TECHNISCHE
UNIVERSITÄT
DARMSTADT

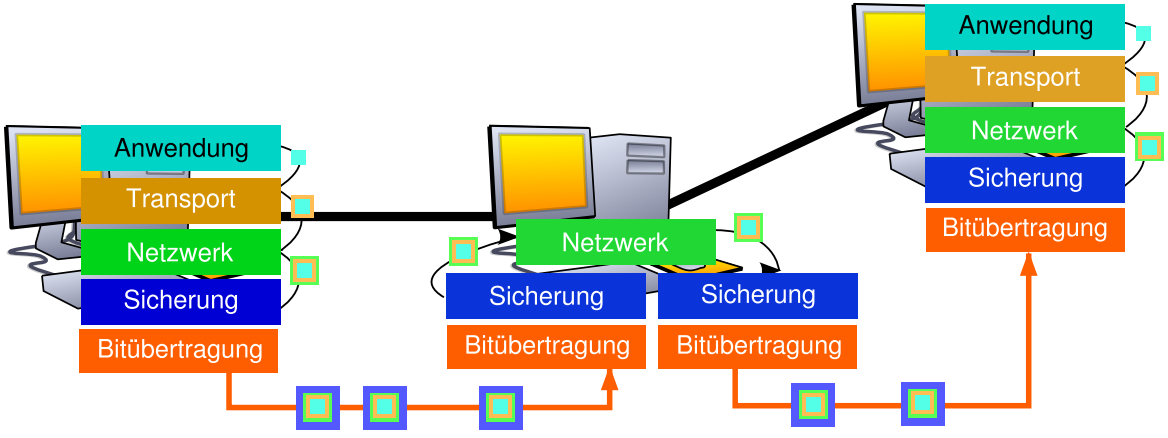
Kommunikationsnetze 1

Kapitel 2: Anwendungsprotokolle

Sommersemester 2024
Björn Scheuermann

Einordnung

TECHNISCHE UNIVERSITÄT DARMSTADT



Netzwerkanwendungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Was sind „Netzwerkanwendungen“ in unserem Sinne?

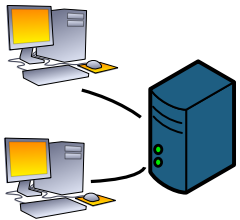
Programme, die zum Zweck einer Diensterbringung bzw. Dienstnutzung innerhalb eines Rechnernetzwerkes miteinander kommunizieren

- Diese Anwendungen umfassen sowohl den eigentlichen Anwendungs„zweck“ bzw. -algorithmus als auch das Anwendungsprotokoll
- Sie setzen auf einem Netzwerkprotokollstapel auf und nutzen dessen Dienste

Anwendungsarchitekturen

TECHNISCHE UNIVERSITÄT DARMSTADT

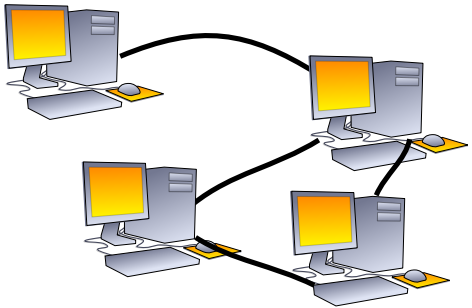
- Der Entwickler einer Netzwerkanwendung muss die *Architektur* seiner Anwendung festlegen
- Die klassische, weit verbreitete Architektur von Netzwerkanwendungen ist die *Client-Server-Architektur*
- Dabei existiert ein zentrales Gerät, das (in der Theorie) immer verfügbar ist: der *Server*
- Der Server beantwortet Anfragen von anderen Geräten, den sogenannten *Clients*
- Clients kommunizieren nicht direkt miteinander, sondern immer nur mit dem Server



Peer-to-Peer-Anwendungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Peer-to-Peer-Anwendungen gehen einen anderen Weg
- Hier kommunizieren Paare von Hosts, die nicht ständig miteinander verbunden sind
- Es gibt also keine Unterscheidung zwischen „Clients“ und „Servern“, sondern jeder teilnehmende Host übernimmt beide Rollen



Peer-to-Peer-Anwendungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Vor allem Anwendungen mit großem Datenverkehrsaufkommen verwenden gerne Peer-to-Peer-Architekturen
  - Dateiverteilung mit BitTorrent
  - Internettelefonie mit Skype (früher…)
  - Filesharing, z. B. mit Gnutella oder LimeWire (ganz früher…)
  - …
- Aber auch die gängigen Kryptowährungen (Bitcoin, Ethereum,…)
- Keine Server notwendig $\Rightarrow$ Kostenvorteile
- Die Vorteile erkauft man mit einem typischerweise sehr viel komplexeren Anwendungsdesign und Anwendungsprotokoll

Wahl der Transportschicht

TECHNISCHE UNIVERSITÄT DARMSTADT

- Für die Kommunikation zwischen zwei Programminstanzen muss sich ein Anwendungsentwickler dann auch für ein Transportprotokoll entscheiden
- Verschiedene Arten von Anwendungen können sehr unterschiedliche Anforderungen stellen
- Verschiedene Transportprotokolle bieten unterschiedliche Dienste an, z. B.
  - Zuverlässigkeit (Dateitransfer: gefordert, Multimedia: Aussetzer können toleriert werden)
  - Datenratensteuerung (Dateitransfer: beliebig, Multimedia: meist Mindestdatenrate gefordert)
  - Echtzeitfähigkeit: Paketlaufzeit oder Paketankunftsintervalle sind vorgeschrieben (Dateitransfer: egal, Multimedia: gefordert)
  - ...
- Im Internet werden fast ausschließlich die Transportprotokolle TCP und UDP verwendet

Datenverkehr im Internet

TECHNISCHE UNIVERSITÄT DARMSTADT

Was verursacht den meisten Datenverkehr im Internet?

Top Content Categories by Downstream Volume – Fixed

	Downstream Volume		
	Content Category	% DS Vol	Sub. Volume
1	On-Demand Streaming	54%	7.9 GB
2	Live Streaming	14%	2.0 GB
3	File Delivery	13%	2.0 GB
4	Browsing	3%	441 MB
5	Game Play	3%	398 MB
6	Video Call	2%	300 MB
7	Messaging	0.6%	81 MB
8	Voice Call	0.5%	74 MB
9	Machine to Machine	0.01%	2 MB
10	AR/VR	0.00%	18 KB
11	Other	10%	1.4 GB

Top Content Categories by Downstream Volume – Mobile

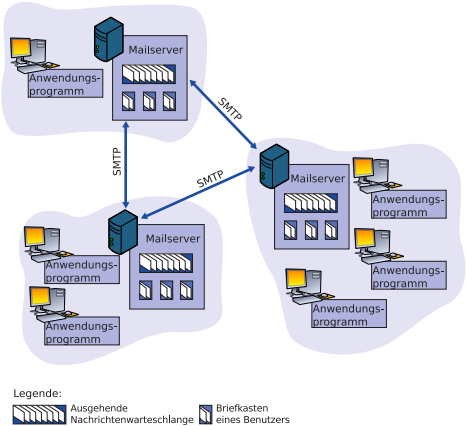
	Downstream Volume		
	Content Category	% DS Vol	Sub. Volume
1	On-Demand Streaming	57%	900 MB
2	File Delivery	11%	173 MB
3	Live Streaming	7%	107 MB
4	Game Play	5%	75 MB
5	Video Call	5%	73 MB
6	Browsing	2%	38 MB
7	Messaging	2%	29 MB
8	Voice Call	1%	19 MB
9	Machine to Machine	0.00%	68 KB
10	AR/VR	0.00%	1 KB
11	Other	10%	160 MB

(Abbildung: [Sandvine Inc.: Global Internet Phenomena Report 2024])

E-Mail im Internet

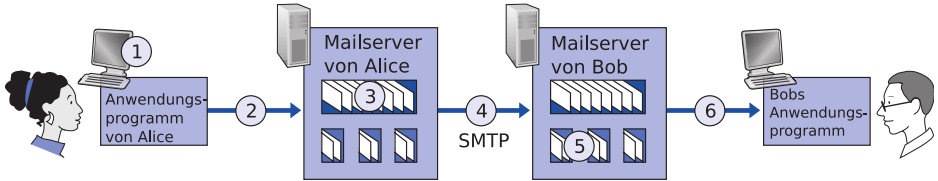
TECHNISCHE UNIVERSITÄT DARMSTADT

- Eine der ältesten Anwendungen im Internet: E-Mail
- Zentrale Komponenten:
  - E-Mail-Server (mit Postfächern für Benutzer)
  - E-Mail-Anwendungen auf Anwender- rechnern
- Server „sprechen“ SMTP miteinander



(Abbildung: [Kurose, Ross])

Zustellung einer E-Mail

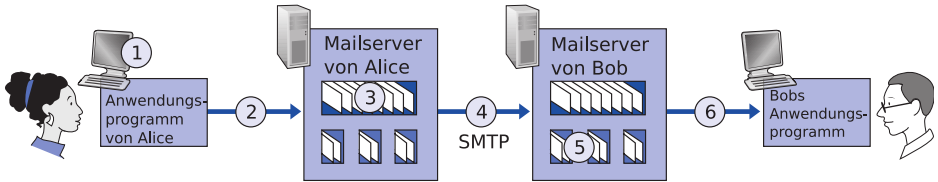


Prinzipieller Ablauf:

