Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Kapitel 2

Anwendungsschicht

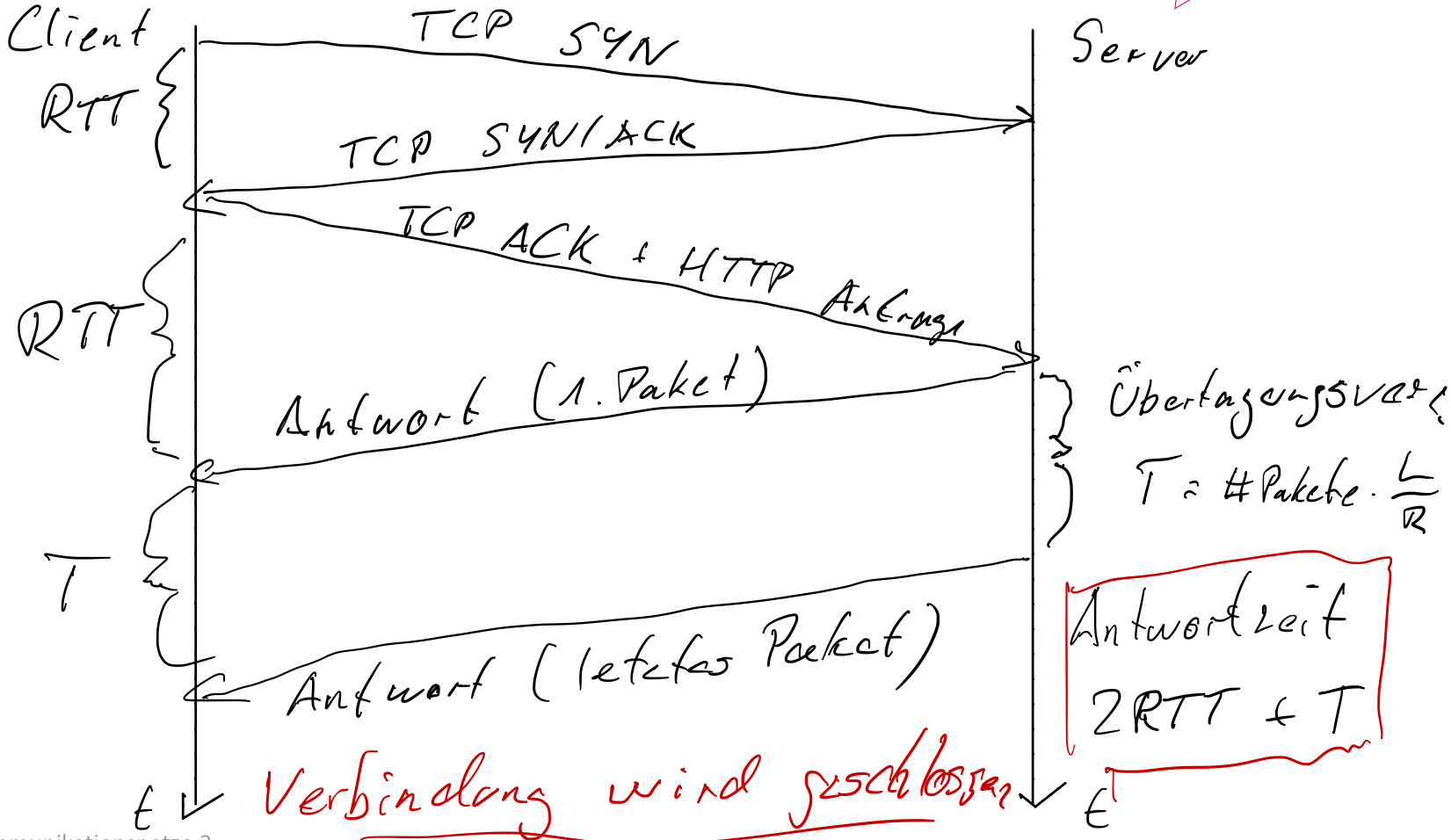
Vorlesung Kommunikationsnetze 2
Wintersemester 2018/19

Oliver P. Waldhorst

(Basierend auf Materialien von J. Kurose und K. Ross © 1996-2018)

Parallele Übertragung von Datenströmen

Wdh. HTTP/1.0


Client
Server
TCP SYN
RTT
TCP SYN/ACK
TCP ACK + HTTP Anfrage
RTT
Antwort (1. Paket)
Übertragungsverz.
T = #Pakete · L/R
T
Antwortzeit
2RTT + T
Antwort (letztes Paket)
Verbindung wird geschlossen
t
t

HTTP/1.0

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

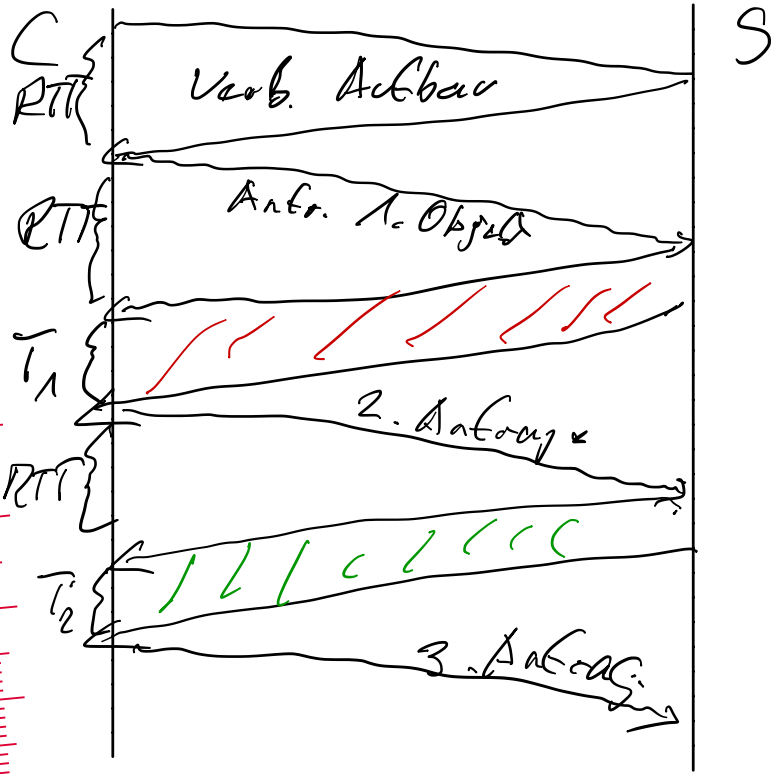
Bei n Objekten, sequentiell:

