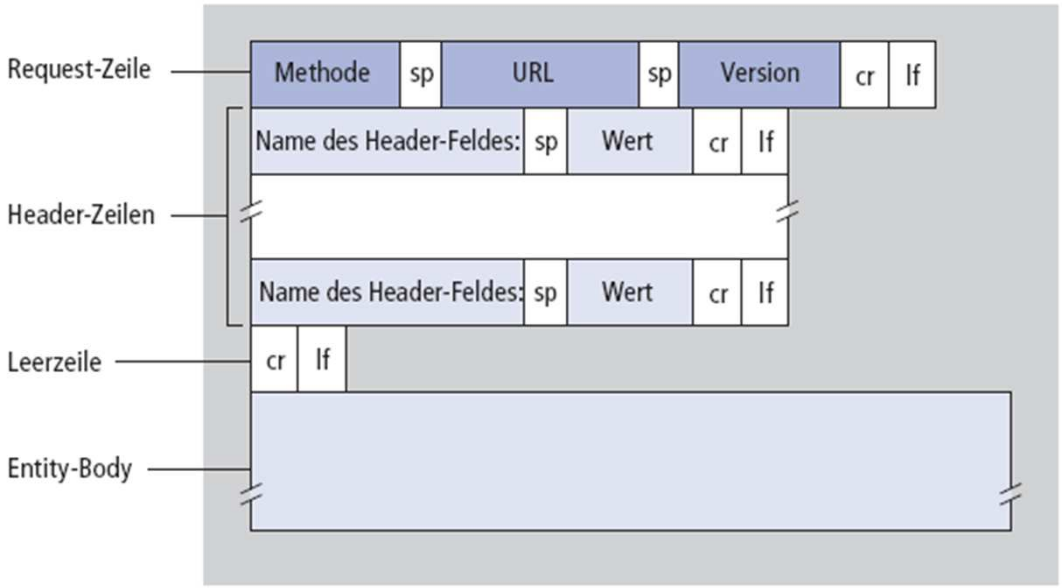
Header-Zeilen

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

```

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

2.2.3 Request-Nachricht Format

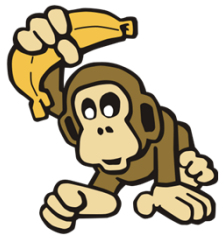


Der Entity-Body wird bei der GET-Methode nicht verwendet (voriges Beispiel), aber bei der POST-Methode um Daten zu versenden.

→ 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

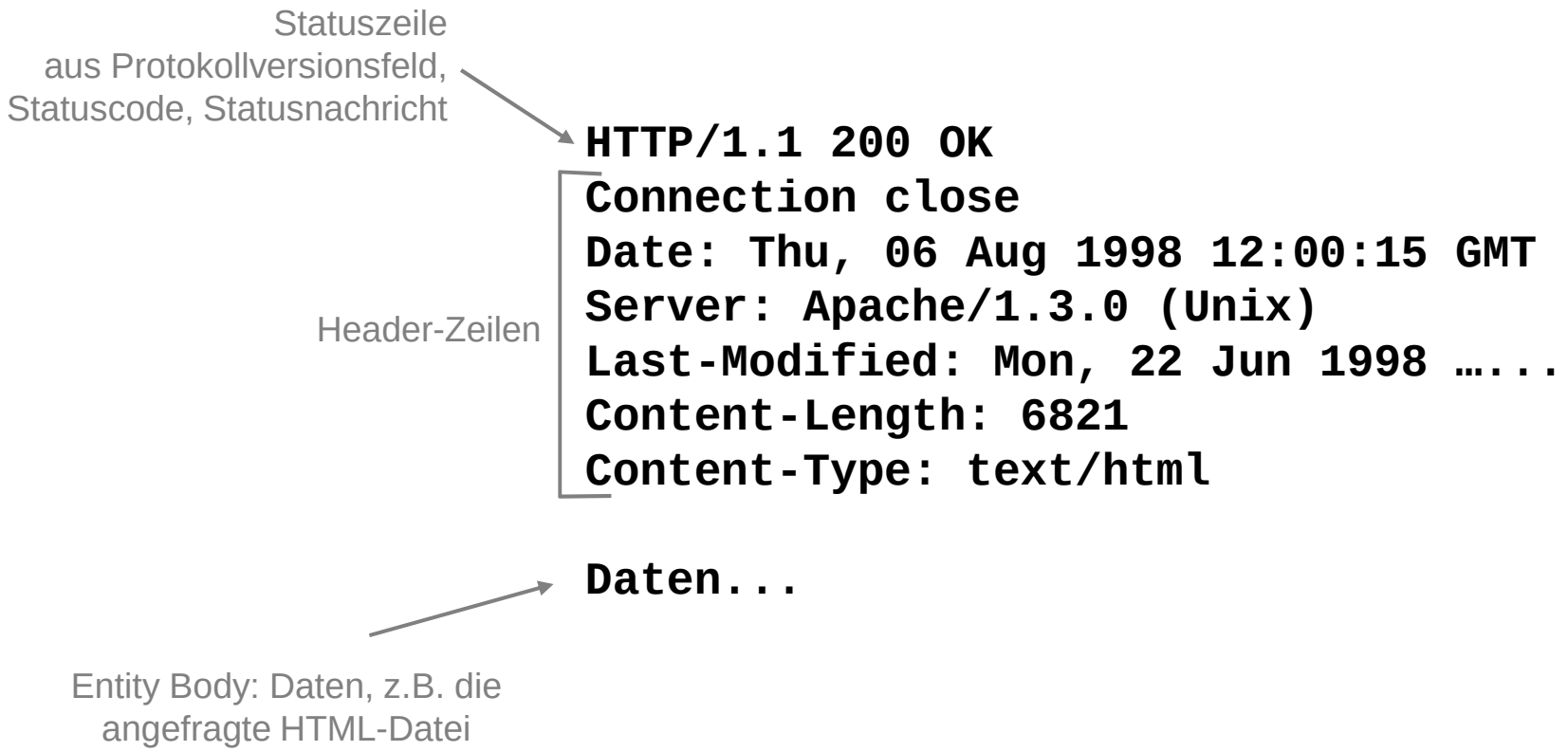
- GET
- POST
