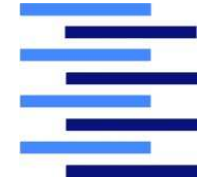


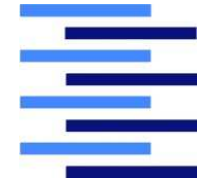
Kapitel 5

Anwendungsschicht



1. Prinzipien von Protokollen der Anwendungsschicht
2. Socket-Programmierung
3. Das World Wide Web (HTTP)
4. Domain Name System (DNS)

Anwendungen und Anwendungsschicht-Protokolle

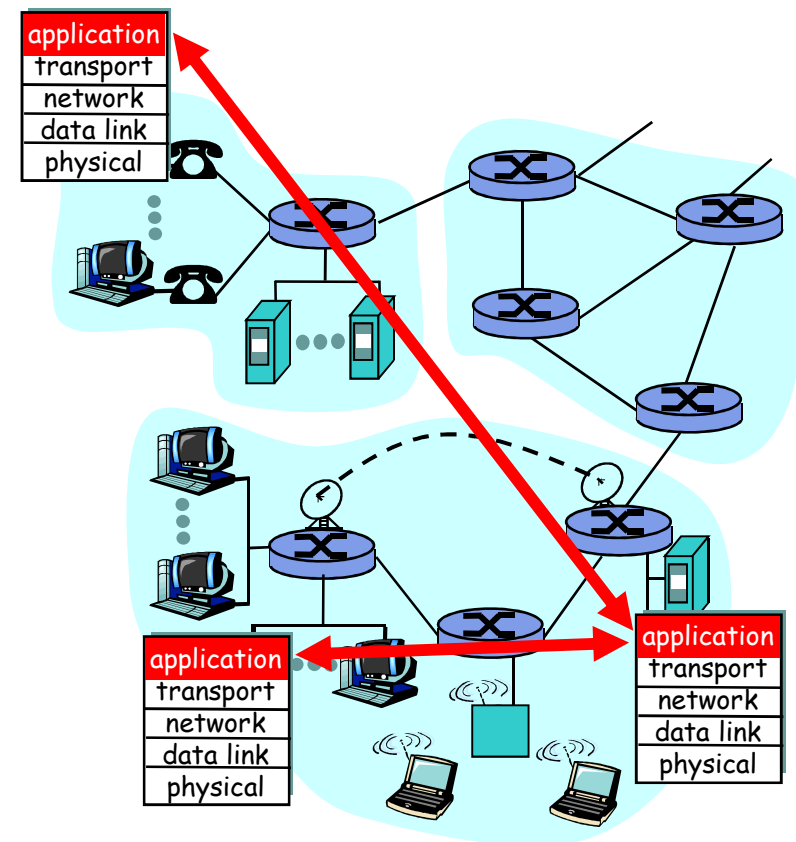


Anwendungen: kommunizierende, verteilte Prozesse

- laufen auf Endsystemen als Benutzermodus-Prozess
- tauschen Nachrichten aus, um eine verteilte Anwendung zu implementieren (→ “verteiltes System”)

Anwendungsschicht-Protokolle:

- sind ein Teil einer Anwendung
- definieren Nachrichtenformate und Aktionen
- benutzen Kommunikationsdienste der unteren Schichten (TCP, UDP)



Das Client-Server Prinzip

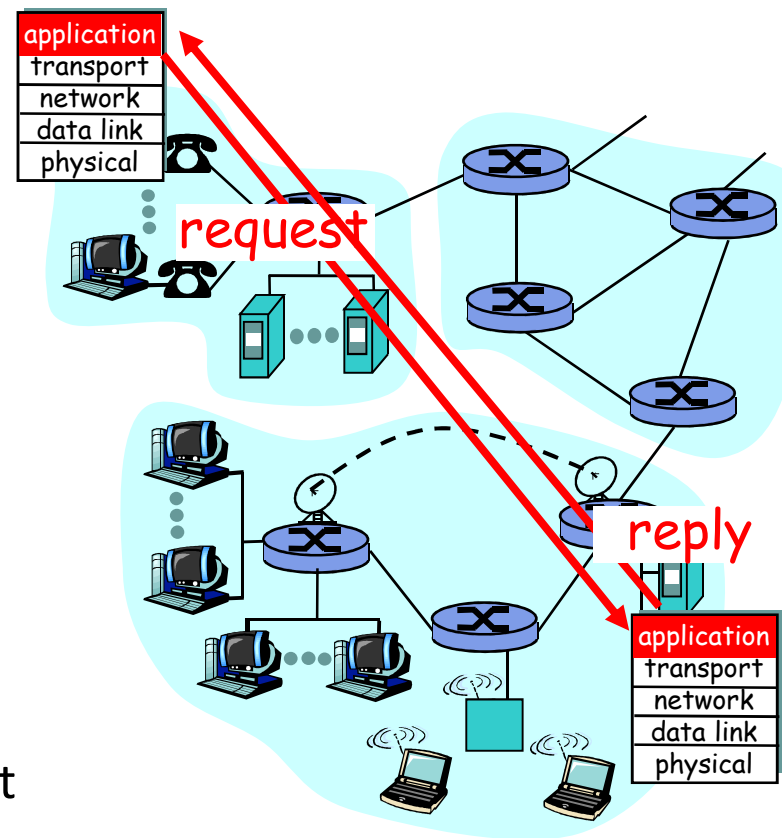
Typische Netzerkanwendungen
bestehen aus zwei Teilen:
Client und *Server*

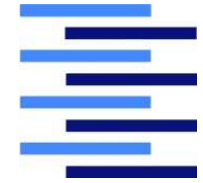
Client:

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

Server:

- liefert den verlangten Dienst zum Client
- *Bsp.: Web-Server sendet die gewünschte Web-Seite, Mail-Server liefert E-Mail, ...*





Anwendungsschicht: Wie wird die Interprozess-Kommunikation realisiert?

API: Application Programming Interface

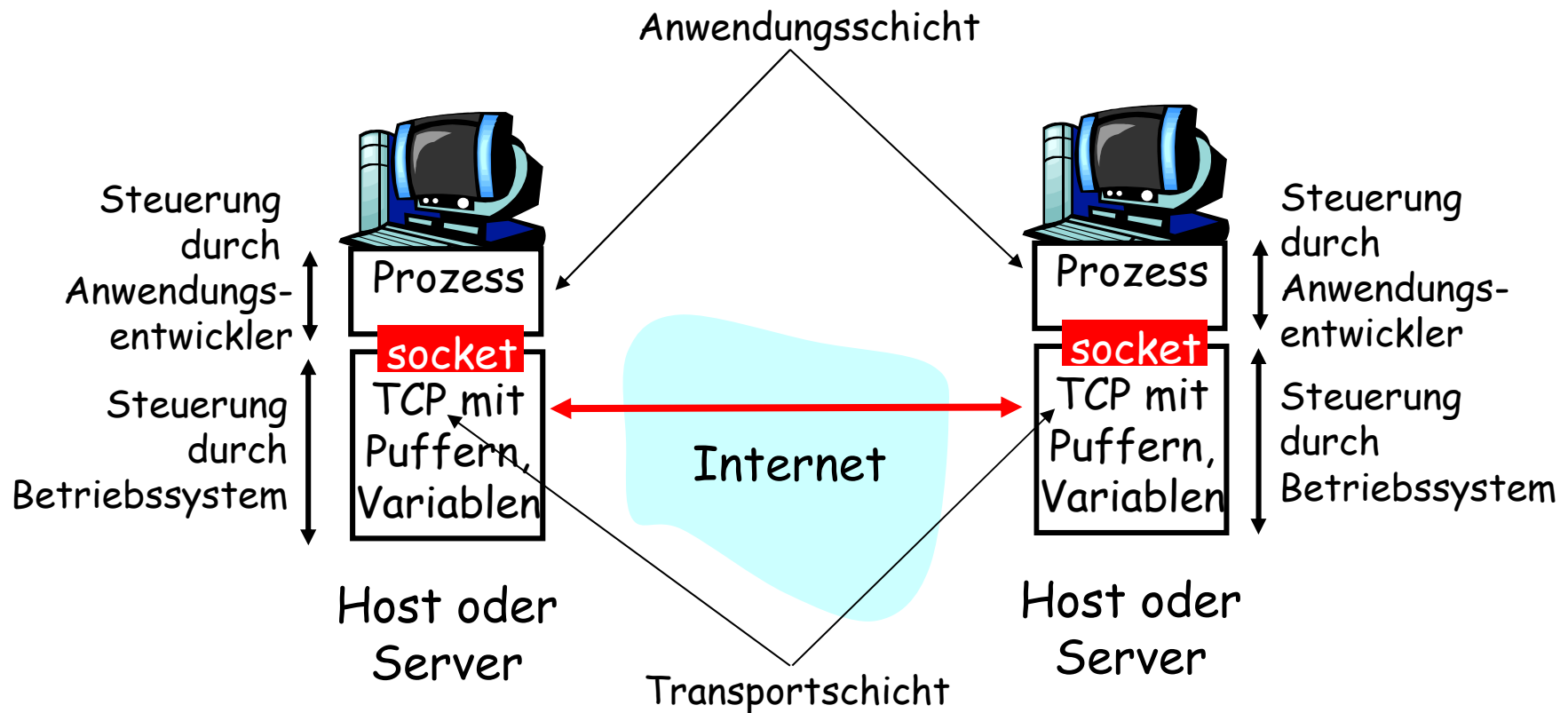
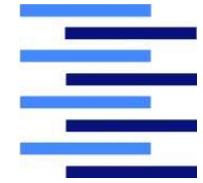
- Definiert eine Schnittstelle zwischen der Anwendungsschicht und der Transport-Schicht (*“Service Access Point”*)
- Internet API: “Socket”
 - Zwei Prozesse kommunizieren durch Senden von Daten ins Socket und Lesen von Daten aus dem Socket

Wie identifiziert ein Anwendungsprozess den Partnerprozess auf dem anderen Rechner?

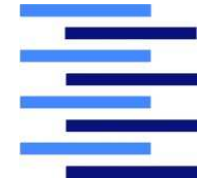
1. **IP-Adresse** des entfernten Rechners (weltweit eindeutig)
2. **“Portnummer”** - Information für den empfangenden Rechner, an welchen lokalen Prozess die Nachricht weitergeleitet werden soll.

... lots more on this later.

Interprozess-Kommunikation über Sockets



Statt TCP kann auch UDP verwendet werden!



Welchen Transportdienst braucht die Anwendung ?

Datenverlust

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

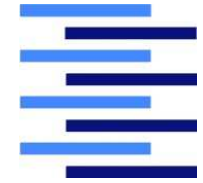
Zeitanforderungen

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

Bandbreite (Übertragungsrate)

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

Qual der Wahl für Anwendungsprogrammierer: TCP oder UDP?

