Vorlesung Kommunikationssysteme
Wintersemester 2026/27

Anwendungen und Netzwerkprogrammierung

Christoph Lindemann

Zeitplan

Nr.	Datum	Thema
01	16.10.26	Organisation, Internet Trends und Grundlagen von Android
02	23.10.26	Protokolldesign und das Internet
03	30.10.26	Anwendungen und Netzwerkprogrammierung
04	06.11.26	LAN und Medienzugriff
05	13.11.26	Ethernet und drahtlose Netze
06	20.11.26	LAN Komponenten und WAN Technologien
07	27.11.26	Routingverfahren und Protokolle
08	04.12.26	Internetworking und Adressierung mit IP
09	11.12.26	IP Datagramme
10	18.12.26	Zusätzliche Protokolle und Technologien
11	08.01.27	User Datagram Protocol und Transmission Control Protocol
12	15.01.27	TCP Überlastkontrolle / Internet Routing und Routingprotokolle
13	22.01.27	TCP für Hochgeschwindigkeitsnetze
14	29.01.27	Review der Vorlesung

Überblick

Ziele:

- Einblick in die Socket-API
- Einführung in essentielle Internetanwendungen

Themen:

- Socket API
- WWW: HTTP, HTML, URL
- FTP, Mail, DNS

Socket API

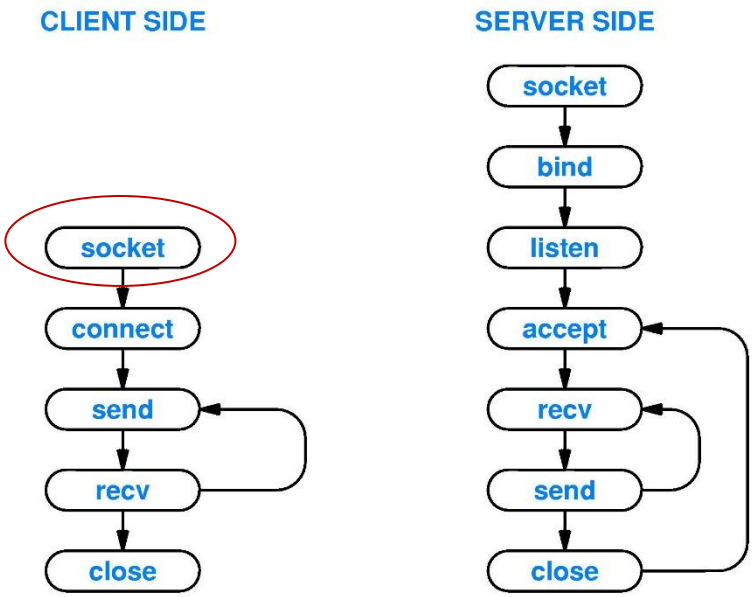
Einleitung (1)

- Anwendungen nutzen das Internet → Betriebssysteme bieten den **Service Internet** über APIs (Application Programming Interfaces) an
- Wichtigste Internet-API: **Socket API**
- **Socket ist Bezeichner für einen spezifischen Kommunikationskanal auf dem Computer**
  - TCP/UDP-Port und IP-Adresse
  - Daten empfangen, senden über den Socket
- **Auf allen wichtigen Betriebssystemen verfügbar**
  - MS Windows, Unixoide wie Linux und macOS, …

Einleitung (2)

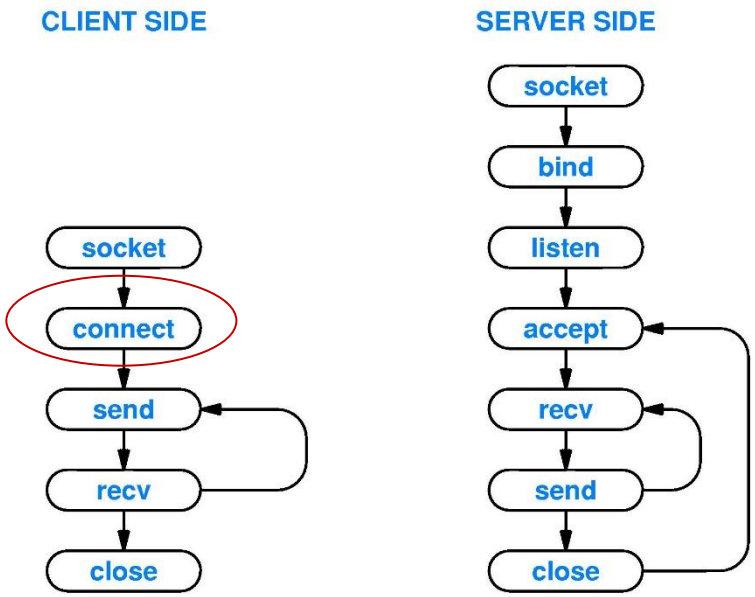
- Sockets sind File Descriptors
  - Standard-Dateioperationen wie read und write lassen sich auch auf Sockets anwenden
- Im Gegensatz zu lokalen Dateien komplexe Parameter
  - Port
  - Transport Layer Protokoll
  - …
- Komplexe API mit vielen Funktionen, einfacher Datentyp (Integer) identifiziert socket

