Hochschule RheinMain

DAS WORLD WIDE WEB

Verteilte Systeme

Prof. Dr. Georg Hinkel
03.05.2024

GLIEDERUNG

Hochschule RheinMain

Datum	Vorlesung	Übungsblatt	Abgabe
19.04.2024	Einführung	HamsterLib	06.05.2024
26.04.2024	Netzwerkprogrammierung	Theorie	
03.05.2024	World Wide Web	HamsterRPC 1	20.05.2024
10.05.2024	Remote Procedure Calls	Theorie	
17.05.2024	Webservices	HamsterRPC 2	03.06.2024
24.05.2024	Fehlertolerante Systeme	Theorie	
31.05.2024	Transportsicherheit	HamsterREST	17.06.2024
07.06.2024	Architekturen für Verteilte Systeme	Theorie	
14.06.2024	Internet der Dinge	HamsterIoT	01.07.2024
21.06.2024	Namen- und Verzeichnisdienste	Theorie	
28.06.2024	Authentifikation im Web	HamsterAuth	15.07.2024
05.07.2024	Infrastruktur für Verteilte Systeme	Theorie	
12.07.2024	Wrap-Up	HamsterCluster (Bonus)	16.08.2024

AGENDA UND LERNZIELE

Hochschule RheinMain

Agenda

- HTTP 1.0/1.1
  - Adressen
  - Methoden
  - Cookies
- HTTPS/HSTS
- HTTP 2.0
- HTTP 3.0
- Programmierung von Webservern

Lernziele

- HTTP-Versionen 1.0 bis 3.0 charakterisieren können

Hochschule RheinMain

WIE KÖNNEN WIR WISSEN ÜBER EIN NETZWERK ZUGÄNGLICH MACHEN?

HTTP/0.9 (1991)
HyperText Transfer Protocol

Hochschule RheinMain

- Grundidee: Verbreite Wissen über das Internet
  - Hypertext ~ strukturierter & formatierter Text, Links
  - Separate Protokolle für Darstellung und Übertragung
- Textuelles Protokoll
- Entwickelt von Tim Berners-Lee am CERN
  - Zusammen mit HTML, URL
- Gilt als Geburtsstunde des World Wide Web
  - Name abgeleitet von erstem Browser

Tim Berners-Lee [Bild: Paul Clarke, CC BY 2.0]

HTTP/1.0 (1996)

Hochschule RheinMain

- Anfrage
  - HTTP-Version
  - Adresse
  - Methode
  - Header
  - (Entity Body optional)

- Antwort
  - HTTP-Version
  - Statuscode
  - Begründung (durch Statuscode vorgegeben)
  - Header
  - (Entity Body optional)

HTTP VERSION

Hochschule RheinMain

- Version 1.0 des HTTP-Standards definiert genau, wie der Standard versioniert wird

- HTTP/<Major>.<Minor>
  - ~Semantic Versioning
  - Höhere Major-Version ➔ höhere Version, ggf. brechende Änderungen
  - Höhere Minor-Version, gleiche Major-Version ➔ höhere Version, keine brechenden Änderungen

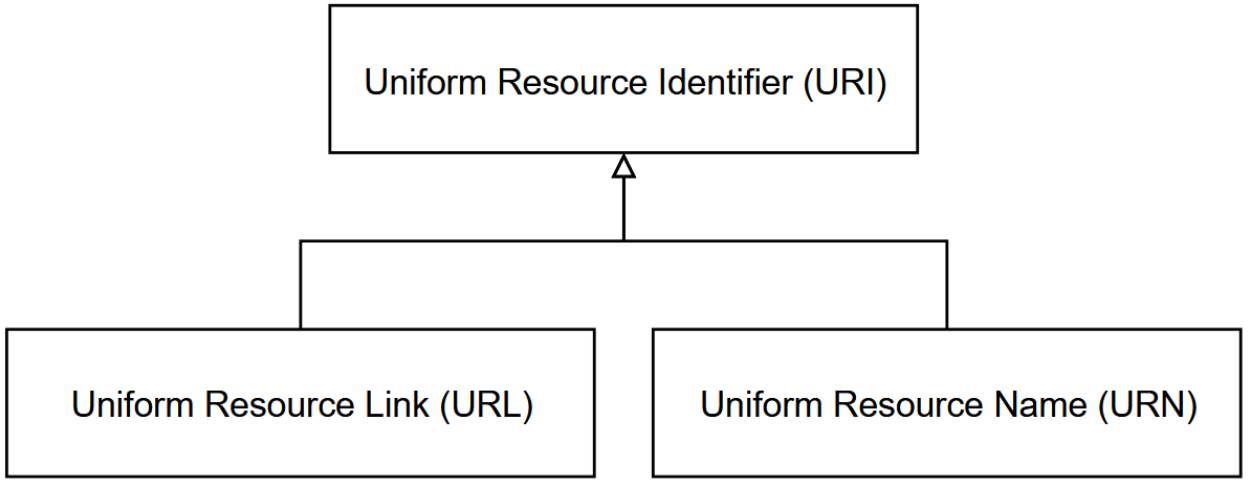
UNIFORM RESOURCE IDENTIFIER (URI)