- HEAD
  Bittet den Server, nur die Kopfzeilen der Antwort (und nicht das Objekt) zu übertragen

HTTP/1.1

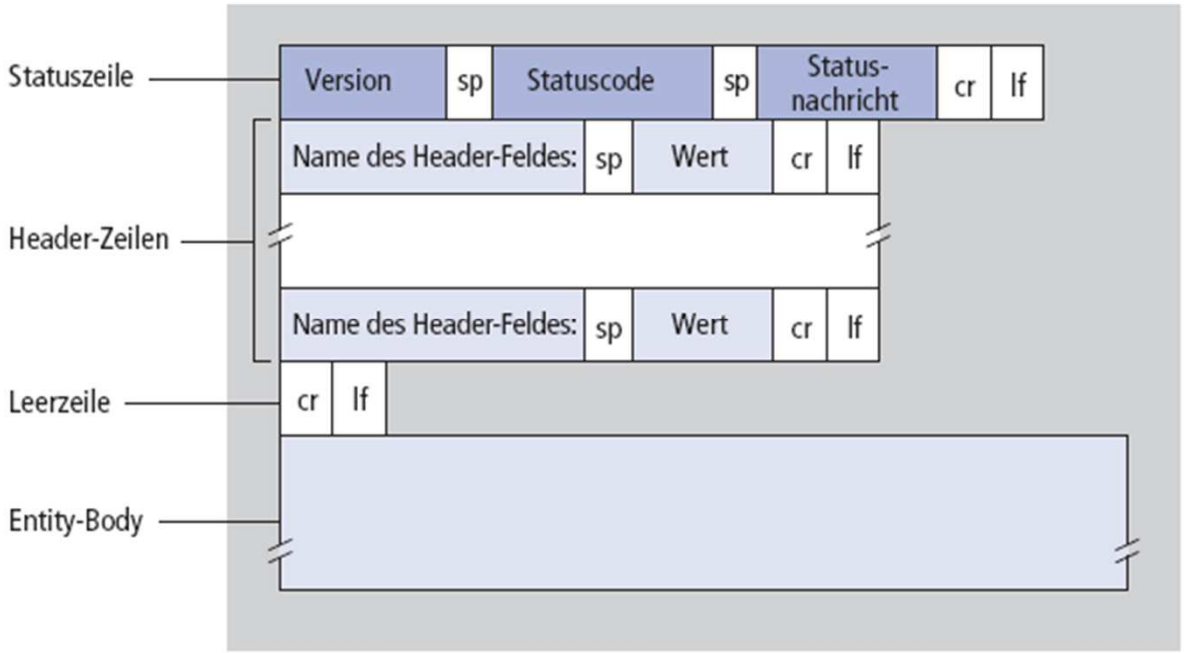
- GET
- POST
- HEAD
- PUT
  Lädt die im Datenteil enthaltene Datei an die durch eine URL bezeichnete Position hoch
- DELETE
  Löscht die durch eine URL angegebene Datei auf dem Server

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 keine Informationen über frühere Anfragen von Clients

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)