Socket API: Client (1)



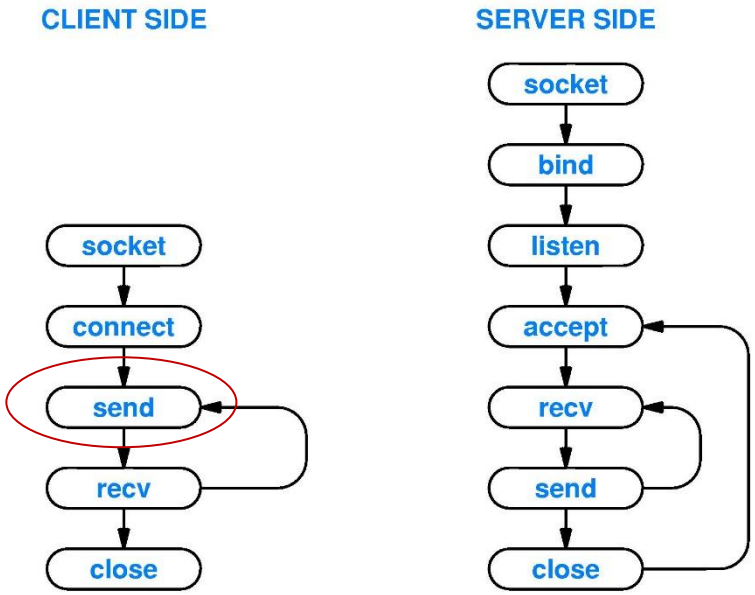
- Erzeugung des Socket
- **descriptor = socket(domain, type, protocol)**
- Bsp.: socket(AF_INET, SOCK_STREAM, 0) erzeugt einen TCP Socket im IPv4-Netz

Socket API: Client (2)



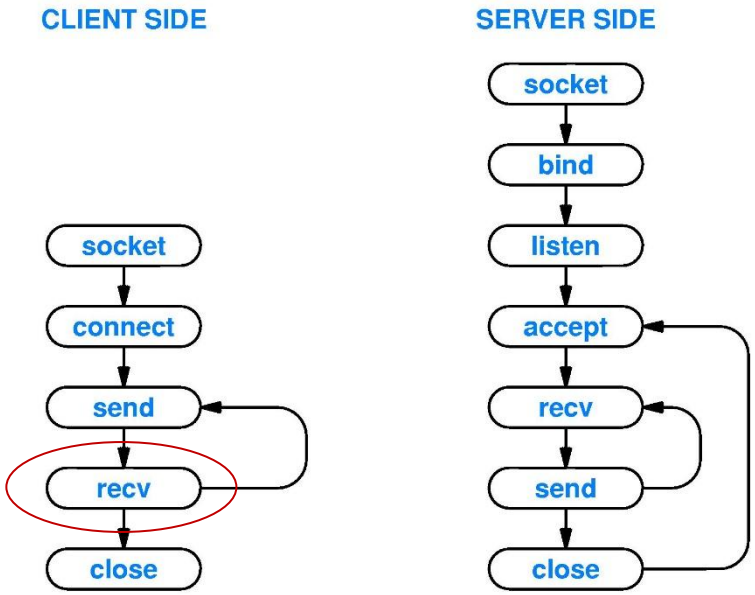
- Verbindungsaufbau zu einem bestimmten Server
  - Folglich TCP Socket bzw. Streaming Typ
- *connect(descriptor, saddress, saddresslen)*
- saddress enthält alle Server-Adress-Informationen

Socket API: Client (3)



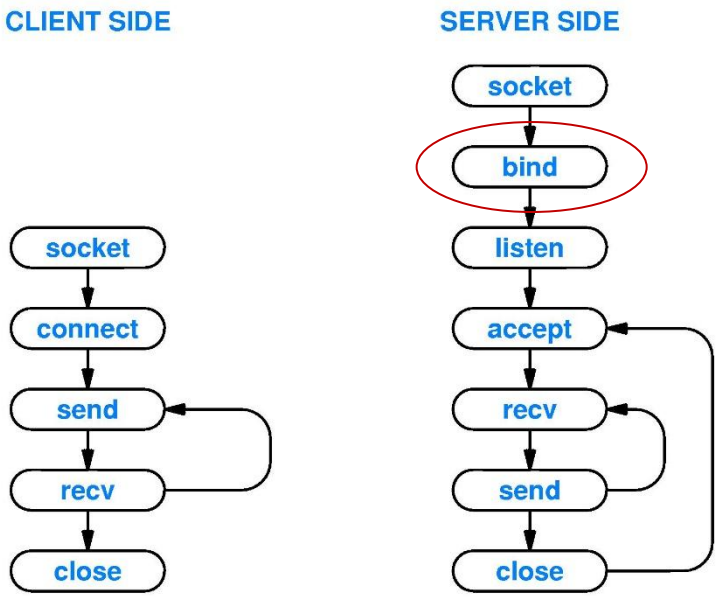
- Übertragung der Daten → typischerweise der Request
- *send(descriptor, data, length, flags)*