1. Alice verfasst ihre E-Mail auf Ihrem Rechner
2. Die E-Mail wird an Alices Mailserver übertragen
3. Dort wird sie in eine Warteschlange gestellt

(Abbildung: [Kurose, Ross])

Zustellung einer E-Mail



4. Alices Server untersucht die Zieladresse

bob@hamburger.edu

Ziel-Benutzer/Postfach — Ziel-Mailserver

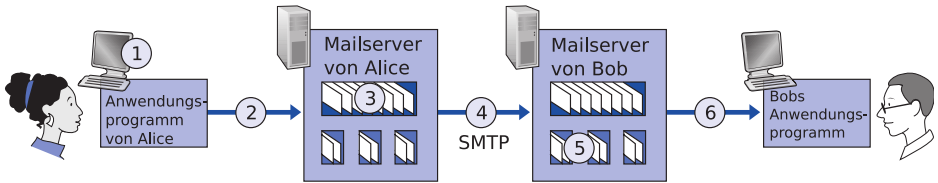
- der Ziel-Mailserver wird zu einer IP-Adresse aufgelöst (DNS MX-Record, später mehr)

…und baut eine TCP-Verbindung zu Bobs Mailserver (Portnummer 25) auf; über diese Verbindung wird mit dem SMTP-Protokoll die E-Mail übertragen

(Abbildung: [Kurose, Ross])

Zustellung einer E-Mail

TECHNISCHE UNIVERSITÄT DARMSTADT



5. Bobs Mailserver packt die Mail in Bobs Postfach

6. Von dort kann Bob sie (irgendwann später, über andere Protokolle) mit seinem E-Mail-Programm abholen

(Abbildung: [Kurose, Ross])

Simple Mail Transfer Protocol

TECHNISCHE UNIVERSITÄT DARMSTADT

- Das *Simple Mail Transfer Protocol (SMTP)* ist ein sehr altes Protokoll
- Ursprünglich definiert in RFC 821 und RFC 822, später in weiteren
- Man kann daran viele Entwurfsprinzipien erkennen, die auch in anderen Protokollen aufgegriffen wurden
- Man merkt aber auch, dass es „Altlasten“ in SMTP gibt – Dinge, die man heute sicherlich anders lösen würde
- Schauen wir uns die SMTP-„Unterhaltung“ zwischen Alices und Bobs Mailserver im Beispiel einmal genauer an

Simple Mail Transfer Protocol

TECHNISCHE UNIVERSITÄT DARMSTADT

```
220 hamburger.edu
HELO crepes.fr
250 Hello crepes.fr, pleased to meet you
```

- Farben stehen hier für Übertragungen des Servers (hier: Mailserver von Bob) bzw. des Clients (Mailserver von Alice)
- Nachdem die TCP-Verbindung aufgebaut wurde, meldet sich zunächst der Server und „grüßt“
- Wie bei vielen anderen SMTP-Nachrichten besteht die Meldung aus einem *Statuscode* (einer dreistelligen Zahl, in diesem Fall 220) und einer optionalen (!) Fließtext- Übersetzung
- Danach „grüßt“ der Client mit einer *HELO*-Nachricht, die mit Statuscode 250 (ungefähr: „alles ok“) bestätigt wird
  - heute genutzte Erweiterung: *EHLO* („extended HELO“) statt *HELO* zur Begrüßung, aktiviert Erweiterungen

Simple Mail Transfer Protocol

TECHNISCHE UNIVERSITÄT DARMSTADT

```
220 hamburger.edu
HELO crepes.fr
250 Hello crepes.fr, pleased to meet you
```

Wieso verwendet man die numerischen Statuscodes?

- Einfacher automatisch auszuwerten

Wieso dann zusätzlich die Textübersetzungen?
Wieso überhaupt Übertragung als Text (statt der binären Darstellung des Statuscodes)?

- Einfacher zu debuggen!

Wir beobachten oft Kompromisse zwischen Effizienz vs. Verständlichkeit!

Simple Mail Transfer Protocol

TECHNISCHE UNIVERSITÄT DARMSTADT

```
220 hamburger.edu
HELO crepes.fr
250 Hello crepes.fr, pleased to meet you
MAIL FROM: <alice@crepes.fr>
250 alice@crepes.fr ... Sender ok
RCPT TO: <bob@hamburger.edu>
250 bob@hamburger.edu ... Recipient ok
```

- Alices Mailserver gibt nun an, wer der Absender („MAIL FROM“) und wer der Empfänger („RCPT TO“) der E-Mail ist
- Das nennt man auch den *Envelope* der Nachricht
- Beides wird von Bobs Server jeweils mit einem Statuscode 250 bestätigt

Simple Mail Transfer Protocol

TECHNISCHE UNIVERSITÄT DARMSTADT

```
220 hamburger.edu
HELO crepes.fr
250 Hello crepes.fr, pleased to meet you
MAIL FROM: <alice@crepes.fr>
250 alice@crepes.fr ... Sender ok
RCPT TO: <bob@hamburger.edu>
250 bob@hamburger.edu ... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
Do you like ketchup?
How about pickles?
.
250 Message accepted for delivery
```

- Jetzt folgt die Übertragung der eigentlichen Mail, eingeleitet mit „DATA“ und abgeschlossen mit einer Zeile, die nur einen Punkt enthält

Ist das mit dem einzelnen Punkt eine gute Idee?

Simple Mail Transfer Protocol

TECHNISCHE UNIVERSITÄT DARMSTADT

```
220 hamburger.edu
HELO crepes.fr
250 Hello crepes.fr, pleased to meet you
MAIL FROM: <alice@crepes.fr>
250 alice@crepes.fr ... Sender ok
RCPT TO: <bob@hamburger.edu>
250 bob@hamburger.edu ... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
Do you like ketchup?
How about pickles?
.
250 Message accepted for delivery
QUIT
221 hamburger.edu closing connection
```

- Zum Schluss signalisiert Alices Server, dass die Verbindung geschlossen werden soll
- Alternativ könnte eine weitere Mail übertragen werden

Kommandos

TECHNISCHE UNIVERSITÄT DARMSTADT

In SMPT gibt es (natürlich…) noch sehr viel mehr Kommandos und Antwort-Statuscodes, als wir eben gesehen haben. Zum Beispiel:

Kommando	Beschreibung
DATA	Begins message composition
EXPN <string>	Returns names on the specified mail list
HELO <domain>	Returns identity of mail server
HELP <command>	Returns information on the specified command
MAIL FROM <host>	Initiates a mail session from host
QUIT	Terminates the mail session
RCPT TO <user>	Designates who receives mail
RSET	Resets mail connection
SAML FROM <host>	Sends mail to user terminal and mailbox
SEND FROM <host>	Sends mail to user terminal
SOML FROM <host>	Sends mail to user terminal or mailbox
TURN	Switches role of receiver and sender
VRFY <user>	Verifies the identity of a user

Antworten

TECHNISCHE UNIVERSITÄT DARMSTADT

Statuscode	Beschreibung
211	(Response to system status or help request)
214	(Response to help request)
220	Mail service ready
221	Mail service closing connection
250	OK, Mail transfer completed
251	User not local, forward to <path>
354	Start mail message, end with <CRLF>.<CRLF>
421	Mail service unavailable
450	Mailbox unavailable
451	Local error in processing command
452	Insufficient system storage

Antworten

TECHNISCHE UNIVERSITÄT DARMSTADT

Statuscode	Beschreibung
500	Unknown command
501	Bad parameter
502	Command not implemented
503	Bad command sequence
504	Parameter not implemented
550	Mailbox not found
551	User not local, try <path>
552	Storage allocation exceeded
553	Mailbox name not allowed
554	Mail transaction failed

Mail-Header

TECHNISCHE UNIVERSITÄT DARMSTADT

- In der normalen Briefpost gibt es im Kopf von Briefen Raum für alle möglichen „Header-Informationen“, z. B.
  - Absenderadresse
  - Datum
  - Geschäftszeichen
  - …
- Das gibt es auch bei einer E-Mail: Dem eigentlichen Text geht ein Header aus mehreren Kopfzeilen voran
- Kopfzeilen beginnen mit einem Schlüsselwort, dann folgen ein Doppelpunkt, ein Leerzeichen und schließlich der Wert
- Welche Kopfzeilen es gibt, ist in RFC 822 festgelegt (plus jede Menge spätere Erweiterungen)

Mail-Header

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Kopfzeilen „From:“ und „To:“ sind Pflicht, viele weitere sind optional möglich
- Fast immer werden zumindest „Subject:“ und „Date:“ verwendet
- Dem Header-Teil folgt eine Leerzeile, danach dann der eigentliche Text

Beispiel

```
From: Alice <alice@crepes.fr>
To: bob@hamburger.edu
Subject: Searching for the meaning of life

Any ideas?
```

Vorsicht: Verwechseln Sie nicht Header und Envelope!