$$\sum_{i=1}^{n} (2RTT + T_i)$$

T_i ist Übertragungszeit von Objekt i

HTTP/1.1

Persistente Verbindungen

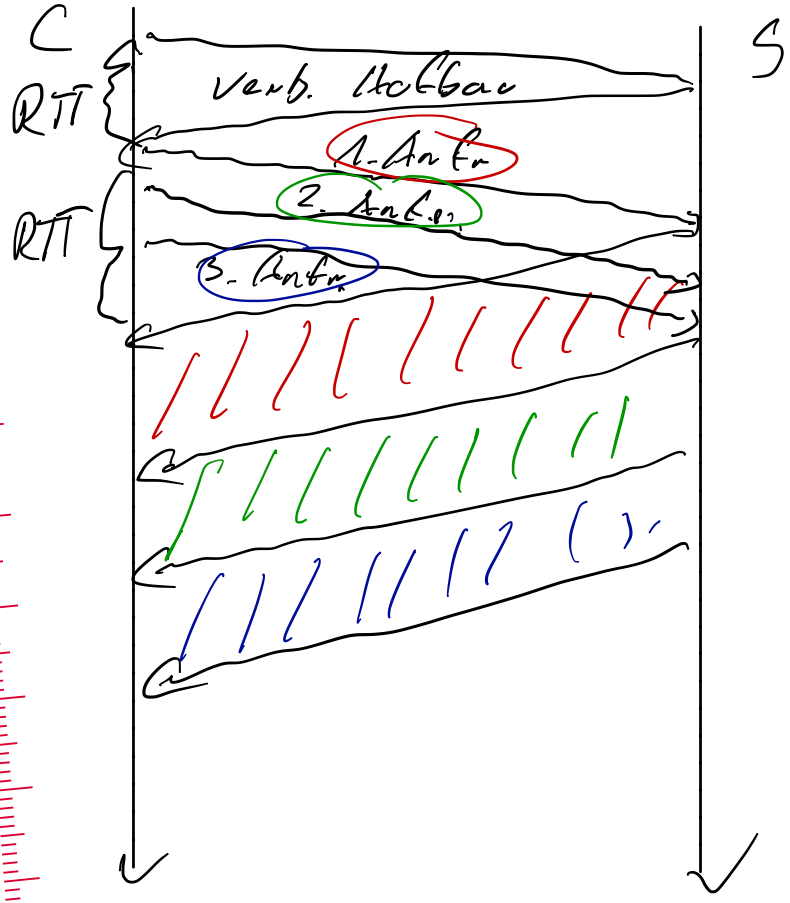


$$(n+1)\,RTT + \sum_{i=1}^{n} T_i$$

$\Rightarrow$ $n \cdot RTT$ schneller

wg. Wegfall Verbindungsaufbau

HTTP/1.1 Pipelining



$$2RTT + \sum_{i=1}^{n} T_i$$

nochmals

$\approx n \cdot RTT$ schneller!

Außerdem nur 1x Slow Start!