Socket API: Client (4)



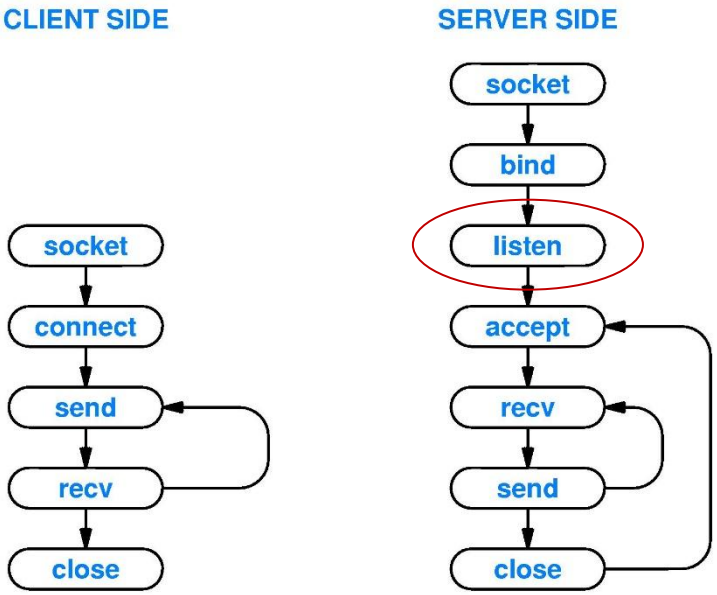
- Empfangen der Antwort → Response des Servers
- *recv(descriptor, buffer, length, flags)*

Socket API: Server (1)



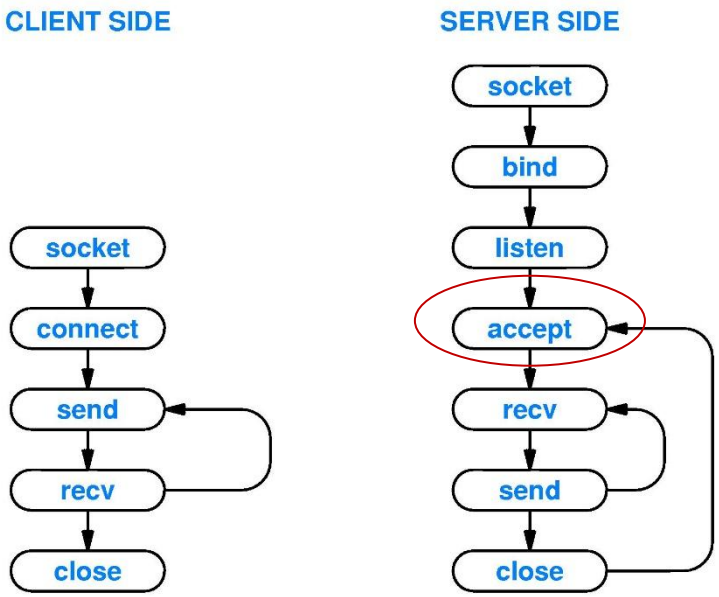
- Festlegen des Ports und der IP-Adresse des Sockets
- *bind(descriptor, localaddr, addrlen)*

Socket API: Server (2)



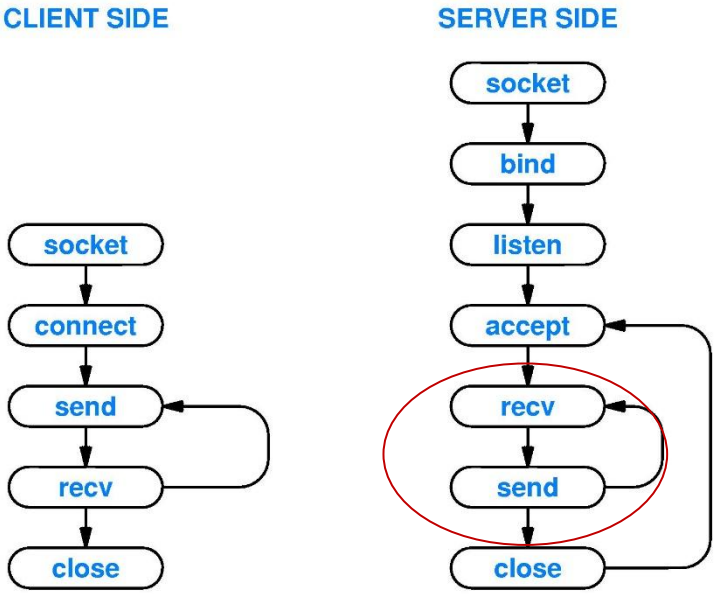
- Schaltet den Socket in den passiven Modus in dem auf eingehende Verbindungen gelauscht wird
- *listen(descriptor, queuesize)*

