
Netze

Modul 11: HTTP 2 und QUIC

👤 Prof. Dr. Hannes Tschofenig



11. 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

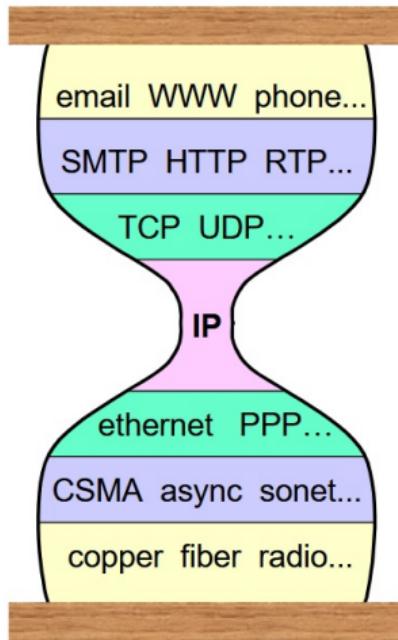
ID	KW	Art	Thema
	40	/	/
UE-1	41	Übung	Topologien und OSI
UE-2	42	Übung	Übertragungen bspw. Kabel
P-1	43	Praktikum	Laboreinführung und Netzwerktools
S-1	44	Video	IPv4
P-2	45	Praktikum	Adressierung
P-3	46	Praktikum	IPv4 und Autokonfiguration
P-4	47	Praktikum	IPv6 und Autokonfiguration
P-5	48	Praktikum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

UE - Übung laut Stundenplan in den Seminarräumen

P - Praktikum in C055

S - Selbststudium KEINE Präsenz

The Waist of the Hourglass ...



Steve Deering. **Watching the Waist of the Protocol Hourglass.**

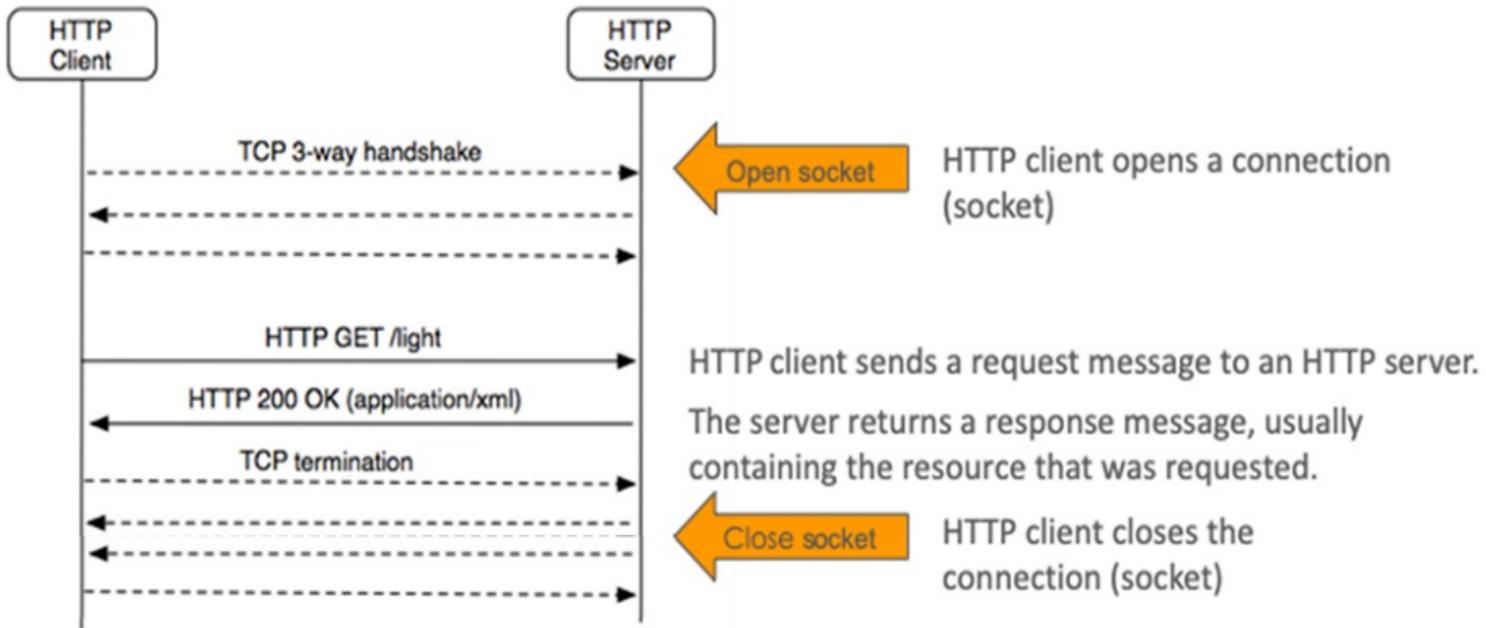
Keynote, IEEE ICNP 1998, Austin, TX, USA.



Steve Deering, former Cisco Fellow,

Erfinder von IP Multicast und Co-Autor von IPv6.