Hochschule RheinMain

- Zeichenkette, die eine Ressource einheitlich darstellt
  - Nur ASCII-Zeichen, keine Leerzeichen, keine Kontrollzeichen

- <schema>:<Aufbau von Schema vorgegeben>
  - https://www.hs-rm.de
  - file:///d/lehre/verteilte%20systeme/
  - about:blank
  - urn:isbn:978-1543057386

- URL ➔ Schema definiert Zugriff
- URN ➔ Eindeutiger Name



HTTP ADRESSEN

Hochschule RheinMain

- Anfrage an einen HTTP-Server immer an bestimmte Adresse, als URL

- Weiterer Bestandteil: Fragment (#) bezeichnet Anker (a) im Dokument, optional

HTTP METHODEN

Hochschule **RheinMain**

- Methoden von HTTP/1.0 definiert
  - **GET**: Abrufen der angefragten Ressource
  - **HEAD**: Nur Header abfragen
  - **POST**: Ressource übertragen
  - (extension): Beliebige andere Methode
- Ergänzungen durch HTTP/1.1
  - **PUT**: Ressource aktualisieren
  - **DELETE**: Ressource löschen
  - **OPTIONS**, **CONNECT**, **TRACE**: kaum verwendet

HTTP HEADER

Hochschule RheinMain

- Zusätzliche Informationen, die Client/Server bereitstellen
  - In HTTP/1.0 hauptsächlich den Anhang betreffend, erweiterbar
  - Content-Encoding: Komprimierung des Anhangs, bspw. x-gzip
  - Content-Length: Größe in Bytes
  - Content-Type: MIME-Typ des Anhangs, bspw. text/html
  - Expires
  - Last-Modified
- Weitere Header möglich
  - Ab HTTP/1.1: Spezieller Header für Authentifizierung