Received-Header

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wenn ein Mailserver eine E-Mail empfängt, fügt er eine `Received:`-Kopfzeile hinzu
- Mit diesen Kopfzeilen kann der Empfänger den Weg der Nachricht nachvollziehen
- Eine E-Mail könnte dann beispielsweise so aussehen:

Beispiel

```
Received: from crepes.fr by hamburger.edu; 12 Oct 21 15:27:39 GMT
From: Alice <alice@crepes.fr>
To: bob@hamburger.edu
Subject: Hungry?

Want to get something to eat?
```

Versand von Nicht-ASCII-Daten

TECHNISCHE UNIVERSITÄT DARMSTADT

- E-Mail unterstützt ursprünglich nur die Übertragung von Daten im 7-Bit-ASCII-Zeichensatz
- Das ist sehr eingeschränkt – im Wesentlichen (englischsprachiger) Text
- …ohne Formatierungen, nicht einmal Sonderzeichen wie etwa deutsche Umlaute
- Zur Zeit der Entstehung von SMTP war das vielleicht ok – heute reicht es nicht mehr
- Wir wollen heute:
  - einen größeren Zeichenvorrat
  - E-Mails mit Anhängen
  - …
- Aber man wollte (bzw. brauchte) auch Abwärtskompatibilität – also müssen bis heute alle E-Mail-Übertragungen 7-Bit-ASCII verwenden

Und nun?

Erweiterungsmöglichkeiten in SMTP

TECHNISCHE UNIVERSITÄT DARMSTADT

- Es war notwendig, *im Rahmen des existierenden SMTP-Standards* Erweiterungen einzuführen
- …die aber zugleich *für existierende SMTP-Server transparent sind*
- Klare Trennung zwischen „Nachrichtenübertragung“ (Envelope) und „Nachrichteninhalt“ (Message) sowie die erweiterbare Header-Struktur haben dabei sehr geholfen
- Unbekannte Headerzeilen werden von (auch älterer) Mail-Software ignoriert, aber weitergeleitet

Postel's Law

TECHNISCHE UNIVERSITÄT DARMSTADT

- Hier erkennt man einen wichtigen Design-Grundsatz, den man in sehr vielen (guten) Protokollen wiederfindet:

„Be conservative in what you do, be liberal in what you accept from others"

- Manchmal auch wiedergegeben als: *„Be conservative in what you send, be liberal in what you accept"*
- Eingeführt in RFC 760 von Jon Postel (Autor unter anderem auch des ursprünglichen SMTP-RFCs)
- Auch bekannt als *Robustness Principle* oder *Postel's Law*

Multipurpose Internet Mail Extensions (MIME)

TECHNISCHE UNIVERSITÄT DARMSTADT

- MIME (RFCs 2045, 2046) nutzt die Erweiterbarkeit durch zusätzliche Header aus; die wichtigsten:
  - `Content-Type`: beschreibt die Art der enthaltenen Daten (z. B. `image/jpeg`)
  - `Content-Transfer-Encoding`: gibt an, wie diese dargestellt werden (z. B. `base64`)
- MIME-E-Mails können aus mehreren Teilen bestehen
  - Teile können selbst wieder (potentiell weiter verschachtelte) MIME-Nachrichten sein

Beispiel für eine MIME-E-Mail

```
From: Alice <alice@crepes.fr>
To: bob@hamburger.edu
Subject: Picture of yummy crepe
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64-codierte Daten.............
```

Mailzugriffsprotokolle

TECHNISCHE UNIVERSITÄT DARMSTADT

- SMTP deckt nur die Zustellung der E-Mail bis zum Mailserver des Empfängers ab
- Für die Übertragung zwischen dem Postfach und dem E-Mail-Programm des Empfängers werden andere Protokolle verwendet
  - POP3 („Post Office Protocol v.3“, RFC 1939) ist ein einfaches Protokoll, zur Abholung einer E-Mail, wobei sie vom Server gelöscht wird
  - IMAP („Internet Mail Access Protocol“, RFC 3501) bietet sehr viel mehr Funktionen, kann z.B. die E-Mails auf dem Server belassen und sie so universeller zugreifbar machen

Das WWW und HTTP

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die wahrscheinlich wichtigste Internet-Anwendung heute ist das World Wide Web
- Im Folgenden schauen wir uns das Protokoll an, das dem WWW zugrunde liegt: das *HyperText Transfer Protocol (HTTP)*
- Wichtigste Aufgabe von HTTP: Webseiten von Webservern an Browser übertragen
- Spezifiziert in RFCs 1945 (HTTP/1.0) und 2616 bzw. 7230–7235 (HTTP/1.1)
- Nachfolger: HTTP/2 in RFC 7540, mit deutlich anderer Grundstruktur

Terminologie: HTTP, Objekte und HTML

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bevor wir die Details des Protokolls anschauen, erst ein bisschen Terminologie
- Eine Webseite besteht aus *Objekten* (HTML-Dateien, Bildern, Audio/Video-Dateien,...)
- Typischerweise (mindestens) ein HTML-Objekt
- HTML (HyperText Markup Language) ist eine Spezifikationssprache für formatierte Dokumente mit Text, eingebetteten Bildern, Links,...
- Die Aufgabe eines Browsers (= HTTP-Client) ist das Herunterladen von solchen Dokumenten und das Anzeigen von HTML-Dokumenten
- Für das Herunterladen wird HTTP verwendet, das wir nun näher betrachten

HTTP: Terminologie

TECHNISCHE UNIVERSITÄT DARMSTADT

- *Wie* ein Web-Objekt (also z. B. eine HTML-Seite) angezeigt wird, ist unabhängig vom Protokoll – HTTP kümmert sich nur um die Übertragung
- Wie HTML im Detail funktioniert, interessiert uns deshalb hier nicht (aber: gerne mal selbst anschauen!)
- Grundschema: eine Basis-HTML-Datei enthält Verweise auf andere Objekte
- In HTTP ist jedes Objekt über eine *URL* (Uniform Resource Locator) adressierbar, z. B.

```
https://www.etit.tu-darmstadt.de/fachbereich/index.de.jsp
```

HTTP: Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

Angenommen wir rufen die URL

`http://www.etit.tu-darmstadt.de/fachbereich/index.de.jsp`

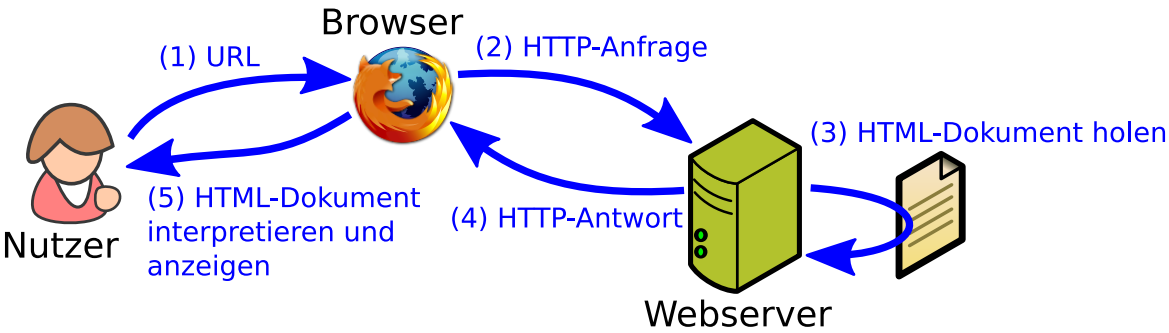
Name des Servers — Pfad auf dem Server

auf. Was passiert?

1. Der Browser analysiert die URL und besorgt sich die zum Servernamen gehörige IP-Adresse
2. Der Browser baut eine TCP-Verbindung zu dieser IP-Adresse auf, normalerweise zu Port 80
3. Der Server nimmt diese Verbindung an
4. Über die TCP-Verbindung kann der Browser dann eine HTTP-Anfrage senden, die der Server beantwortet
5. Schließlich wird die TCP-Verbindung wieder geschlossen

Laden eines HTML-Dokuments

TECHNISCHE UNIVERSITÄT DARMSTADT



Komplexere Webseiten

TECHNISCHE UNIVERSITÄT DARMSTADT

- Heute bestehen praktisch alle Webseiten aus mehr als einem Objekt
- Diese Objekte sind (möglicherweise mehrfach verschachtelt) in ein Hauptdokument eingebettet
- Der Browser empfängt und verarbeitet zunächst dieses Hauptdokument
- Dabei erfährt er, welche weiteren Objekte er laden muss
- Die besorgt er dann mit separaten, unabhängigen HTTP-Anfragen
- Das wird fortgesetzt, bis der Browser alle benötigten Objekte geladen hat

HTTP-Anfragen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eine HTTP-Anfrage (Request) beginnt mit einer Anfragezeile, die die verwendete *Methode* (GET, POST, HEAD, PUT, DELETE) enthält

```
GET /fachbereich/index.de.jsp HTTP/1.1
```

- „Methoden“ heißen in HTTP die verschiedenen Arten von Anfragen
- GET-Anfragen sind bei weitem die häufigsten; sie werden verwendet, um Dokumente von einem Server abzurufen
- Jede HTTP-GET-Anfrage holt *genau ein* Objekt
- Außer der Methode gibt die Anfragezeile an, welches Dokument auf dem Server angerufen wird und welche HTTP-Protokollversion verwendet werden soll

HTTP-Anfragen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Der Anfragezeile können noch HTTP-Header-Zeilen folgen, z. B.

```
Host: www.etit.tu-darmstadt.de
User-agent: Mozilla/5.0 (X11; Linux i686; rv:99.0) \
  Gecko/20100101 Firefox/99.0
Connection: keep-alive
...
```

- Eine leere Zeile zeigt das Ende einer HTTP-Anfrage an

HTTP-Antworten

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eine HTTP-Antwort beginnt mit einer *Statuszeile* (Protokollversion + Statuscode + Beschreibung)
  ```
  HTTP/1.1 200 OK
  ```
- Es folgen wieder Headerzeilen
  ```
  Content-Type: text/html; charset=UTF-8
  Date: Thu, 24 Oct 2021 11:14:24 GMT
  Connection: keep-alive
  ...
  ```
- Dann eine Leerzeile
- Und schließlich das eigentliche angefragte Objekt
- Parallelen zu SMTP (und vielen anderen Internet-Protokollen) sind unübersehbar
- Aber einige Dinge sind anders gelöst als in SMTP
  - z. B. keine Einschränkung des Zeichensatzes (zum Glück!)