HTTP: Wiederholung



Beispiel HTTP GET Anfrage

HTTP GET

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
```

HTTP 200 OK Antwort

```
HTTP/1.1 200 OK
Date: Wed, 03 Oct 2024 10:15:30 GMT
Server: Apache/2.4.41 (Ubuntu)
Last-Modified: Tue, 12 Sep 2024 15:30:20 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 3456
... (HTML content)
```

HTTP/1.0 - Wichtige Merkmale

- HTTP/1.0 wurde in RFC 1945 [21] veröffentlicht und definierte die grundlegenden Anfragemethoden GET, POST und HEAD.
- Es wurden HTTP-Header eingeführt, die in Anfragen und Antworten verwendet werden konnten, um zusätzliche Informationen zu übermitteln.
- Die Header wurden in einem textbasierten Format codiert.
- Eine optionale HTTP-Versionsnummer wurde eingeführt, die in allen Anfragen angegeben werden konnte.
- Für Antworten wurde ein dreistelliger Statuscode definiert.

HTTP / Web – Erfinder: Tim Berners-Lee



Tim Berners-Lee beim IAB Privacy Workshop, December 2010 [10].
Professor am MIT, Gründer und Direktor des
World Wide Web Consortium (W3C).

2004 zum Ritter geschlagen; Turing Award 2016.



MIT Stata Center, finanziert von Ray und Maria Stata.
(Ray Stata: Gründer von Analog Devices, Inc.)

HTTP Methoden und deren Semantik

- Die HTTP-Methoden definieren das Verhalten einer Anfrage:
 - HEAD ermöglicht es, Metadaten über eine Ressource anzufordern, ohne die Ressource selbst herunterzuladen.
 - GET fordert eine Ressource an.
 - POST erlaubt es einem Client, Daten an den Server zu senden.
- Obwohl auch mit GET Daten übermittelt werden können - als Teil der Query-Parameter in der **Uniform Resource Identifier (URI)** - bot die Einführung der POST-Methode eine Möglichkeit, beliebige Datenmengen effizient an den Server zu übertragen.
- Während die GET-Methode als **idempotent** und sicher gilt, trifft dies auf POST nicht zu.
 - Eine Methode ist idempotent, wenn mehrere identische Anfragen an dieselbe URL immer zum gleichen Ergebnis führen - ohne Nebeneffekte.
 - **Sicher** bedeutet, dass die Methode nur lesend auf Ressourcen zugreift.
- Vollständige Liste aller Methoden wird von IANA verwaltet:
<https://www.iana.org/assignments/http-methods/http-methods.xhtml>



Begriffe

- **URI (Uniform Resource Identifier)** Allgemeiner Identifikator für Ressourcen.
- **URL (Uniform Resource Locator)** Eine URI, die den Ort einer Ressource angibt (z. B. Schema + Host + Pfad).
- **URN (Uniform Resource Name)** Eine URI, die eine Ressource unabhängig vom Ort benennt.

Merksatz:

URL ⊂ URI (jede URL ist eine URI, aber nicht jede URI ist eine URL)

Allgemeine URI-Syntax (RFC 3986 [8]):

scheme:[//authority]path[?query][#fragment]

Beispiel für eine HTTP-URL:

<http://example.com:80/products/list?category=books&page=2#top>

HTTP-Methoden und Idempotenz

HTTP-Methode	Beschreibung	Idempotenz