Socket API: Server (3)



- Verbindungsannahme
- Es wird ein neuer Socket erzeugt, der zur Kommunikation mit dem Client dient → ursprüngliche Socket lauscht weiter auf Verbindungen
- *newsock = accept(descriptor, caddress, caddresslen)*

Socket API: Server (4)



- Requests lesen und mit einem Response antworten

UDP

- Vorheriges Beispiel war für TCP-Sockets

- UDP benötigt keinen Verbindungsaufbau!
  - Der Funktion *sendto* werden die Nachricht und die notwendigen Ziel-Adressen direkt übergeben
  - Empfänger lauscht mit *recvfrom* auf eingehende Nachrichten

Funktionsreferenz

Name	Used By	Meaning
accept	server	Accept an incoming connection
bind	server	Specify IP address and protocol port
close	either	Terminate communication
connect	client	Connect to a remote application
getpeername	server	Obtain client's IP address
getsockopt	server	Obtain current options for a socket
listen	server	Prepare socket for use by a server
recv	either	Receive incoming data or message
recvmsg	either	Receive data (message paradigm)
recvfrom	either	Receive a message and sender's addr.
send	either	Send outgoing data or message
sendmsg	either	Send an outgoing message
sendto	either	Send a message (variant of sendmsg)
setsockopt	either	Change socket options
shutdown	either	Terminate a connection
socket	either	Create a socket for use by above

Anwendungsprotokolle

Einleitung

- Anwendungen, die Daten austauschen wollen, müssen die Art und Weise der Kommunikation festlegen
  - Syntax und Semantik der Nachrichten
  - Client / Server Rollen und Verbindungsaufbau
  - Fehlerbehandlung
  - Verbindungsabbau
- Ergebnis: Application Layer Protokoll

Anwendungstypen

- **Privater Dienst**: Anwendungsprotokoll muss nicht in einem formellen Dokument festgehalten werden
  - „der Quellcode ist die Dokumentation“
  - Dritte Parteien müssen / sollen keine passenden Clients oder Server bauen
- **Standardisierter Dienst**: Andere Parteien sollen in der Lage sein Clients oder eigene Server anhand einer öffentlichen Spezifikation zu bauen (z.B. HTTP, SMTP)
  - Präzise und eindeutige Formulierungen um Interoperabilität zu gewährleisten

Protokollaspekte

- Zwei konzeptionell getrennte Teile des Anwendungsprotokolls:
  - **Data Representation**
    - Beschreibt wie die zu übertragenden Nachrichten als Bytes dargestellt werden
    - Plattformunabhängig → Mittels dieser Beschreibung können die Nachrichten auf jedem System gelesen werden
  - **Data Transfer**
    - Beschreibt welche Nachrichten ausgetauscht werden, in welcher Reihenfolge, und was die Nachrichten für Information beinhalten → **Syntax und Semantik der Nachrichten**
    - **Fehlererkennung, Fehlerkorrektur**
- Komplizierte Dienste trennen dies strikt in zwei Dokumente auf

WWW Technologien

- **HyperText Markup Language (HTML)**
  - Inhalt und Layout einer Website
- **Uniform Resource Locator (URL)**
  - Format und Bedeutung der Adressen von Ressourcen
- **Hypertext Transfer Protokoll (HTTP)**
  - Kommunikation zwischen Browser und Web Server

HTML (1)

- Textuelle Beschreibung des Inhalt und Layouts einer Website
  - Unterstützt Multimedia-Inhalte: Bilder, Videos, …
  - Markup Sprache: Weist Texten und Daten bestimmten Eigenschaften und Darstellungsformen zu
    - Allgemeine Richtlinien für die Darstellung
    - Browser können den Inhalt z.B. an die Displaygröße anpassen

```
<HTML>
  <HEAD>
      <TITLE>
            text that forms the document title
      </TITLE>
  </HEAD>
  <BODY>
       body of the document appears here
  </BODY>
</HTML>
```

HTML (2)