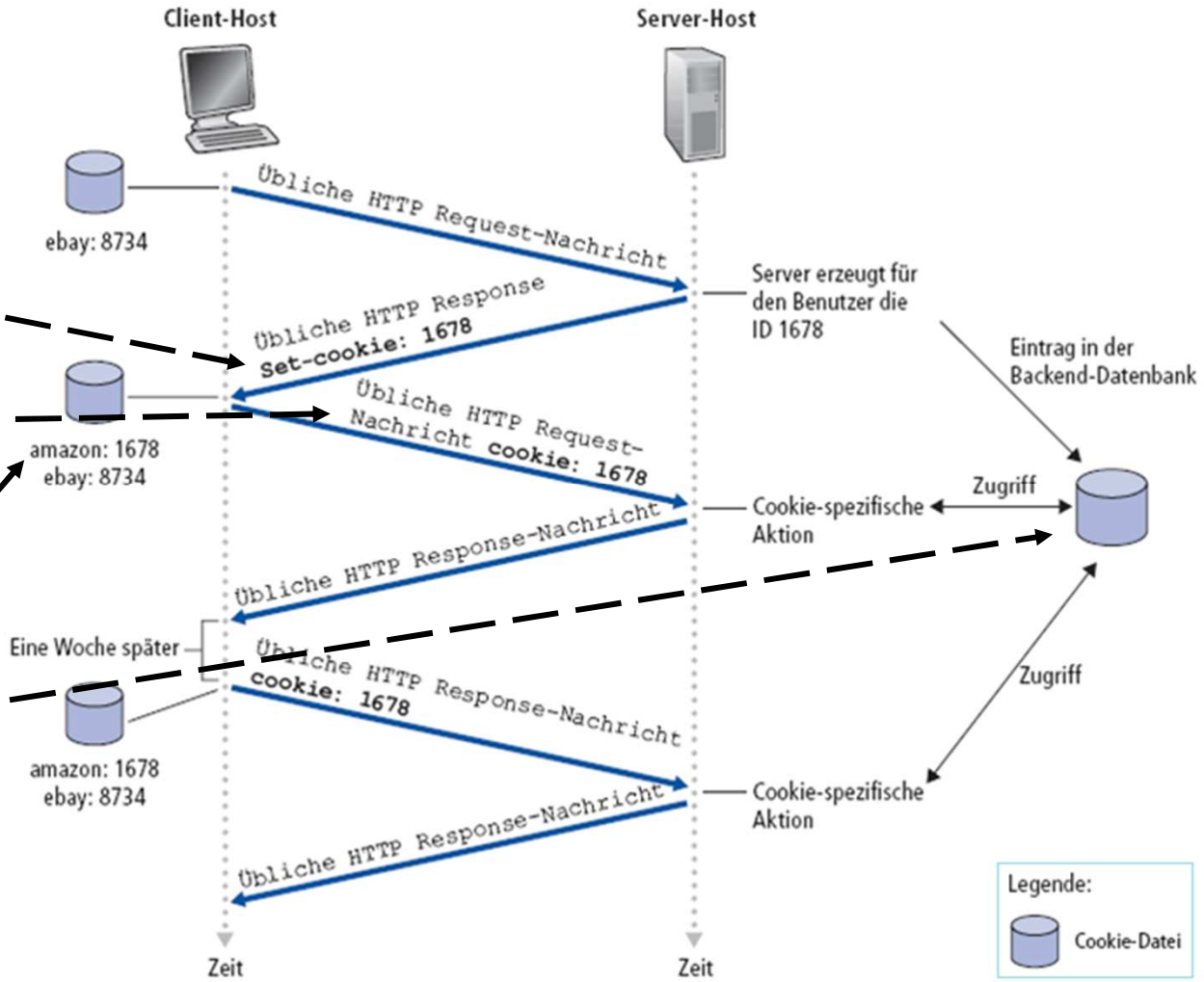
2.2.4 Benutzer-Server-Interaktion: Cookies

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*

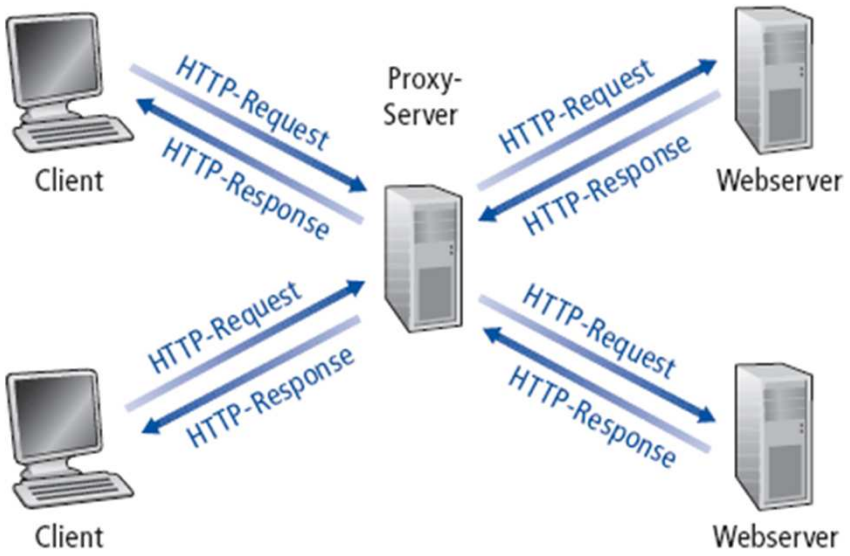
- In den Endsystemen: Zustand wird im Protokoll auf dem Client oder Server gespeichert und für mehrere Transaktionen verwendet
- Cookies: HTTP-Nachrichten beinhalten den Zustand

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

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