GET	Ruft Daten vom Server ab.	Idempotent
HEAD	Wie GET, aber ohne die Ressource abzurufen. Prüft, ob eine Ressource existiert.	Idempotent
POST	Sendet Daten zur Verarbeitung an den Server.	Nicht idempotent
PUT	Aktualisiert eine bestehende Ressource oder erstellt eine neue. Mehrfache Anfragen erzeugen keine Duplikate.	Idempotent
DELETE	Löscht eine Ressource vom Server, falls sie existiert. Andernfalls passiert nichts.	Idempotent

Representational State Transfer (REST)

- Moderne Web-, Mobile- und Cloud-Anwendungen bestehen aus vielen verteilten Komponenten, die über HTTP kommunizieren.
- Um diese Kommunikation vorhersehbar und interoperabel zu gestalten, hat sich REST als De-facto-Standard für Web-APIs etabliert.
- REST bietet:
 - **Einheitliche Konventionen** für HTTP-Methoden, URLs und Statuscodes
 - **Hohe Interoperabilität** zwischen Systemen, Programmiersprachen und Plattformen
 - **Lose Kopplung** — Client und Server können unabhängig weiterentwickelt werden
 - **Skalierbarkeit** durch Statelessness und Caching
 - **Einfache Lernbarkeit** durch klare und intuitive Semantik

REST und CRUD im Überblick

- REST nutzt HTTP als Transportmechanismus, wobei die CRUD-Operationen direkt den HTTP-Methoden entsprechen:
 - **Create** → POST
 - **Read** → GET
 - **Update** → PUT / PATCH
 - **Delete** → DELETE
- REST ist daher eines der wichtigsten Architekturmuster für Web- und Cloud-APIs.
- Siehe auch "Building Protocols with HTTP"[\[23\]](#).



Roy Fielding ist Co-Autor der HTTP-Spezifikation und entwickelte REST als Teil seiner Dissertation an der University of California. Er ist der Mitbegründer des Apache HTTP Server Projekts.

HTTP/1.1 - Standardisierung und Persistente Verbindungen

- HTTP/1.1 wurde in der IETF in drei Schritten dokumentiert:
 - Zunächst als **RFC 2068** [12], kurz nach HTTP/1.0 veröffentlicht.
 - Anschließend ersetzt durch **RFC 2616** [22].
 - Schließlich wurde die Spezifikation in sechs RFCs aufgeteilt: **RFC 7230** [16], **RFC 7231** [17], **RFC 7232** [15], **RFC 7233** [11], **RFC 7234** [13] und **RFC 7235** [14].
- Das ständige Schließen und Wiederöffnen von TCP-Verbindungen war offensichtlich ineffizient.
- HTTP/1.1 formalisierte daher das Konzept der **persistenten Verbindungen** - eine Funktion, die in vielen HTTP/1.0-Servern bereits implementiert war.
- Über die Connection-Headerzeile mit dem Wert **Keep-Alive** konnte signalisiert werden, dass die Verbindung offen bleiben soll.
- In HTTP/1.1 wurde dieses Verhalten zum Standard.

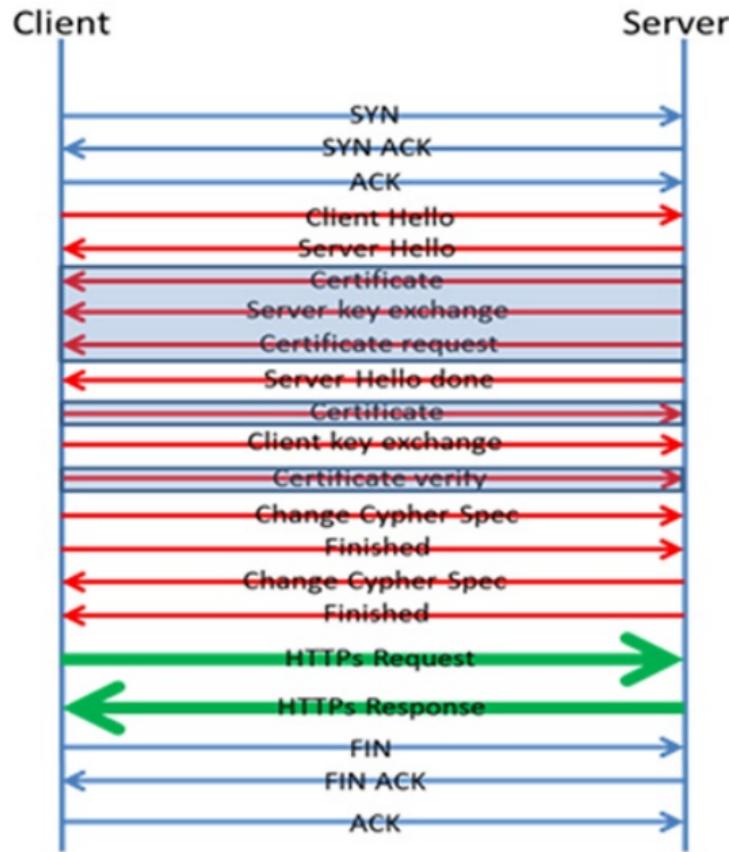
- **Virtuelle Server** wurden häufiger verwendet: Da die URL in der HTTP-Anfragezeile (z. B. bei einem GET) keine absolute URL ist, wurde der **Host-Header** eingeführt, um anzugeben, welcher HTTP-Server angesprochen werden soll.