- Textuelle Beschreibung des Inhalt und Layouts einer Website
  - Deklarativ: Beschreibt was dargestellt werden soll und nicht wie
  - Nutzt Hyperlinks (Anchor) um zu anderen Dokumenten zu verweisen → auch Medieninhalte können Links beinhalten
  - Dokumente können Meta-Daten beinhalten

```
<HTML>
  <HEAD>
      <TITLE>
            text that forms the document title
      </TITLE>
  </HEAD>
  <BODY>
       body of the document appears here
  </BODY>
</HTML>
```

HTML (3)

- HTML-Dokumente werden aus verschachtelten Tags zusammen gestellt:
  - Meist öffnender Tag <tag> und schließender Tag </tag>
  - HEAD enthält Meta-Informationen und BODY den eigentlichen Seiteninhalt

- Anchor-Tags enthalten die Hyperlinks:
  <a href="index.html">

```
<HTML>
  <HEAD>
      <TITLE>
            text that forms the document title
      </TITLE>
  </HEAD>
  <BODY>
       body of the document appears here
  </BODY>
</HTML>
```

Uniform Resource Locators (1)

- **Adresse eines Objekts im Internet**
- Allgemeines Format:
  - **protocol://computer_name:port/document_name?parameters**
  - Z.B. http://www.uni-leipzig.de:80/index.html?stundenplan
- Bestandteile teilweise optional (implizite Default Werte)
  - Z.B. www.uni-leipzig.de/index.html?stundenplan → Port 80 und Protokoll HTTP „vermutet“ bzw. HTTPS und Port 443

Uniform Resource Locators (2)

- Browser nutzt *computer_name und port* um den richtigen Server zu identifizieren
- *protocol* spezifiziert mit welchem Protokoll der Browser mit dem Server sprechen wird (z.B. HTTP, HTTPS, FTP)
- *document_name* und *parameters* werden vom Server ausgewertet und identifizieren das gewünschte Objekt

HTTP (1)

- Kommunikation zwischen Browser und Web Server
  - Browser sendet **HTTP Request** zum Server
    - **HTTP Request Header**, eine leere Zeile + eventuell Daten
  - Server antworten mit einem **HTTP Response**
    - **HTTP Response Header**, eine leere Zeile + angeforderte Daten
- Kontrollnachrichten **(HTTP Header)** bestehen aus Text
  - Zeilenbasiert
  - (Fast) jede Zeile ein Headerfeld: <Feldname>: <Feldwert>
- Dateien werden binär übertragen
- Download und Upload wird unterstützt

HTTP: Methoden

- Mehrere HTTP Request Methoden
  - GET: Abruf eines Dokumentes
  - HEAD: Abruf von Statusinformationen zu einem Dokument
    - Server liefert den gleichen HTTP Header wie bei GET, aber nicht das eigentliche Dokument
  - POST: Übermittlung von Daten zum Server
  - PUT: Ersetzt Daten auf dem Server
- Im Request Header in der ersten Zeile angegeben

HTTP (3)

- Erste Zeile im HTTP Response Header ist ein Statuscode → Informationen über den Erfolg der Anfrage

- Beispiele:
  - 200: OK
  - 400: Syntax-Fehler in der Anfrage
  - 404: Dokument existiert nicht

```
HTTP/1.1 200 OK
Date: Sat,  1 Aug 2013 10:30:17 GMT
Server: Apache/1.3.37 (Unix)
Last-Modified: Thu, 15 Mar 2012 07:35:25 GMT
ETag: "78595-81-3883bbe9"
Accept-Ranges: bytes
Content-Length: 16
Connection: close
Content-Type: text/plain

This is a test.
```

HTTP (4)