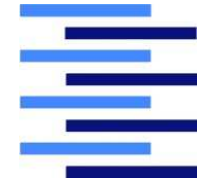


TCP – Dienste:

- Zuverlässige, reihenfolge-erhaltende Übertragung eines Byte-Stroms
- Flusskontrolle
(Überlaststvermeidung des Empfängers)
- Staukontrolle
(Überlaststvermeidung des Netzwerks)
- *Nicht geboten:*
Garantien über Verzögerung und Übertragungskapazität

UDP - Dienste:

- Unzuverlässiger Datentransfer einzelner Datenpakete
- geringer Overhead
- *Nicht geboten:*
 - Verbindungsaufbau
 - Flusskontrolle
 - Staukontrolle
 - Garantien über Verzögerung und Übertragungskapazität

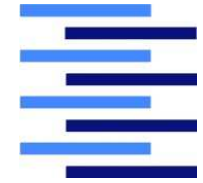


Anwendungen, Anwendungsschicht-Protokoll und Transportprotokoll

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 821]	TCP
remote terminal access	TELNET[RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	RTP[RFC 3550]	UDP
remote file server	NFS[RFC 3530]	TCP or UDP
Internet telephony	SIP[RFC 3261] + RTP	TCP + UDP

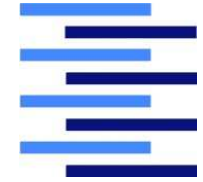
Kapitel 5

Anwendungsschicht



1. Prinzipien von Protokollen der Anwendungsschicht
2. **Socket-Programmierung**
3. Das World Wide Web (HTTP)
4. Domain Name System (DNS)

Socket Programmierung



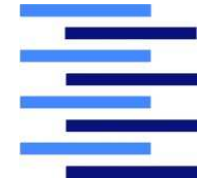
Ziel: Lerne, wie man Client/Server Applikationen entwickelt, die über Sockets kommunizieren

Socket API

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

socket

Eine lokale **Schnittstelle** als "Tür" oder "Steckdose", durch die ein Anwendungsprogramm Nachrichten zu/von einem anderen Prozess (lokal oder Remote) sowohl senden als auch empfangen kann



Socket Programmierung *mit TCP*

Client muss den Server kontaktieren

- Der Server-Prozess muss als Erster laufen
- Der Server muss ein Socket erzeugt haben, das den Client-Kontakt bemerken kann

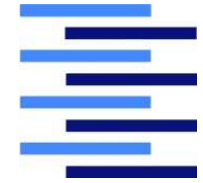
Client kontaktiert den Server durch:

- Erzeugung eines TCP- Sockets
- Spezifizierung der IP-Adresse + Port-Nummer des Server- Prozesses

- Nach Erzeugung des **Client-Socket**: Client-TCP baut Verbindung zum Server-TCP auf (3-Wege-Handshake)
- Wenn durch den Client kontaktiert, erzeugt das Server-TCP einen neuen "Arbeits"-Socket, mit dem danach Server und Client kommunizieren (*→ eigener Thread!*)
 - Erlaubt dem Server, gleichzeitig mit vielen Clients zu kommunizieren

Sicht der Anwendung

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

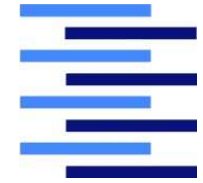


JAVA: Socket-Methoden (TCP)

Klasse Socket im Package java.net:

Die Daten werden über einen Stream versendet / empfangen!


- **Konstruktor: public Socket (String host, int port)**
 - Creates a stream socket and connects it to the specified port number on the named host
- **Konstruktor: public Socket (InetAddress address, int port)**
 - Creates a stream socket and connects it to the specified port number at the specified IP address
- **public InputStream getInputStream()**
 - Returns an input stream for this socket.
- **public OutputStream getOutputStream()**
 - Returns an output stream for this socket.
- **public void close()**



JAVA: ServerSocket-Methoden (TCP)

*Klasse ServerSocket im Package java.net:
Zusätzliche Methode für das "Lauschen"*

- *Konstruktor:* `public ServerSocket (int port)`
 - Creates a server socket, bound to the specified port.
- `public Socket accept()`
 - Listens for a connection to be made to this socket and accepts it. *The method blocks until a connection is made.* A new Socket is created for that connection and returned.
- `public void close()`

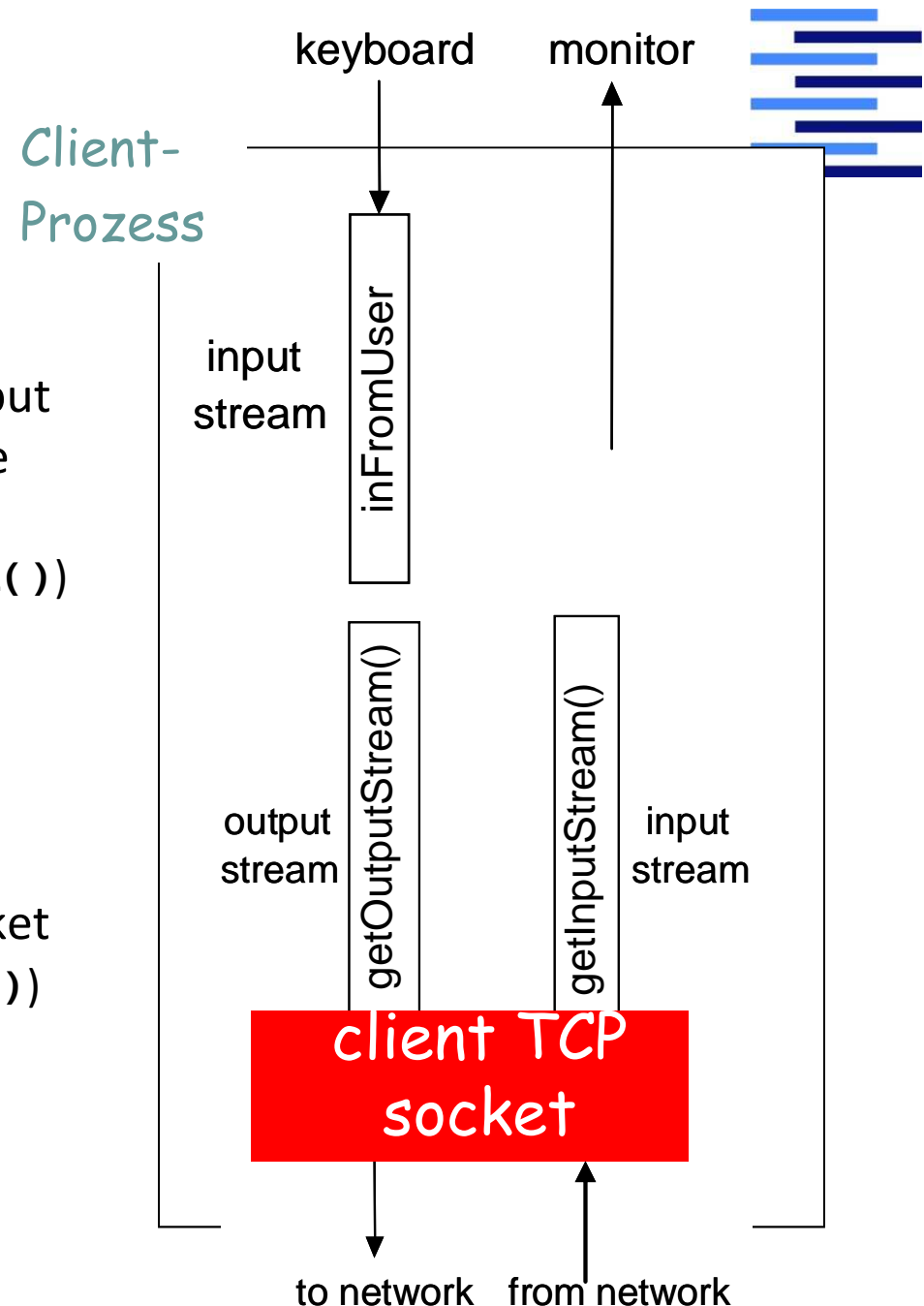


Standard-Socket!

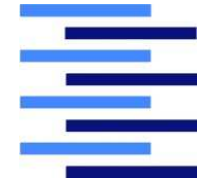
Socket Programmierung mit TCP

Beispiel: Client-Server Anwendung mit JAVA:

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



Client/Server Socket Interaction: TCP



Server (running on `hostid`)

Client

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

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

read request from
`connectionSocket.getInputStream()`

write reply to
`connectionSocket.getOutputStream()`

close
`connectionSocket.close()`

TCP
connection setup

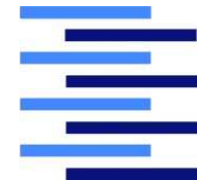
create socket,
connect to `hostid`, port=`x`
`clientSocket =`
`Socket(hostid,x)`

send request using
`clientSocket.getOutputStream()`

read reply from
`clientSocket.getInputStream()`

close
`clientSocket.close()`

RB SS



JAVA: TLS (SSL) - Sockets

Klassen **SSLSocket** und **SSLSocketFactory** in `javax.net.ssl`

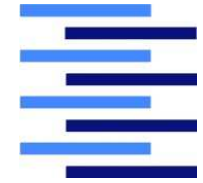
- SSLSockets verwendbar wie (TCP-) Sockets
- Vor Weitergabe (an TCP) werden die Daten verschlüsselt und gegen Veränderung geschützt
- Erzeugung über SSLSocketFactory, da verschiedene Provider-abhängige Implementierungen möglich sind
- Beispiel:

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

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


Kapitel 5

Anwendungsschicht

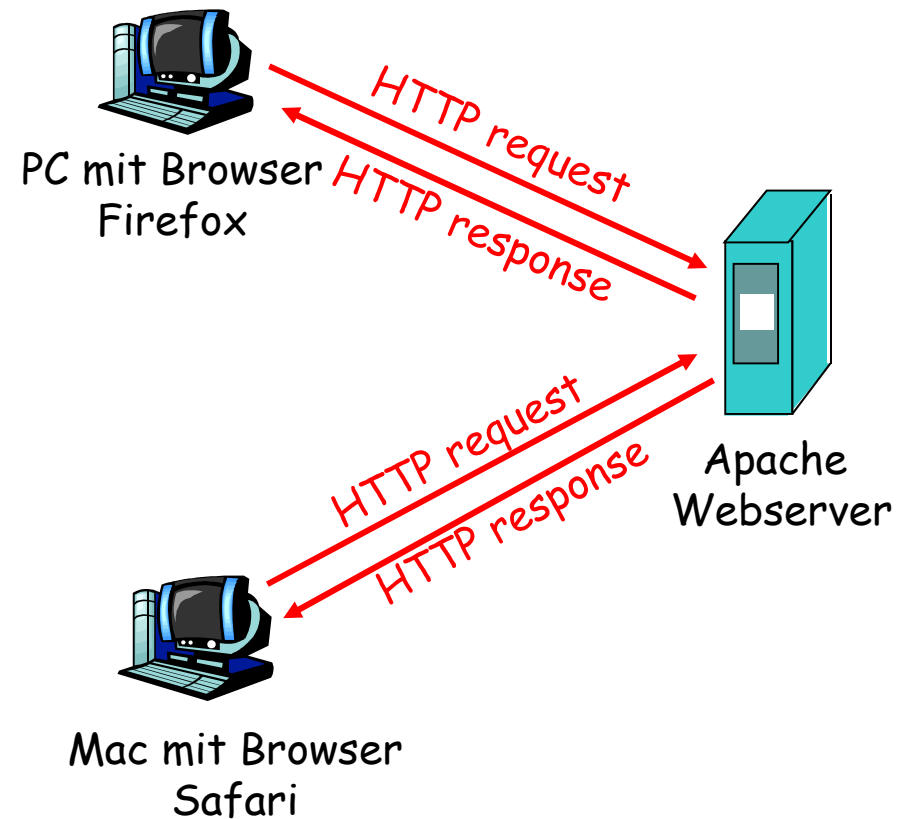


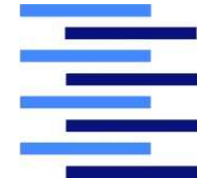
1. Prinzipien von Protokollen der Anwendungsschicht
2. Socket-Programmierung
3. Das World Wide Web (HTTP)
4. Domain Name System (DNS)