- HTTP/1.1 führte außerdem die Unterstützung für **Proxies** ein, also zwischengeschaltete HTTP-Server.
- Proxies waren damals nützlich, da sie Inhalte **zwischenspeichern** konnten. Das Verhalten beim Caching musste daher durch Header wie Cache-Control genau definiert werden.
- Mit dem zunehmenden Einsatz von **TLS** wurden Caching-Proxies jedoch weniger brauchbar - stattdessen kamen **Content Distribution Networks (CDNs)** zum Einsatz.
- Zusätzlich zu neuen Statuscodes wurden auch neue Methoden eingeführt: **CONNECT**, **OPTIONS** und **TRACE**.

HTTP-Methoden und Idempotenz

HTTP-Methode	Beschreibung	Idempotenz
GET	Ruft Daten vom Server ab.	Idempotent
HEAD	Wie GET, aber ohne die Ressource abzurufen. Prüft, ob eine Ressource existiert.	Idempotent
POST	Sendet Daten zur Verarbeitung an den Server.	Nicht idempotent
PUT	Aktualisiert eine bestehende Ressource oder erstellt eine neue. Mehrfache Anfragen erzeugen keine Duplikate.	Idempotent
DELETE	Löscht eine Ressource vom Server, falls sie existiert. Andernfalls passiert nichts.	Idempotent
CONNECT	Stellt einen bidirektionalen Tunnel zum Server her.	Nicht idempotent
TRACE	Debugging-Methode für Loopback-Tests, bei der die Anfrage zurückgespiegelt wird.	Idempotent
OPTIONS	Ermöglicht das Ermitteln aller vom Server unterstützten Methoden.	Idempotent

HTTP und Sicherheit: Der Weg zu TLS

- Ursprünglich bot HTTP keine Kommunikationssicherheit - Nachrichten wurden unverschlüsselt übertragen.
- Mit dem Aufkommen von E-Commerce im Web wurde klar, dass Sicherheit ein unverzichtbares Merkmal ist.
- Netscape entwickelte daraufhin das Protokoll **Secure Socket Layer (SSL)**.
- SSL wurde später zur Standardisierung an die **IETF** übergeben und in **Transport Layer Security (TLS)** umbenannt.
- Die Verwendung von TLS mit HTTP wurde in **RFC 2818 [25]** spezifiziert.



HTTP/1.1 und das Head-of-Line Blocking

- Beim Abrufen einer Webseite werden häufig zusätzliche Ressourcen benötigt (z.B. Bilder).
- Diese Ressourcen erfordern jeweils eigene HTTP-Anfragen vom Client an den Server.
- HTTP/1.1 führte **Pipelining** ein, bei dem mehrere Anfragen gesendet werden konnten, bevor Antworten empfangen wurden.
- Die Implementierung von Pipelining war jedoch komplex.
- Antworten mussten in der ursprünglichen Reihenfolge zurückgegeben werden.
- Eine langsame Antwort blockierte alle nachfolgenden - dieses Problem wird als **Head-of-Line (HOL) Blocking** bezeichnet.

Warum HTTP/2?

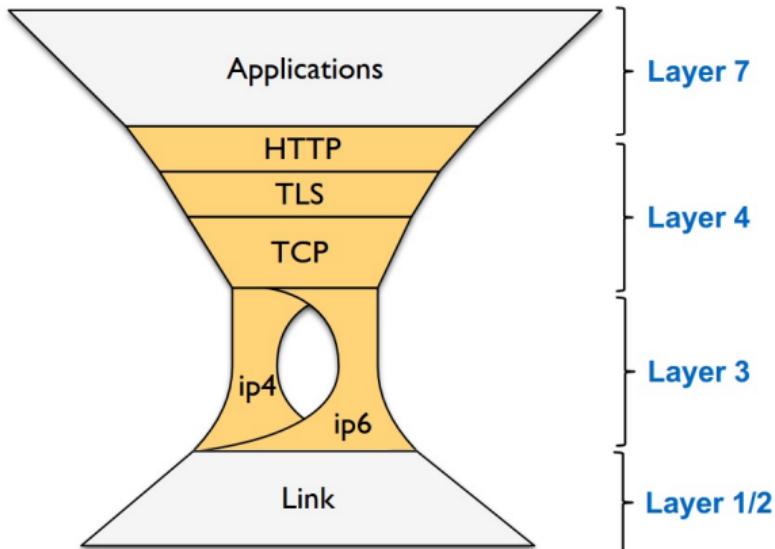
- In der Praxis etablierten Entwickler mehrere parallele TCP-Verbindungen, um gleichzeitige HTTP-Anfragen durchzuführen.
- Dies führte zu hohem Ressourcenverbrauch - nicht nur durch den TCP-Overhead, sondern auch durch wiederholte TLS-Handshakes.
- Ein weiteres Problem von HTTP/1.1 war die **textbasierte Kodierung** der Header. Viele davon müssen bei jeder Anfrage erneut übertragen werden.