HTTP STATUSCODES

Hochschule RheinMain

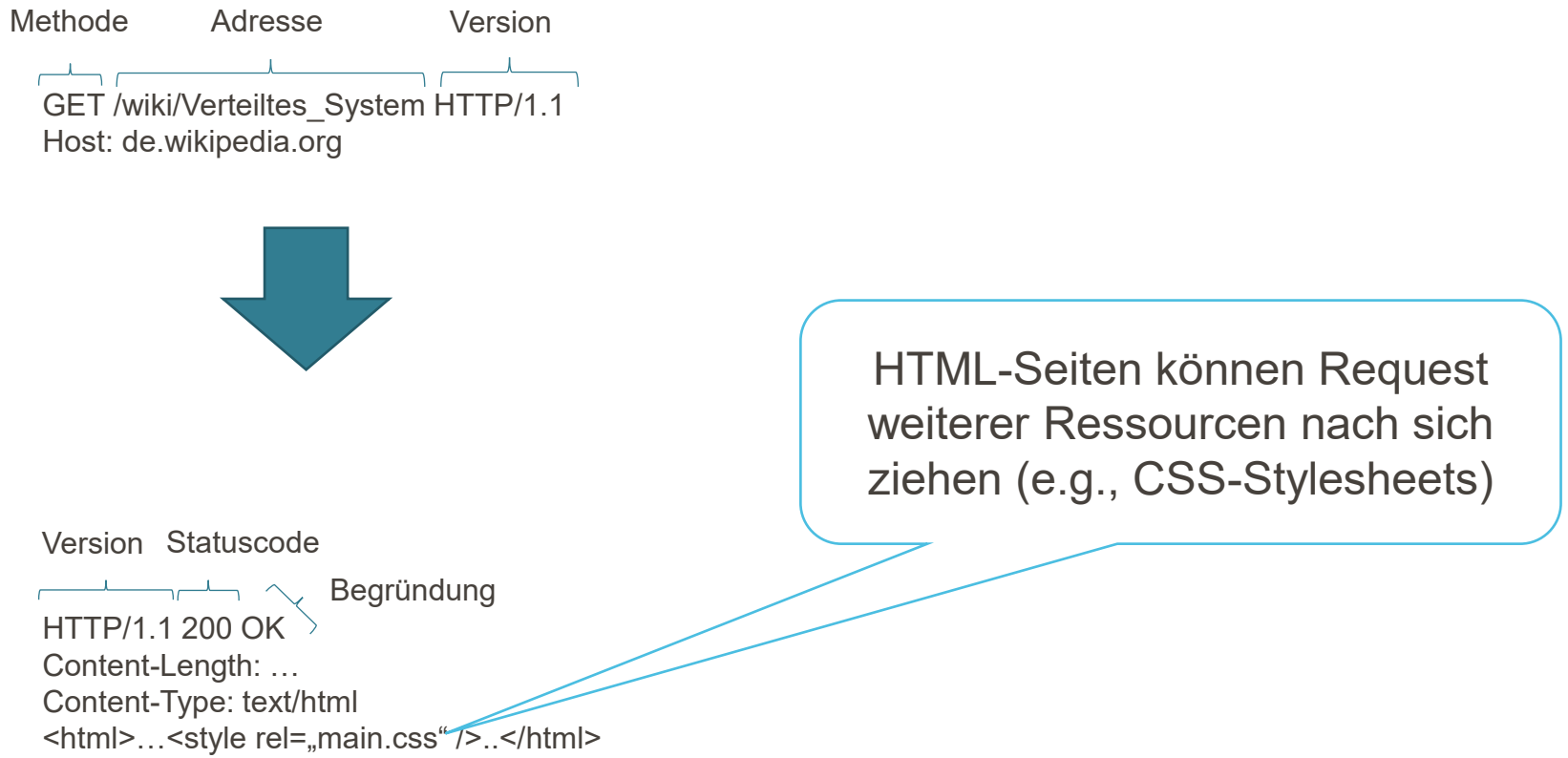
- Erfolg
  - 200 OK
  - 201 Created
  - 202 Accepted
  - 204 No Content
  - (weitere in HTTP/1.1)
- Umleitung
  - 301 Moved Permanently
  - 302 Moved Temporarily
  - 304 Not Modified
  - (weitere in HTTP/1.1)
- Client-Fehler
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found
  - (weitere in HTTP/1.1)
- Server-Fehler
  - 500 Internal Server Error
  - 501 Not Implemented
  - 502 Bad Gateway
  - 503 Service Unavailable
  - (weitere in HTTP/1.1)

Weitere Statuscodes möglich wenn Client und Server sich einig sind

Beispiel: 418 I am a teapot (RFC 2324, HTCPCP/1.0)

HTTP ROUNDTRIP - BEISPIEL

Hochschule RheinMain



HTTP: ABLAUF

Hochschule RheinMain

HTTP/1.1 (1999)

Hochschule RheinMain

- Problem: HTTP/1.0 schließt zugrundeliegende TCP-Verbindung nach jedem Request
  - Website mit HTML, CSS, 3 Java Script Files, 7 Bildern ➔ 12 TCP Verbindungen
- Lösung: Verbindung offen lassen
  - Zusätzlicher Client-Header „keepalive“
  - Damit nur noch eine 1 Verbindung notwendig, aber Requests nur sequentiell
  - Browser machen häufig bis zu 6 Verbindungen parallel auf
- Zusätzlich neue Methoden, neue Header

HTTP COOKIES

Hochschule RheinMain

- HTTP/1.1 ist zustandslos
  - TCP Verbindung wird nach dem Laden der Website geschlossen
- Heutige Webanwendungen erfordern Zustand
- Implementierung mit Headern
  - Server kann speziellen Header „Set-Cookie“ setzen, um Browser mitzuteilen, Informationen als Cookie zu speichern
  - Client sendet Cookie mit jedem Request mit
  - Cookie ist Overhead für jeden Request ➔ typischerweise Begrenzung auf Identifier