Das HTTP-Protokoll

HTTP: Hypertext Transfer Protokoll

- Grundlage: Client-/Server-Prinzip
 - *Client*: “Browser” -Anforderung, Interpretation und Anzeige von Web-Objekten
 - *Server*: Web-Server – sendet Web-Objekte als Antwort auf eine Anfrage
- HTTP/1.0: RFC 1945
 - *nur noch selten verwendet*
- HTTP/1.1: RFC 2616
 - *aktueller Standard*
- HTTP/2.0: RFC 7540
 - *neu (Mai 2015!), hauptsächlich Performance-Optimierungen bzgl. HTTP/1.1*





Eigenschaften von HTTP

HTTP benutzt TCP als Transport-Protokoll:

- Der Client ermittelt aus der übergebenen URL den Rechnernamen des Servers und erfragt über DNS die zugehörige IP-Adresse
- Der Client initiiert eine TCP-Verbindung zum Server mit Portnummer 80 (erzeugt Socket) *[bei TLS (SSL)-Verschlüsselung Port 443]*
- Der Server akzeptiert die TCP-Verbindung des Client
- HTTP-Nachrichten (Anwendungsschicht-Messages) werden zwischen Browser und Web Server ausgetauscht (ASCII-Klartext)
- Anschließend wird die TCP-Verbindung geschlossen

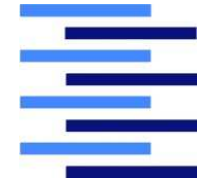
HTTP ist “zustandslos”

- Der Server speichert keine Informationen über vergangene Anfragen

nebenbei

Protokolle, die Zustände speichern, sind komplex!

- Die Vergangenheit muss protokolliert und verwaltet werden.
- Wenn beim Server oder Client Fehler auftreten (Absturz), können Zustände leicht inkonsistent werden!

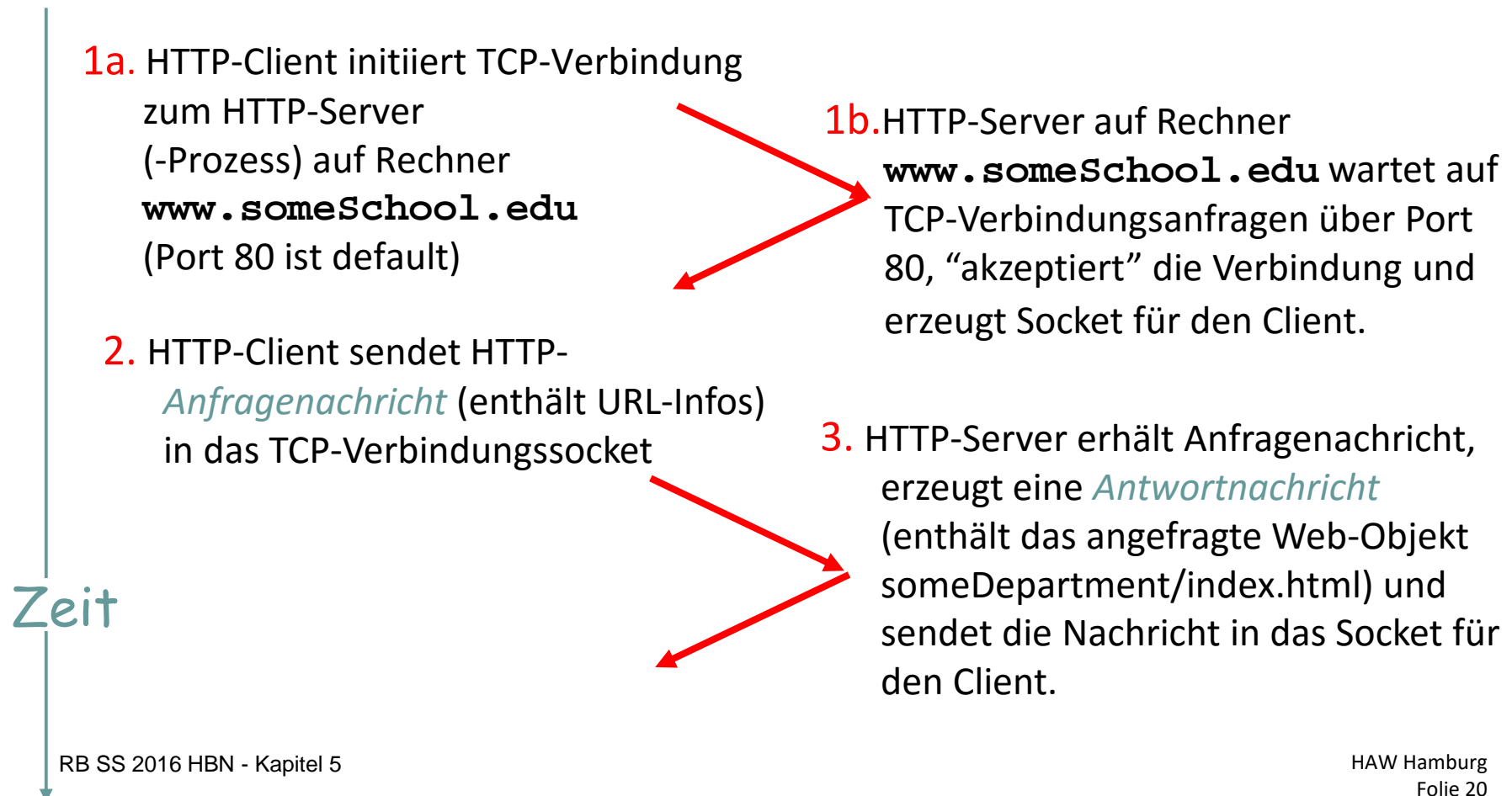


HTTP 1.0: Beispiel

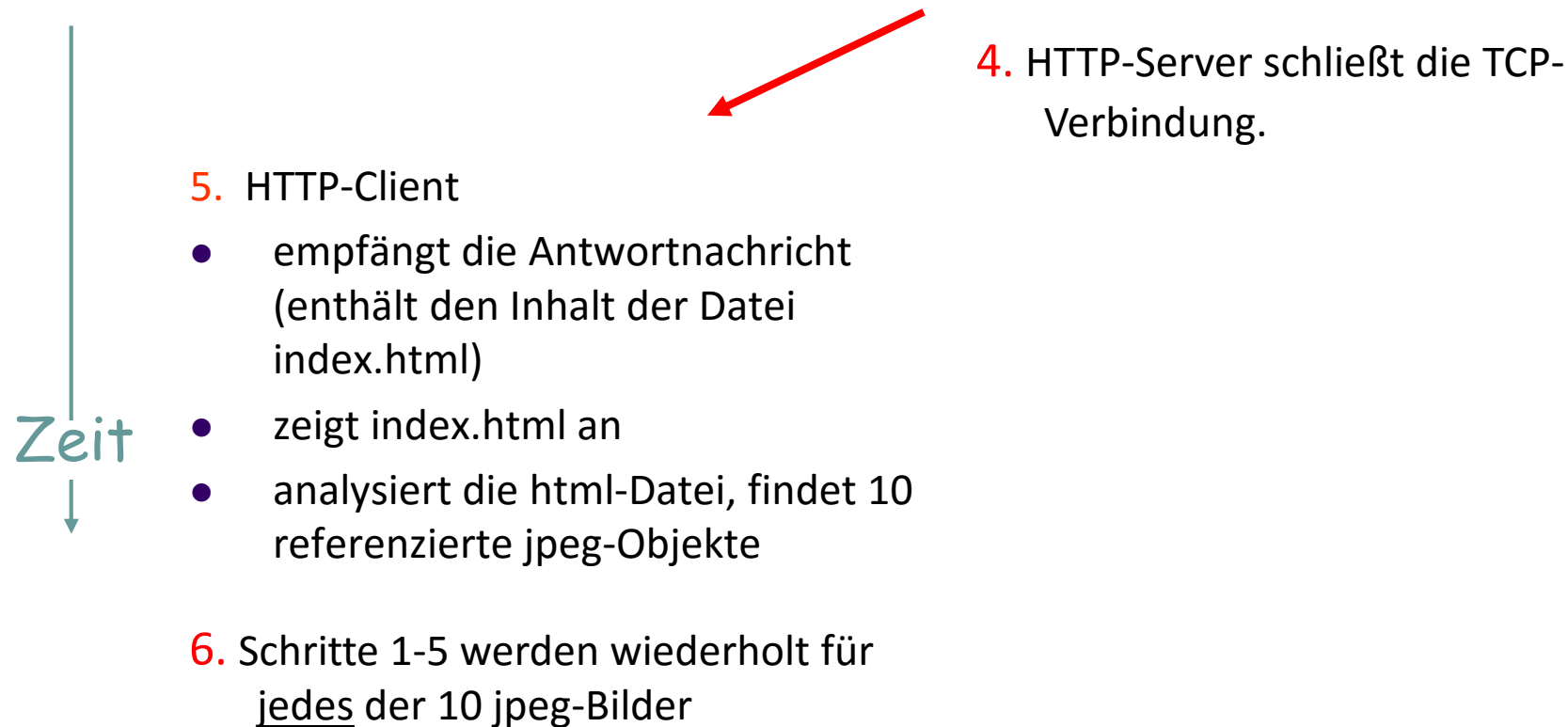
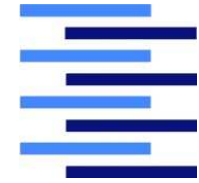
Benutzer gibt URL ein

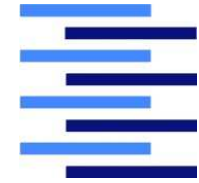
<http://www.someSchool.edu/someDepartment/index.html>

(enthält Text und Referenzen zu 10 jpeg-Bildern)



HTTP 1.0: Beispiel (Forts.)