Problem HTTP 1/1

Größe der Objekte

Objekt 1

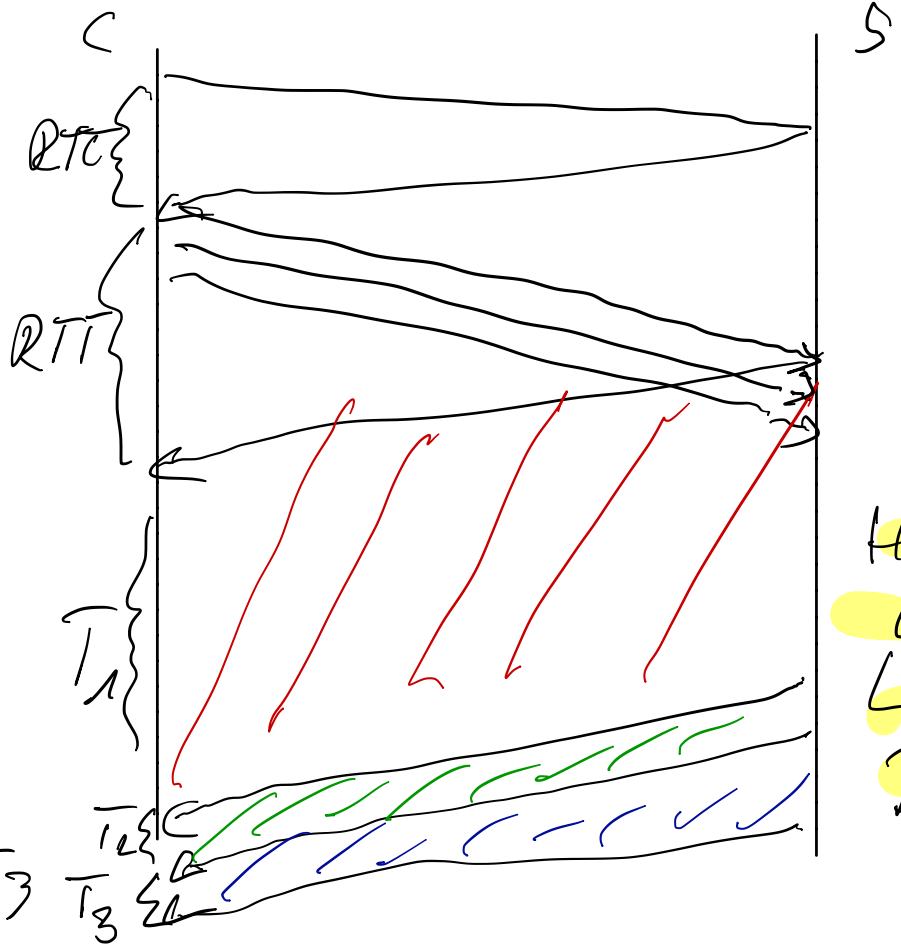
Objekt 2

Objekt 3

1. Objekt: $2RTT + T_1$

2. Objekt:
$2RTT + T_1 + T_2$

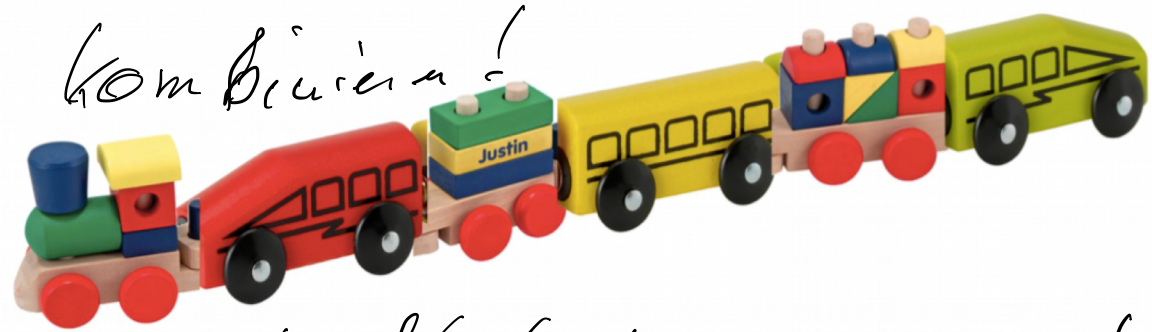
3. Objekt:
$2RTT + T_1 + T_2 + T_3$



Head of Line Blocking

Sequentiell: Züge nacheinander

Datenströme kombinieren!



Vorteil bei unterschiedlich langen Zügen!

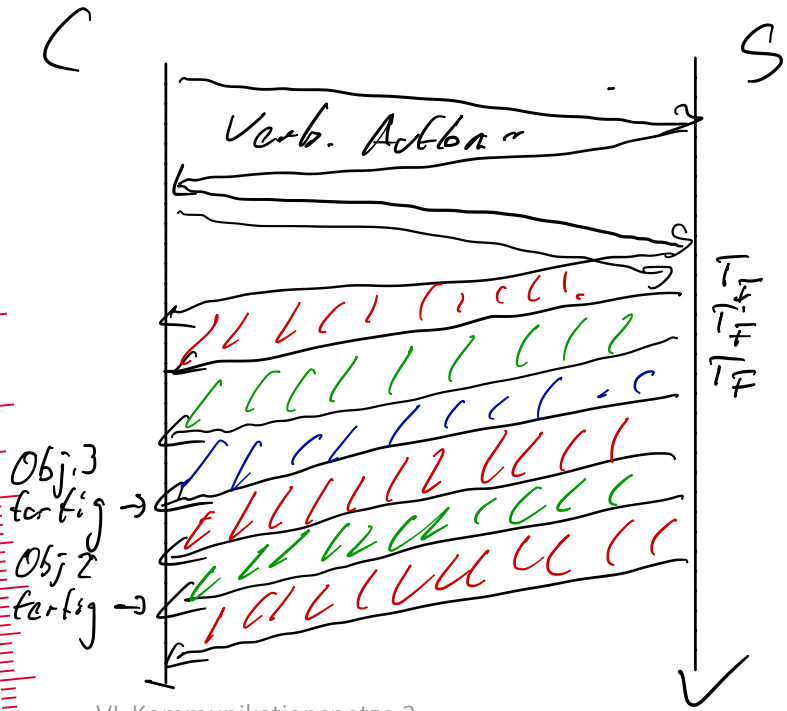
Quelle: https://www.myf5.net/upload/2015/05/http2-v1.12.pdf

Wie geht es besser?

→ Objekte S_i in Stücke fester Größe zerteilen

Frame mit Größe S_F

→ Stücke reihum (round-robin) übertragen



$S_1 = 3 \cdot S_F$

$S_2 = 2 \cdot S_F$

$S_3 = 1 \cdot S_F$

⇒ Gerechter!

→ Fragmentierung + Flußkontrolle!

HTTP/2 [RFC 7540]

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

→ Designed für Problem: Webseite mit vielen Objekten

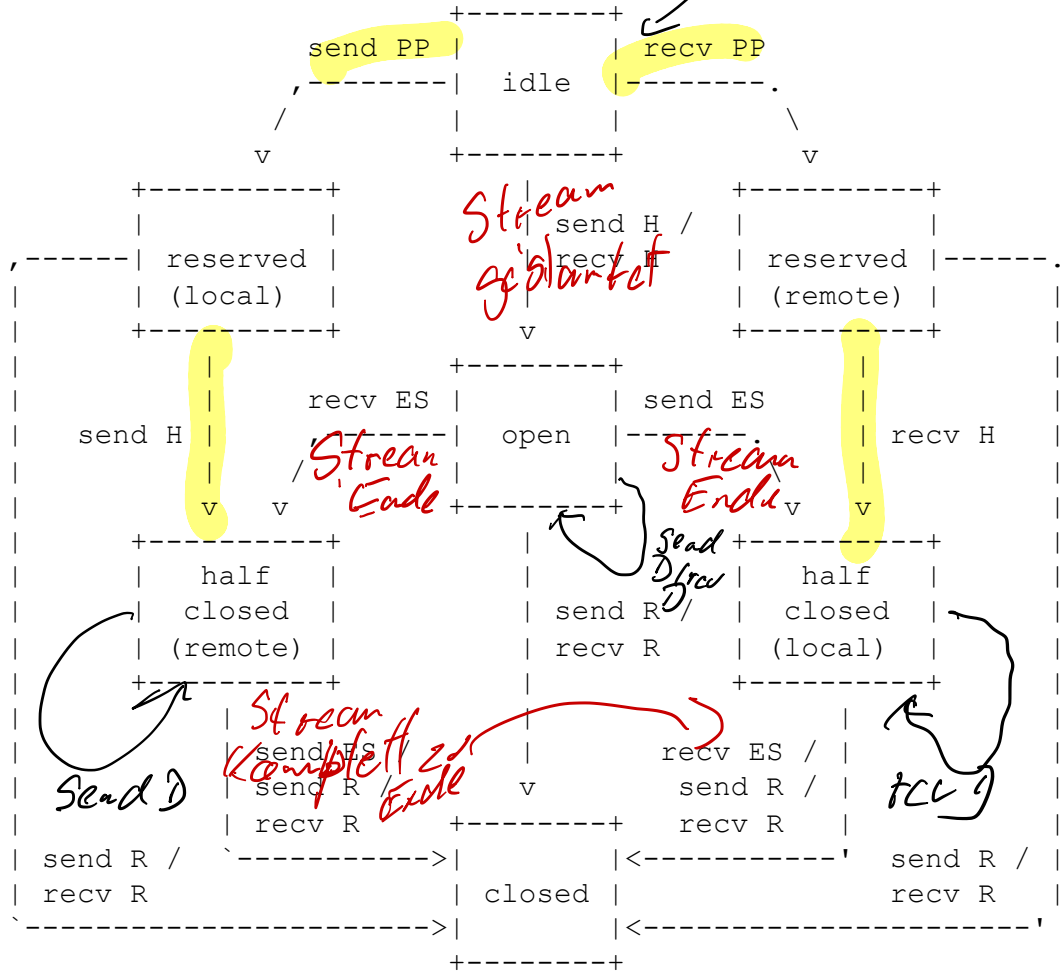
→ Historie: Google SPDY

→ Binärformat mit FRAMES
- Length min. 2^{14} Byte max. $2^{61}-1$ Byte
- Type (Data, Header, ...)
- Flags (8 Bit)
- Stream Identifier (31 Bit)
- Nutzdaten

