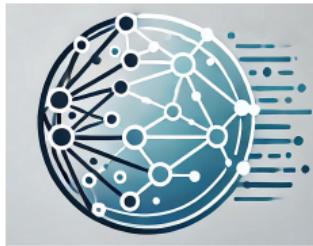

Netze

Modul 10: DNS und HTTP 1

👤 Prof. Dr. Michael Rademacher



4. Dezember 2025

Semesterplanung — Vorlesungen

Modul	Dozent	Datum	Thema
1	Rademacher	2. Oktober 2025	Einführung, OSI-Referenzmodell und Topologien
2	Rademacher	9. Oktober 2025	Übertragungsmedien und Verkabelung
3	Rademacher	16. Oktober 2025	Ethernet und WLAN
4	Tschofenig	23. Oktober 2025	IPv4, Subnetze, ARP, ICMP
5	Tschofenig	30. Oktober 2025	IPv6 und Autokonfiguration
6	Tschofenig	6. November 2025	Netzwerksegmentierung
7	Tschofenig	13. November 2025	Routing
8	Rademacher	20. November 2025	Transportschicht und UDP
9	Rademacher	27. November 2025	TCP
10	Rademacher	4. Dezember 2025	DNS und HTTP 1
11	Tschofenig	11. Dezember 2025	HTTP 2 und QUIC
12	Tschofenig	18. Dezember 2025	TLS und VPN
/	/	8. Januar 2026	Bei Bedarf / TBA
13	Tschofenig	15. Januar 2026	Messaging
14	Rademacher	22. Januar 2026	Moderne Netzstrukturen

Semesterplanung — Übungen und Praktika

ID	KW	Art	Thema
	40	/	/
UE-1	41	Übung	Topologien und OSI
UE-2	42	Übung	Übertragungen bspw. Kabel
P-1	43	Praktikum	Laboreinführung und Netzwerktools
S-1	44	Video	IPv4
P-2	45	Praktikum	Adressierung
P-3	46	Praktikum	Adressierung und IPv4
P-4	47	Praktikum	IPv4 und Autokonfiguration
P-5	48	Praktikum	IPv4 und IPv6
P-6	49	Praktikum	IPv6 und Autokonfiguration
P-7	50	Praktikum	Routing
S-2	51	Experiment	VPN
	52	Experiment	VPN
S-2	2		
P-8	3	Praktikum	Switching
P-9	4	Praktikum	DNS, Transportprotokolle, Webkommunikation

UE - Übung laut Stundenplan in den Seminarräumen

P - Praktikum in C055

S - Selbststudium **KEINE** Präsenz

- **Niemand möchte sich IP-Adressen merken.**
- Trotzdem brauchen Knoten (Nodes) in größeren Netzwerken (bspw. dem Internet) einen **Namen**, damit Nutzer (Menschen) diese adressieren können.
- Anstatt einer IP-Adresse in Punkt-Notation werden alphanumerische Internet-Adressen bevorzugt.
- $\text{www.example.com} \hat{=} 93.184.216.34$

Wie kann eine solche Abbildung zwischen IP-Adresse und Namen realisiert werden?



- **Niemand möchte sich IP-Adressen merken.**
- Trotzdem brauchen Knoten (Nodes) in größeren Netzwerken (bspw. dem Internet) einen **Namen**, damit Nutzer (Menschen) diese adressieren können.
- Anstatt einer IP-Adresse in Punkt-Notation werden alphanumerische Internet-Adressen bevorzugt.
- $\text{www.example.com} \hat{=} 93.184.216.34$

Wie kann eine solche Abbildung zwischen IP-Adresse und Namen realisiert werden?

Idee:

- Eine **zentrale** Datenbank für alle Namen und IP-Adressen
- Bei **jeder** namensbasierten Adressierung eines Knotens wird ein IP-Paket mit dem Namen an **diese** Datenbank geschickt

- Niemand möchte sich IP-Adressen merken.
- Trotzdem brauchen Knoten (Nodes) in größeren Netzwerken (bspw. dem Internet) einen **Namen**, damit Nutzer (Menschen) diese adressieren können.
- Anstatt einer IP-Adresse in Punkt-Notation werden alphanumerische Internet-Adressen bevorzugt.
- www.example.com $\hat{=}$ 93.184.216.34

Wie kann eine solche Abbildung zwischen IP-Adresse und Namen realisiert werden?

Idee:

- Eine **zentrale** Datenbank für alle Namen und IP-Adressen
- Bei **jeder** namensbasierten Adressierung eines Knotens wird ein IP-Paket mit dem Namen an **diese** Datenbank geschickt

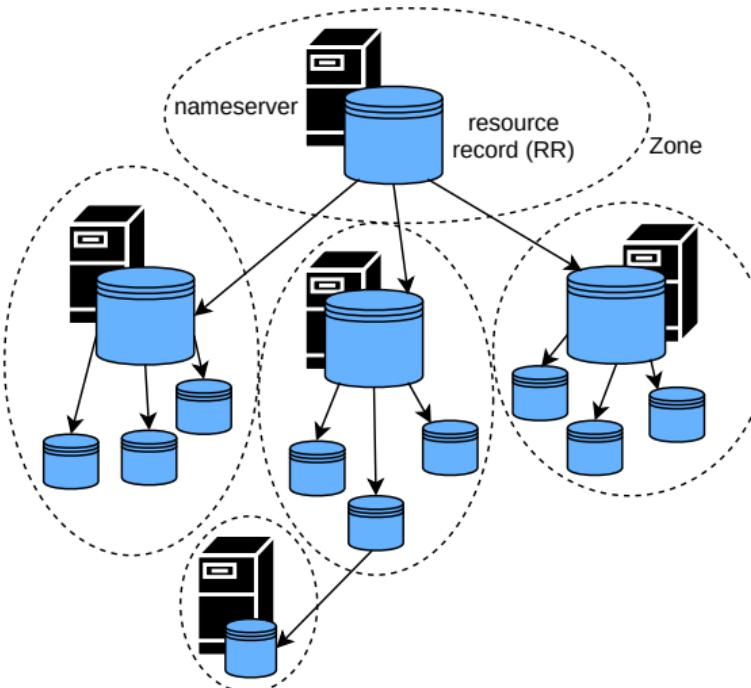
Probleme:

- **Wer** betreibt diese Datenbank?
- **Wo** steht diese Datenbank?
- **Wie** werden Namen und IP-Adressen hinzugefügt und entfernt?
=> **Eher keine gute Idee.**



DNS als verteilte Datenbank

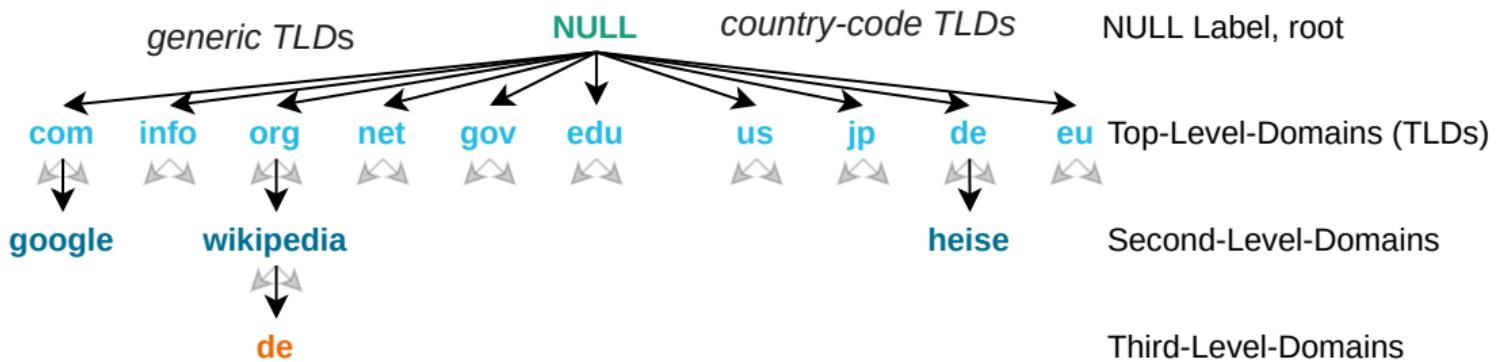
- Das DNS kann als eine **verteilte, hierarchische** Datenbank mit **baumförmiger Struktur** beschrieben werden.
- Die DNS-Informationen (**Resource Records**) werden in Zonen aufgeteilt und auf verschiedenen Knoten (**Namensservern**) vorgehalten und verwaltet [8].
- Eine Zone kann dabei für eine Domain und ihre Subdomains verantwortlich sein (also für viele Namen).
- Namensserver sind im Internet öffentlich erreichbar.
- Es existiert eine Vielzahl an **Namensservern**, welche über das DNS-Protokoll (UDP **und** TCP, Port 53) angesprochen werden können.



Domänennamen Syntax im Internet

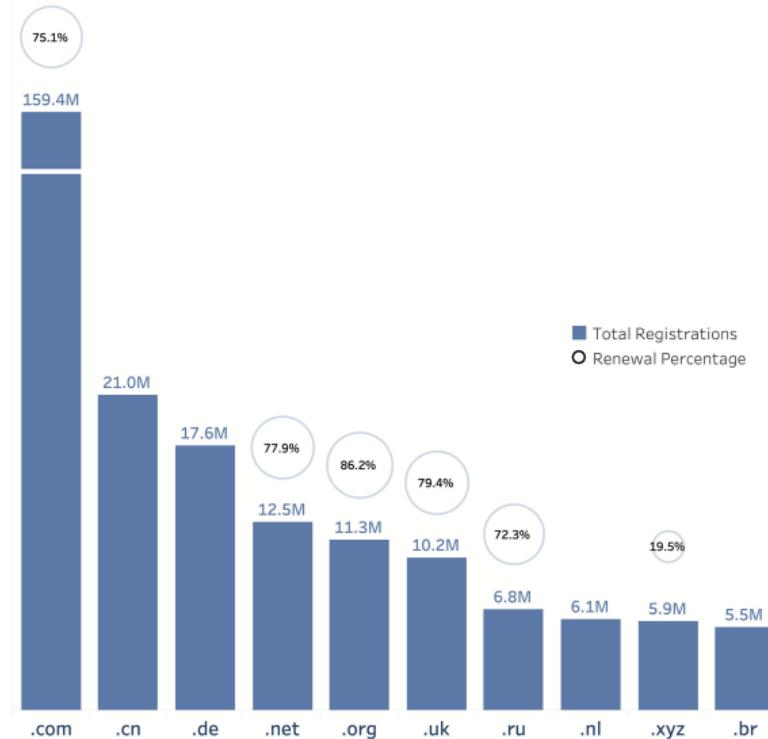
- Ein Domänenname besteht aus einem oder mehreren Teilen („Labels“).
 - Labels sind durch Punkte voneinander getrennt.
 - Die Hierarchie von Labels nennt man auch Level (gelesen von **rechts nach links**).
 - Das Null-Label (Länge ist 0) ist für die Wurzelzone reserviert.
 - Ein Label kann bis zu 63 Zeichen lang sein, die gesamte Domain bis zu 255 Byte (ca. 253 Zeichen Nutzlänge).

Beispiel: **en.wikipedia.org.NULL**



- Mittlerweile existieren mehr als 1000 TLDs [6].