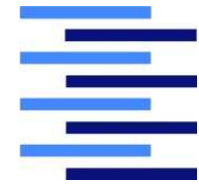
Nicht-persistente und persistente Verbindungen

Nicht-persistent

- HTTP/1.0: nach jedem Anfrage / Antworttausch wird die TCP-Verbindung geschlossen!
 - 2 RTT (Round Trip Time) nötig, um ein Objekt zu holen:
 - TCP-Verbindungsaufbau
 - Anfrage / Antwortübermittlung
 - TCP-"Slow Start"-Phase trifft jedes Objekt!
- ➔ Viele Browser öffnen mehrere TCP-Verbindungen parallel

Persistent

- default für HTTP/1.1 + HTTP/2.0
- mehrere Anfrage / Antwort – Zyklen über dieselbe TCP-Verbindung möglich
- HTTP/1.1: Pipelining
Der Client kann Anfragen über die TCP-Verbindung sofort senden, ohne auf die Antwort einer früheren Anfrage zu warten (*allerdings: feste Reihenfolge der Antworten!*)
- HTTP/2.0: mit mehreren unabhängigen "Streams" innerhalb derselben TCP-Verbindung (*Vermeidung des "Head-of-Line-Blocking"-Problems, d.h. kein Warten auf vorherige lange Antworten nötig*)



HTTP-Nachrichtenformat: Beispiel-Anfrage

Anfragezeile

GET /someDepartment/index.html HTTP/1.0

Header-
Zeilen

Host: www.someSchool.edu

User-agent: Mozilla/4.0

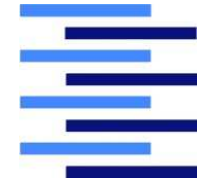
Accept: text/html, image/gif, image/jpeg

Accept-language: fr

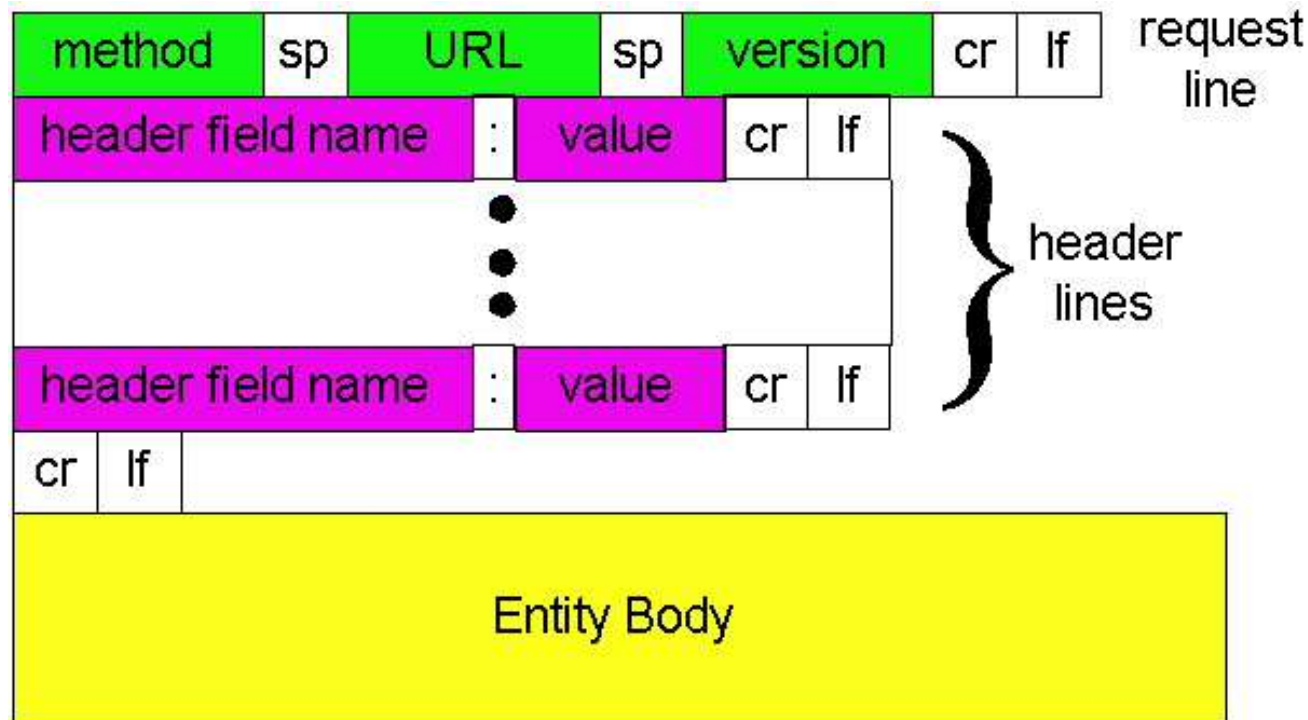
Carriage return +
line feed

(Leerzeile) zeigt
Headerende an

(extra carriage return, line feed)



HTTP-Anfragenachricht: Generelles Format



HTTP 1.0: method = GET|HEAD|POST

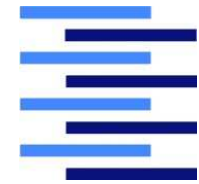
URL = HTTP://*host* [:*port*] *abs_path* für Browser / Proxies

URL = *abs_path* bei direkter Anfrage

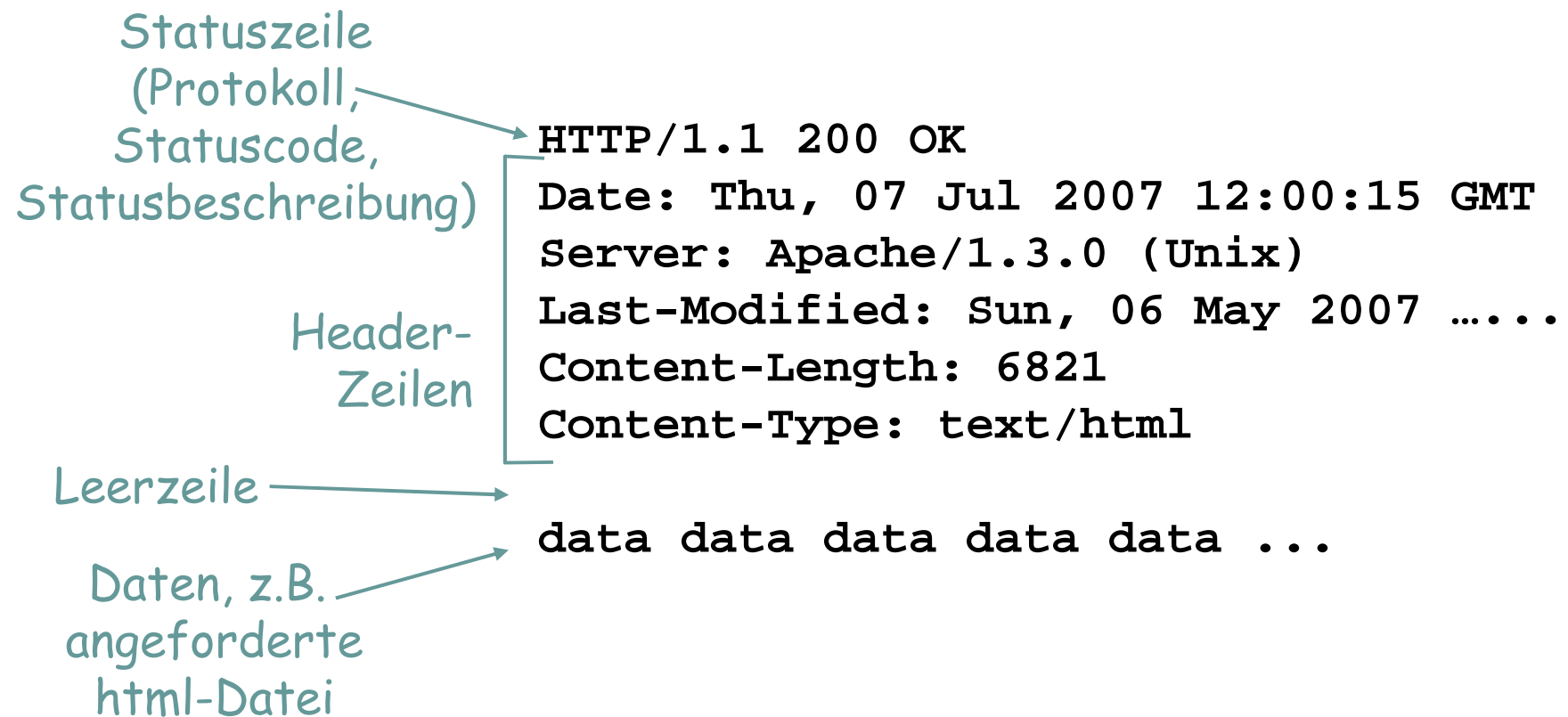
HTTP 1.1: method = GET|HEAD|POST|PUT|DELETE|TRACE|CONNECT

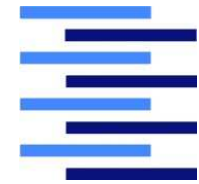
URL = HTTP://*host* [:*port*] [*abs_path* [?*query*]] für Browser / Proxies

URL = *abs_path* [?*query*] bei direkter Anfrage



HTTP-Nachrichtenformat: Beispiel-Antwort





HTTP Antwort - Statuscodes

Beispiele:

200 OK

- Anfrage war erfolgreich, Objektdaten kommen im Datenbereich der Antwortnachricht

301 Moved Permanently

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

400 Bad Request

- Anfragenachricht konnte vom Server nicht interpretiert werden

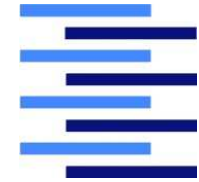
404 Not Found

- Angeforderte Datei auf dem Server nicht vorhanden

505 HTTP Version Not Supported

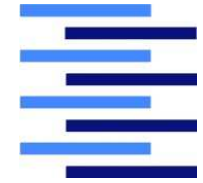
- Die verwendete HTTP-Version wird vom Server nicht unterstützt

Übersicht HTTP-Nachrichtenheader



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

Anleitung zum Selbsttest ...



```
C:> ncat www.informatik.haw-hamburg.de 80  
GET /index.html HTTP/1.0 <CR> <CR>
```

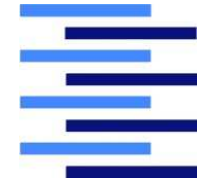
Anfrage

```
HTTP/1.1 200 OK  
Date: Wed, 02 Sep 2015 13:29:46 GMT  
Server: Apache/2.2.3 (Linux/SUSE)  
Last-Modified: Fri, 28 Jan 2011 09:45:37 GMT  
Accept-Ranges: bytes  
Content-Length: 6372  
Connection: close  
Content-Type: text/html  
<html>  
... Inhalt  
</html>
```

Antwort

Tools zum Herstellen einer interaktiven TCP-Verbindung:

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

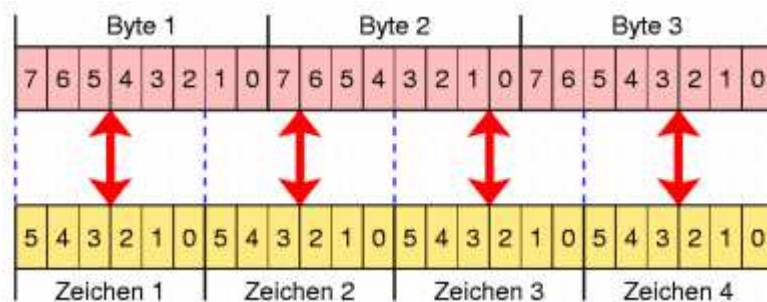


Benutzer-Identifizierung: Authentifikation

Authentifikation:

Zugriffsschutz für
Serverinhalte

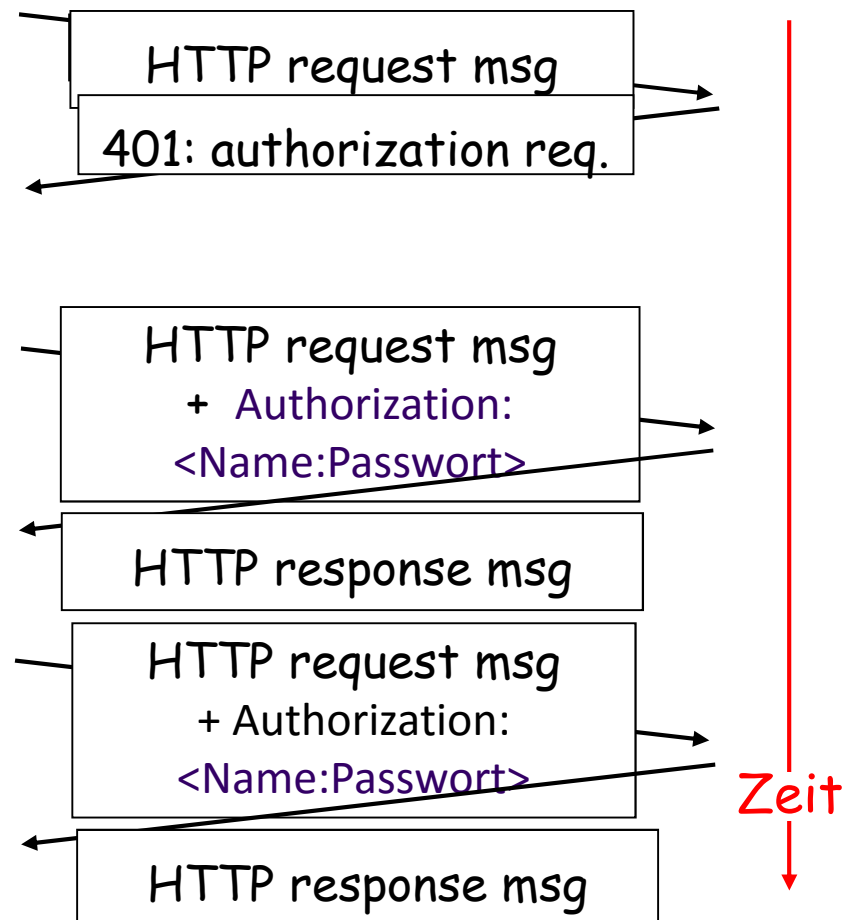
- Basis-Authentifikation
durch Name und Passwort
(**base64-codiert**)

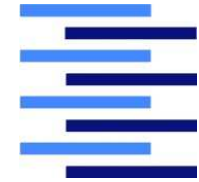


- **zustandslos**: der Client
muss die Autorisations-
informationen in jeder
Anfrage angeben

Client

Server

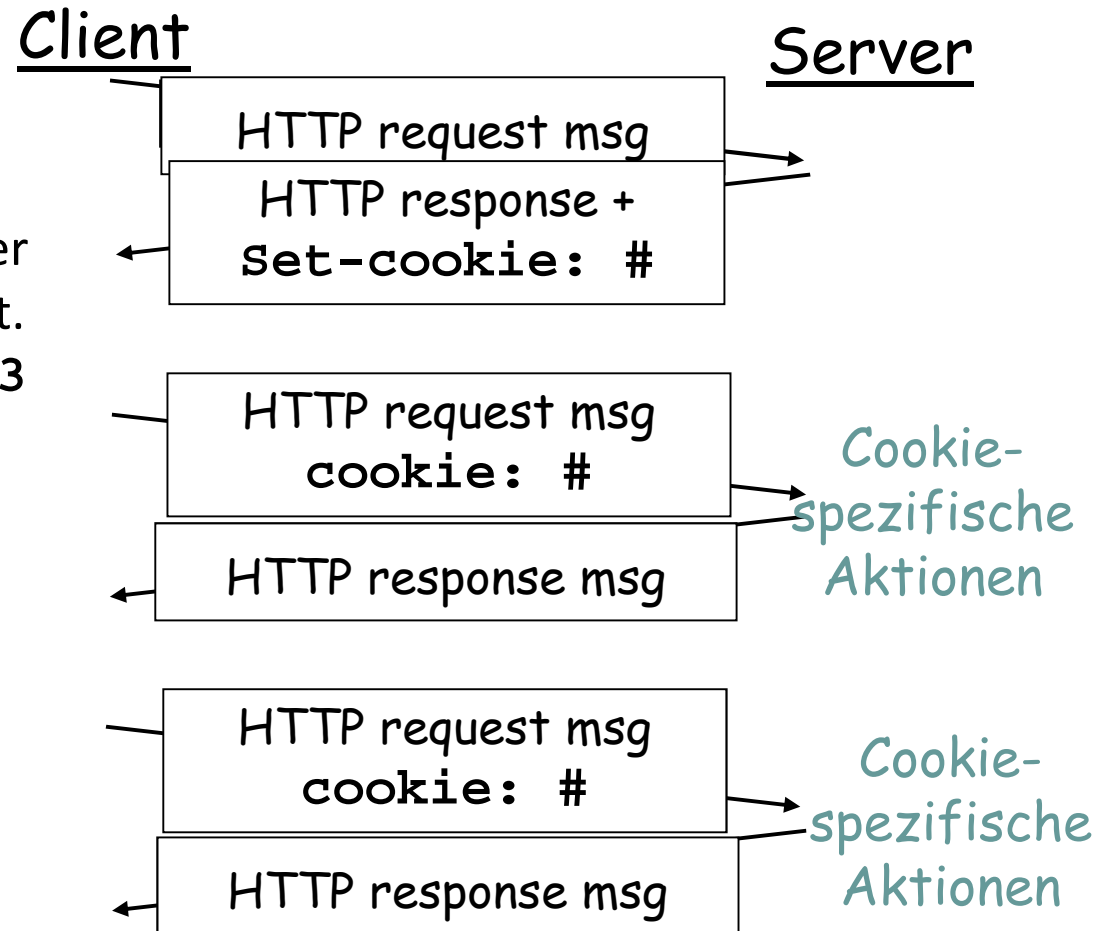


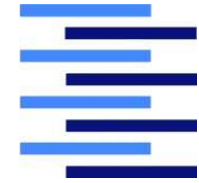


Benutzer-Identifizierung: Cookies [RFC 2109]

Zustandsinformation wird auf dem Client in einer Datei gespeichert!

- Server erzeugt Identifizierungsstring, z.B. **ID1678453**
- Server sendet "Cookie" in der Antwortnachricht zum Client. **Set-cookie: ID1678453**
- Client speichert Cookie in lokaler Datei und gibt den Wert bei späteren Anfragen an den Server mit: **cookie: ID1678453**





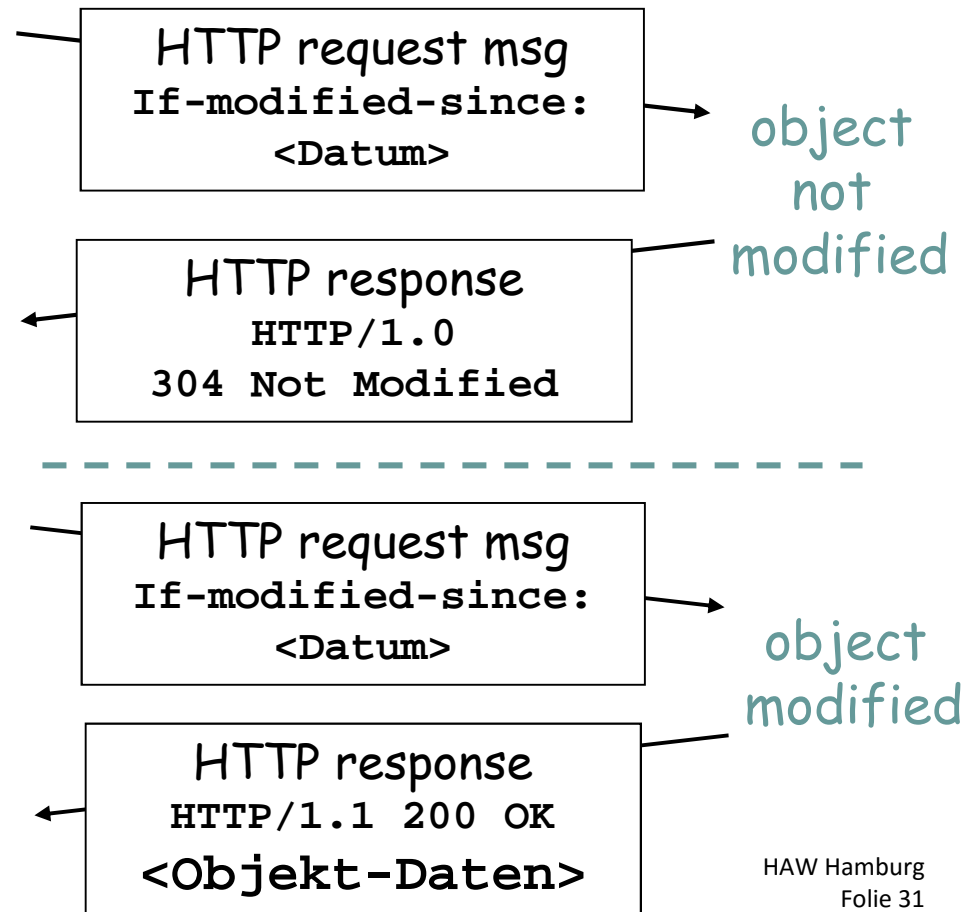
Caching auf Clientseite: Bedingtes GET

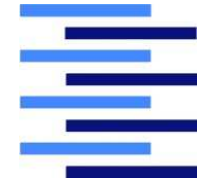
Angeforderte Web-Objekte werden vom Browser in einem lokalen Cache-Verzeichnis gespeichert!