Streams + Multiplexing

- Stream: Einer pro Objekt der Webseite
  Logische Assoziation:
  Bidirektionale Folge von Frames
  mit gleichem Stream Identifier.
- Viele Streams gleichzeitig geöffnet

Stream nicht aktiv



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

H: Handshake (Stream starten)

ES: End of Stream

R: Reset

PP: Push Promise

D: Data

Frame-Typen

Prioritäten + Abhängigkeiten

- Jeder Stream hat Priorität
- Stream kann Abhängigkeiten zu anderen Streams haben!

→ Browser kann Reihenfolge beeinflussen

→ Dynamisch änderbar über

PRIORITY Frame für Stream ID

Enthält:
- Stream ID
- Priorität
- Liste anderer Stream IDs, zu denen Abh. besteht

Flusskontrolle (vgl. TCP Flusskontrolle)

- Sender darf nur so viel Senden, wie Empfänger verarbeiten kann
- Flusskontrollfenster ∃ Soviel kann noch empfangen werden
  - → Pro Stream
  - → Insgesamt für alle Streams (ID 0)
- Angezeigt durch WINDOW_UPDATE Frame für Stream ID!

Flusskontrolle (2)

C

S

SETTINGS-FRAME

Fenstergröße pro Stream Y

Gesamtfenster X

DATA(1), SIZE=S

$W(1) := Y - S$

$W(0) := X - S$

Verarbeitet

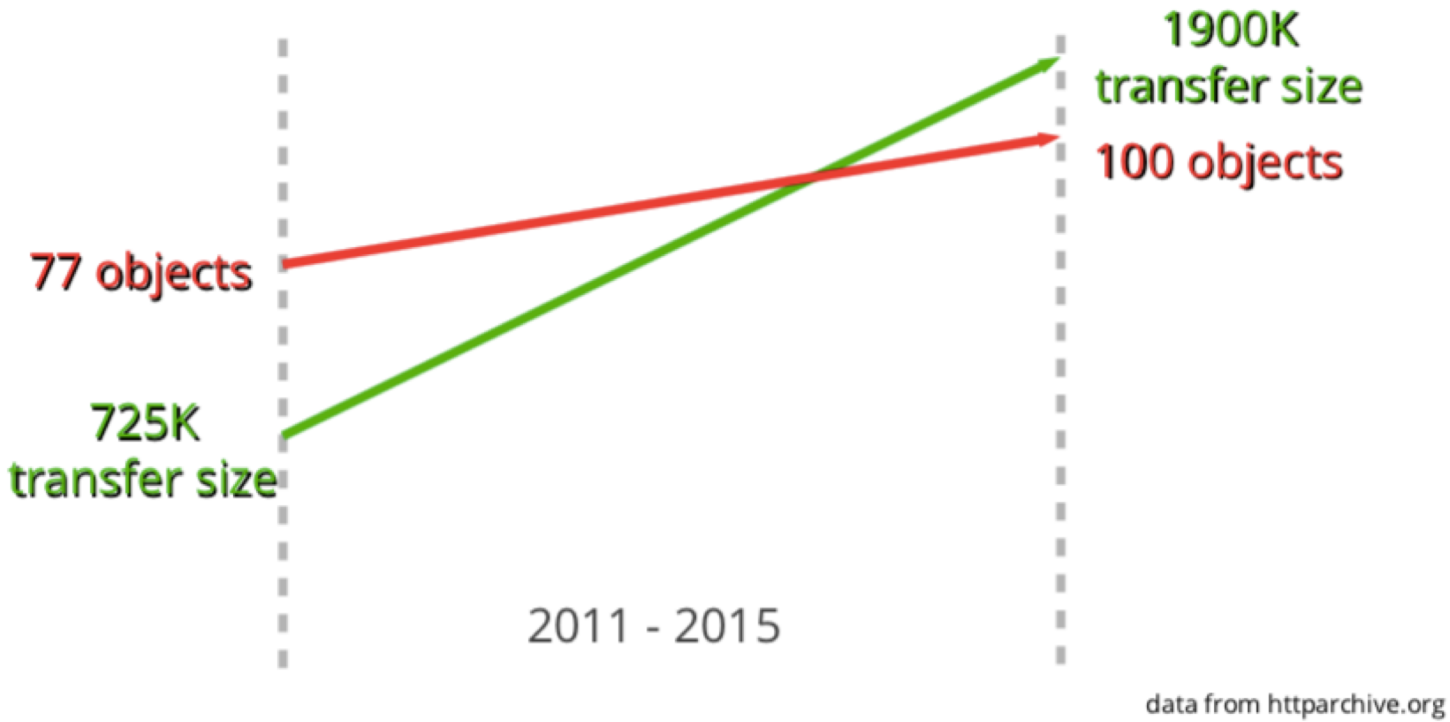
WINDOW_UPDATE(1,S)

Kann vom Server selbst berechnet werden

$W(1) := Y - S + S = Y$

$W(0) := X - S + S = X$

Einschub: Objekte pro Web-Seite



Quelle: https://www.myf5.net/upload/2015/05/http2-v1.12.pdf

Einschub: RFC-Terminologie [RFC 2119]

1. MUST This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.

2. MUST NOT This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.

3. SHOULD This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

4. SHOULD NOT This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

5. MAY This word, or the adjective "OPTIONAL", mean that an item is truly optional. [...]

Einschub: Parallele Verbindungen in HTTP/1.1 [RFC 2616]

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Clients that use persistent connections SHOULD limit the number of simultaneous connections that they maintain to a given server. A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy. A proxy SHOULD use up to 2*N connections to another server or proxy, where N is the number of simultaneously active users. These guidelines are intended to improve HTTP response times and avoid congestion.