Top 10 der TLDs - Im Jahr 2025

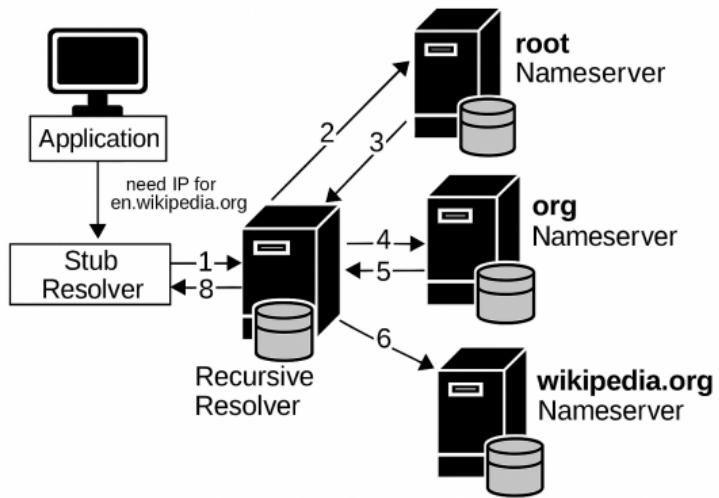


Quelle: <https://www.dnib.com/articles/the-domain-name-industry-brief-q3-2025>, „The DNIB Quarterly Report Q3 2025“

DNS Namensauflösung mit Resolver

- Der Client sendet eine **rekursive DNS-Anfrage** an seinen konfigurierten **rekursiven Resolver** (z. B. im ISP oder lokalen Router), um die IP-Adresse von en.wikipedia.org. zu erhalten.
- Der Eintrag liegt *nicht* im Cache. Der rekursive Resolver startet deshalb eine **iterative Auflösung** beginnend bei einem Root-Nameserver.
- Der Root-Server liefert einen Verweis (NS + Glue) auf die Nameserver der **org**-TLD.
- Der rekursive Resolver fragt einen **org**-TLD-Nameserver nach en.wikipedia.org.
- Der TLD-Server verweist auf die autoritativen Nameserver der Zone **wikipedia.org**.
- Der rekursive Resolver fragt einen dieser autoritativen Nameserver nach en.wikipedia.org.
- Der autoritative Nameserver **wikipedia.org** liefert die IP-Adresse von en.wikipedia.org.
- Der rekursive Resolver speichert die Antwort im Cache und sendet sie an den Client zurück.

→ Der Client kann nun IP-Pakete (z. B. mit TCP/HTTP) an en.wikipedia.org. senden.



Typen von DNS Namensserver

■ **Autoritativer** Namensserver: gibt **gesicherte** Antworten für Domainanfragen einer Zone.

- Root-Namensserver (13 logische Instanzen a–m.root-servers.net., weltweit hunderte physische Server via Anycast)
- TLD-Namensserver
- Domain-Namensserver
- „primary“ und „secondary“ autoritative Namensserver
 - „primary“ werden manuell vom Administrator der Zone konfiguriert
 - „secondary“ beziehen ihre Informationen über automatische Update-Mechanismen – Asynchronous Full Zone Transfer (AXFR) und Incremental Zone Transfer (IXFR)

■ **Nicht-autoritativer** Namensserver:

- Rekursiver Resolver, DNS-Informationen aus zweiter oder dritter Hand.
- DNS-Informationen werden im Cache gespeichert. **Kein iteratives Verfahren notwendig.**
- Wird meist vom ISP auf Routern vorkonfiguriert
- Viele Unternehmen bieten öffentliche DNS-Server an:
 - Google (8.8.8.8, 8.8.4.4), Cloudflare (1.1.1.1, 1.0.0.1), OpenDNS (208.67.222.222)

DNS Resource Records

- DNS-Informationen werden als **Resource Records** gespeichert [1].
- Diese liegen als Text-Darstellung in Zone-Dateien auf den Namensservern oder in komprimierter Form in DNS-Paketen oder in DNS-Caches vor.

Typische Textform: $< name > < ttl > < class > < type > < rdata >$

- $< name >$ Domänenname, zu dem der RR gehört
- $< ttl >$ Gültigkeit des RR in Sekunden
- $< class >$ IN für Internet
- $< type >$ Art des Resource Records
 - A: IPv4-Adresse eines Knotens
 - AAAA: IPv6-Adresse eines Knotens
 - NS: Hostname eines autoritativen Namensservers
 - CNAME: Abbildung eines Hostnamens auf einen anderen (canonical = anerkannt)
- $< rdata >$ Daten des RR



- **h.root-servers.net. 518400 IN A 198.97.190.53**

- IPv4-Adresse (A), unter welcher der Root-Server „h“ erreichbar ist

- **h.root-servers.net. 518400 IN AAAA 2001:500:1::53**

- IPv6-Adresse (AAAA), unter welcher der Root-Server „h“ erreichbar ist

- **org. 172800 IN NS b2.org.afilias-nst.org.**

- Hostname des autoritativen Namensservers „b2.org.afilias-nst.org.“ für die „org“-Zone

DNS explorativ mit dig – Die DNS-Hierarchie

Wie löst DNS einen Namen auf?

DNS arbeitet hierarchisch. Jeder Nameserver kennt nur einen Teil des Baums. Mit `dig` kann man diesen Auflösungsweg manuell nachverfolgen.

1. Root-Server abfragen

Die Root-Server kennen alle TLD-Nameserver:

- `dig A a.root-servers.net`

2. Domain über einen Root-Server auflösen

Wir fragen einen Root-Server direkt, welchen .com-TLD-Server wir kontaktieren sollen:

- `dig @198.41.0.4 A google.com`

3. Domain über einen TLD-Server auflösen

Der .com-Nameserver liefert die zuständigen Authoritative Nameserver:

- `dig @192.41.162.30 A google.com`

So kann man die gesamte DNS-Delegationskette schrittweise nachvollziehen – vom Root bis zum autoritativen Nameserver.

Direkte Abfragen bestimmter Nameserver

Mit `dig` können wir gezielt einzelne Nameserver ansprechen, um die Zuständigkeiten und Antworten verschiedener Ebenen zu untersuchen.