HTTPS

Hochschule RheinMain

- Problem: Kommerzialisierung des Internets lockt Kriminelle an
  - Beispiel E-Commerce ➔ Stehlen von Zugangsdaten, …
- Lösung: Transportsicherheit via TLS (schauen wir uns später genauer an)
  - Sämtliche Nachrichten zwischen Client und Server werden verschlüsselt
  - Standardmäßig Port 443 statt 80
  - Sonst keinerlei Änderungen gegenüber HTTP
- HTTP Strict Transport Security (HSTS): Server kann per Header Verschlüsselung erbeten
  - Implementiert in spezifischem Header *Strict-Transport-Security*
  - Browser speichert Information und baut zukünftige Verbindungen mit HTTPS auf

Hochschule RheinMain

16 JAHRE „RUHE“

WAS IST PASSIERT?

Hochschule RheinMain

- HTTP entworfen, um statische Dokumente abzufragen / auszutauschen
  - Im Original zustandslos
  - Kaum Authentifizierung / Autorisierung
  - Statische Webseiten

- Web in den 2010er Jahren
  - „Web 2.0“ / Soziale Medien ➔ Eigentliche Inhalte von Nutzern generiert
  - Webseiten häufig beim Zugriff generiert
  - Webseiten bestehen aus einer Vielzahl von Ressourcen
  - Geschwindigkeit Seitenaufbau ist wichtiger Wettbewerbsfaktor ➔ 8-second rule

HTTP/2.0 (2015)

Hochschule RheinMain

- Probleme mit HTTP/1.1
  - Ausufernde Header
  - Browser können immer nur eine Ressource pro TCP-Verbindung nachladen
  - Jede Antwort vom Server erfordert eine vorherige Anfrage vom Client
- Aber: HTTP/1.1 sehr verbreitet, Kompatibilität wichtig
- Lösung
  - Multiplexing
  - Header-Komprimierung
  - Server Streaming