HTTP-Statuscodes

TECHNISCHE UNIVERSITÄT DARMSTADT

Ein paar Beispiele für HTTP-Statuscodes:

- *200 OK* – Alles in Ordnung
- *400 Bad Request* – Der Server hat die Anfrage nicht verstanden
- *404 Not Found* – Das angefragte Objekt wurde auf dem Server nicht gefunden
- *505 HTTP Version Not Supported*
- ...

Header-Zeilen

TECHNISCHE UNIVERSITÄT DARMSTADT

Es gibt sehr viele mögliche Kopfzeilen in HTTP-Anfragen und -Antworten. Ein paar Beispiele haben wir oben schon gesehen:

```
GET /fachbereich/index.de.jsp HTTP/1.1
Host: www.etit.tu-darmstadt.de
User-agent: Mozilla/5.0 (X11; Linux i686; rv:99.0)\
  Gecko/20100101 Firefox/99.0
Connection: keep-alive
```

- *Host:* Von welchem Webserver wird etwas angefordert
  - der Host-Header ist Pflicht seit der HTTP-Version HTTP/1.1

Header-Zeilen

TECHNISCHE UNIVERSITÄT DARMSTADT

Es gibt sehr viele mögliche Kopfzeilen in HTTP-Anfragen und -Antworten. Ein paar Beispiele haben wir oben schon gesehen:

```
GET /fachbereich/index.de.jsp HTTP/1.1
Host: www.etit.tu-darmstadt.de
User-agent: Mozilla/5.0 (X11; Linux i686; rv:99.0)\
  Gecko/20100101 Firefox/99.0
Connection: keep-alive
```

- *User-Agent:* Welcher Browser wird in welcher Version verwendet
  - erlaubt es dem Server, evtl. speziell angepasste Versionen der Objekte auszuliefern

Header-Zeilen

TECHNISCHE UNIVERSITÄT DARMSTADT

Es gibt sehr viele mögliche Kopfzeilen in HTTP-Anfragen und -Antworten. Ein paar Beispiele haben wir oben schon gesehen:

```
GET /fachbereich/index.de.jsp HTTP/1.1
Host: www.etit.tu-darmstadt.de
User-agent: Mozilla/5.0 (X11; Linux i686; rv:99.0)\
  Gecko/20100101 Firefox/99.0
Connection: keep-alive
```

- *Connection: keep-alive* bedeutet, dass der Client gerne möchte, dass der Server nach dem Beantworten dieser Anfrage die TCP-Verbindung offen lässt (*persistentes HTTP*)
- Andernfalls würde der Server nach Beantworten der Anfrage die Verbindung schließen
  - das war das Standardverhalten im ursprünglichen HTTP
- Über den Sinn von persistentem HTTP reden wir noch ausführlicher

Header-Zeilen

TECHNISCHE UNIVERSITÄT DARMSTADT

Anfang der HTTP-Antwort:

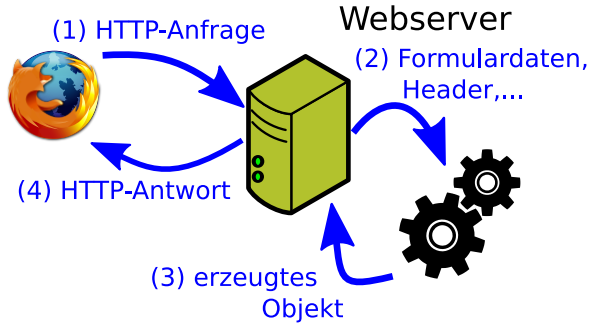
```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 3712
Date: Thu, 24 Okt 2021 11:14:24 GMT
Connection: keep-alive
```

- *Content-Type:* Welche Art von Daten ist enthalten (hier auch: welcher Zeichensatz)
  - 7-Bit-Codierung wie in SMTP ist bei HTTP nicht nötig
- *Content-Length:* Wie viele Bytes enthält der Datenteil?
- *Date:* Wann wurde die Anfrage beantwortet?
  - *nicht* wann wurde das Dokument erstellt/geändert!
- *Connection: keep-alive* – der Server bestätigt die Verwendung von Persistenz

Dynamisch erzeugte Webseiten

TECHNISCHE UNIVERSITÄT DARMSTADT

- Statt eine feste Datei auszuliefern kann ein Webserver ein Objekt auch dynamisch erzeugen
- Dafür: Übergabe der Daten zum HTTP-Request an ein (beliebiges) Programm, das im Gegenzug das Objekt generiert
- Eine Standard-Schnittstelle dafür: Common Gateway Interface (CGI) (RFC 3875)
- Alternativ können Webseiten auch Client-seitig dynamisch erzeugt werden, z. B. durch JavaScript



Hochladen von Formulardaten: GET vs. POST

TECHNISCHE UNIVERSITÄT DARMSTADT

- Häufig kann man auf einer Webseite irgendwo Daten eingeben
- Diese Daten müssen dann mit einer HTTP-Anfrage an den Server geschickt werden
  - ... um dort beispielsweise von PHP verarbeitet zu werden
- Dafür gibt es zwei unterschiedliche Mechanismen:
  1. die Daten können in einer GET-Anfrage in den angefragten Pfad hineincodiert werden

Beispiel für eine GET-Anfrage mit Formulardaten

```
GET /ein/pfad?a=irgendwas&b=wasanderes HTTP/1.1
Host: www.meineseite.de
...
```

Hochladen von Formulardaten: GET vs. POST

TECHNISCHE UNIVERSITÄT DARMSTADT

- Häufig kann man auf einer Webseite irgendwo Daten eingeben
- Diese Daten müssen dann mit einer HTTP-Anfrage an den Server geschickt werden
- Dafür gibt es zwei unterschiedliche Mechanismen:
  1. die Daten können in einer GET-Anfrage in den angefragten Pfad hineincodiert werden
  2. es kann die POST-Methode verwendet werden; dabei stehen die Formulardaten im Datenteil der HTTP-Anfrage

Beispiel für eine POST-Anfrage mit Formulardaten

```
POST /login.jsp HTTP/1.1
Host: www.meineseite.de
Content-Length: 28
Content-Type: application/x-www-form-urlencoded

userid=alice&password=geheim
```

Zustandslosigkeit

TECHNISCHE UNIVERSITÄT DARMSTADT

HTTP ist ein *zustandsloses* Protokoll: Der Server merkt sich keine Informationen über vorherige Anfragen eines Clients

Wieso kann es sinnvoll sein, ein Protokoll zustandslos zu entwerfen?

- Protokolle, die Zustand verwalten, sind komplex!
  - der Zustand muss gespeichert und verwaltet werden
  - wenn Server oder Client abstürzen, muss der Zustand wieder synchronisiert werden
  - ...

Zustandslosigkeit macht ein Protokoll also viel einfacher zu implementieren und deutlich weniger fehleranfällig

Zustandsloses HTTP?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Trotz Zustandslosigkeit sehen wir, dass Web-Anwendungen (die ja auf HTTP aufbauen) häufig Zustand halten
  - eine Anmeldung bleibt auch für das Laden der nächsten Seite gültig, Webshops speichern den momentanen Inhalt des Einkaufswagens,...

Wie kann das funktionieren?

- Nicht der HTTP-Server (im engeren Sinne) erkennt den Zusammenhang zwischen den Anfragen
- Aber andere Mechanismen (z. B. vom Webserver zum Erzeugen von dynamischen HTTP-Objekten aufgerufene Programme) „erkennen“ einen Zusammenhang zwischen mehreren HTTP-Anfragen
- Die Kunst: Schlüssel für den Zugriff auf Zustandsinformationen über ein zustandsloses Protokoll transportieren

HTTP-Cookies

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eine bekannte Möglichkeit sind sogenannte *Cookies*
- Ein HTTP-Server kann mit einer HTTP-Antwort einen Cookie „mitliefern“
- Dafür wird ein spezieller HTTP-Header verwendet:

  `Set-Cookie: irgendeincookietext`
- Der Browser sieht diesen Header und speichert den Cookie für diese Webseite
- Jedesmal, wenn zukünftig eine Webseite von diesem Server angefordert wird, wird der Cookie mitgeschickt

  `Cookie: irgendeincookietext`
- Der HTTP-Server kann den Cookie an die Webanwendung weitergeben; sie kann so den Benutzer wiedererkennen

HTTP-Cookies

TECHNISCHE UNIVERSITÄT DARMSTADT

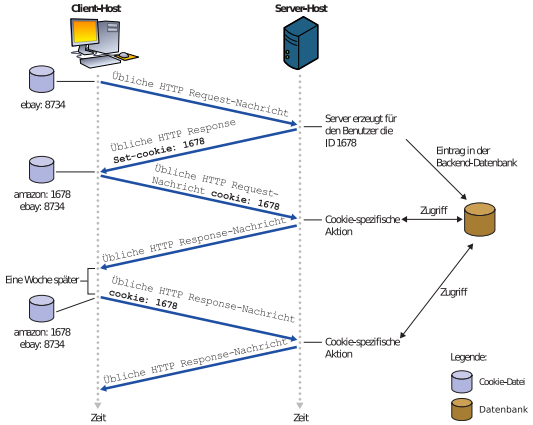
- Die Informationen werden also im Client gespeichert