- **Ziel:** Übertragung von Objekten vermeiden, die der Client bereits in seinem lokalen Cache hat.
- Client: spezifiziert Datum der gecachten Kopie in der HTTP-Anfrage
If-modified-since: <Datum>
- Server: Antwort enthält kein Web-Objekt, wenn die Kopie aktuell ist:
HTTP/1.0 304 Not Modified

Client

Server

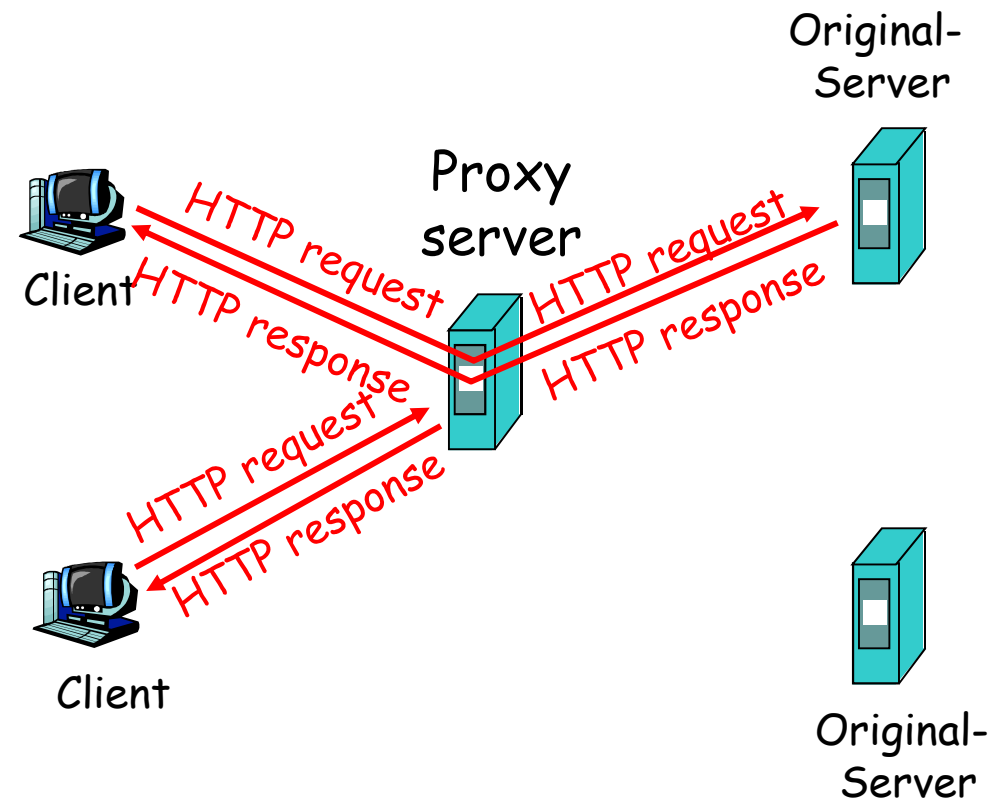




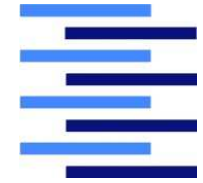
Caching auf Serverseite: Proxy-Server

Ziel: Beantwortung von Client-Anfragen durch "lokalen" Server

- Browser-Konfiguration:
"Proxy-Server verwenden"
- Client sendet alle HTTP-Anfragen an den Proxy Server
 - Object im Web-Cache des Proxy Servers vorhanden: liefere Web-Objekt zurück
 - sonst: Proxy Server stellt Anfrage an Original-Server und leitet die Antwort an den Client weiter

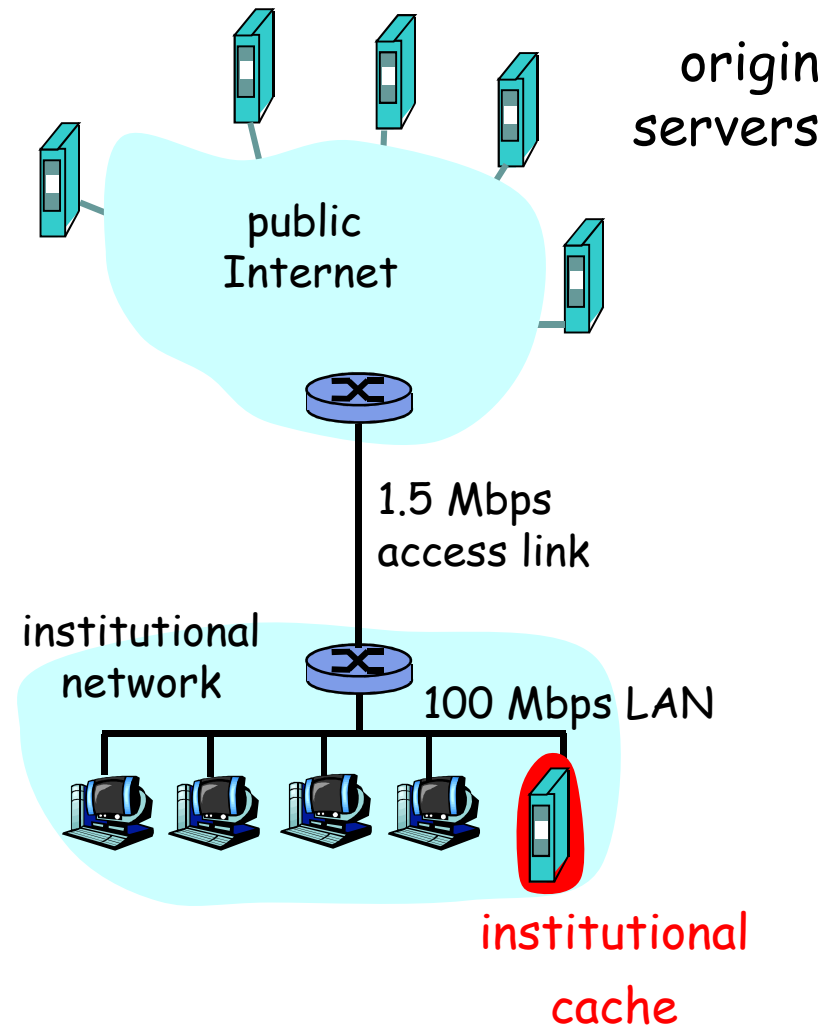


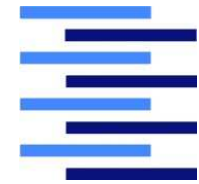
Vorteile des Caching durch Proxy Server



Voraussetzung: Proxy Server ist in der “Nähe” des Client (z.B. im selben Firmennetzwerk)

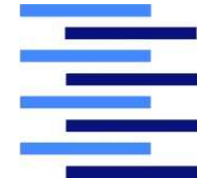
- geringere Antwortzeit
- weniger Verkehr ins Internet (Kosten und Verfügbarkeit der Verbindung zum ISP)
- Sicherheitsvorteile für geschlossene Netze (Zugriffsfilterung möglich)
→ *kommt später*





Sicherheit des WWW (HTTP/HTML): ☹️

- HTTP / HTML – Sicherheitsprobleme:
 - Cookies
 - Mobiler Code / Skripte auf Clients
 - Web-Server-Zugriff / Server-Skripte
- Beispiele:
 - Cross-Site-Scripting
 - Webserver-Spoofing



Beispiel: Cross-Site-Scripting (XSS)

- "Normaler" Link auf einer "normalen" Webseite:
``
Click Here ``
Dateiname
- "Böser" Link auf einer "normalen" Webseite:
`<a href="HTTP://www.example.com/`
`<script>..Böser Code..</script>">Click Here`
Dateiname
- Ergebnis: Dateiname
`<HTML>`
404 page not found:
`<script>..Böser Code..</script>`
`</HTML>`
Code wird ausgeführt, wenn die Metazeichen '<' und '>' nicht vom Web-Server ersetzt worden sind!!

Beispiel: Webserver-Spoofing

https://online-service.tv/www.commerzbank.de/portal/de/privatkunden/online-banking/onlinebanking.html

Ihr Suchtext Konzern | English

PIN **Anmelden**

Privatkunden Geschäftskunden Firmenkunden

Online Banking Produkte Beratung Wertpapiere & Märkte Service & Hilfe Kontakt

Start Privatkunden Online Banking

Online Banking

Hier erhalten Sie schnellen Zugriff auf die wichtigsten Funktionen im Online Banking. Sind nicht Ihre gewünschten Funktionen dabei? Dann klicken Sie oben im Navigationsmenü auf „Online Banking“ – dort finden Sie alle Funktionen im Überblick.

Meine Startseite

Finanzübersicht

Umsätze & Digitales Haushaltsbuch

Überweisung

Referenzkontoüberweisung

Depotübersicht

Wertpapiere kaufen

Wertpapiere verkaufen

Orderbuch

Mein Postfach

paydirekt

Online sicher bezahlen

Ab Ende 2015 direkt vom Girokonto.

Mehr zu paydirekt

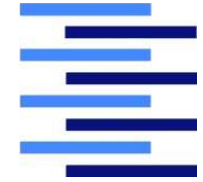
Sicherheit an unterster Stelle

Mit dem Online Banking haben Sie komfortablen Zugriff auf Ihre Konten und Depots – egal ob mit dem PC, Smartphone oder Tablet.

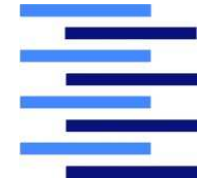
Dabei bieten Ihnen unsere innovativen TAN-Verfahren und die Sicherheitsstandards der Commerzbank höchsten Schutz in Verbindung mit einer einfachen Nutzung (hä, hä ..).

Kapitel 5

Anwendungsschicht



1. Prinzipien von Protokollen der Anwendungsschicht
2. Socket-Programmierung
3. Das World Wide Web (HTTP)
4. **Domain Name System (DNS)**

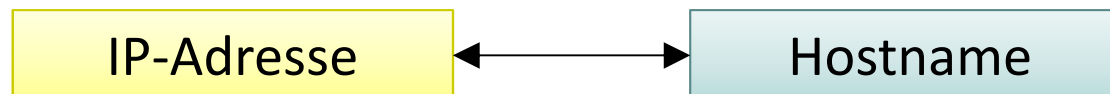


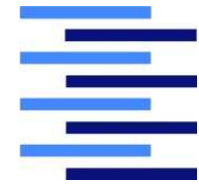
DNS: Problemstellung

Identifizierung von

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

➔ Wer verwaltet die Abbildung zwischen IP-Adresse und Hostnamen?



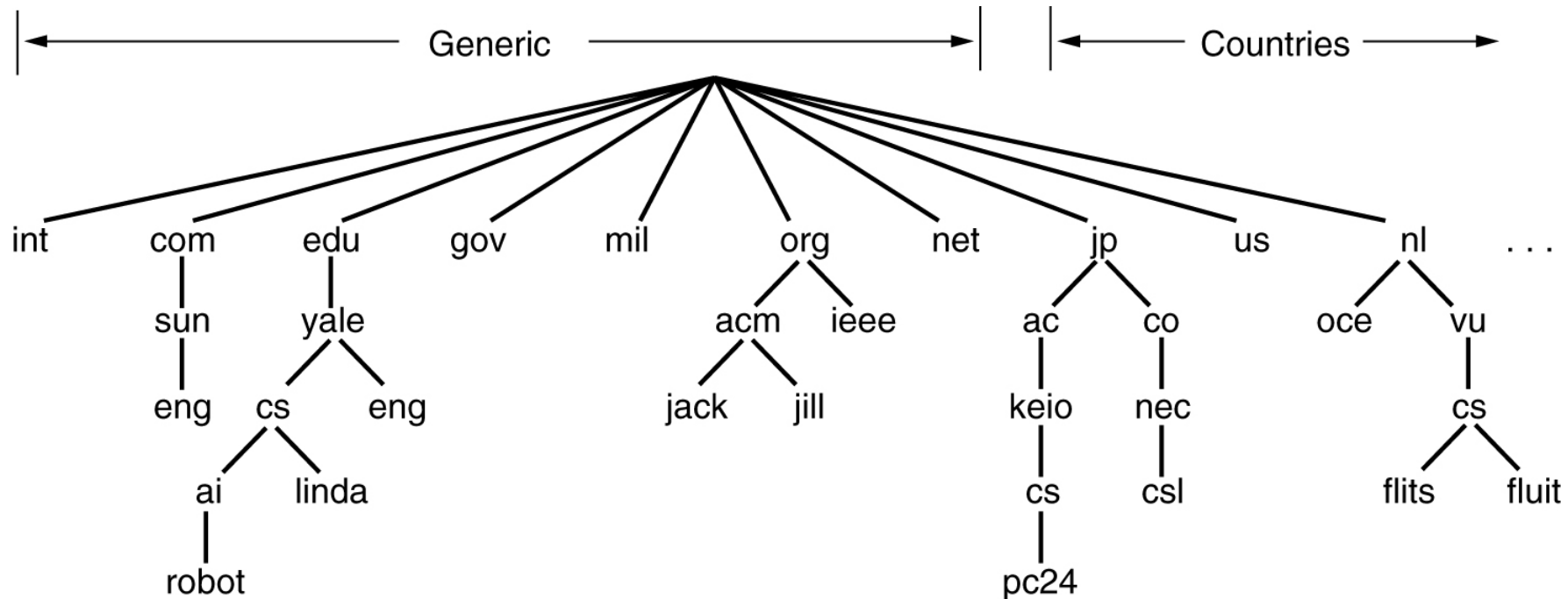
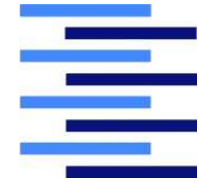


DNS: Domain Name System – was ist das?