- HTTP/2 ist nicht rückwärtskompatibel, verwendet aber denselben Port wie HTTP/1.1.
- Die **Protokollverhandlung** erfolgt meist über **ALPN (Application-Layer Protocol Negotiation)** im TLS-Handshake [18].

Neuerungen in HTTP/2

- Spezifiziert in RFC 7540 [7]
- **Binäre Kodierung** der HTTP-Nachrichten statt textbasierter Darstellung
- **Header-Kompression** mittels HPACK (RFC 7541 [24])
- **Server Push**: Der Server kann mehrere Antworten auf eine einzelne Anfrage senden (nicht zu verwechseln mit server-initiierten Anfragen!)
- Einführung einer **Framing-Schicht**, die mehrere parallele Streams über eine einzige TCP-Verbindung erlaubt
- **Streams** sind logisch voneinander getrennt, besitzen Prioritäten und werden durch **Frames** übermittelt
- **Flusskontrolle** erfolgt sowohl auf Verbindungs- als auch auf Stream-Ebene



Unsere schönen Schichten ...



Brian Trammell und Joe Hildebrand in

Evolving Transport in the Internet [27].



Brian Trammell war lange Zeit an der ETH Zürich

und arbeitet jetzt für Google.

Das Internet hat sich verändert ...



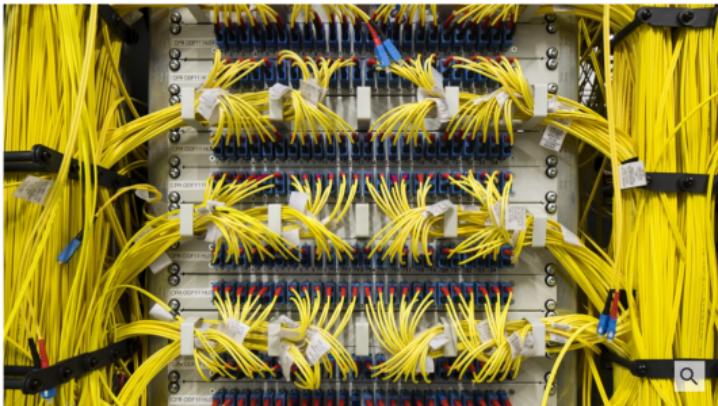
- Erste Experimente im ARPANET mit nur wenigen verbundenen Geräten
- 2023 ca. 1 Milliarde Geräte [3]

Revolution im Internet!

Neues Netz-Protokoll Quic

Revolution in den Tiefen des Internets

13. Januar 2021, 13:31 Uhr | Lesezeit: 4 min



Hier sausen die Daten: Am Frankfurter Internetknoten Decix werden viele Einzelnetze des Internets über Glasfaser miteinander verbunden. (Foto: DE-CIX Management GmbH/oh)

Quelle: [2]

Internetbeschleuniger: Die IETF lässt das QUIC-Protokoll vom Stapel

Nach Jahren der Entwicklung, 26 Konferenzen, fast 2000 Tickets und einigen tausend E-Mails ist das neue Transportprotokoll nun offiziell fertig.

Lesezeit: 4 Min.

¶ | Print | 130



(Bild: wk1003mike/Shutterstock.com)

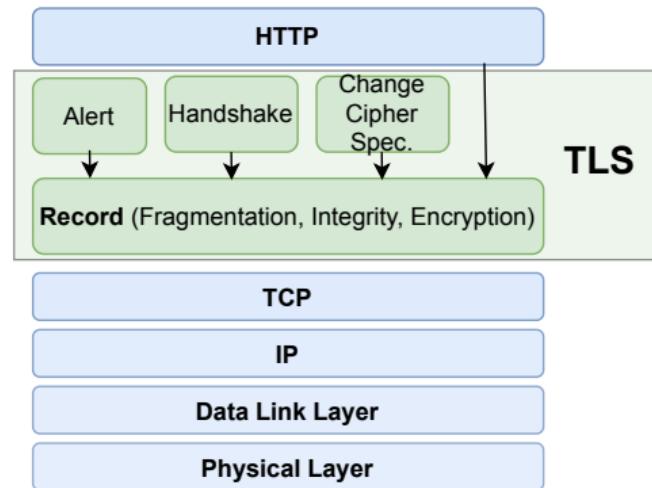
Quelle: [4]

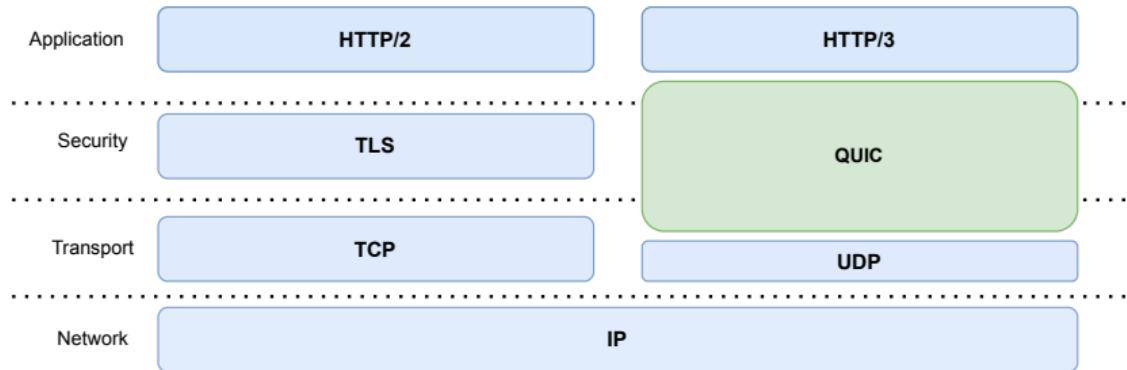
- HTTP/3 wurde im Jahr 2022 als **RFC 9114 [9]** veröffentlicht und entstand aus dem Wunsch, **TCP** durch ein neues Transportprotokoll zu ersetzen.
- Aufgrund der **Netzwerk-Versteinerung (network ossification)** war es praktisch nicht möglich, ein neues Transportprotokoll direkt einzuführen.
- Daher wurde **QUIC** als neues Protokoll entwickelt, das auf **UDP** basiert.
- Parallel dazu wurde **TLS 1.3** in der IETF TLS-Arbeitsgruppe entwickelt, um eine sichere Kommunikation mit optimiertem Roundtrip-Verhalten zu ermöglichen.
- QUIC integriert TLS 1.3 direkt in das Protokoll, ersetzt jedoch den **TLS Record Layer** (siehe RFC 9001).
- Da QUIC Funktionen enthält, die zuvor von HTTP/2 bereitgestellt wurden, wurde darauf aufbauend **HTTP/3** spezifiziert.



Eine Revolution?