Der HTTP-Server (nicht zwangsläufig die Web-Anwendung „dahinter“!)
ist also weiterhin zustandslos

Typische Anwendung:

- Die Web-Anwendung wählt für jeden neuen Benutzer eine eindeutige ID als Cookie
- Alle HTTP-Anfragen dieses Benutzers können so als zusammengehörig erkannt werden
- Der Benutzer kann auf seinem „Weg“ durch die Webseite „verfolgt“ werden

HTTP-Cookies – Beispiel



(Abbildung: [nach Kurose, Ross])

Round Trip Times

TECHNISCHE UNIVERSITÄT DARMSTADT

- Für das Folgende ist ein kurzer Vorgriff auf das Kapitel zu Transportschichtprotokollen notwendig
- Bevor über eine TCP-Verbindung Daten übertragen werden können, muss die Verbindung zunächst aufgebaut werden
- Dafür muss zunächst ein Paket vom Client zum Server, danach eine Antwort zurück geschickt werden
  - genauer gesagt gibt es noch ein drittes Paket – mehr dazu später in der Vorlesung; für den Moment reicht die einfache Sicht
- Erst nach Abschluss dieses „Handshakes“ können Daten über TCP übertragen werden
- Dieser Verbindungsaufbau kostet Zeit: Die Laufzeit für ein Paket einmal hin und zurück
  - diese (natürlich nicht konstante) Dauer nennt man die *Round Trip Time (RTT)* einer Verbindung

Latenzen und Übertragungszeiten

TECHNISCHE UNIVERSITÄT DARMSTADT

Was schätzen Sie: Wie groß sind typische RTTs im Internet?

- Innerhalb Deutschlands: ca. 5–20 ms
- Innerhalb Europas: bis ca. 50 ms
- Nach Nordamerika: ca. 100–150 ms
- Weltweit: bis ca. 500 ms

Wie stark lässt sich das durch schnellere Technik verringern?

- Kaum – denn kein Netzwerkpaket kann schneller unterwegs sein als mit Lichtgeschwindigkeit!
- Nach Nordamerika und zurück (ca. 16 000 km) sind deshalb 53 ms eine absolute untere Schranke

„No matter how hard you push and no matter what the priority,
you can't increase the speed of light." (RFC 1925)

Latenzen und Übertragungszeiten

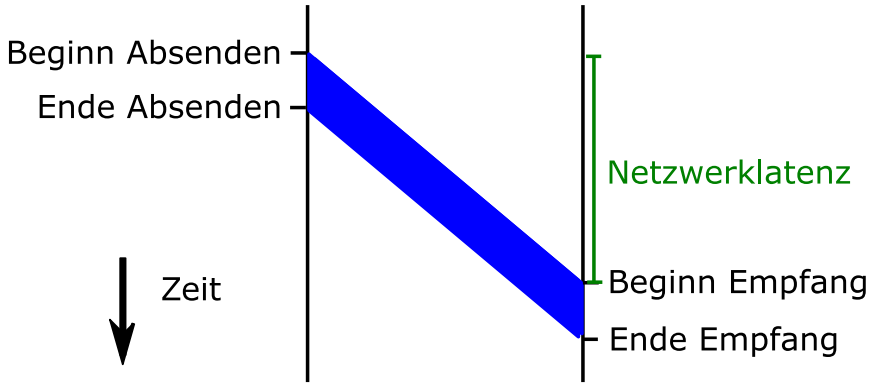
TECHNISCHE UNIVERSITÄT DARMSTADT

Wie lange dauert das Absenden eines 10 KB großen Objektes über eine VDSL-Leitung mit 80 MBit/s Übertragungsrate?

$$\frac{10\,\text{KB}}{80\,\text{MBit/s}} = 1\,\text{ms}$$

- Das Absenden (bzw. Empfangen) des Objektes (Übertragungszeit) braucht also nur einen Bruchteil der Reisezeit durch das Netz (Latenz)!
- Auch Verarbeitungszeiten (in Client und Server oder auch in den Zwischenstationen bei der Paketweiterleitung) sind im Vergleich vernachlässigbar klein

Latenzen und Übertragungszeiten

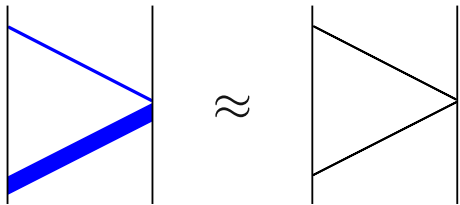


Performanz von HTTP

TECHNISCHE UNIVERSITÄT DARMSTADT

Was bedeutet das für Protokolle wie HTTP?

- Web-Objekte haben meist eine (im Vergleich zu typischen Datenraten) überschaubare Größe
- Die Übertragungszeiten sind also eher kurz
- Die Latenzen dominieren die Wartezeiten bis zum Eintreffen einer Webseite



Performanz von HTTP

TECHNISCHE UNIVERSITÄT DARMSTADT

- Heutige Webseiten können aus Dutzenden von Objekten bestehen – Wartezeiten, bis alle Teile abgeholt sind, werden trotz hoher Datenraten sehr lang

Was muss man tun, um die Performanz von Protokollen wie HTTP zu verbessern?

- Wie bereits gesehen können wir die Latenz (und damit die RTT) nicht beliebig verringern
- Schlussfolgerung:

Interaktive Protokolle wie HTTP müssen die Zahl der notwendigen RTTs gering halten!

HTTP-Performanz

TECHNISCHE UNIVERSITÄT DARMSTADT

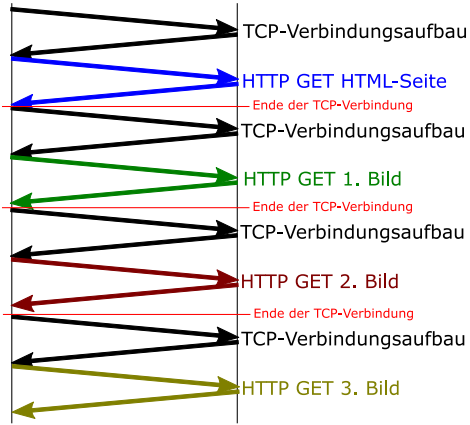
- Angenommen, ein Browser ruft eine Webseite mit drei kleinen eingebetteten Objekten von einem Server (mit klassischem, nicht-persistentem HTTP/1.0) ab

Wie lange dauert das (gemessen in RTTs)?

- Jedes Objekt in einer separaten TCP-Verbindung
- Verbindungsaufbau benötigt jeweils 1 RTT
- Dann nochmals 1 RTT bis zum Eintreffen des angeforderten Objektes (für HTTP-Anfrage und -Antwort)
- Also: $4\ \text{Objekte} \cdot 2\frac{\text{RTTs}}{\text{Objekt}} = 8\,\text{RTTs}$

HTTP-Performanz

TECHNISCHE UNIVERSITÄT DARMSTADT



Persistentes HTTP

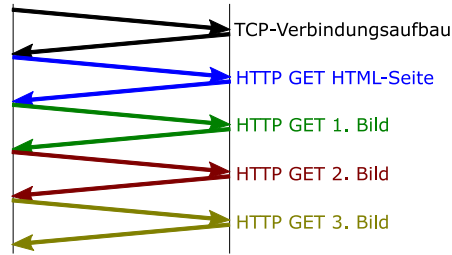
TECHNISCHE UNIVERSITÄT DARMSTADT

- Als Webseiten komplexer wurden, hat man das Problem erkannt und HTTP so erweitert, dass RTTs eingespart werden können
- Idee hinter Persistentem HTTP: nicht für jeden Request eine neue TCP-Verbindung öffnen
- Erfordert entsprechende Funktionen im HTTP-Protokoll ($\Rightarrow$ `Connection: keepalive`)
- Ursprüngliches HTTP: Server schließt Verbindung nach Ende des Requests, Client schließt danach ebenfalls
- Spätere Erweiterung: Client und Server können sich darauf einigen, die Verbindung offen zu halten

Persistentes HTTP

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ohne Persistenz: 8 RTT
- Mit Persistenz:
  - Aufbau: 1 RTT
  - Je Objekt 1x Request+Response:
    $4 \cdot 1\,\text{RTT} = 4\,\text{RTT}$
  - Insgesamt: 5 RTT



Da geht doch bestimmt noch was? ;-)

Anfragen über persistente Verbindungen

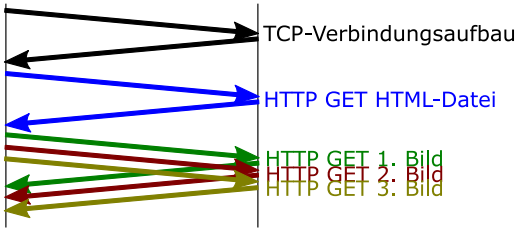
TECHNISCHE UNIVERSITÄT DARMSTADT

- Wie arbeitet der Server bei persistenten Verbindungen:
  - liest einen Request von der TCP-Verbindung
  - beantwortet ihn
  - liest den nächsten Request
  - usw.
- Es macht für den Server keinen Unterschied, wann die gelesenen Requests abgeschickt wurden!
- Das Warten auf die Antwort zu früheren Requests ist also per se nicht notwendig