Einschub: Parallele Verbindungen in der Realität

Browser	HTTP/1.1	HTTP/1.0
IE 6,7	2	4
IE 8	6	6
Firefox 2	2	8
Firefox 3	6	6
Safari 3,4	4	4
Chrome 1,2	6	?
Chrome 3	4	4
Chrome 4+	6	?
iPhone 2	4	?
iPhone 3	6	?
iPhone 4	4	?
Opera 9.63,10.00alpha	4	4
Opera 10.51+	8	?

Quelle: http://www.stevesouders.com/blog/2008/03/20/roundup-on-parallel-connections/

Einschub: Sharding als Lösung?

GET	/	www.aftonbladet.se	html	205.71 KB	→ 32 ms
GET	general.css?57785785	www.aftonbladet.se	css	193.22 KB	→ 10 ms
GET	d-head.js?57785785	www.aftonbladet.se	js	190.40 KB	→ 20 ms
GET	aftonbladet.gif	gfx.aftonbladet-cdn.se	gif	6.33 KB	→ 44 ms
GET	sanna_lundell_white.jpg	gfx.aftonbladet-cdn.se	jpeg	3.01 KB	→ 44 ms
GET	pappa01.jpg	gfx2.aftonbladet-cdn.se	jpeg	18.66 KB	→ 228 ms
GET	forskare.jpg	gfx.aftonbladet-cdn.se	jpeg	6.88 KB	→ 44 ms
GET	story26jan.jpg	gfx.aftonbladet-cdn.se	jpeg	2.66 KB	→ 45 ms
GET	osbalk.jpg	gfx2.aftonbladet-cdn.se	jpeg	4.11 KB	→ 238 ms
GET	Bank.jpg	gfx2.aftonbladet-cdn.se	jpeg	16.09 KB	→ 247 ms
GET	Tatu.JPG	gfx.aftonbladet-cdn.se	jpeg	7.38 KB	→ 44 ms
GET	sve8.jpg	gfx.aftonbladet-cdn.se	jpeg	7.39 KB	→ 45 ms
GET	otto-tvaa.jpg	gfx.aftonbladet-cdn.se	jpeg	6.71 KB	→ 46 ms
GET	AStridTT.JPG	gfx.aftonbladet-cdn.se	jpeg	6.52 KB	→ 66 ms
GET	nimoy.jpg	gfx.aftonbladet-cdn.se	jpeg	7.87 KB	→ 67 ms
GET	PSYK0§.jpg	gfx2.aftonbladet-cdn.se	jpeg	18.28 KB	→ 257 ms
GET	parislgh.jpg	gfx2.aftonbladet-cdn.se	jpeg	21.89 KB	→ 267 ms
GET	ProjectRunwayvinnare.jpg	gfx2.aftonbladet-cdn.se	jpeg	7.83 KB	→ 276 ms
GET	par_strand-opt.jpg	gfx2.aftonbladet-cdn.se	jpeg	5.72 KB	→ 286 ms
GET	mack.jpg	gfx2.aftonbladet-cdn.se	jpeg	16.27 KB	→ 299 ms
GET	skildaVpuffmitt.jpg	gfx1.aftonbladet-cdn.se	jpeg	4.49 KB	→ 144 ms
GET	socker_ny.jpg	gfx1.aftonbladet-cdn.se	jpeg	3.79 KB	→ 153 ms
GET	andreetta.jpg	gfx1.aftonbladet-cdn.se	jpeg	5.53 KB	→ 163 ms
GET	skidor_50.gif	gfx.aftonbladet-cdn.se	gif	3.20 KB	→ 70 ms

Quelle: https://www.myf5.net/upload/2015/05/http2-v1.12.pdf

Header Compression

- Viele Requests haben die gleichen Header
- Viele Header passen nicht in erste TCP Pakete bzw. initialer cwnd.
- Kompression der Header in HEADER-Frame
- „Header Compression for HTTP/2 HPACK"
  [RFC 7541]

Kompatibilität

- Keine neuen URLs → gut für Nutzer!
- Keine neuen Std.-Ports: 80 bzw. 443 (http://... https://...)
- Wie erkennt man HTTP/2 Unterstützung?

HTTP/2-fähiger Browser sendet:

```
GET / HTTP/1.1
Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c
HTTP2-Settings: <base64url encoding of HTTP/2 SETTINGS payload>
```

z.B. Flusskontroll-Fenster

→ HTTP/2 im Klartext

Kompatibilität (2)

HTTP/2 - fähiger Web-Server:

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: h2c

[ HTTP/2 connection ...
```

„Alter" Server:

```
HTTP/1.1 200 OK
Content-Length: 243
Content-Type: text/html

...
```

https://

→ Aushandlung bei TLS-Verbindungsaufbau (später)

Verbreitung

→ Unterstützung in allen wichtigen Browsern 2015 [Wikipedia]

https://w3techs.com/technologies/details/ce-http2/all/all

→ ≈ 30% umgestellt!

Zusammenfassung HTTP/2

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Motivation: Head-of-Line-Blocking

- → Objekte in Frames geteilt
- → Mehrere Streams „parallel" (round-robin)
- → Flusskontrolle
- → Prioritäten
- → URLs kompatibel zu HTTP/1.x

Übertragung von Multimedia-Inhalten (Audio / Video)

Motivation

- Video on demand (Netflix, Amazon Prime, Maxdome, ...)

- Internet-Telefonie (inkl. Video) (Skype, Google Talk, Facetime, ...)

Funktionsweise digitales Video

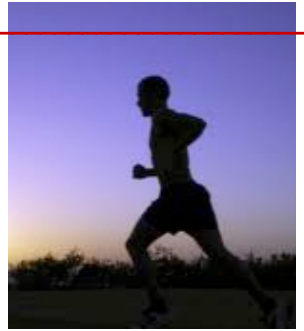


Bild 1

→ alle blau

→ 1x Senden (räumlich)

Bild

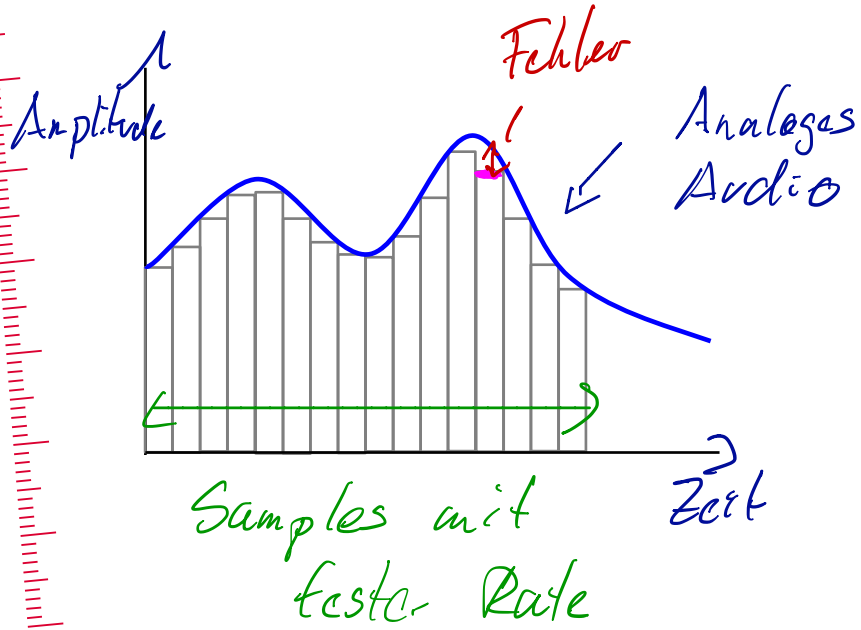
Fast alle Punkte gleich

⇒ Unterschiedlich Senden (zeitlich)

- Bild = Array von Bildpunkten
- Bildpunkt = Bit
- Bilderfolge = Video
  (z.B. 24 Bilder/s)
  Bild = Frame
- Video-Kodierung nutzt räumliche / zeitliche Redundanz
- Konstante oder variable Bitrate

Bsp.: MPEG1 (CD-ROM) < 1,5 Mb/s
MPEG2 (DVD) < 3-6 Mb/s
MPEG4 (Internet) < 1 Mb/s

Funktionsweise Digitales Audio



- Abtastung (Sampling) mit fester Rate
- Sample-Werte 8 Bit