- **Kosten der Abstraktion:** Eine Schicht kann nur auf die direkt benachbarten Schichten Einfluss nehmen.
 - Beispiel: Fragmentierung kann in mehreren Schichten erfolgen und erfordert eine komplexe Koordination im Protokollstack.
- **Beobachtung:** Die überwiegende Mehrheit des Internetverkehrs besteht aus:
 - HTTP+TLS+TCP+IP
- Idee: Durch die **Kombination** einiger dieser Protokolle kann Performance erheblich verbessert werden





- QUIC ersetzt TCP und integriert TLS 1.3.
- **QUIC bietet die gleichen Services wie TCP (Zuverlässigkeit, Flusskontrolle, Staukontrolle) sowie zusätzlich Authentifizierung und Verschlüsselung.**
- **ABER:**
 - QUIC läuft in der Anwendung (user-space) nicht im Betriebssystem (kernel space)!
 - QUIC nutzt UDP zur Übertragung, weil Änderungen an TCP nicht möglich waren.

Geschichte von QUIC

2012 Google arbeitet intern an QUIC (mit dem Namen SPDY)

2013 Erste öffentlich Experimente mit Chrome

2015 Google bringt Ideen zur IETF und eine Arbeitsgruppe wird gegründet.

2023 Gruppe veröffentlicht ersten RFCs:

- **9000 [20]: QUIC: A UDP-Based Multiplexed and Secure Transport**
- 9001 [26]: Using TLS to Secure QUIC
- 9002 [19]: QUIC Loss Detection and Congestion Control

2025 ca. 30% der Übertragungen nutzen QUIC [1]



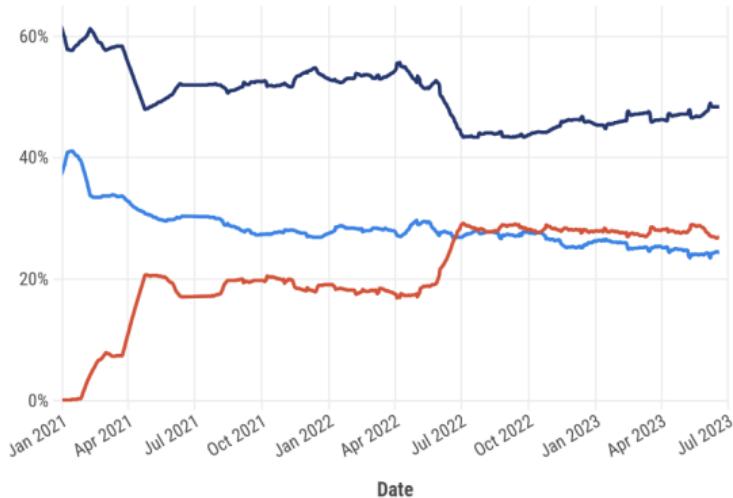
Martin Thomson, Mozilla Engineer, Mitglied des Internet Architecture Boards (IAB) und der W3C Technical Architecture Group (TAG), Co-Designer von QUIC und vielen anderen RFCs.

HTTP Protokolle im Einsatz

HTTP Versions In Use 2021 - 2023

HTTP Version v1.1 v2.0 v3.0

Percentage



Quelle: [6]



Mark Nottingham arbeitet für Cloudflare, ist seit 2007 Vorsitzender der HTTP-Arbeitsgruppe. Er leitete die QUIC-Arbeitsgruppe und war Mitglied des W3C Board of Directors, des Internet Architecture Board und der W3C Technical Architecture Group.

Ausgewählte Aspekte zu QUIC: UDP

- UDP ermöglicht es der Anwendung jedes Paket zu kontrollieren
- QUIC implementiert den zuverlässigen Transport in einer Anwendungsbibliothek, anstatt ihn der TCP-Bibliothek des Betriebssystems zu überlassen.
 - Dies ermöglicht es (bspw. Google in Chrome), QUIC zu integrieren, ohne das zugrunde liegende Betriebssystem zu ändern
 - Basiert auf (schlechten) Erfahrungen bei der Umstellung von IPv4 auf IPv6. Benötigte Updates von Routern, Firewalls, ...

Google Git

[chromium](#) / [chromium](#) / [src](#) / [net](#) / [refs/heads/main](#) / [.](#) / [quic](#)

tree: efb4783fd40fbalecb18d810d47c239eee556c2f [[path history](#)] [[tgz](#)]

[crypto/](#)
[platform/](#)
[address_utils.h](#)
[bidirectional_stream_quic_impl.cc](#)
[bidirectional_stream_quic_impl.h](#)
[bidirectional_stream_quic_impl_unittest.cc](#)
[COMMON_METADATA](#)
[crypto_test_utils_chromium.cc](#)
[crypto_test_utils_chromium.h](#)
[dedicated_web_transport_http3_client.cc](#)