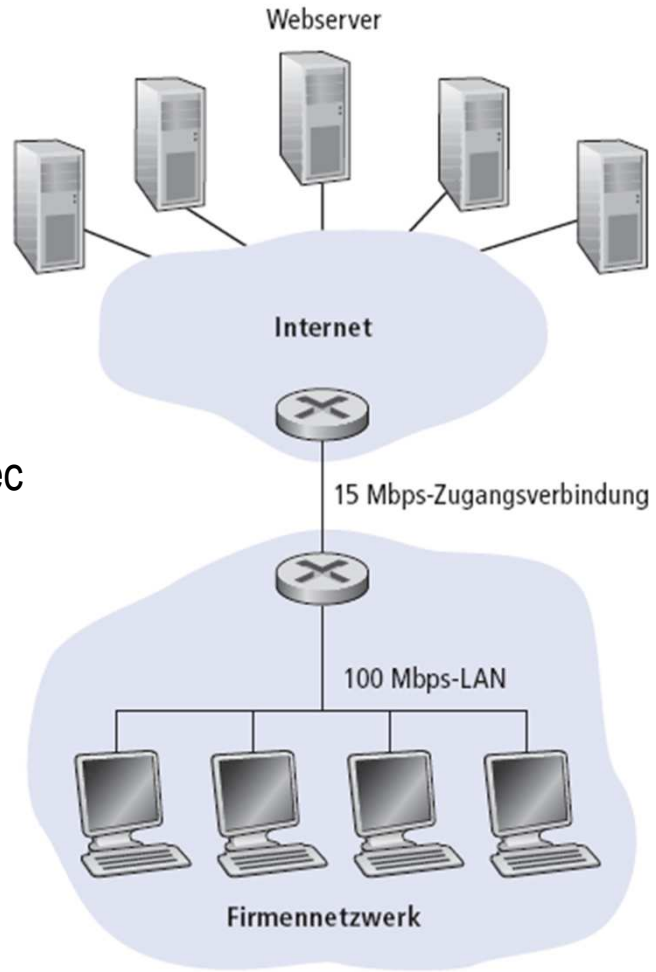
2.2.5 Webcaching

- 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 von Webcaching

- Verringert die Antwortzeit
  - Oft 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
  - Inhalts-anbieter können durch die Nutzung vieler Caches ihre Inhalte gut verbreiten (Ähnliches kann durch P2P-Filesharing erreicht werden)

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

Resultat

Auslastung des LAN = 15%

Auslastung der Zugangsleitung = 100%