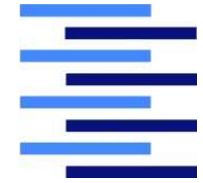
DNS [RFC 1034/1035] ist ...

- *eine verteilte Datenbank*,
implementiert durch eine Hierarchie von Name - Servern
- *ein Anwendungsschicht-Protokoll*,
über das Endsysteme, Router und Name-Server miteinander kommunizieren, um eine Abbildung von Hostnamen auf IP-Adressen zu erreichen.
 - Hinweis: Dies ist eine *Kernfunktion* für das Internet, die als Protokoll auf der Anwendungsschicht implementiert ist.
 - DNS stellt einen Dienst für andere Anwendungsschicht-Protokolle zur Verfügung (ohne Benutzerschnittstelle), stellt also selbst *keine Anwendung* dar.
 - DNS verwendet UDP

Namensraumstrukturierung: Domains und Subdomains



- Hierarchische Baumstruktur
- Generische und länderspezifische Top-Level Domains (*März 2014: 475*)
- Hostnamen (Rechnernamen) sind auf jeder Ebene möglich
- Der gesamte Pfad gehört zum Namen
- Beispiele: *flits.cs.vu.nl.* oder *users.informatik.haw-hamburg.de.*

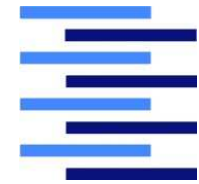


Verwaltung durch verteilte DNS - “Name - Server”

Warum gibt es keinen zentralen Name-Server für das gesamte Internet?

- Single point of failure
- Verkehrslast
- Entfernung der zentralen Datenbank
- Wartung

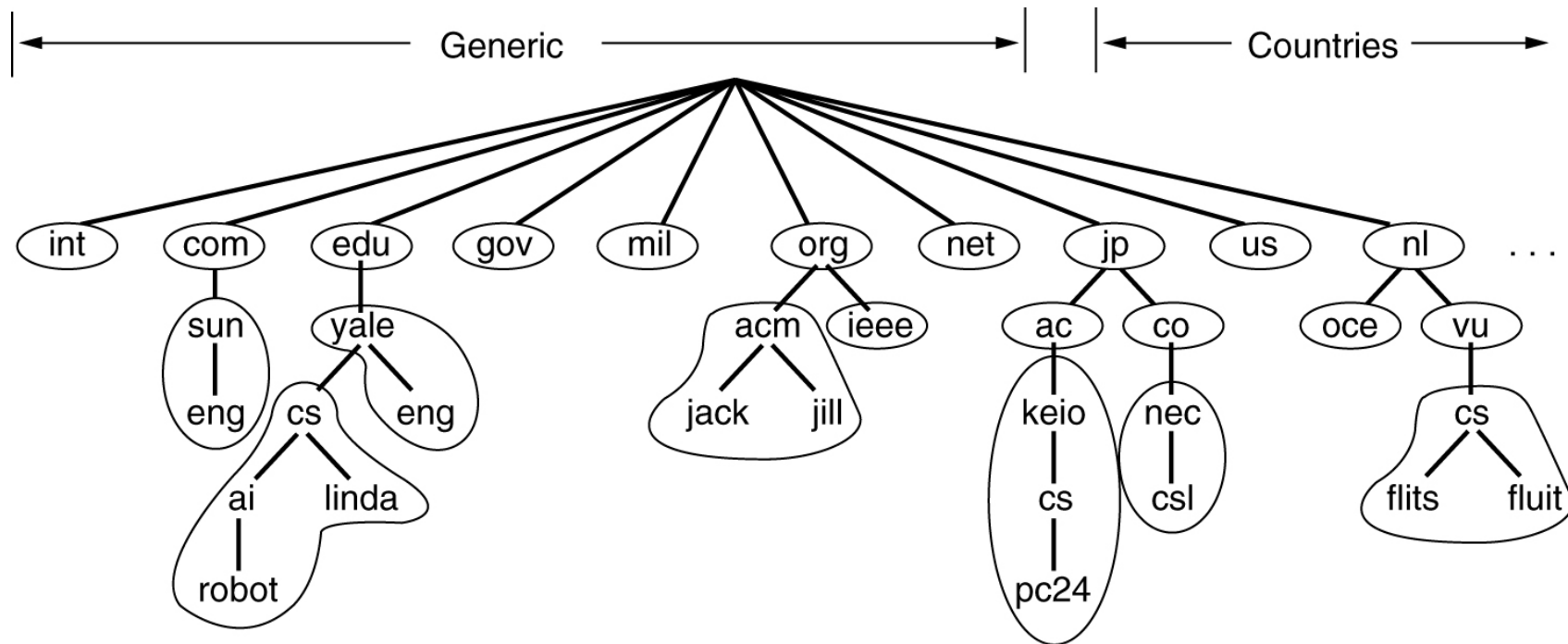
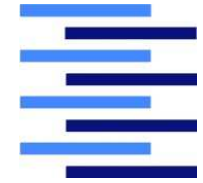
➔ Diese Architektur wäre nicht *skalierungsfähig*!



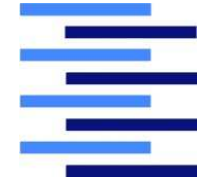
Arten von Name-Servern

- **Lokale Name-Server:**
 - jeder ISP / jede Firma hat üblicherweise einen lokalen Name-Server ("*DNS-Server*")
 - Alle DNS-Anfragen eines Hosts gehen zuerst zum lokalen Name-Server (*IP-Adresse des lokalen DNS-Servers gehört zur IP-Konfiguration eines Hosts*)
 - Lokale Nameserver erfragen IP-Adressen bei autoritativen Nameservern
- **Autoritative Name-Server:**
 - sind verantwortlich für die Speicherung von Name und IP-Adresse von Hosts
 - können Anfragen zur Umrechnung von Namen in IP-Adressen beantworten (*und noch mehr ...*)
 - verwalten eine "Zone" des Namensraums als "Autorität"
- **Root-Name-Server:**
 - verweisen auf autoritative Name-Server für die Top-Level Domains (*com, org, edu, de, nl, uk, jp, ...*)

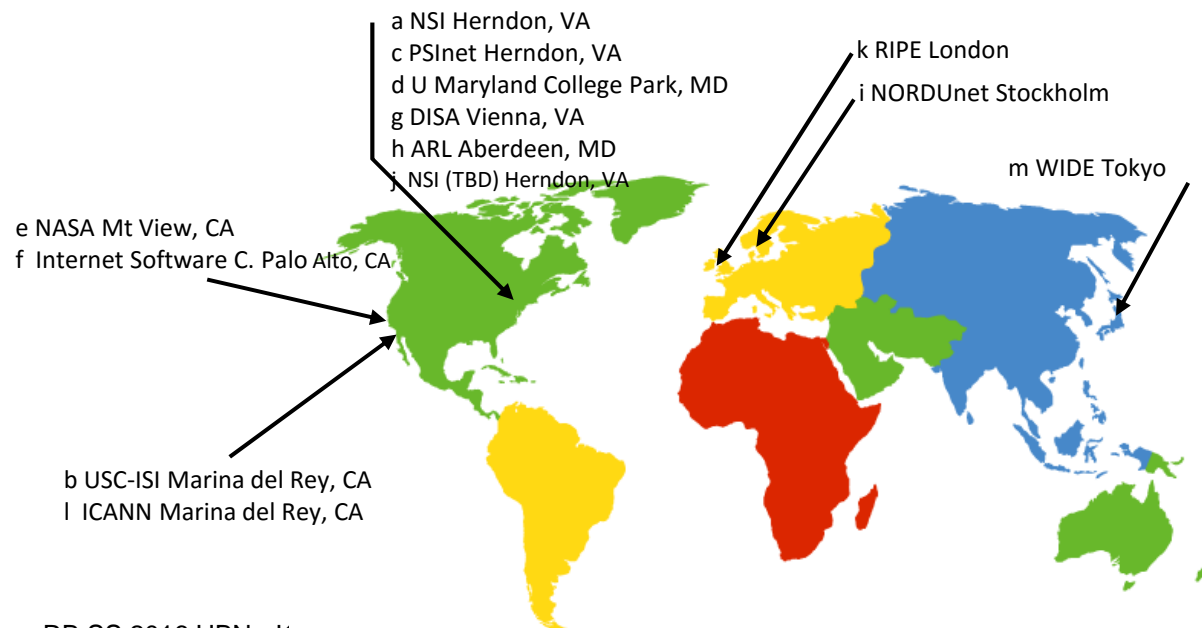
Verwaltung von **Zonen** durch Autoritative Name-Server