[dedicated_web_transport_http3_client.h](#)
[dedicated_web_transport_http3_client_test.cc](#)
[DIR_METADATA](#)
[mock_crypto_client_stream.cc](#)
[mock_crypto_client_stream.h](#)
[mock_crypto_client_stream_factory.cc](#)

Quelle: [5]



Results Filter

Client:	quic-go	ngtcp2	mvfst	quiche	kwik	picoquic	aioquic	neqo	ng				
Server:	quic-go	ngtcp2	mvfst	quiche	kwik	picoquic	aioquic	neqo	ng				
Test:	3 172	6 218	H 215	DC 216	LR 215	C20 136	M 196	S 157	R 168	Z 145	B 217	U 154	T 203

Screenshot des QUIC Interop Runner Interfaces

Der **QUIC Interop Runner** führt automatisierte Interoperabilitäts- tests zwischen verschiedenen QUIC-Implementierungen durch.

Live verfügbar unter:

<https://interop.seemann.io>

Weitere Informationen im IETF-Blog:

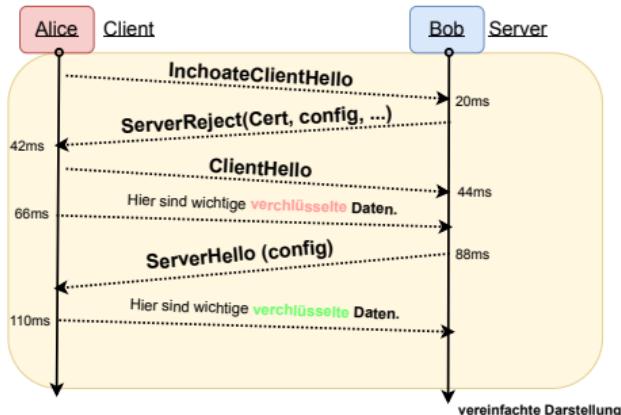
<https://www.ietf.org/blog/quic-automated-interop-testing/>

Ausgewählte Aspekte zu QUIC: Handshake

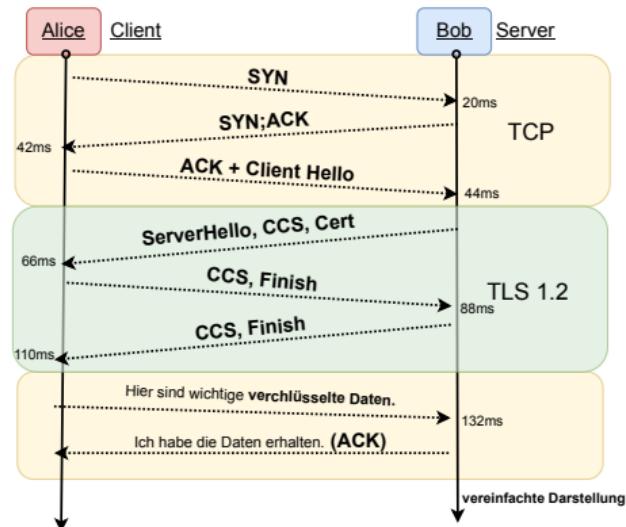
- Beobachtung:

- TLS kann seinen Schlüsselaustausch erst **nach dem** TCP-Handshake beginnen.
- Latenzzeit ist einer der wichtigste Faktor für die Leistung künftiger Netze.

QUIC:



TLS:



Ausgewählte Aspekte zu QUIC: Multi-Homing und Mobilität

- Beobachtung:

- Mobile Verbindungen werden abgeschaltet
- Mobile Geräte wechseln das Netzwerk (WiFi ⇔ 4G/5G)

- Problem:

- TCP Verbindungen haben ein Timeout, danach Neuaufbau
- TCP erkennt eine Verbindung anhand von IPs und Ports

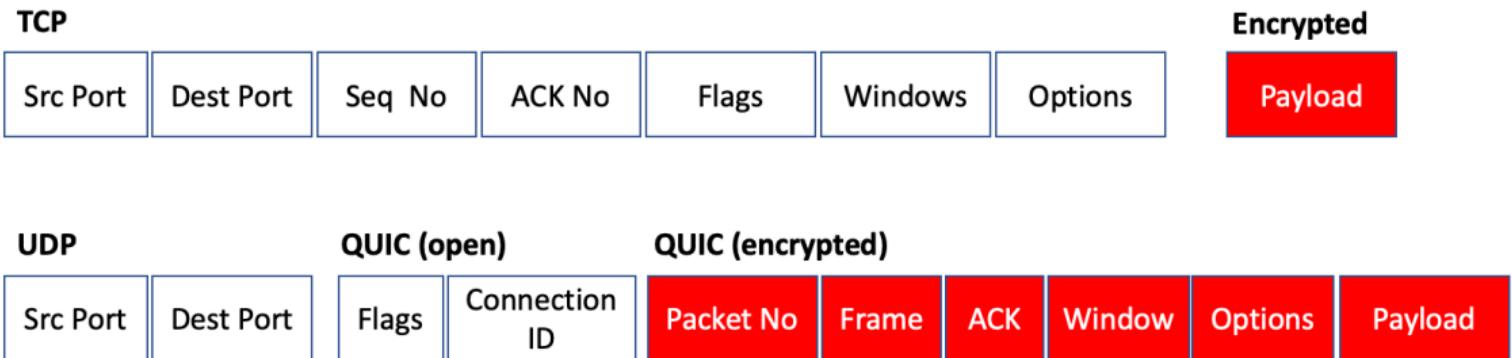
■ QUIC verwendet eindeutige Verbindungs-IDs:

- Wenn ein Gerät das Netzwerk wechselt (bspw. von WiFi ⇒ 4G/5G), kann es eine Verbindung wieder aufnehmen (bspw. Video-Streaming), indem es die gleiche QUIC Verbindungs-ID unter einer anderen IP-Adresse (und einem anderen UDP-Port) wieder verwendet.

UDP	QUIC (open)	QUIC (encrypted)							
Src Port	Dest Port	Flags	Connection ID	Packet No	Frame	ACK	Window	Options	Payload

Ausgewählte Aspekte zu QUIC: Implementierung und Security

- Implementierung von Algorithmen in der Applikation (bspw. Webserver/Browser)
 - Bessere Update Fähigkeit, Anpassung an den Kontext der Daten, ...
- Verschlüsselung und Authentifizierung ist Standard



Akronyme I

ALPN Application-Layer Protocol Negotiation

CDN Content Delivery Network

HOL Head-of-line

SSL Secure Socket Layer

TLS Transport Layer Security

URI Uniform Resource Identifier

URL Uniform Resource Locator

URN Uniform Resource Name

Quellen I

- [1] Cloudflare radar.
<https://radar.cloudflare.com/adoption-and-usage>.
(Accessed on 05/18/2023).
- [2] Das internet schneller machen - digital - sz.de.
<https://www.sueddeutsche.de/digital/internet-schneller-google-tcp-protokoll-verschlüsselung-1.5171790>.
(Accessed on 05/18/2023).
- [3] Internet hosts count log - geschichte des internets – wikipedia.
https://de.wikipedia.org/wiki/Geschichte_des_Internets#/media/Datei:Internet_Hosts_Count_log.svg.