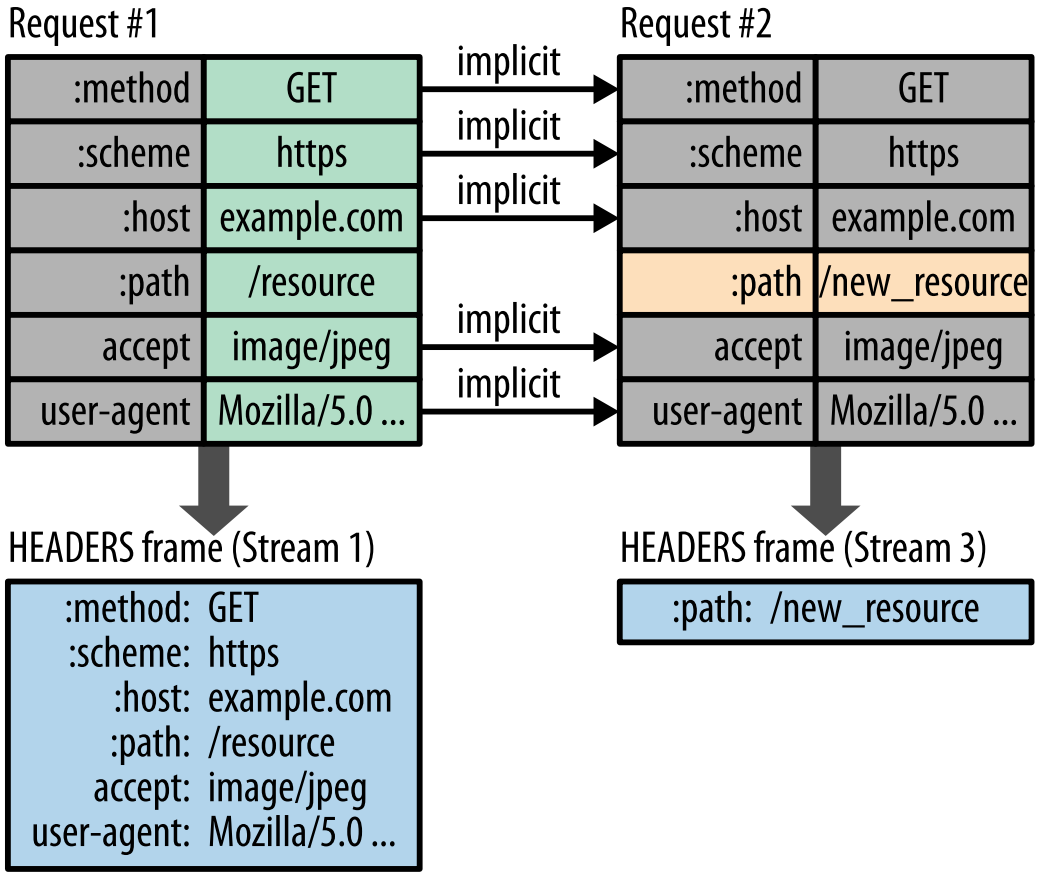
HEADERKOMPRIMIERUNG

Hochschule RheinMain

- HTTP/0.9-1.1 kodiert Header-Informationen als Text
  - Separiert durch Leerzeichen
  - Einfach für Menschen lesbar, aber Parsing nicht trivial
  - Typischerweise 500-800 Byte pro Request, bei Einsatz von Cookies noch mehr
- Lösung in HTTP/2.0: Frames, komprimierte Header
  - Separate Datenblöcke für Header und Payload ➔ Header für mehrere Frames gültig
  - Header binär kodiert, komprimiert

HEADERKOMPRIMIERUNG

Hochschule RheinMain



[https://web.dev/performance-http2/]

- Header werden mit Huffman-Code komprimiert (HPACK)
- Header werden pro Verbindung gecacht
  - Erneute Übertragung nicht notwendig

MULTIPLEXING

Hochschule RheinMain

- Problem: Beim Aufruf einer Website sind sehr viele Requests nötig
  - HTML
  - CSS
  - Skripte
  - Bilder
  - …
- HTTP/1.1: Mehrere parallele Verbindungen notwendig
  - Obwohl unterliegende TCP-Verbindung eigentlich duplexfähig
- Lösung: Multiplexing
  - Frames (Requests und Responses) enthalten Stream ID
  - Client kann eine Response mittels Stream ID dem entsprechendem Request zuordnen

SERVER STREAMING

Hochschule RheinMain

- Problem: Nachfolgende Zugriffe erfordern separaten Request, obwohl sehr vorhersagbar
  - Nach dem HTML wird der Browser wahrscheinlich die CSS abrufen und dann enthaltene Bilder
- Lösung: Server Streaming, Server kann von sich aus Nachrichten schicken, die nicht zu einem Request gehören
  - Dedizierter Frametyp, der genutzt wird, um dem Client eine Ressource zu versprechen (Promise)
  - Client kann Promise widersprechen (e.g. Ressource bereits im Cache)
- Typischerweise nicht direkt vom Webserver implementiert
  - Anwendung muss selbst spezifizieren, welche Ressourcen für eine Anfrage mitgeliefert werden sollen
  - Anwendung typischerweise nur für statische Inhalte: Client könnte die weiteren Ressourcen ablehnen

BEISPIELRECHNUNG

Hochschule RheinMain

- Annahme: Website besteht aus 1 HTML, 1 CSS, 10 Scripte, 88 Bilder
  - Verarbeitungszeit im Server und im Browser werden ignoriert
  - Roundtripdauer angenommen 12ms, Bandbreite unendlich

- HTTP/1.0
  - 100 mal TCP-TLS-Verbindung aufmachen, 100 mal Request/Response
  - Verbindungsaufbau mit TCP/TLS: 3 Roundtrips (einschließlich Request/Response) + Verbindungsabbau

$$100 * 4 * 12ms = 4{,}8s$$

BEISPIELRECHNUNG

Hochschule RheinMain

- Annahme: Website besteht aus 1 HTML, 1 CSS, 10 Scripte, 88 Bilder
  - Verarbeitungszeit im Server und im Browser werden ignoriert
  - Roundtripdauer angenommen 12ms, Bandbreite unendlich

- HTTP/1.1
  - 1 mal TCP-TLS-Verbindung aufmachen, 100 mal Request/Response
  - Verbindungsaufbau mit TCP/TLS: 3 Roundtrips (einschließlich Request/Response) + Verbindungsabbau

$$(3 + 100) * 12ms = 1{,}24s$$

BEISPIELRECHNUNG

Hochschule **RheinMain**

- Annahme: Website besteht aus 1 HTML, 1 CSS, 10 Scripte, 88 Bilder
  - Verarbeitungszeit im Server und im Browser werden ignoriert
  - Roundtripdauer angenommen 12ms, Bandbreite unendlich