DNS: Root Name-Server



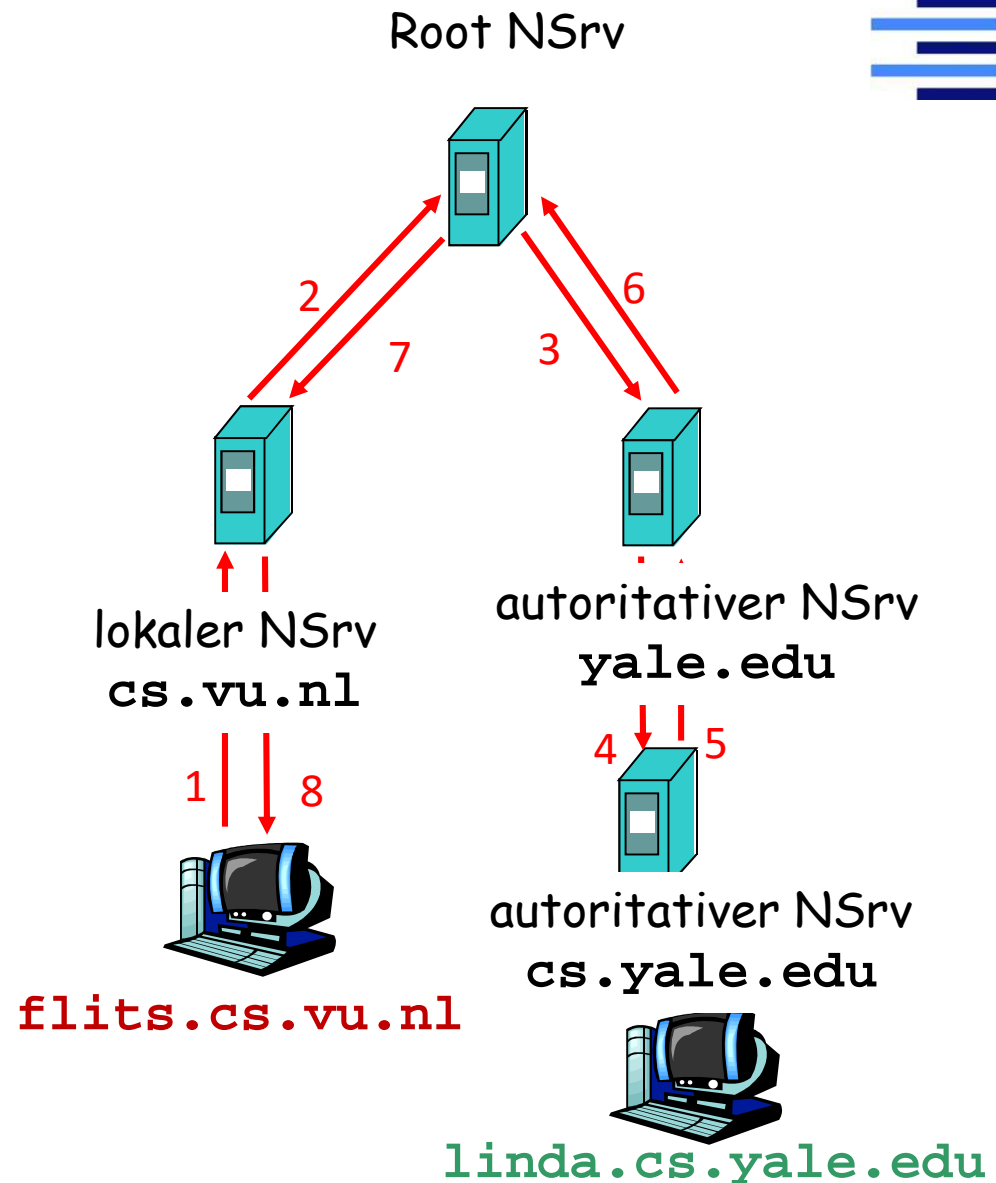
- ... wird von lokalen Name-Servern kontaktiert, die einen Hostnamen nicht auflösen können
 - Die Adressen der Top-Level-Domain-Nameserver sind jedoch meist im Cache der lokalen Name-Server vorhanden
- Root Name-Server:
 - liefert den im Pfad nächsten autoritativen Name-Server, falls der angefragte Name nicht gespeichert ist



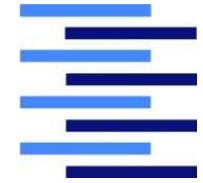
- Weltweit gibt es 13 Root Name-Server (*jeweils mehrere Maschinen*)
 - <http://root-servers.org/>
 - <http://www.orsn.org/de/>
- IP-Adressen der Root-Name-Server sind durch eine spezielle Konfigurationsdatei allen anderen Name-Servern bekannt!

Beispiel: DNS - Anfrage

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



DNS: Iterative Anfragen

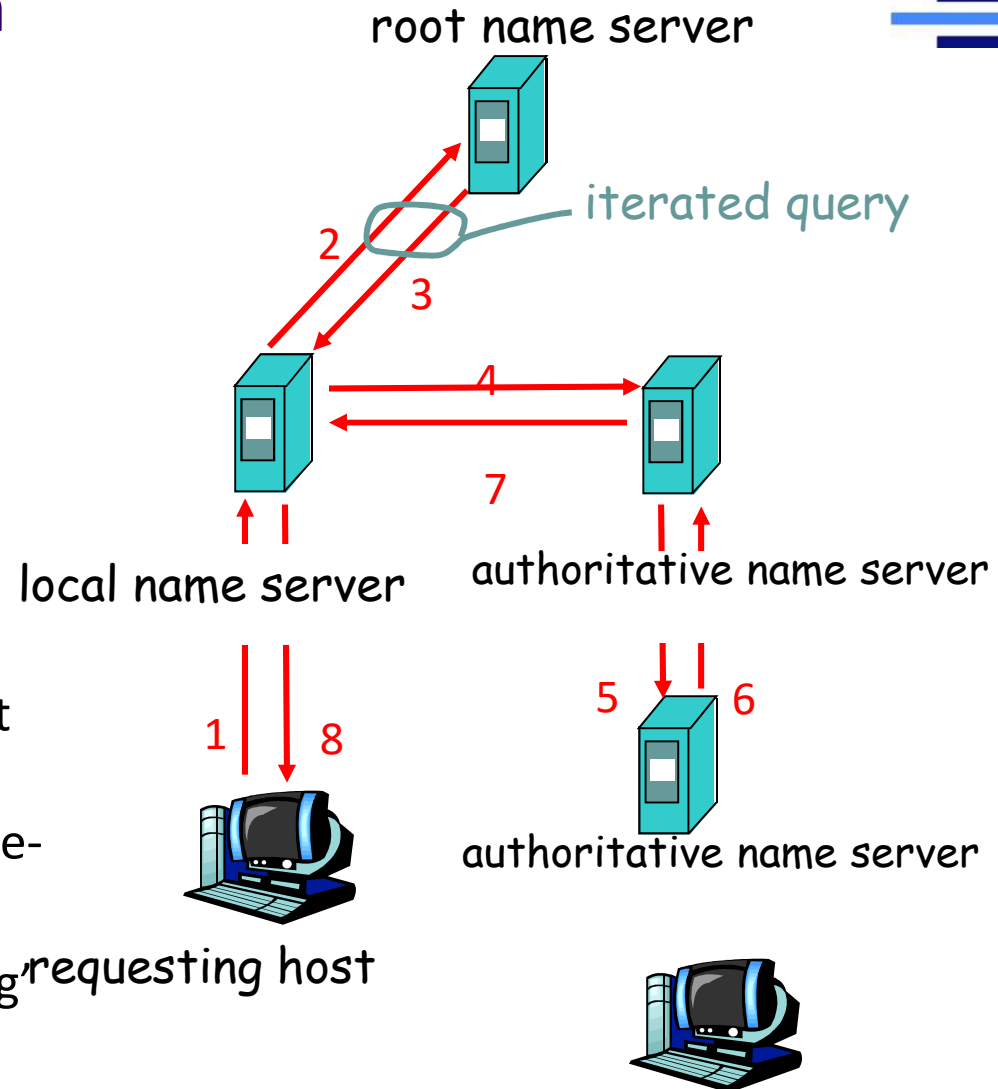


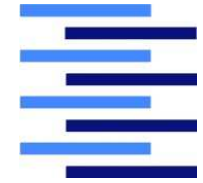
Rekursive Anfrage ("query"):

- Aufgabe der Namensauflösung wird komplett "delegiert"
- Lastverhalten (→ Root Server)?

Iterative Anfrage:

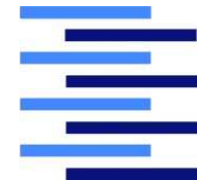
- Der angefragte Server liefert nur die IP-Adresse des nächsten autoritativen Name-Servers zurück
- "Ich habe keine Ahnung, frag requesting host Server nnn.nnn.nnn.nnn !"





DNS Resource Records

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



Resource Record-Typen

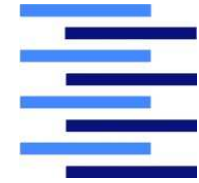
Typ	Bedeutung	Wert
SOA	Start of Authority	Parameter für diese Zone (mehrere Attribute)
A	Adresse eines Hosts (IPv4)	32-Bit integer
AAAA	Adresse eines Hosts (IPv6)	128-Bit integer
MX	Mail eXchange	16-Bit integer (Prioritätsangabe), danach Name eines Mailservers
NS	Name Server	Name-Server-Adresse für eine Domain
CNAME	Canonical name	Domainname
PTR	Pointer	Name für eine IP-Adresse ("Reverse Lookup")
HINFO	Host description	CPU und BS in ASCII
TXT	Text	Uninterpretierter ASCII-Text



DNS Resource Records: Beispiel

; Authoritative data for cs.vu.nl

cs.vu.nl.	86400	IN	SOA	star boss (952771,7200,7200,2419200,86400)
cs.vu.nl.	86400	IN	TXT	"Divisie Wiskunde en Informatica."
cs.vu.nl.	86400	IN	TXT	"Vrije Universiteit Amsterdam."
cs.vu.nl.	86400	IN	MX	1 zephyr.cs.vu.nl.
cs.vu.nl.	86400	IN	MX	2 top.cs.vu.nl.
flits.cs.vu.nl.	86400	IN	HINFO	Sun Unix
flits.cs.vu.nl.	86400	IN	A	130.37.16.112
flits.cs.vu.nl.	86400	IN	A	192.31.231.165
flits.cs.vu.nl.	86400	IN	MX	1 flits.cs.vu.nl.
flits.cs.vu.nl.	86400	IN	MX	2 zephyr.cs.vu.nl.
flits.cs.vu.nl.	86400	IN	MX	3 top.cs.vu.nl.
www.cs.vu.nl.	86400	IN	CNAME	star.cs.vu.nl
ftp.cs.vu.nl.	86400	IN	CNAME	zephyr.cs.vu.nl



DNS-Protokoll: Nachrichtenformat

Es gibt Anfrage- und Antwortnachrichten; beide haben dasselbe Nachrichtenformat

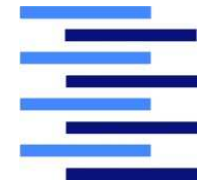
12 Byte Header:

- **identification**: 16 bit-Zahl zur Identifizierung einer Anfrage und zugehöriger Antwort
- **flags**:
 - Anfrage oder Antwort?
 - Rekursion gewünscht?
 - Rekursion verfügbar?
 - Antwort ist autoritativ!

Tools: **nslookup** oder **dig**

identification	flags
number of questions	number of answer RRs
number of authority RRs	number of additional RRs
questions (variable number of questions)	
answers (variable number of resource records)	
authority (variable number of resource records)	
additional information (variable number of resource records)	

↑
12 bytes
↓



Sicherheit des Domain Name Service (DNS): ☹️

Keine Gewährleistung von Geheimhaltung /
Integritätssicherung (→ Spoofing möglich)

Beispiel:

- Zur Optimierung von Caches verwendete Verfahren können missbraucht werden: „DNS Cache Pollution“
- Zwischengelagerter Server versendet zusätzlich falsche Reply-ResourceRecords
- Andere Server / Hosts nehmen dies evtl. als gültige Antwort hin (je nach Konfiguration)