1. Nameserver einer Domain abfragen

Wir fragen einen Google-Nameserver selbst – wie jede andere Domain:

- `dig A ns1.google.com`

2. Normale DNS-Auflösung durchführen

Standard-Resolver führt rekursive Auflösung aus:

- `dig A google.com`

3. Domain gezielt über einen autoritativen Google-Nameserver auflösen

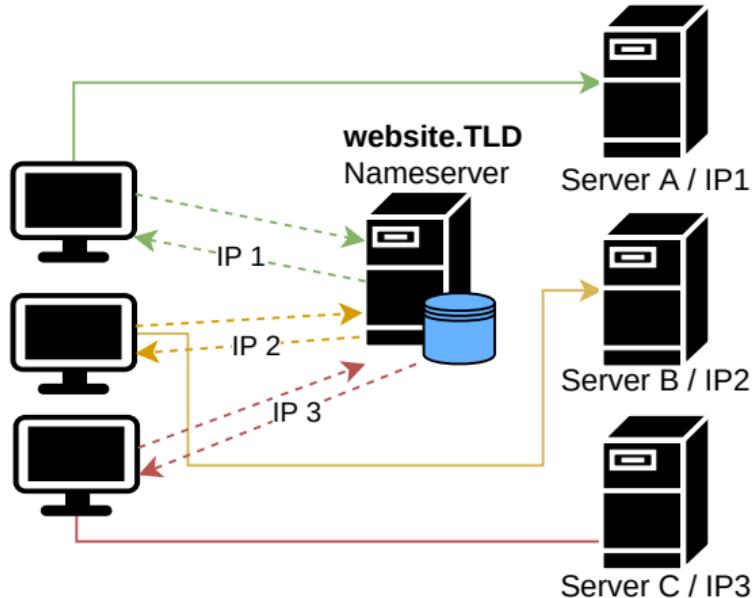
Damit umgehen wir den rekursiven Resolver und sehen die „echte“ Antwort von Google:

- `dig @216.239.32.10 A google.com`

So lässt sich exakt überprüfen, welche Antwort ein autoritativer Nameserver liefert – unabhängig von lokalen Caches oder Resolvern.

Lastverteilung mit DNS

- Namensserver verteilen unterschiedliche IP-Adressen je nach Anfrage
- Typischerweise im Round-Robin-Verfahren
- Pro: Sehr einfach zu implementieren
- Contra: Sehr unflexibel durch viel genutzte öffentliche DNS-Server (Caching)



■ DNS ist eine sehr kritische Komponente des Internets.

- „It's not DNS. There's no way it's DNS. It was DNS.“
- Viele Anwendungen verlassen sich auf ein funktionales DNS.

Datenschutz:

- DNS-Anfragen beinhalten Informationen, mit welchen anderen Knoten ein Knoten (Nutzer) interagiert
- DNS-Anfragen sind nicht geschützt:
 - Können mitgelesen werden
 - Können ggf. verändert werden
- Ein öffentlicher DNS-Server kennt ggf. alle DNS-Anfragen eines Knotens

Blocklisting:

- DNS-Anfragen können (bspw. lokal) blockiert werden
- Dies kann bspw. die Verbreitung von Schadsoftware (oder anderen ungewollten Inhalten) verhindern

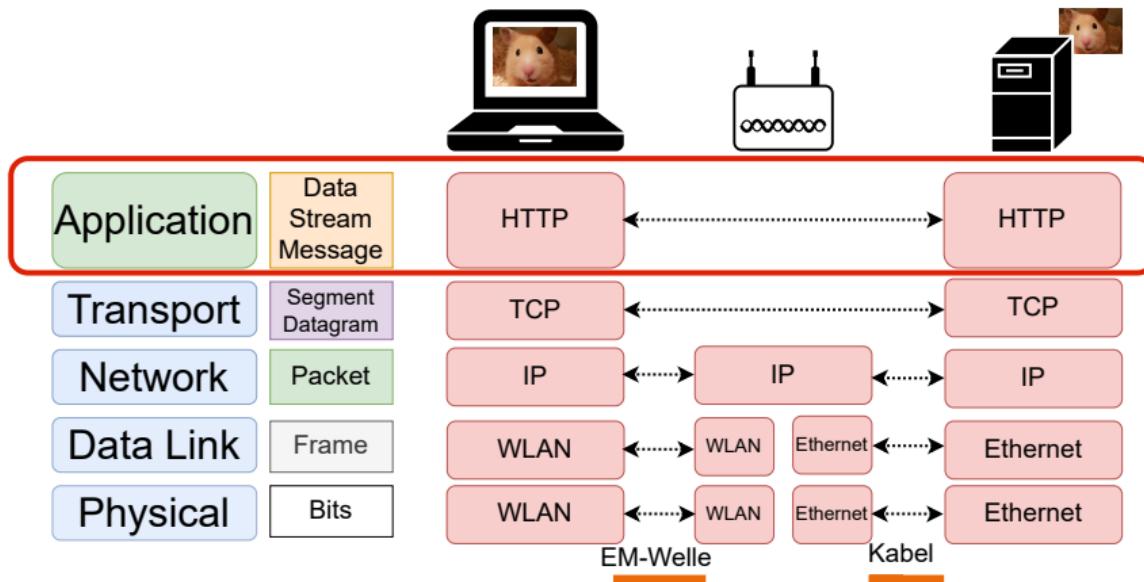


- DNSSEC (DNS Security Extensions) schützt gegen Manipulation von DNS-Daten.
- Idee: Resource Records werden kryptografisch signiert.
- Aufbau einer Vertrauenskette:
 - Root-Zone signiert TLD-Zonen (DS-Records)
 - TLD-Zonen signieren darunterliegende Domains
- Wichtige zusätzliche RR-Typen:
 - **DNSKEY**: öffentlicher Schlüssel der Zone
 - **DS**: Delegation Signer (Verweis auf Schlüssel der Child-Zone)
 - **RRSIG**: Signatur eines RR-Sets
 - **NSEC/NSEC3**: kryptografischer Beweis für Nicht-Existenz eines Namens
- Schützt die **Integrität** und **Authentizität** von DNS-Daten, aber nicht deren Vertraulichkeit.



HTTP

Das HTTP ist ein zustandsloses Protokoll der Anwendungsschicht, dessen erste Version 1991 eingesetzt wurde. Die erste offizielle Standardisierung erfolgte 1996 mit HTTP/1.0. HTTP bildet die Grundlage des World Wide Web und wird genutzt, um Hypertext-Dokumente sowie weitere Webressourcen an Browser und andere Clients zu übertragen.



Webseite

- Eine Webseite besteht aus Objekten.
 - HTML-Dateien, Bilder, CSS, Java-Applets, Audiodateien, ...
- Eine Webseite hat eine Basis-HTML-Datei, die (in der Regel) mehrere referenzierte Objekte beinhaltet.
- Objekte werden durch eine **URL** adressiert.