- HTTP/1.1, *6 Verbindungen parallel*
  - 1 mal TCP-TLS-Verbindung aufmachen, 100 mal Request/Response
  - Verbindungsaufbau mit TCP/TLS: 3 Roundtrips (einschließlich Request/Response) + Verbindungsabbau
  - 1 Roundtrip für HTML, CSS und Skripte parallel (2 Runden), dann Bilder parallel (15 Runden)

$$(3 + 1 + 2 + 15) * 12ms = 252ms$$

BEISPIELRECHNUNG

Hochschule RheinMain

- Annahme: Website besteht aus 1 HTML, 1 CSS, 10 Scripte, 88 Bilder
  - Verarbeitungszeit im Server und im Browser werden ignoriert
  - Roundtripdauer angenommen 12ms, Bandbreite unendlich

- HTTP/2.0
  - 1 mal TCP-TLS-Verbindung aufmachen, 3 Requests / Responses parallel
  - Verbindungsaufbau mit TCP/TLS: 3 Roundtrips (einschließlich Request/Response) + Verbindungsabbau
  - 1 Request für HTML, parallele Request für CSS + Skripte, parallele Requests für Bilder

$$(3 + 1 + 1 + 1) * 12ms = 72ms$$

BEISPIELRECHNUNG

Hochschule RheinMain

- Annahme: Website besteht aus 1 HTML, 1 CSS, 10 Scripte, 88 Bilder
  - Verarbeitungszeit im Server und im Browser werden ignoriert
  - Roundtripdauer angenommen 12ms, Bandbreite unendlich

- HTTP/2.0, Server sendet sofort alle Dateien per Server Streaming
  - 1 mal TCP-TLS-Verbindung aufmachen, 1 mal Request / 100 mal Response parallel
  - Verbindungsaufbau mit TCP/TLS: 3 Roundtrips (einschließlich Request/Response) + Verbindungsabbau

$$(3 + 1) * 12ms = 48ms$$

Faktor 100 schneller als HTTP/1.0!

HEAD OF LINE BLOCKING

Hochschule RheinMain

- Stellen Sie sich vor, Sie bestellen Pizza zusammen mit Kommilitonen 1..N
  - Lieferdienst ist reihenfolgetreu
  - Garantiert also, wer früher bestellt bekommt auch früher Pizza
- Bei der Auslieferung kommt es zu einem Unfall
  - Wer muss alles warten, wenn die Pizza für Kommilitone N nicht ausgeliefert werden konnte?
  - Wer muss alles warten, wenn die Pizza für Kommilitone 1 nicht ausgeliefert werden konnte?
    - Alle anderen, Pizzen werden kalt und müssen erneut geliefert werden
- Effekt wird Head-of-Line-Blocking genannt
- **Achtung**: Reihenfolgentreue trotzdem wichtig, übertragene Dateien könnten größer als ein TCP Paket sein. Problematisch ist die Reihenfolgentreue über verschiedene Streams hinweg

Hochschule RheinMain

HTTP/1.1 ODER HTTP/2.0, WAS DAVON NEHMEN WIR JETZT?

Application-Layer Protocol Negotiation (ALPN)

APPLICATION-LAYER PROTOCOL NEGOTIATION (ALPN)

Hochschule RheinMain

- Problem: Wie bekommt der Browser heraus, ob ein Webserver HTTP/2.0 kann oder HTTP/1.1?
- Idee: Faktisch alle Webseiten per HTTPS ausgeliefert, Zertifikate erweiterbar
- Lösung:
  - Auflistung unterstützter Protokolle als TLS-Erweiterung
  - Jedes Protokoll registriert bei IANA ALPN-Bezeichner
    - HTTP/1.1: http/1.1
    - HTTP/2.0: h2
    - HTTP/3.0: h3
  - Server antwortet dann mit gewählter Protokollversion

HTTP/3.0 (2022)

Hochschule RheinMain

- Problem: HTTP-Verbindung ist auf darunterliegende TCP-Verbindung angewiesen
  - Aber TCP-Verbindung bricht zusammen wenn User das Netzwerk wechselt
  - TCP ist reihenfolgentreu, verlorenes Paket führt daher zu Stau ➔ Head-of-Line-Blocking

- Lösung: Wir wechseln auf UDP


Süddeutsche Zeitung

Neues Netz-Protokoll Quic

Revolution in den Tiefen des Internets

[https://www.sueddeutsche.de/digital/internet-schneller-google-tcp-protokoll-verschluesselung-1.5171790]

  - Wie prüfen wir dann, dass Pakete auch angekommen sind?
  - Wie prüfen wir dann, dass keine Duplikate ausgeliefert worden?