Verzögerung von Internet + Zugangsleitung + LAN =

2s + Minuten (!) + Millisekunden

→ **Wartezeit untragbar!**

Offensichtlicher Lösungsansatz:
Bandbreite der Zugangsleitung erhöhen

2.2.5 Beispiel für Webcaching

Annahmen

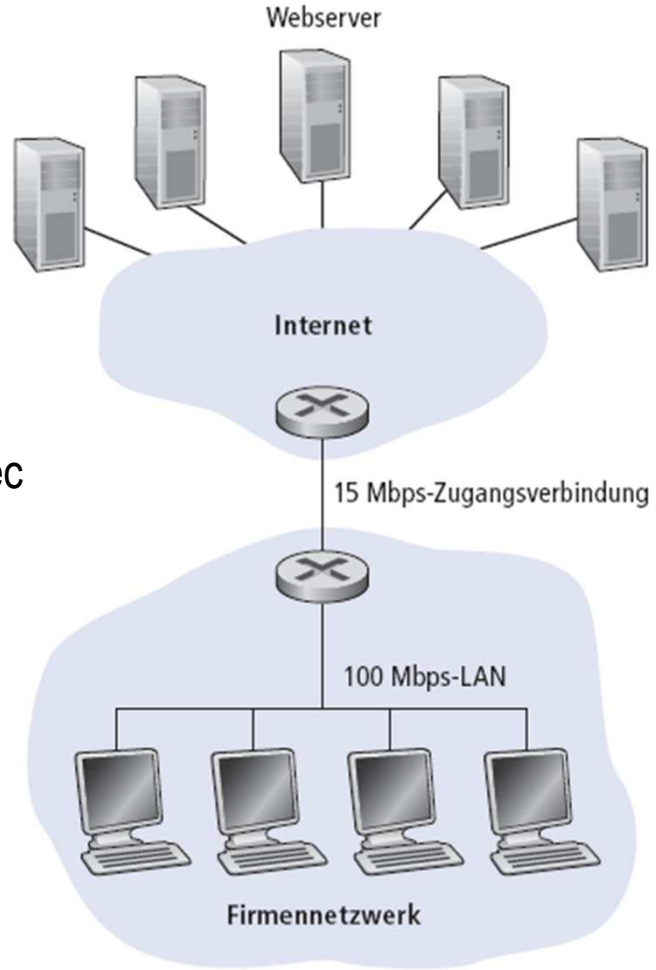
Bandbreite der Zugangsleitung jetzt = **100 Mbps**

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

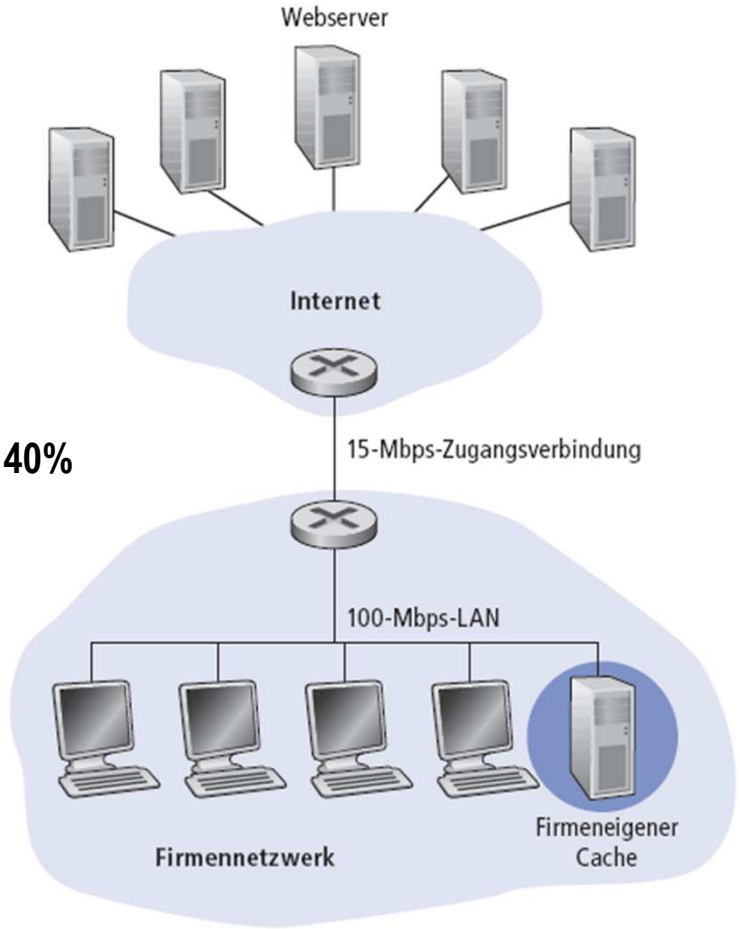
Annahmen

Annahmen bleiben gleich

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%