<https://netze-demo.mc-lab.de/>



Ein HTML-Dokument

```
<html>
<head>
<title>Hamster Horsti</title>
</head>
<body>
<h1>Willkommen auf meiner Website</h1>

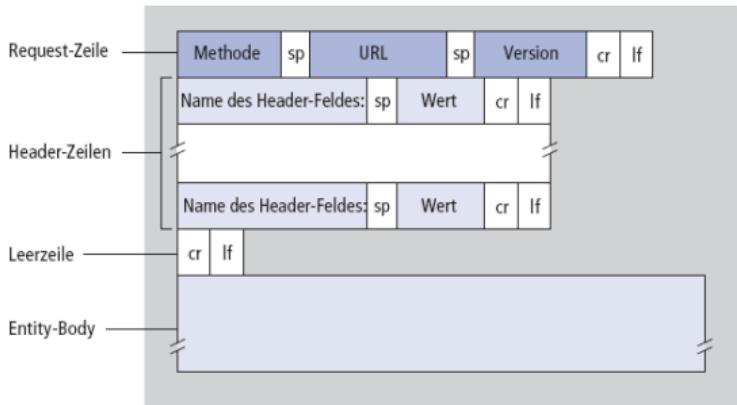
<h2>Content</h2>
Folgt mir fuer mehr spannenden Hamster Content!
<br>
Aktuelles Statement:
<br>
<audio controls>
<source src="horsti.mp3" type="audio/mpeg">
</audio>
</body>
</html>
```



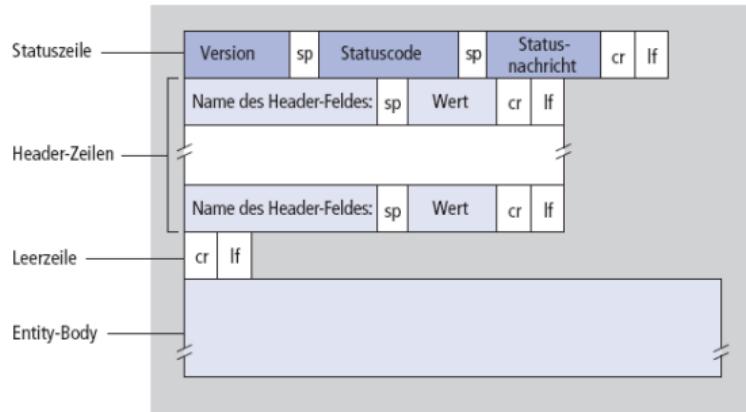
- HTTP ist ein „Request → Response“ Protokoll
- HTTP ist **zustandslos (stateless)**
 - Jeder **Request** enthält alle Informationen, die für die **Response** notwendig sind
 - Die Semantik eines Requests ist unabhängig von vorherigen Requests; der Server speichert keinen Sitzungszustand
- Typischer Ablauf einer einfachen HTTP-Kommunikation:
 1. Client (Browser wie Firefox, Chrome, Edge) stellt einen **Request**
 2. Beispiel: eine HTML-Datei wird angefordert
 3. Server (z. B. Apache HTTP Server oder nginx) antwortet mit einer **Response**
 4. Beispiel: die angefragte HTML-Datei wird ausgeliefert
 5. Der Client interpretiert die Response
 6. Beispiel: Darstellung der Seite, Nachladen weiterer Ressourcen (Bilder, CSS, JS)

Request und Response Protokoll Format

Request:



Response:



Methode:

- GET: Anfrage, um Daten beim Server abzurufen
- POST: Daten senden (bspw. Login)
- PUT, PATCH, DELETE: Weitere Methoden (v.a. bei REST-APIs)

Statuscode:

- 200 OK
- 301 Moved Permanently
- 401 Unauthorized
- **404 Not Found**
- 500 Internal Server Error



Beispiel Request und Response

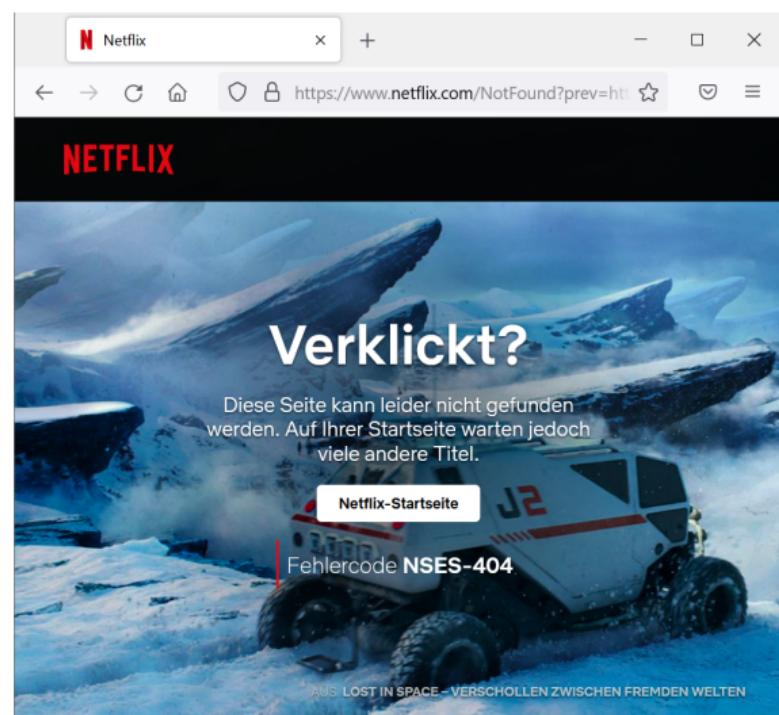
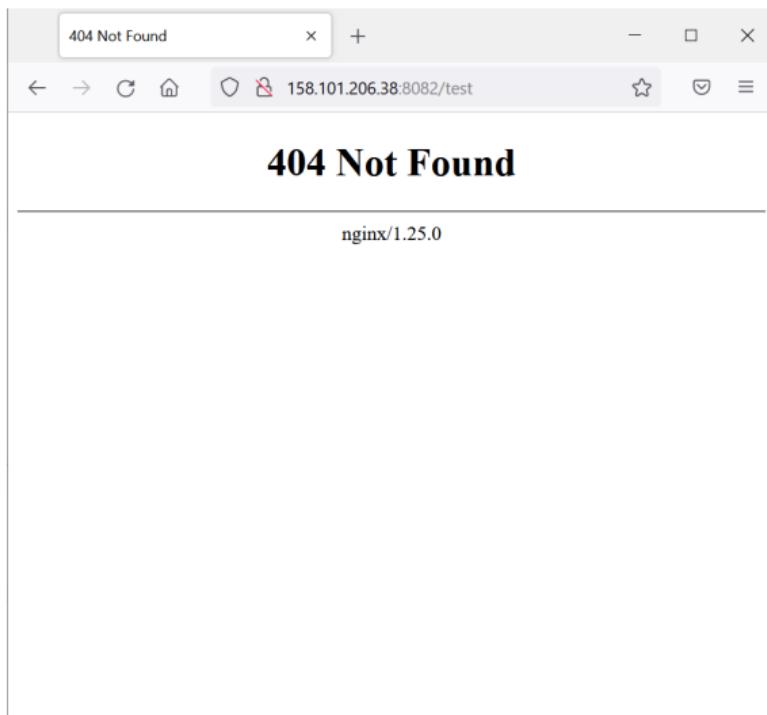
Request:

```
GET /index.html HTTP/1.1
Accept: text/html
Accept-Language: de-DE
Accept-Encoding: gzip, deflate
Host: 158.101.206.38:8082
User-Agent: Mozilla/5.0
```