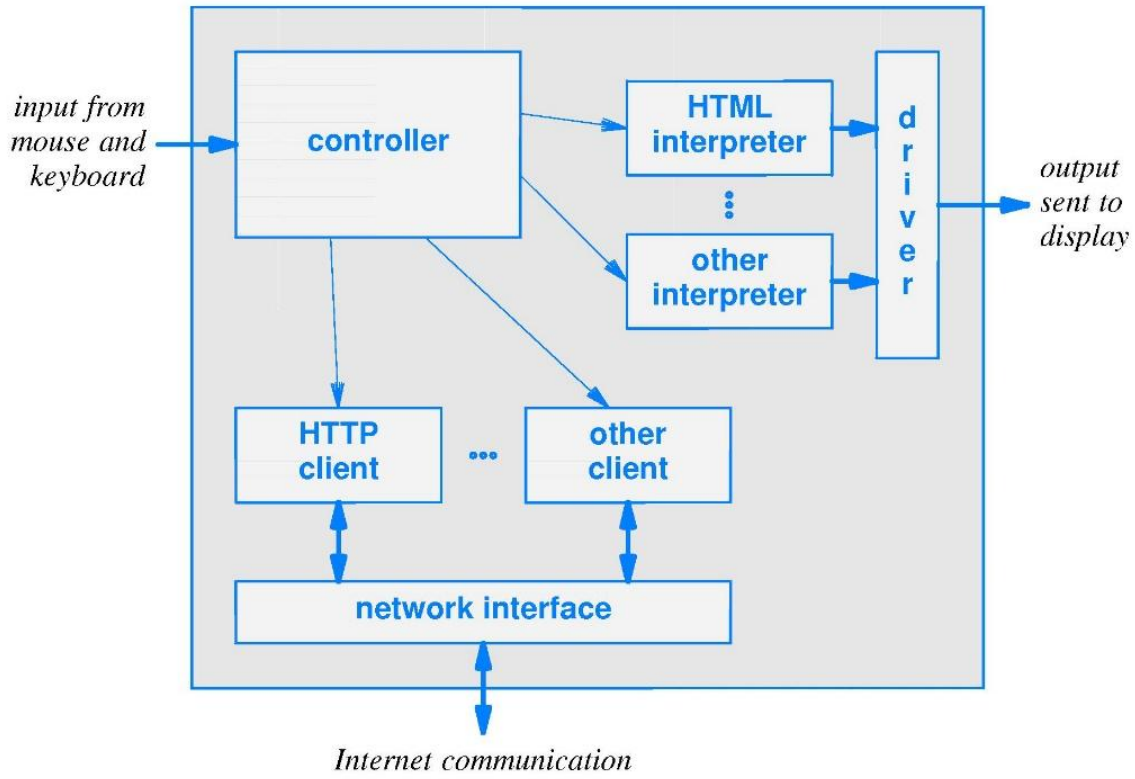
- **Lokales Caching mittels GET, HEAD und dem Last-Modified Header Feld**

- Viele Objekte beim Besuch einer Website immer wieder benötigt: Bilder, Sounds, CSS, JS, … → **selten geändert**

- Einmal geladen (mit GET) kann mittels HEAD und dem empfangenen HTTP Reponse Header der Zeitpunkt der letzten Bearbeitung bestimmt werden

  - **Pro Objekt muss nur der HTTP Response Header übertragen werden, NICHT das ganze Objekt**

Browser-Architektur



File Transfer Protocol (1)

- Computer im „Internet“ sind sehr heterogen, da mehrere
  - Rechnerarchitekturen
  - Betriebssysteme
  - Interpretation von Dateiendungen (\n vs. \r\n)
  - Darstellungen von Zeilenenden in Textdateien
  - …
- **Dateiaustausch im Internet ist komplex → FTP behandelt all diese Probleme und ermöglicht den Austausch von beliebigen Daten zwischen beliebigen Computern**

File Transfer Protocol (2)

- Eigenschaften von FTP
  - Beliebige Dateien austauschen
  - Upload und Download
  - Authentifizierung und Rechteverwaltung
  - Verzeichnisse
  - Kontrollnachrichten sind Text-basiert
  - Versteckt die Heterogenität der verschiedenen Betriebssysteme und Technologien

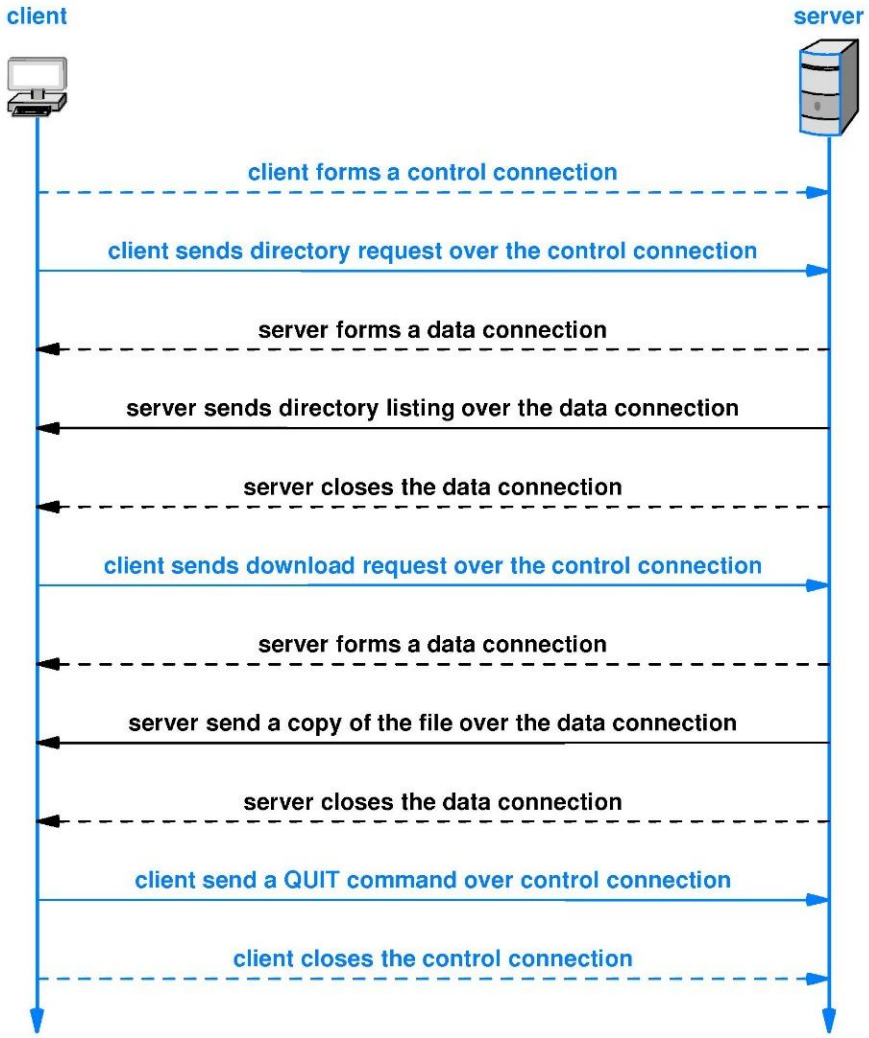
File Transfer Protocol (3)