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

→ Kostengünstiger als Erhöhung der Bandbreite

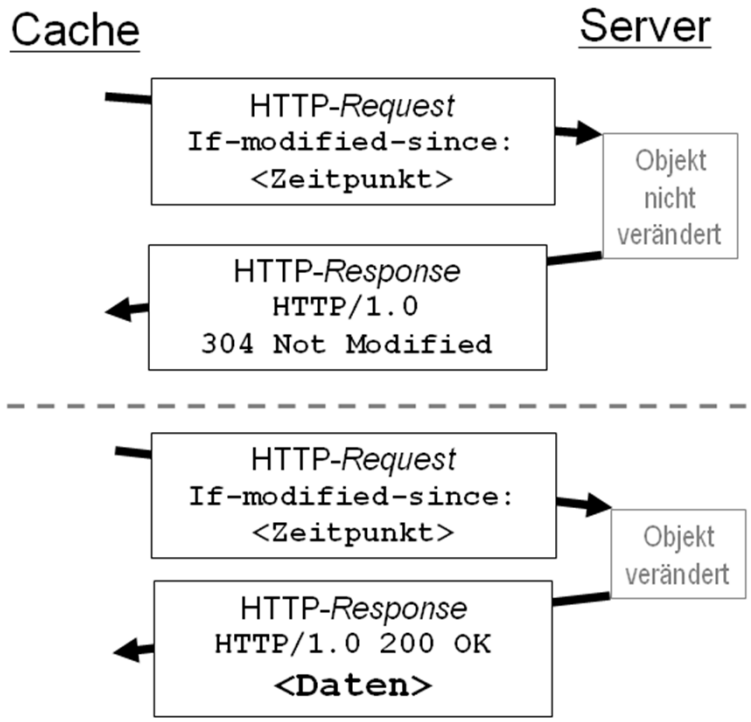
2.2.6 Bedingtes GET

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:

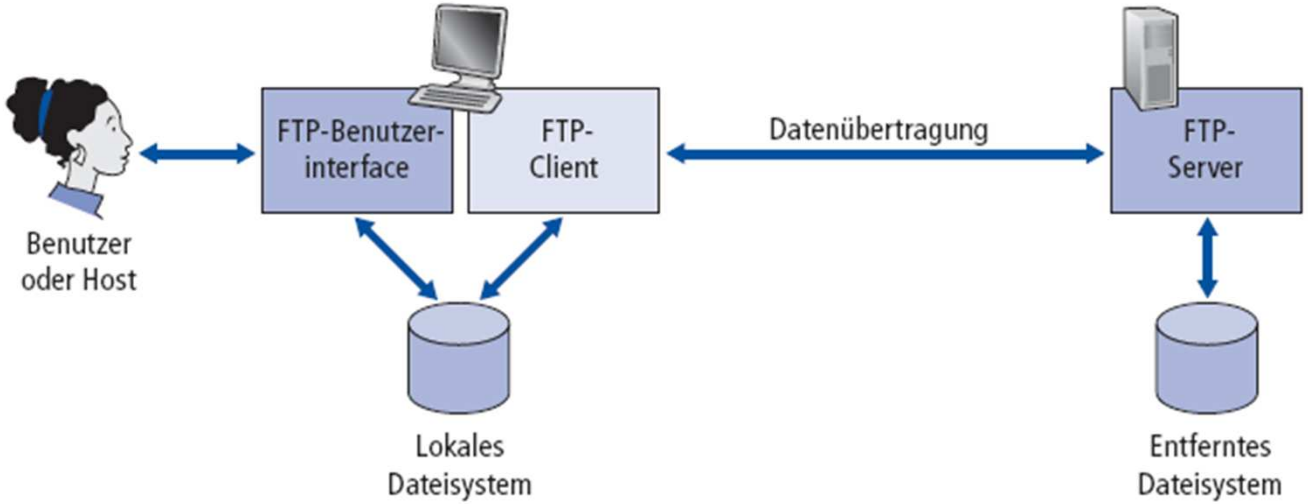
- Die GET-Methode und
- Die Header-Zeile `If-Modified-Since:`



- **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



- 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 über 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