Response:

```
HTTP/1.1 200 OK
Server: nginx/1.25.0
Date: Thu, 01 Jun 2023 11:51:26 GMT
Content-Type: text/html
Content-Length: 460
Last-Modified: Mon, 29 May 2023 16:20:32 GMT
```

```
<html>
<head>
<title>Hamster Horsti</title>
</head>
```



Wireshark: Website Beispiel

The screenshot shows a Wireshark capture window with the following details:

- Filter Bar:** ip.src==158.101.206.38 or ip.dst==158.101.206.38
- Table Headers:** No., Time, Source, Destination, Protocol, Length, Info
- Selected Frame:** 256 (17...) 10.20.111.72 > 158.101.206.38 HTTP [GET / HTTP/1.1]
- Frame Details:** Shows the raw hex and ASCII data for the selected frame, which contains the HTML content of the website.
- Text Data:** Line-based text data: text/html (18 lines) showing the HTML code for the website.

Raw Hex and ASCII Data for Selected Frame (Frame 256):

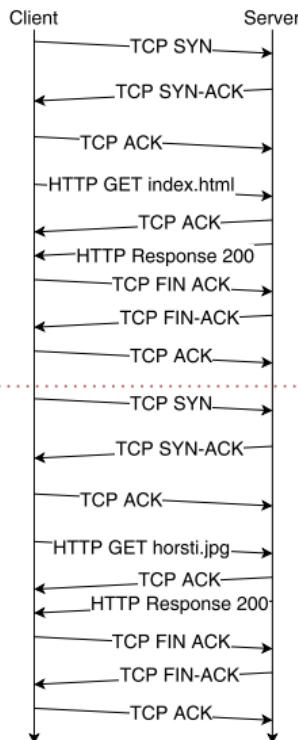
Hex	ASCII
0100	20 20 20 20 20 20 20 20 3c 74 69 74 6c 65 3e 48
0110	61 6d 73 74 65 72 20 48 6f 72 73 74 69 3c 2f 74
0120	69 74 6c 65 3e 0a 20 20 20 20 3c 2f 68 65 61 64
0130	3e 0a 20 20 20 20 3c 62 6f 64 79 3e 0a 20 20 20
0140	20 20 20 20 20 3c 68 31 3e 57 69 6c 6c 6b 6f 6d
0150	6d 65 6e 20 61 75 66 20 6d 65 69 6e 65 72 20 57
0160	65 62 73 69 74 65 3c 2f 68 31 3e 0a 20 20 20 20
0170	20 20 20 20 3c 69 6d 67 20 73 72 63 3d 22 68 6f

Reassembled TCP (698 bytes):

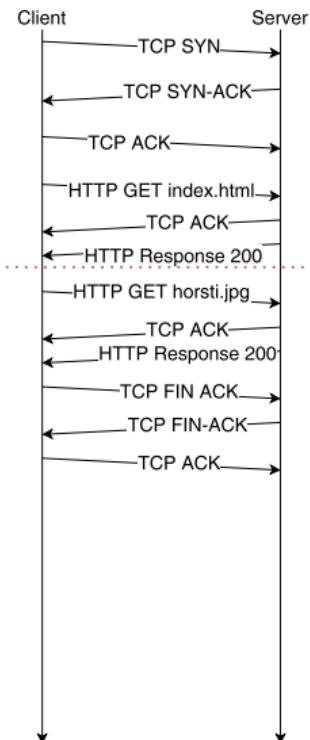
```
<title>Hamster Horsti</title>
</head>
<body>
<h1>Willkommen auf meiner Website</h1>
```

Abruf von mehreren HTML-Objekten via HTTP

Verbindlungspersistenz



← Non-Persistent (Default bei
HTTP/1.0)