Pipelining

TECHNISCHE UNIVERSITÄT DARMSTADT

- Der Browser kann also über dieselbe Verbindung Folge-Requests sofort absenden und dadurch weitere RTTs sparen
- Das nennt man *Pipelining*
- Ergebnis: weitere Zeitersparnisse möglich – im Beispiel:
  - 1x Verbindungsaufbau
  - 1x HTTP-Request/Response für die HTML-Seite
  - dann ohne Warten auf weitere Antworten direkt drei Requests
  - zusammen: 3 RTTs
- Potentielles Problem: *Head-of-Line Blocking*
  - Verzögerung durch noch nicht ausgelieferte Objekte



Parallele TCP-Verbindungen

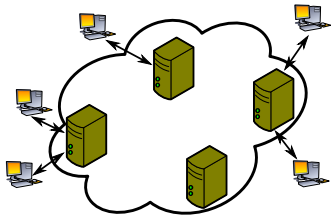
TECHNISCHE UNIVERSITÄT DARMSTADT

- Mehrere parallele TCP-Verbindungen können genutzt werden, um mehrere Objekte gleichzeitig abzurufen
- Aber: Diese Verbindungen müssen sich natürlich die Netzwerkkapazität teilen – Gewinne sind dadurch begrenzt
- Zusätzliche Verbindungen erzeugen zusätzliche Last auf dem Server
- IETF-Standards für HTTP/1.1 empfehlen: nicht mehr als zwei parallele Verbindungen (RFC 2616)
- Browser halten sich allerdings nicht daran
- Schwierige Optimierungsaufgabe für Browserprogrammierer: Wann wie viele parallele Verbindungen, und welche Objekte in welcher Reihenfolge über welche Verbindung holen?

Content Distribution Networks

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wegen der wichtigen Rolle, die die Länge der RTT für die Performanz von Web-Angeboten spielt, sind Inhaltsanbieter stark an niedrigen RTTs interessiert
- Ziel deshalb: Webserver „nahe zu den Browsern“ bringen
  - Idee:
    - *mehrere* Server mit (i. d. R.) identischen Inhalten
    - Nutzer werden zum nächstgelegenen geleitet
  - Das nennt man ein *Content Distribution Network (CDN)*
  - Weit verbreitetes Konzept, mehrere kommerzielle Anbieter
    - Akamai, Limelight, Amazon CloudFront, CDNetworks,...



Content Distribution Networks

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wie entscheidet man, was die „beste“ Kopie für eine gegebene Benutzer-IP-Adresse ist?
- Hier sind viele Kriterien möglich – Geschäftsgeheimnis der CDN-Firmen :-)
  - erwartete RTT
  - Netzwerk- und/oder Serverlast
  - Kosten der jeweiligen Verbindung (Peering-/Transit-Vereinbarungen!)
  - . . .
- Wenn man einen geeigneten Server identifiziert hat, leitet man den anfragenden Browser dahin um
  - es gibt mehrere Möglichkeiten, das technisch zu realisieren

Caching

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eventuell lässt sich die Zeit für eine Anfrage ja manchmal vollständig vermeiden?
- Idee: HTTP-Clients können Objekte von Webservern in einem lokalen *Cache* zwischenspeichern
- Wenn der Benutzer dann dieselbe Seite noch einmal besucht, müssen sie nicht erneut übertragen werden

Aber: Woher soll der Browser wissen, ob die Version des Objektes, die im Cache liegt, noch aktuell ist?

Expires-Header

TECHNISCHE UNIVERSITÄT DARMSTADT

- Beim Ausliefern eines Objektes kann ein Server einen Zeitpunkt angeben, bis zu dem das Objekt gültig bleiben soll
- Bis dahin können Anfragen nach diesem Objekt also vollständig vermieden werden
- Der `Expires`-Header gibt einen Zeitpunkt an, bis zu das Objektes sicher verwendet werden kann
- Problem: Das erfordert, dass der Server weiß, wann sich das Objekt zum nächsten Mal ändern wird...
- Um dieses Problem zu vermeiden, gibt es noch einen zweiten Mechanismus in HTTP

Caching: Bedingtes GET

TECHNISCHE UNIVERSITÄT DARMSTADT

- Angenommen, ein Client fragt folgendes Objekt an:

```
GET /ein/pfad.jpg HTTP/1.1
Host: www.einserver.de
```

- Dann kann der Server beim Beantworten einer Anfrage den „Stand“ eines Objektes mit einer speziellen Header-Zeile angeben:

```
HTTP/1.1 OK
Date: Thu, 15 Nov 2021 10:56:12
Last-Modified: Thu, 14 Nov 2012 14:44:32
Content-Type: image/jpeg
```

Caching: Bedingtes GET

TECHNISCHE UNIVERSITÄT DARMSTADT

- Der Client kann nun eine GET-Anfrage stellen, die das Objekt nur dann überträgt, wenn es sich gegenüber der Version im Cache geändert hat

```
GET /ein/pfad.jpg HTTP/1.1
Host: www.einserver.de
If-modified-since: Thu, 14 Nov 2012 14:44:32
```

- If-modified-since entspricht genau der Header-Zeile Last-Modified aus der vorangegangenen HTTP-Antwort
- Falls sich nichts geändert hat, überträgt der Server das Objekt nicht erneut, sondern antwortet mit Statuscode 304:

```
HTTP/1.1 304 Not Modified
Date: Thu, 15 Nov 2021 11:03:12
```

Last-modified: Performanzimplikationen

TECHNISCHE UNIVERSITÄT DARMSTADT

Wie viel kann man mit bedingten GETs gewinnen?

- If-modified-since kann die übertragene Datenmenge (und damit Übertragungszeiten) reduzieren
- Es verringert aber *nicht* die Zahl notwendiger RTTs!

SSL/TLS und HTTPS

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die unverschlüsselte Übertragung von Webseiten (und Formulardaten) ist in vielen Fällen keine gute Idee
  - Anmeldedaten
  - Zahlungsmitteldaten
  - Banking
  - ...
- Standardtechnik zum Schutz von HTTP-Verbindungen: SSL/TLS bzw. HTTP*S*
- Eine auch nur halbwegs vollständige Behandlung des Themas ist hier zeitlich nicht möglich
- ...aber wenigstens die Grundzüge schauen wir uns an

Kryptographie

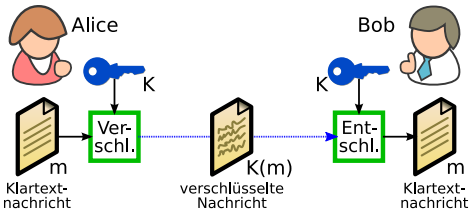
TECHNISCHE UNIVERSITÄT DARMSTADT

- Dafür zunächst nötig: ein grober (und sehr anwendungsorientierter) Überblick über kryptographische Grundlagen
- Das Folgende soll und kann keine Kryptographie-Vorlesung ersetzen
- Wir schauen nur (sehr kurz, sehr oberflächlich, sehr unvollständig), *was* man mit unterschiedlichen Standardtechniken („kryptographische Primitive“) machen kann
- Bewusst nicht: *Wie* funktioniert die Kryptographie dahinter
- Danach: Wie wird das im Internet (und speziell im WWW) eingesetzt

Symmetrische Verschlüsselung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bei *symmetrischen* Kryptoverfahren kennen Sender und Empfänger beide denselben *Schlüssel*
- Der Schlüssel ist eine Bitfolge fester Länge (z. B. 128 Bit)
- Nachrichten, die mit Schlüssel *K* verschlüsselt wurden, können nur mithilfe von *K* wieder entschlüsselt werden

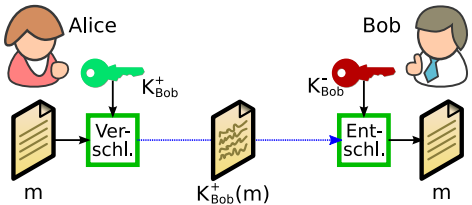


- Beispiele für symmetrische Verschlüsselungsalgorithmen: AES, Blowfish, Serpent, Twofish; historisch: DES, RC4

Asymmetrische Verschlüsselung

TECHNISCHE UNIVERSITÄT DARMSTADT

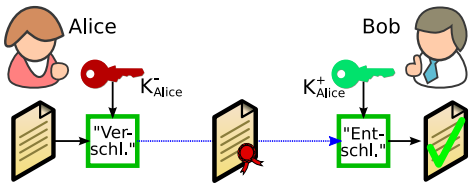
- Separate Schlüssel für alle *Paare* von Teilnehmern sind oft nicht möglich
- Dann helfen *asymmetrische* („Public Key“) Verfahren (z. B. RSA, ElGamal, verschiedene EC-Algorithmen,...)
- Jeder Benutzer hat einen *privaten* und einen *öffentlichen* Schlüssel (hier mit K^- und K^+ bezeichnet)
- Was mit dem öffentlichen Schlüssel verschlüsselt wurde, kann nur mit dem privaten wieder entschlüsselt werden