Kontrollverbindung seperat zum Datenkanal:
→ Nennt man **Out-of-Band** Übermittlung der Steuerinformationen

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

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

→ Gesamtanzahl von Sitzungen, die gleichzeitig verwaltet werden können dadurch stark 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:

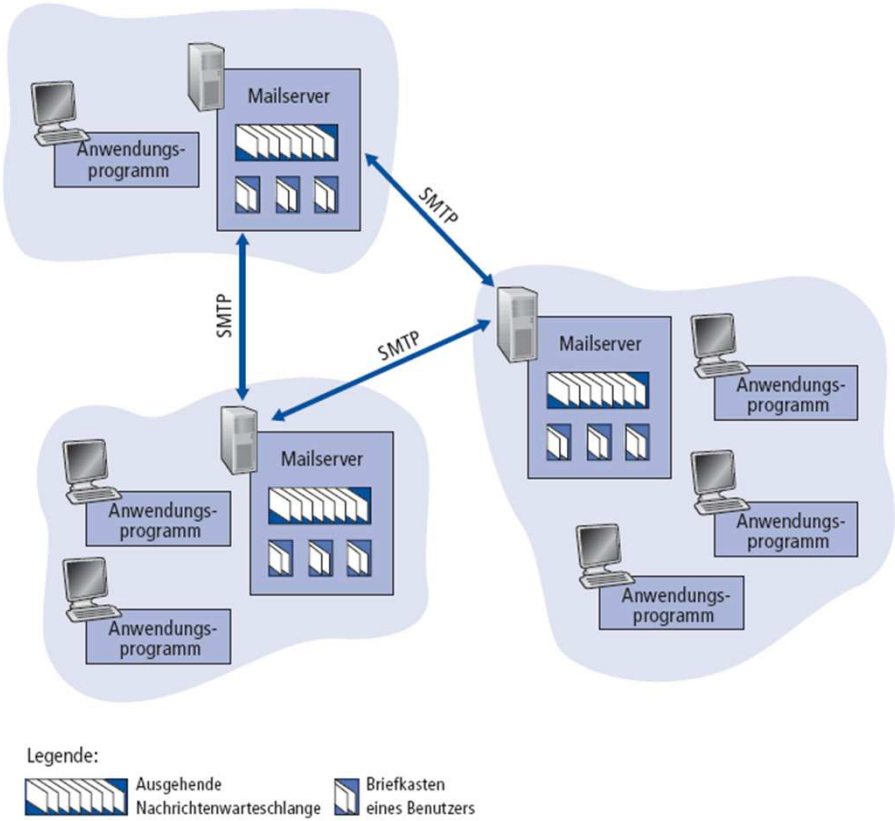
Statuscodes und Erläuterungen (wie bei HTTP)