- FTP nutzt zwei Kommunikationskanäle

1. **Control Connection**
   - Authentifizierung
   - Anforderung eines File Listing
   - Anforderung eines Upload / Download
   - Client kontaktiert Server

2. **Data Connection**
   - Genutzt für den wirklichen Austausch von Daten
   - Eigentliche FTP-Client öffnet einen Listening-Socket zu dem der FTP-Server sich verbindet
   - FTP-Client teilt Server über Control Connection den offenen Port mit

File Transfer Protocol (4)

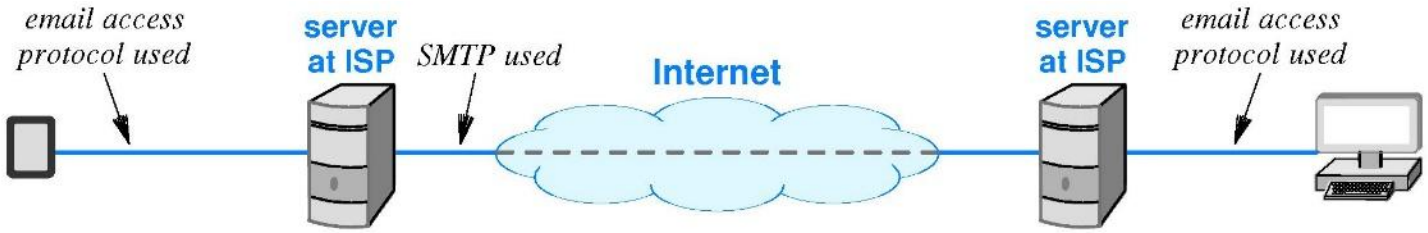


E-Mail

- Ähnlich zu WWW gibt es mehrere Standards die im Zusammenspiel den Dienst E-Mail ermöglichen
  - **Simple Mail Transfer Protocol (SMTP)**
    - Transfer von E-Mails über das Internet zur Mailbox des Empfängers
  - **POP3 / IMAP**
    - Remote-Zugriff auf die Mailbox des Nutzers
  - **Email Representation Standards (RFC2822 und MIME)**
    - Encodierung (Binärdarstellung) der Nachrichten

E-Mail Komponenten

- Nutzer besitzt Mailbox bei seinem Mailanbieter
  - E-Mail-Versand: E-Mail Client des Nutzers spricht SMTP mit Server des Mailanbieter, dieser SMTP mit dem Server des Empfängers
  - E-Mail lesen: E-Mail Client spricht POP3 oder IMAP mit dem Server des Mailanbieter



SMTP

- **Kann nur Text übertragen: sowohl Kontroll-Nachrichten als Nutzdaten**
- Server-Nachrichten beginnen mit einem numerischen Code
- **Beliebige Dateien mit MIME im Body der E-Mail encodiert**

```
Server:  220 somewhere.com Simple Mail Transfer Service Ready
Client:  HELO example.edu
Server:  250 OK
Client:  MAIL FROM:<John_Q_Smith@example.edu>
Server:  250 OK
Client:  RCPT TO:<Matthew_Doe@somewhere.com>
Server:  550 No such user here
Client:  RCPT TO:<Paul_Jones@somewhere.com>
Server:  250 OK
Client:  DATA
Server:  354 Start mail input; end with <CR><LF>.<CR><LF>
Client:  ...sends body of mail message, which can contain
Client:  ...arbitrarily many lines of text
Client:  <CR><LF>.<CR><LF>
Server:  250 OK
Client:  QUIT
Server:  221 somewhere.com closing transmission channel
```

Domain Name System (1)