QUIC

Früher: Quick UDP Internet Connections, heute nicht mehr als Akronym

Hochschule RheinMain

- Ursprünglich von Google entwickelt
- Definiert zuverlässiges, verbindungsorientiertes Transportprotokoll oberhalb von UDP

- Idee: Bandbreite häufig kein Problem, aber Latenz
  - VDSL: 100Mbit/s, aber Latenz durch Lichtgeschwindigkeit limitiert
  - Glasfaser: bis zu 1000Mbit/s möglich
  - Außerdem hohe Verarbeitungsgeschwindigkeit der Endgeräte

- Daher: Entwurfsziel Reduktion der Roundtrips, keine Begrenzung der Paketgröße
  - Integration der Sicherheit in Transportprotokoll ➔ nur ein Roundtrip bis Verbindung fertig aufgebaut
  - Stärkeres Caching ➔ kein Roundtrip für Verbindungsaufbau notwendig, wenn Server bereits bekannt
  - Streng monotone Sequenznummern ➔ genauere Schätzung der Roundtrip-Zeit möglich, aber separate Acknowledgements
  - Staukontrolle ähnlich wie bei TCP

NOCHMAL BEISPIELRECHNUNG

Hochschule RheinMain

- Entfernung Sydney-Lissabon: 18.186km
- Lichtgeschwindigkeit in Glasfaser: 200.000km/s

➔Licht auf direkter Verbindung

$$\text{etwa } \frac{18.186km}{200.000\frac{km}{s}} = 0{,}09093s \sim 91ms$$

- Tatsächliche Ping-Statistik: 277ms (für Roundtrip) [https://wondernetwork.com/pings/Sydney]
  - Entspricht ca. 65% der Lichtgeschwindigkeit auf direkter Strecke

➔ Potential für weitere Verbesserungen der Latenz ist sehr begrenzt!

QUIC

Früher: Quick UDP Internet Connections, heute nicht mehr als Akronym

Hochschule RheinMain

- Grundidee: Dedizierte Connection ID für jede Verbindung
  - Wird vom Client gewürfelt
  - Aushandeln eines symmetrischen Schlüssels direkt bei Verbindungsaufbau
    - Sehen wir uns beim Thema Sicherheit noch etwas genauer an
  - Connection ID und Verbindungsparameter beibehalten, auch wenn die darunterliegende UDP-Verbindung abbricht

  ➔ QUIC Verbindung bleibt bestehen selbst wenn der Client das Netzwerk wechselt

  ➔ Paradeanwendung: Nutzer wechselt von LTE ins WLAN

- Verhindert Head-of-Line-Blocking
  - Multiplexing wie bei HTTP/2.0
  - Reihenfolgentreue wird nur pro Stream ID gewährleistet

QUIC
Probleme mit NAT

Hochschule RheinMain

- IPv4 Adressen sind sehr begrenzt
  - Adressraum von nur 32bit

- Teil der Lösung: Native Address Translation (NAT)
  - Switch bildet Verbindungen von verschiedenen Clients auf nicht genutzte Ports ab
  - Drastische Erweiterung des verfügbaren Adressraums
  - Gültigkeit der Zuordnung bei TCP eindeutig (Switch „sieht“ Verbindungsauf/-abbau)

- Problem: QUIC ist zu neu
  - QUIC hat expliziten Connection_Close Frame, aber viele Switches kennen QUIC (noch) nicht
  - Behandlung daher als UDP Datagram
  - Typischerweise wesentlich kürzere Timeouts als TCP
  - Pakete vom Server kommen beim Switch u.U. erst an, wenn NAT-Mapping bereits entfernt