Digitale Unterschriften

TECHNISCHE UNIVERSITÄT DARMSTADT

- Public-Key-Kryptographie lässt sich „umgekehrt“ anwenden: Was mit dem privaten Schlüssel „verschlüsselt“ wurde, kann nur mit dem passenden öffentlichen Schlüssel „entschlüsselt“ werden
- Das ermöglicht Absenderauthentifizierung und digitale Signaturen: Die Nachricht kann nur vom Besitzer des privaten Schlüssels „unterschrieben“ worden sein

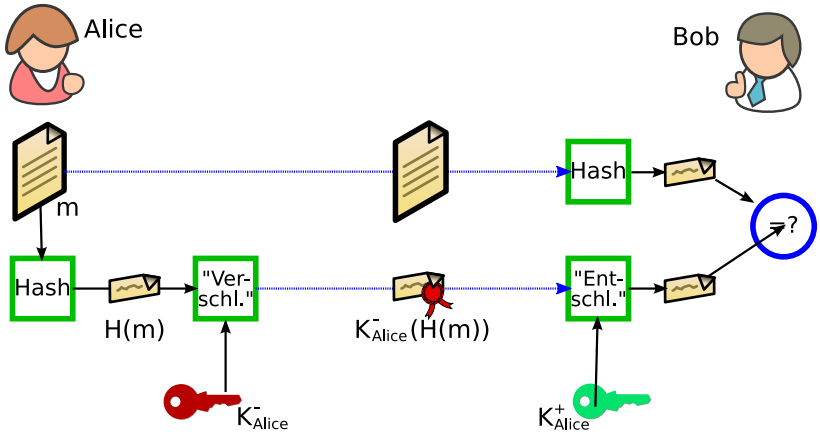


Kryptographische Hashfunktionen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Public-Key-Kryptographie ist rechenaufwändig und deshalb *langsam*
- Techniken für digitale Unterschriften werden daher oft mit kryptographischen Hashfunktionen kombiniert
- Eine Hashfunktion ist eine Funktion, die eine Eingabe auf einen Hashwert fester Länge abbildet
- Bei einer *kryptographischen* Hashfunktion ist es nicht möglich, zu einem gegebenen Hashwert y eine Nachricht x zu finden, für die $H(x) = y$ (eine *Kollision*)
- Aktuelle Beispiele: SHA-3, SHA-2, Skein, Whirlpool
- Historisch (Komplexität der besten Kollisionsattacke): SHA-1 ($2^{61,2}$), MD5 (2^{21}), MD4 (2)

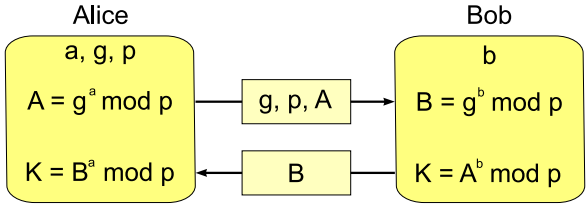
Digitale Signaturen mit Hashes



Schlüsselaustauschprotokolle

TECHNISCHE UNIVERSITÄT DARMSTADT

- Mit Schlüsselaustauschprotokollen können sich zwei Kommunikationspartnern über eine nicht abhörsichere Leitung auf einen gemeinsamen Schlüssel einigen
- Bekanntestes Beispiel: Diffie-Hellman-Schlüsselaustausch



$K = A^b \bmod p = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p = (g^b \bmod p)^a \bmod p = B^a \bmod p$

- Um sog. Man-in-the-Middle-Angriffe zu verhindern, wird das üblicherweise mit Public-Key-Kryptographie kombiniert

(Abbildung: [Wikipedia 2006])

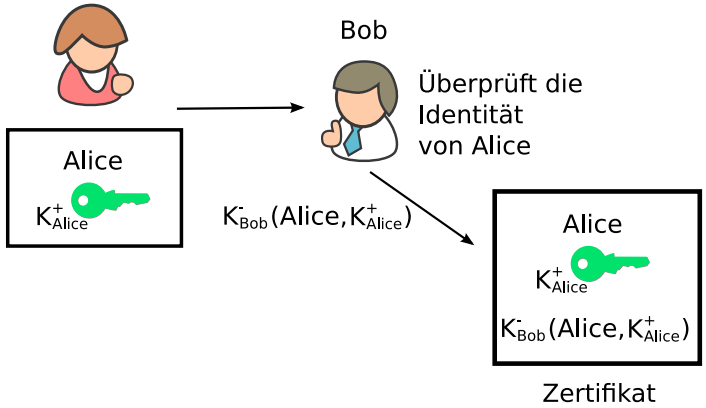
Zertifikate und Identitäten

TECHNISCHE UNIVERSITÄT DARMSTADT

Wie findet man bei asymmetrischer Krytographie den öffentlichen Schlüssel seines Kommunikationspartners? Welche Probleme sehen Sie?

- Verschlüsseln bringt nichts, wenn man nicht sicher sein kann, den richtigen Schlüssel zu benutzen!
- Zentrales Problem daher: Vertrauenswürdige Abbildung einer Identität auf den dazugehörigen öffentlichen Schlüssel
- Lösungsansatz: *Zertifikate* – vertrauenswürdige Dritte unterschreiben (digital), dass ein bestimmter öffentlicher Schlüssel zu einer bestimmten Person(/Organisation/...) gehört

Zertifikate



Vertrauenswürdigkeit von Zertifikaten

TECHNISCHE UNIVERSITÄT DARMSTADT

Kann man einem Zertifikat vertrauen?

Einem Zertifikat kann man höchstens so viel vertrauen, wie demjenigen, der es ausgestellt hat (in unserem Beispiel Bob):

- Vertraue ich darauf, dass er ehrlich ist?
- Vertraue ich darauf, dass er die Identität richtig geprüft hat?
- Vertraue ich darauf, dass er gut auf seinen geheimen Schlüssel aufpasst?
- Weiteres Problem: Woher bekommt man zuverlässig den öffentlichen Schlüssel desjenigen, der ein Zertifikat ausgestellt hat?

Public Key Infrastructure (PKI)

TECHNISCHE UNIVERSITÄT DARMSTADT

Wer sollte für wen Zertifikate ausstellen?

Die Regeln, nach denen Zertifikate ausgestellt werden, und die dafür notwendige Infrastruktur nennt man Public Key Infrastructure (PKI):

- Häufig, unter anderem im WWW: oligopolistisch und hierarchisch (X.509/PKIX)
- Alternative: anarchisch (Web of Trust)

- Zentrale Frage: Wem vertraue ich?

Hierarchisches Vorgehen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Es gibt einige Anbieter, die sogenannte Root-Zertifikate besitzen:
  - mit einem Root-Zertifikat zertifiziert man sich selbst („ich bestätige hiermit, dass ich ich bin“)
  - Root-Zertifikate schaffen kein neues Vertrauen
  - Root-Zertifikaten könnte man z. B. vertrauen, weil sie als Bestandteile von Programmen oder Betriebssystemen ausgeliefert werden
- Jemand, der von einem solchen Anbieter ein Zertifikat erhalten hat, kann dann wieder Zertifikate für andere ausstellen

Wann kann man einem Zertifikat in einem solchen hierarchischen System vertrauen?

Vertrauen beim hierarchischen Vorgehen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Einem Zertifikat *A* in einer hierarchischen PKI kann man nur dann vertrauen, wenn:
  - es eine Kette von Zertifikaten gibt, die von einem Root-Zertifikat zu *A* führt
  - bei dieser Kette jeder Nachfolger vom Vorgänger zertifiziert wird
  - man jedem einzelnen Element dieser Kette vertraut
- Fazit: Je länger die Kette ist, desto schwerer fällt das Vertrauen
- Konsequenz: Meist möchte man sein Zertifikat direkt von einem vertrauenswürdigen Inhaber eines Root-Zertifikats erhalten

Hierarchische PKI im Internet

TECHNISCHE UNIVERSITÄT DARMSTADT

- Im Internet kommen häufig PKI-Systeme zum Einsatz, die auf X.509/PKIX basieren
- Das ist auch das Standardvorgehen im WWW
- X.509 ist ein allgemeiner Standard zur Darstellung von Zertifikaten
- PKIX ist eine IETF-Arbeitsgruppe, die X.509 auf die Bedürfnisse des Internet anpasst
- Zentrales Dokument: RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
- Sehr trocken! Im Folgenden schauen wir uns einfach einmal ein Zertifikat für das Homebanking der Deutschen Bank an

X.509 PKIX Zertifikat (Auszug)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Certificate:
 Version: 3 (0x2)
 Serial Number: 33:a5:22:2a:1f:7a:f4:01:1d:d0:8a:c7:85:e5:48:bf

 Issuer: C=US, O=Symantec Corporation,
 OU=Symantec Trust Network, CN=Symantec Class 3 EV SSL CA - G3

 Validity:
 Not Before: Jun 11 00:00:00 2015 GMT
 Not After : Sep  3 23:59:59 2016 GMT

 Subject: 1.3.6.1.4.1.311.60.2.1.3=DE/1.3.6.1.4.1.311.60.2.1.1=Frankfurt am Main/
 businessCategory=Private Organization/serialNumber=HRB 30000, C=DE/postalCode=60325, ST=Hessen,
 L=Frankfurt am Main/street=Taunusanlage 12, O=Deutsche Bank AG, CN=meine.deutsche-bank.de

 Subject Public Key Info:
 Public Key Algorithm: rsaEncryption
 Public-Key: (2048 bit)
  Modulus: 00:ad:8e:cd: ...