- DNS weist IP Adressen lesbare, hierarchische Namen zu → **Domains**
  - Einzelne Label durch „.“ voneinander getrennt
  - Z.B. www.uni-leipzig.de

- **Namensauflösung: Verfahren um Domains zu IP-Adressen umzuwandeln**

- **Domains werden von rechts nach links aufgelöst**
  - **Leere Domain „.“ ist die Root Zone, 1. Label die Top-Level Domain (TLD; z.B. de oder com)**
  - **Jedem Label sind DNS-Server zugeordnet, die mitteilen können, welche DNS-Server für das nächste (linke) Label zuständig sind**

Domain Name System (2)

- **DNS-Verwaltung ist dezentral**
  - Unternehmen / natürliche Person bewirbt sich für eine bestimmte Domain, z.B. uni-leipzig.de, bei der zuständigen Registrierungsstelle, z.B. DENIC für *.de
    - Muss mindestens zwei DNS-Server bereitstellen, die die Namensauflösung für *.uni-leipzig.de durchführen
  - Uni Leipzig kann wiederum weitere Sub-Domains erstellen und diesen anderen zur Verwaltung überlassen
    - Z.B. *.informatik.uni-leipzig.de
- Der DNS-Server, der schlussendlich für das linkeste Label zuständig ist wird als **authoritative** bezeichnet

Domain Name System (3)

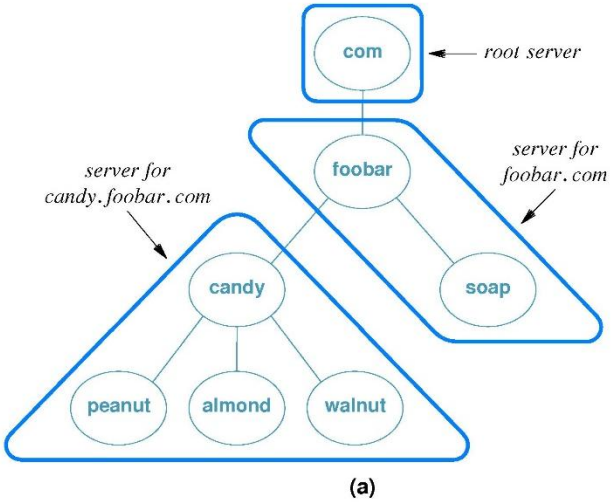
- **Namesauflösung verläuft iterativ**

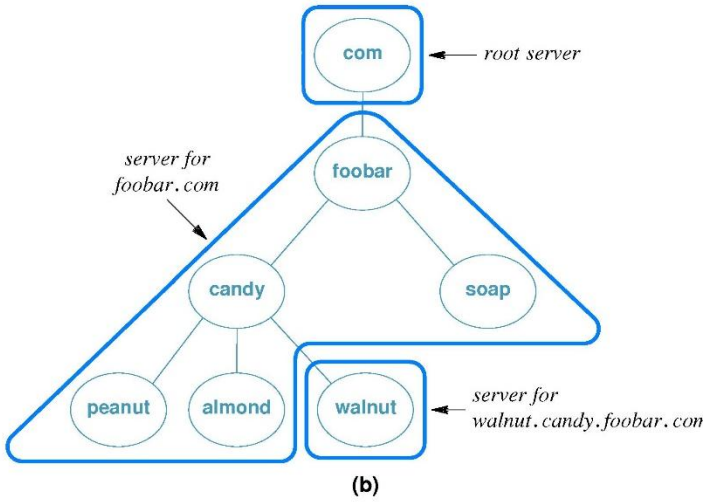
- Beispiel rvs.informatik.uni-leipzig.de
  1. DNS Resolver fragt zunächst die DNS-Server der Root Zone wer für die TLD „de“ zuständig ist
  2. DNS Server von „de“ werden nach „uni-leipzig.de“ gefragt
  3. DNS Server von „uni-leipzig.de“ werden nach DNS-Server für „informatik.uni-leipzig.de“ gefragt
  4. …

- DNS-Einträge werden gecached: im Router, beim ISP

Domain Name System (4)

- DNS-Server können auch für Subdomains einer Domain zuständig sein
  - Flexibel konfigurierbar





Domain Name System (5)

- Es gibt verschiedene Arten von DNS-Einträgen
- A-Records: IP4 Adressen
- AAAA-Records: IPv6 Adressen
- MX-Records: Domain des SMTP-Servers der Domain
- CNAME: Alias

Zusammenfassung

- Für das WWW werden mindestens drei Technologien genutzt: HTTP, HTML und URL
- FTP dient dem Datenaustausch in heterogenen Netzen
- Für E-Mail werden mindestens drei Technologien genutzt: SMTP, IMAP/POP3 und MIME
- DNS ermöglicht die Zuweisung von lesbaren Namen zu IP-Adressen