- **`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

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. Eudora, Outlook, elm, Mozilla Thunderbird

→ Eingehende und ausgehende Nachrichten werden auf dem Server gespeichert

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
  - Server: empfangender Mailserver

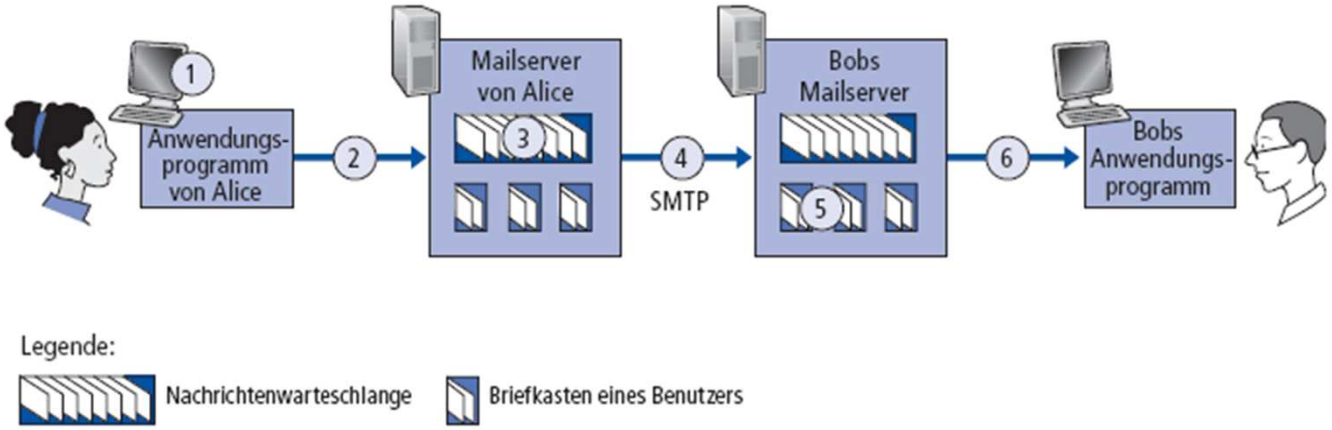
2.4.1 SMTP

- 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) 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