 Signature Algorithm: sha256WithRSAEncryption
 72:b7:57:c3: ...
```

Das dazugehörige Symantec-Zertifikat (Auszug)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Certificate:
 Version: 3 (0x2)
 Serial Number: 7e:e1:4a:6f:6f:ef:f2:d3:7f:3f:ad:65:4d:3a:da:b4

 Issuer: C=US, O=VeriSign, Inc., OU=VeriSign Trust Network,
 OU=(c) 2006 VeriSign, Inc. - For authorized use only,
 CN=VeriSign Class 3 Public Primary Certification Authority - G5

 Validity:
 Not Before: Oct 31 00:00:00 2013 GMT
 Not After : Oct 30 23:59:59 2023 GMT

 Subject: C=US, O=Symantec Corporation, OU=Symantec Trust Network, CN=Symantec Class 3 EV SSL CA - G3

 Subject Public Key Info:
 Public Key Algorithm: rsaEncryption
 Public-Key: (2048 bit)
  Modulus: 00:d8:a1:65: ...

 Signature Algorithm: sha256WithRSAEncryption
 42:01:55:7b: ...
```

Das Root-Zertifikat (Auszug)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Certificate:
 Version: 3 (0x2)
 Serial Number: 18:da:d1:9e:26:7d:e8:bb:4a:21:58:cd:cc:6b:3b:4a

 Issuer: C=US, O=VeriSign, Inc., OU=VeriSign Trust Network,
 OU=(c) 2006 VeriSign, Inc. - For authorized use only,
 CN=VeriSign Class 3 Public Primary Certification Authority - G5

 Validity:
 Not Before: Nov  8 00:00:00 2006 GMT
 Not After : Jul 16 23:59:59 2036 GMT

 Subject: C=US, O=VeriSign, Inc., OU=VeriSign Trust Network, OU=(c) 2006 VeriSign, Inc. - For
 authorized use only, CN=VeriSign Class 3 Public Primary Certification Authority - G5

 Subject Public Key Info:
 Public Key Algorithm: rsaEncryption
 Public-Key: (2048 bit)
  Modulus: 00:af:24:08: ...

 Signature Algorithm: sha1WithRSAEncryption
 93:24:4a:30: ...
```

SSL/TLS und HTTPS

TECHNISCHE UNIVERSITÄT DARMSTADT

- SSL (Secure Sockets Layer) bzw. TLS (Transport Layer Security)
  - SSLv2 ursprünglich von Netscape entwickelt
  - Weiterentwickelt zu SSLv3
  - Von der IETF nochmals weiterentwickelt und standardisiert als TLS (in RFC 2246)
- Protokoll(e) zum Authentifizieren und Verschlüsseln von TCP-Verbindungen
- Hierzu: Verwendung von Public-Key-Infrastrukturen (PKI) zum Ausstellen, Verteilen und Überprüfen von digitalen Zertifikaten

SSL/TLS

TECHNISCHE UNIVERSITÄT DARMSTADT

- SSL bzw. TLS sind (teilweise unnötig) komplexe Protokolle
- Prinzip:
  - zusätzliche Protokollschicht zwischen Transport- und Anwendungsprotokoll
  - sieht nach „oben“ aus wie TCP, nach „unten“ wie ein TCP-basiertes Anwendungsprotokoll
  - nutzt eine darunterliegende TCP-Verbindung
  - verlässt sich insbes. auf Zuverlässigkeit und Reihenfolgeerhaltung durch TCP
- Variante für UDP-basierte Anwendungsprotokolle: DTLS

HTTPS-Protokollstapel

TECHNISCHE UNIVERSITÄT DARMSTADT

- Durch diese Struktur unterstützt SSL/TLS prinzipiell jedes (TCP-basierte) Anwendungsprotokoll
- HTTPS ist tatsächlich ganz „normales“ HTTP, mit der zusätzlichen SSL/TLS-Zwischenschicht
- Ebenso wird SSL/TLS aber auch mit anderen Protokollen genutzt, z. B. mit SMTP, POP3, IMAP,…
- Die SSL/TLS-Zwischenschicht nutzt X.509-Zertifikate für die Überprüfung von Identitäten
- …mit allen sich daraus ergebenden möglichen Problemen und Einschränkungen!

Wichtig: SSL/TLS ist nicht unfehlbar –
und höchstens so vertrauenswürdig wie das verwendete Zertifikat!

HTTP/2

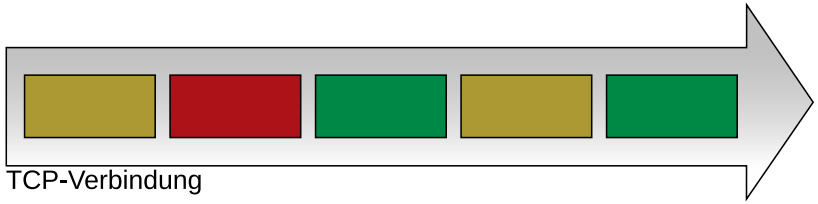
TECHNISCHE UNIVERSITÄT DARMSTADT

- Zunehmend verbreitet: HTTP/2
- Basiert auf Entwürfen von Google („SPDY“)
- Hier: kurzer Blick
- Implementiert von immer mehr Websites, außerdem von einigen Browsern (Chrome, Firefox, Opera, Safari, Edge,...)
- Zentrale Ziele:
  - Beschleunigung des Seitenaufbaus
  - Reduktion der Zahl paralleler TCP-Verbindungen
  - möglichst gute Kompatibilität zu existierenden Anwendungen

HTTP/2: Streams und Rahmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- HTTP/2 unterstützt mehrere unabhängige Streams über dieselbe TCP-Verbindung
- Die Daten eines Streams werden dann in Rahmen (Frames) zerlegt
- Rahmen unterschiedlicher Streams können abwechselnd über dieselbe TCP-Verbindung übertragen werden
- Wann welche Rahmen für welchen Stream übertragen werden, ist flexibel
  - so entstehen Freiheitsgrade, die für Performanzvorteile genutzt werden können
  - insbesondere: Head-of-Line-Blocking kann vermieden werden



HTTP/2: Streams und Rahmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Jeder Rahmen beginnt mit einem Header, der unter anderem eine Stream-ID (31-Bit-Zahl) und die Länge des Frames enthält
  - binär codiert – man ist hier also von der „Human-Readable-Philosophie“ von HTTP abgerückt
- Grundidee: Für jedes angeforderte Objekt öffnet der Client einen Stream über dieselbe TCP-Verbindung
- Entspricht ungefähr einem HTTP-Anfrage/Antwort-Paar
- Innerhalb des Streams semantisch auch sehr ähnlich zu einem HTTP-Austausch

HTTP/2: Streams

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eine Beschleunigung des Seitenaufbaus kann dann durch mehrere Mechanismen erfolgen, u. a.:
  - Streams können priorisiert werden (komplexe Mechanismen!); Objekte, die besonders wichtig sind, können andere „überholen“
  - auch der Server kann Streams öffnen (!); kann dem Client also von sich aus Objekte schicken, die (wahrscheinlich) benötigt werden
  - auch ein Push-Betrieb wird dadurch möglich: die TCP-Verbindung bleibt offen, der Server kann aktualisierte/neue Objekte von sich aus übertragen, ohne dass der Client immer wieder neu anfragen muss
- Jede Seite kann verlangen, dass die TCP-Verbindung geschlossen wird (GOAWAY-Nachricht)

HTTP/2: Header

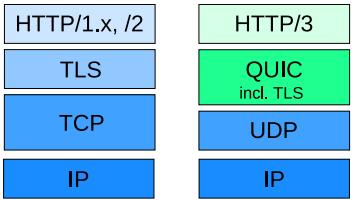
TECHNISCHE UNIVERSITÄT DARMSTADT

- Beim Aufbau eines neuen Streams können in beide Richtungen Header übertragen werden
- Name/Wert-Paare wie bei früheren HTTP-Versionen; existierende Header wurden übernommen und integriert
  - Ziel: Anwendungskompatibilität – Web-Anwendungen und „netzwerkferne“ Teile von Webserver und Browser sollen möglichst nichts vom neuen Protokoll wissen müssen
- Bei HTTP/2 werden auch Header nicht Human-readable übertragen (also keine „Header-Zeilen“ mehr)
- Netter Trick: HTTP/2 verwendet ein Datenkompressionsverfahren, um die Datenmenge für die Übertragung der Header zu reduzieren

HTTP/3 und QUIC

TECHNISCHE UNIVERSITÄT DARMSTADT

- HTTP/3 basiert nicht(!) mehr auf TCP
- Stattdessen ein neues Protokoll, das Aufgaben von TCP und TLS übernimmt: QUIC
- QUIC wird in der Anwendung (also im Browser und Webserver) implementiert und baut selbst wiederum auf UDP auf
- Konzepte von Streams und Rahmen werden im Wesentlichen aus HTTP/2 übernommen
- Weitere Verbesserungen durch Integration von TLS: Einsparung von RTTs



Zusammenfassung

TECHNISCHE UNIVERSITÄT DARMSTADT

- In diesem Abschnitt haben wir uns drei bedeutende Anwendungsprotokolle angeschaut:
  - SMTP
  - HTTP
  - DNS
- Außerdem SSL/TLS als wichtige „Sicherheits-Zwischenschicht“ für viele Anwendungsprotokolle
- Dadurch: erste Eindrücke über Verhaltensweisen von Protokollen für die Netzwerkkommunikation
- Wir bewegen uns nun weiter nach unten im Protokollstapel