  → $2^8 = 256$ versch. Werte
  → nicht alle Werte darstellbar → Fehler
- Telefon: 8.000 Samples/s
  CD 44.100 Samples/s

Telefon Datenrate
64.000 b/s ≈ 64 kb/s

Bsp. digitales Audio

- oben vorgestellt: Pulscode Modulation (PCM)
- andere: z.B. MPEG 1 Audio Layer 3 (MP3)
  Advanced Audio Codec (AAC)

- Datenraten:
  CD: 1,411 Mb/s
  MP3: <320 kb/s
  Telefon: 5,3 kb/s

→ Nutzer sind empfindlicher gegenüber schlechter Audio-Qualität als Video-Qualität

Klassifizierung A/V-Anwendungen

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

→ Streaming von gespeicherten Inhalten (Netflix...)
- Streaming = Abspielen während des Herunterladens
- Anhalten, Vorspulen, Springen ...
- Kontinuierliche Wiedergabe

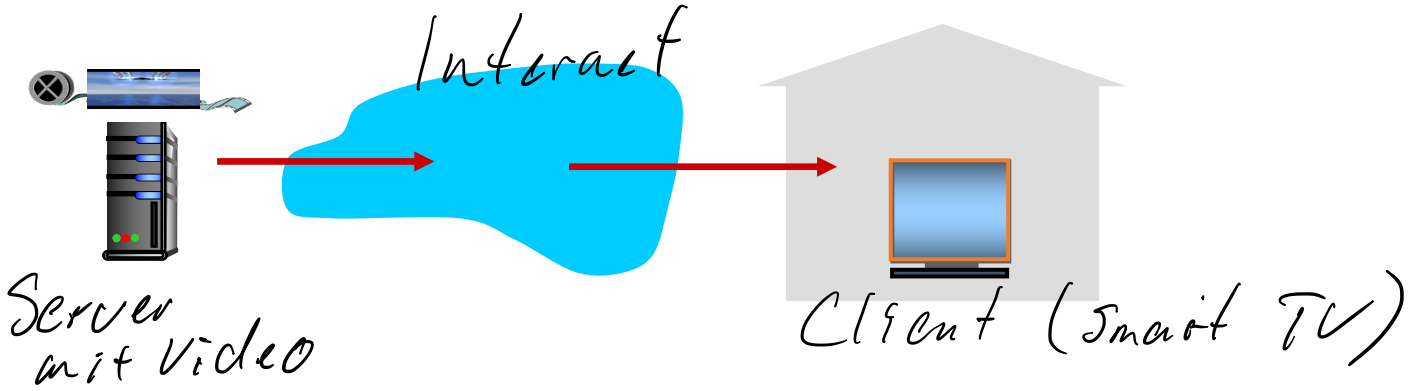
→ Dialogorientiertes A/V (Skype, ...)
- 2 oder mehr Teilnehmer
- Verzögerung <150ms, bei >400ms nicht nutzbar
- Verlusttolerant

→ Streaming von Live A/V (TV-Stationen)
- Verz. von <10s bei Start OK!

Streaming von gespeichertem A/V

Streaming gespeicherter A/V

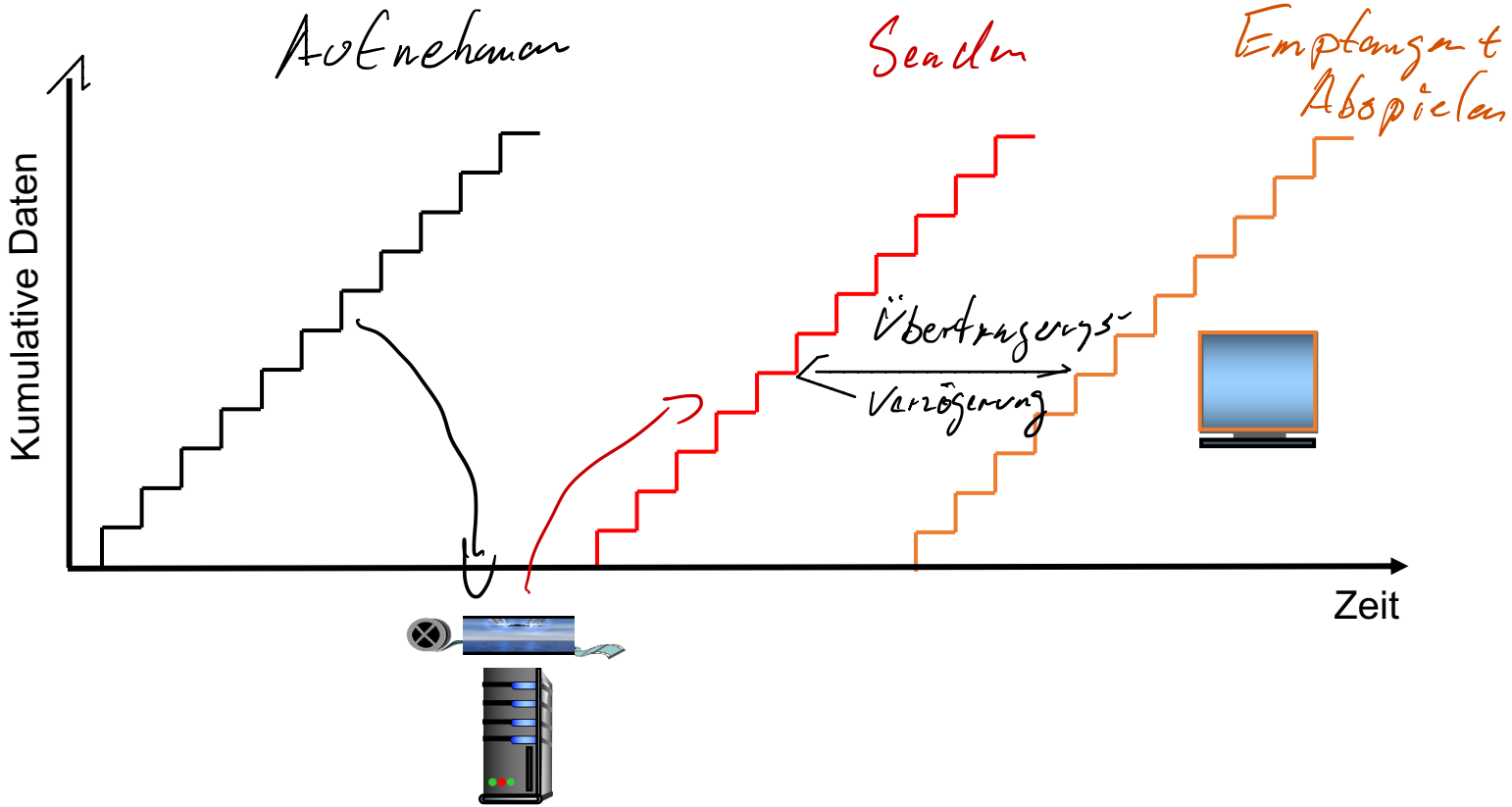
Youtube: 16% Verkehr, 1 Mrd Nutzer
Netflix: 37% Verkehr, 75 Mio Nutzer



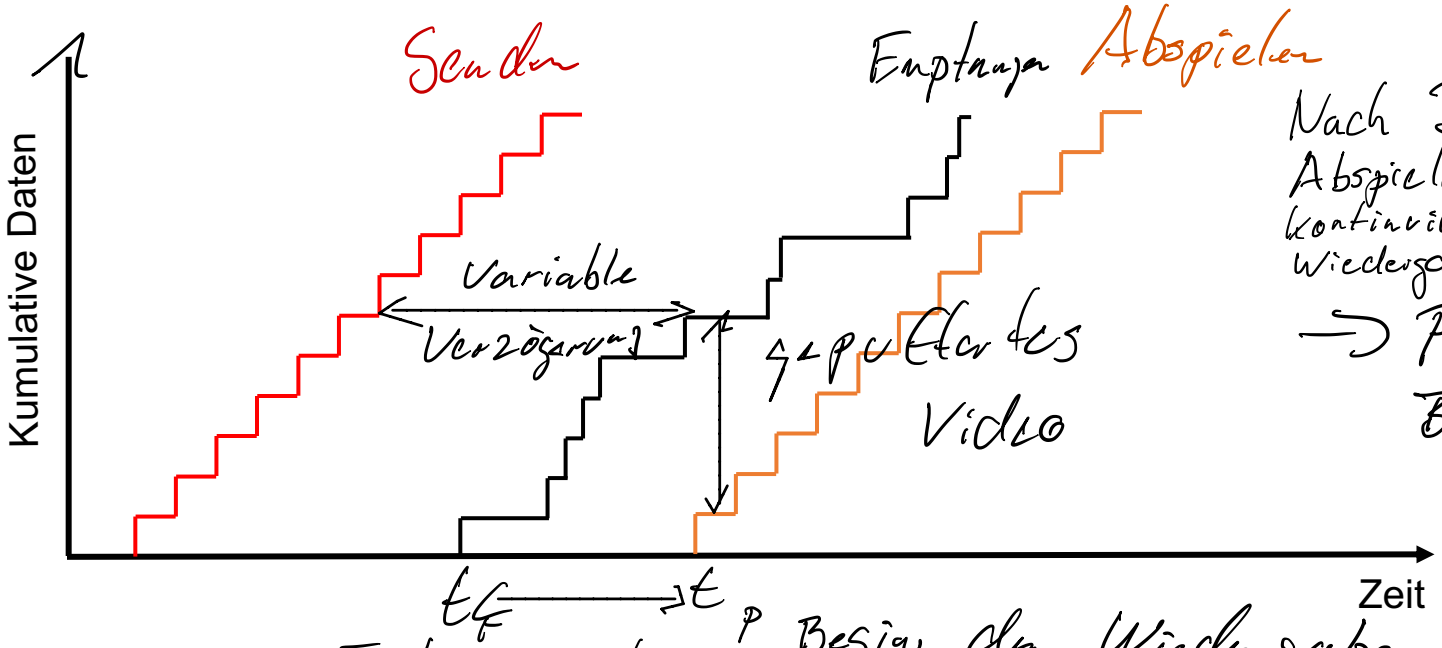
Ansätze

- UDP-Streaming
- HTTP-Streaming
- adaptives HTTP-Streaming

Video-Übertragung (idealisiert)



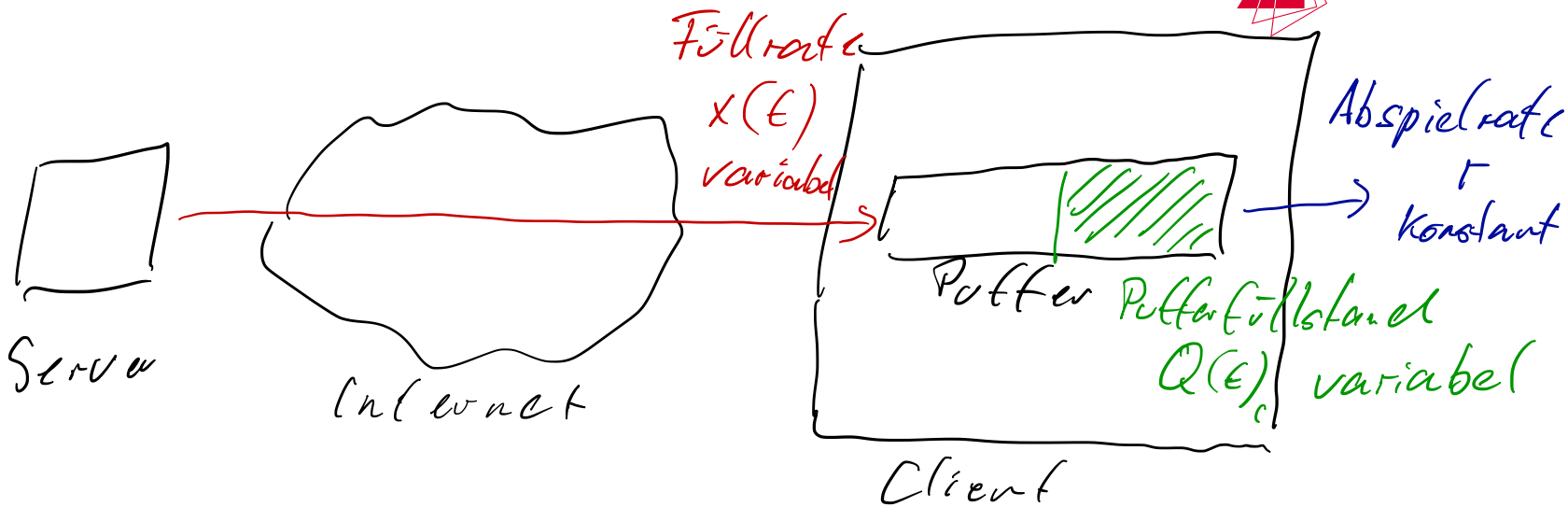
Reale Video-Übertragung