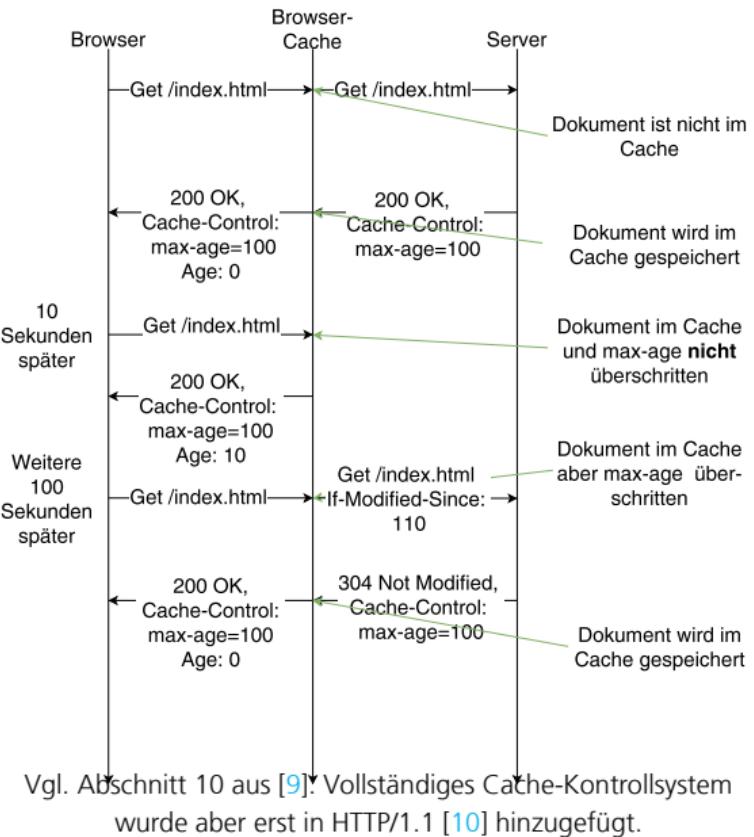


Persistent (Default bei
HTTP/1.1) →

- Definiert in RFC 1945 [9].
- Einführung grundlegender Methoden: GET, POST, HEAD.
- Einführung von HTTP-Headern (textbasierte Kodierung).
- Drei-stellige Statuscodes für Antworten.
- GET: idempotent und „sicher“ (nur lesend), POST: nicht idempotent und nicht sicher.

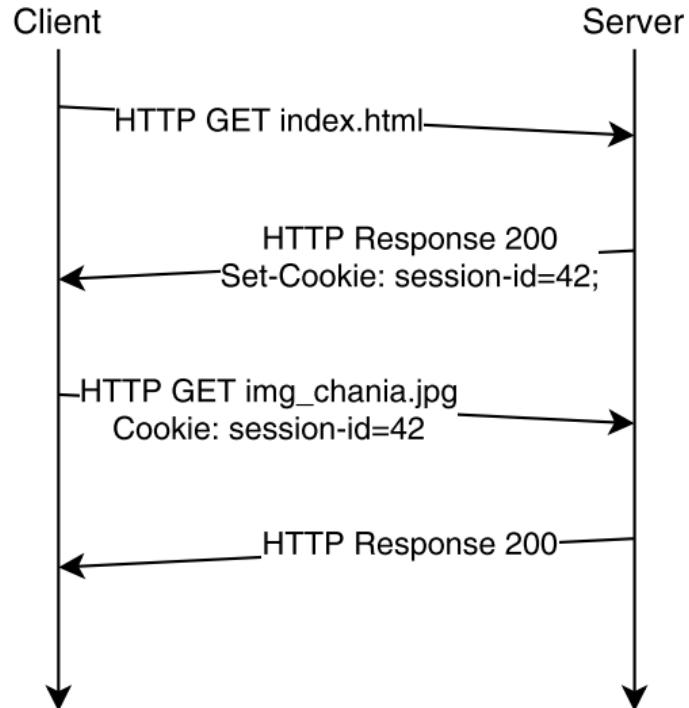
Caching im Browser

- Ziel: Reduziere die Daten, die der Client beim Server anfragen muss. Dadurch schnelleres Anzeigen der Website.
- Viele Daten im Internet sind statisch, bspw. **Logos, Texte, Bilder**.
- Implementierung in HTTP:
 - Server können in der HTTP-Response mitschicken, wie lange Daten gültig sind.
 - Clients können im HTTP-Request mitschicken, dass sie Daten schon haben und diese Version von Zeitpunkt X ist.
 - Der Server prüft beim Request, wann sich die Datei das letzte Mal verändert hat.
 - Response: **304 (Not Modified)**, wenn die Daten beim Client noch aktuell sind.
 - Response: **200 (OK)** und die Daten, wenn die Daten beim Client veraltet sind.



Session trotz zustandslosem HTTP

- **HTTP ist zustandslos.**
- Eine Vielzahl von Szenarien benötigt jedoch „Sessions“ (z. B. Online-Banking, Webshops).
- Zu diesem Zweck wurden Cookies [2] eingeführt.
- Idee: Server gibt dem Client einen Cookie.
 - Client-Requests enthalten diesen Cookie.
 - Server kann die Sitzung bzw. den Client identifizieren.



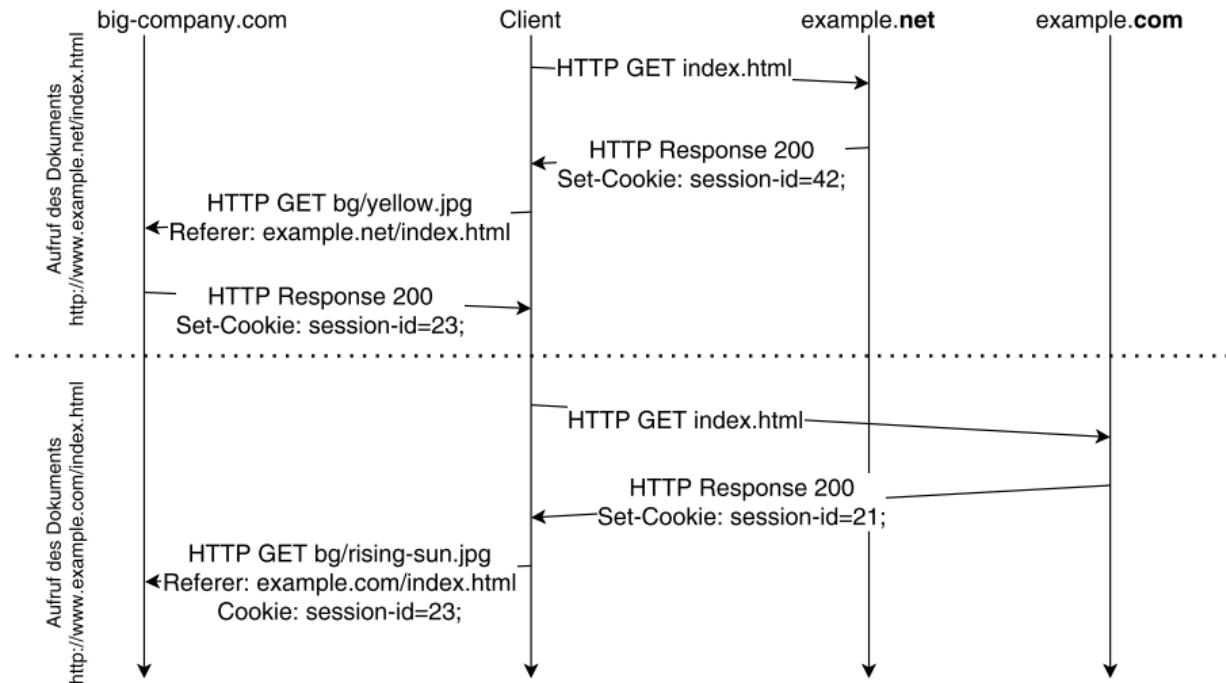
Sind Cookies böse? Tracking...