BEISPIELRECHNUNG

Hochschule RheinMain

- Annahme: Website besteht aus 1 HTML, 1 CSS, 10 Scripte, 88 Bilder
  - Verarbeitungszeit im Server und im Browser werden ignoriert
  - Roundtripdauer angenommen 12ms, Bandbreite unendlich

- HTTP/3.0, Server sendet sofort alle Dateien per Server Streaming
  - 1 mal QUIC-Verbindung aufmachen, 1 mal Request / 100 mal Response parallel
  - Verbindungsaufbau mit QUIC: 1 Roundtrip, Verbindung wird danach offen gelassen

$$(1 + 1) * 12ms = 24ms$$

  - Verbindungsparameter können gespeichert werden ➔ Ggf. Reduktion auf 12ms

- Spätestens mit HTTP/3.0 ist hier doch wieder Bandbreite der limitierende Faktor

PROGRAMMIERUNG VON WEBSERVERN
Einführung

Hochschule RheinMain

- HTTP-Implementierungen heute üblicherweise durch Webserver gekapselt, Programmierung über spezifische Web-Frameworks

- Typische Bestandteile
  - Controller: Verantwortlich für Anfragen an Gruppen von Adressen („Routen“)
  - Dependency Injection: Webserver löst Instanzen für Controller typischerweise per DI auf
  - Request-Pipeline, in die generisch Code injiziert werden kann (e.g. Authentifizierung)
  - Template-Engine: Nutzlast oft template-gesteuert, insb. bei HTML-Seiten
  - Serializer: Typisierte Objekte werden in Netzdatenformat konvertiert, falls kein HTML

PROGRAMMIERUNG VON WEBSEITEN

Beispiel: SpingBoot ➔ Apache Tomcat

Hochschule **RheinMain**

```
package com.example.demo;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class DemoApplication {
    public static void main(String[] args) {
      SpringApplication.run(DemoApplication.class, args);
    }
    @GetMapping("/hello")
    public String hello(@RequestParam(value = "name", defaultValue = "World") String name) {
      return String.format("Hello %s!", name);
    }
}
```

Routen können Platzhalter für Parameter enthalten

PROGRAMMIERUNG VON WEBSEITEN

Beispiel ASP.NET Core ➔ Kestrel, IIS oder http.sys

Hochschule RheinMain

```
using Microsoft.AspNetCore.Mvc;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllers();
var app = builder.Build();
app.MapControllers();
app.Run();

[ApiController]
[Route("hello")]
public class DemoController : ControllerBase
{
    [HttpGet]
    public string Hello( string? name = "World")
    {
        return $"Hello {name}!";
    }
}
```

ZUSAMMENFASSUNG

Hochschule RheinMain

- World Wide Web angefangen mit textbasiertem Protokoll mit einfachen Primitiven
  - Methoden
  - Ressourcen-Adressen
  - URL-Links mit Möglichkeit für Abfragen

- Primitive heute alle noch erhalten, aber technischer Unterbau wesentlich komplexer
  - Headerkomprimierung
  - Server Streaming
  - Multiplexing ➔ Parallelitätstransparenz
  - QUIC ➔ löst Head-of-Line-Blocking plus stabile Verbindung bei Wechsel des Netzwerks

- Features von HTTP/2.0 und HTTP/3.0 in der Praxis (noch) nicht sehr verbreitet
  - Viele „middle-boxes“ (e.g. Switches) sind für TCP optimiert, kennen QUIC nicht

  ➔ Ihre Aufgabe könnte werden, diese Features für das Unternehmen zu nutzen

MÖGLICHE PRÜFUNGSAUFGABEN

Hochschule RheinMain

- Erläutern Sie die Bestandteile einer HTTP-Adresse!
- Wofür werden in HTTP Request- und Response-Header verwendet? Nennen Sie Beispiele!
- Nennen Sie Beispiele für standardisierte HTTP Statuscodes!
- Nennen Sie Beispiele für standardisierte HTTP Methoden!
- Charakterisieren Sie den Unterschied zwischen HTTP/1.1 und HTTP/2.0!
- Charakterisieren Sie den Unterschied zwischen HTTP/2.0 und HTTP/3.0!
- Erläutern Sie den Begriff Head-of-Line Blocking!
- Erläutern Sie für ein gegebenes Beispiel, welche Vorteile der Umstieg von HTTP/1.1 auf HTTP/2.0 (oder HTTP/3.0) zu erwarten wären!