(Accessed on 05/18/2023).
- [4] Internetbeschleuniger: Die ietf lässt das quic-protokoll vom stapel | heise online.
<https://www.heise.de/news/Internetbeschleuniger-Die-IETF-lässt-das-QUIC-Protokoll-vom-Stapel-6058718.html>.
(Accessed on 05/18/2023).
- [5] quic - chromium/src/net - git at google.
<https://chromium.googlesource.com/chromium/src/net/+/master/quic>.
(Accessed on 06/23/2024).
- [6] Why http/3 is eating the world.
<https://pulse.internetsociety.org/blog/why-http-3-is-eating-the-world>.
(Accessed on 06/19/2024).
- [7] Belshe, M., Peon, R., and Thomson, M.
Hypertext Transfer Protocol Version 2 (HTTP/2).
RFC 7540, May 2015.
- [8] Berners-Lee, T., Fielding, R. T., and Masinter, L. M.
Uniform Resource Identifier (URI): Generic Syntax.
RFC 3986, January 2005.
- [9] Bishop, M.
HTTP/3.
RFC 9114, June 2022.

Quellen II

- [10] Cooper, A.
Report from the Internet Privacy Workshop.
RFC 6462, January 2012.
- [11] Fielding, R. T., Lafon, Y., and Reschke, J.
Hypertext Transfer Protocol (HTTP/1.1): Range Requests.
RFC 7233, June 2014.
- [12] Fielding, R. T., Nielsen, H., Mogul, J., Gettys, J., and Berners-Lee, T.
Hypertext Transfer Protocol – HTTP/1.1.
RFC 2068, January 1997.
- [13] Fielding, R. T., Nottingham, M., and Reschke, J.
Hypertext Transfer Protocol (HTTP/1.1): Caching.
RFC 7234, June 2014.
- [14] Fielding, R. T., and Reschke, J.
Hypertext Transfer Protocol (HTTP/1.1): Authentication.
RFC 7235, June 2014.
- [15] Fielding, R. T., and Reschke, J.
Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests.
RFC 7232, June 2014.
- [16] Fielding, R. T., and Reschke, J.
Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing.
RFC 7230, June 2014.
- [17] Fielding, R. T., and Reschke, J.
Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.
RFC 7231, June 2014.
- [18] Friedl, S., Popov, A., Langley, A., and Stephan, E.
Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension.
RFC 7301, July 2014.



Quellen III

- [19] Iyengar, J., and Swett, I.
QUIC Loss Detection and Congestion Control.
RFC 9002, May 2021.
- [20] Iyengar, J., and Thomson, M.
QUIC: A UDP-Based Multiplexed and Secure Transport.
RFC 9000, May 2021.
- [21] Nielsen, H., Fielding, R. T., and Berners-Lee, T.
Hypertext Transfer Protocol – HTTP/1.0.
RFC 1945, May 1996.
- [22] 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.
- [23] Nottingham, M.
Building Protocols with HTTP.
RFC 9205, June 2022.
- [24] Peon, R., and Ruellan, H.
HPACK: Header Compression for HTTP/2.
RFC 7541, May 2015.
- [25] Rescorla, E.
HTTP Over TLS.
RFC 2818, May 2000.
- [26] Thomson, M., and Turner, S.
Using TLS to Secure QUIC.
RFC 9001, May 2021.
- [27] Trammell, B., and Hildebrand, J.
Evolving transport in the internet.
Internet Computing, IEEE 18 (09 2014), 60–64.