- Obwohl Cookies nur an den Server gesendet werden, von dem diese stammen, kann man Nutzende mit Cookies **tracken/nachverfolgen**.
- Szenario: Eine Website bindet andere Websites ein (bspw. Werbung, „Like Buttons“, Bilder, Schriftarten, Scripte, ...)
- Die eingebundene Seite bekommt **ihre Cookies**, kann feststellen, wo sie eingebunden ist (**Referrer**), und so den Nutzer tracken.

```
GET bg/rising-sun.jpg HTTP/1.1
Accept: image/svg
Host: big-company.com
User-Agent: Mozilla/5.0
Referer: example.com
Cookie: session-id=23
```



Tracking



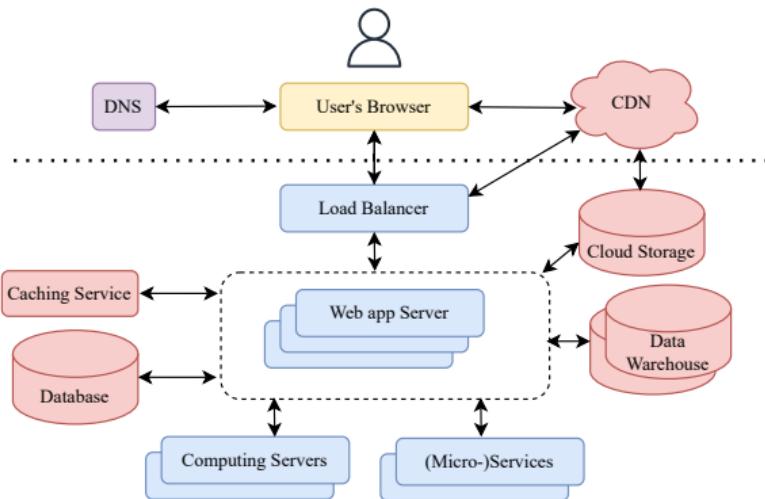
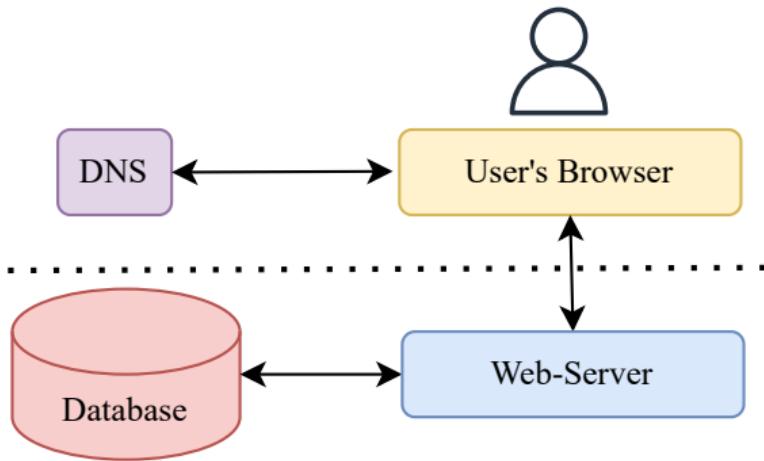
Was hat big-company.org alles über den Nutzer lernen können?

- Betriebssystem und Browser, IP-Adresse, **auf welchen Seiten der Nutzer war.**
- Besonders interessant, wenn big-company.org auch den Klarnamen kennt (via Login)



TLS + HTTP = HTTPS

Webseiten: „früher“ und heute



Quellen I

- [1] Domain names - implementation and specification.
RFC 1035, November 1987.
- [2] Barth, A.
HTTP State Management Mechanism.
RFC 6265, April 2011.
- [3] Bortzmeyer, S., Dolmans, R., and Hoffman, P. E.
DNS Query Name Minimisation to Improve Privacy.
RFC 9156, November 2021.
- [4] Hoffman, P. E., and McManus, P.
DNS Queries over HTTPS (DoH).
RFC 8484, October 2018.
- [5] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and Hoffman, P. E.
Specification for DNS over Transport Layer Security (TLS).
RFC 7858, May 2016.
- [6] iana.
<https://data.iana.org/tld/tlds-alpha-by-domain.txt>.
<https://data.iana.org/TLD/tlds-alpha-by-domain.txt>, 12 2020.
(Accessed on 12/01/2020).
- [7] Kinnear, E., McManus, P., Pauly, T., Verma, T., and Wood, C. A.
Oblivious DNS over HTTPS.
RFC 9230, June 2022.
- [8] Mockapetris, P.
Domain names - concepts and facilities.
STD 13, RFC Editor, November 1987.
<http://www.rfc-editor.org/rfc/rfc1034.txt>.

Quellen II

- [9] Nielsen, H., Fielding, R. T., and Berners-Lee, T.
Hypertext Transfer Protocol – HTTP/1.0.
RFC 1945, May 1996.
- [10] Nielsen, H., Mogul, J., Masinter, L. M., Fielding, R. T., Gettys, J., Leach, P. J., and Berners-Lee, T.
Hypertext Transfer Protocol – HTTP/1.1.
RFC 2616, June 1999.
- [11] Surý, O., Toorop, W., 3rd, D. E. E., and Andrews, M. P.
Interoperable Domain Name System (DNS) Server Cookies.
RFC 9018, April 2021.