Nach Beginn des Abspielens kontinuierliche Wiedergabe
→ Puffer benötigt

Wiedergabeverzögerung
$d_t = t_p - t_r$

Puffern und Abspielen



- Puffer füllen vor Wiedergabestart t_p
- Füllstand $Q(t)$ schwankt, weil r fest, aber $x(t)$ variabel

Puffern und Abspielen (2)

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Für durchschnittliche Füllrate $\bar{x}$:

$\bar{x} < r$: Puffer läuft leer, Bild steht

$\bar{x} > r$: Puffer läuft niemals leer, wenn t_p groß genug gewählt

r über Kodierung bestimmbar

t_p klein: ggf. Puffer leer

t_p groß: Nutzer genervt!

} Bei guter Wahl kontinuierliche Wiedergabe!

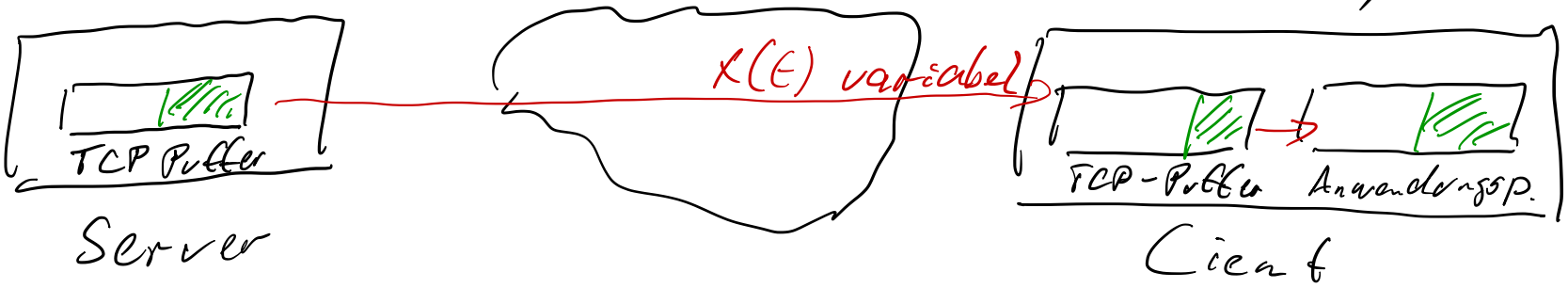
UDP-Streaming

- Senderate = Abspielrate = Bitrate Video
  Bsp.: Video-Rate 2 Mb/s, UDP-Paket 8000b
  ⇒ alle 4ms ein UDP-Paket
- Unabhängig von Stausituation?
- Puffer + kurze Wiedergabeverzögerung (2s-5s)
- UDP in Realtime-Protocoll RTP [RFC 3550]

- Nachteil: UDP geht nicht durch NAT

HTTP-Streaming (TCP!)

- Multimedia-Datei über HTTP-Get holen (vom Client aus → kein NAT-Problem)



→ Maximale TCP-Senderate / Staukontrolle

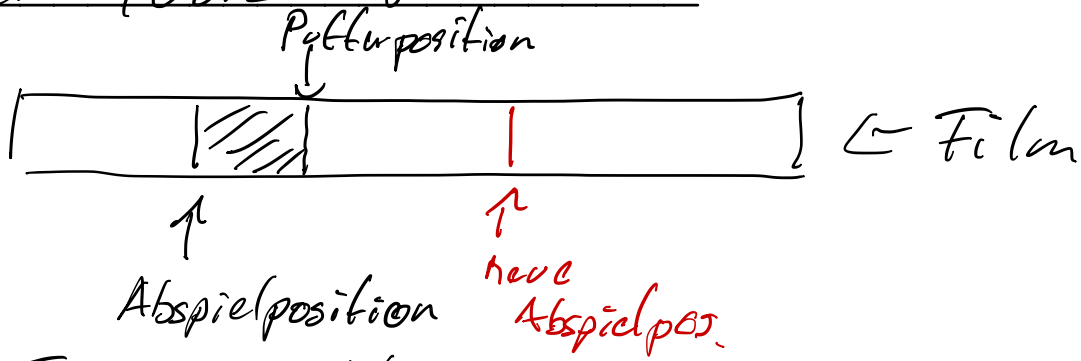
→ Flusskontrolle → Sender stopt automatisch bei Pause

→ Zuverlässigkeit

→ Faustformel: Max. Übertragungsrate > 2 · Wiedergaberate

→ Geht durch NAT!

Neupositionieren / vorzeitiges Beenden



- neue Frames nicht gepuffert.
- verschwendete Ressourcen für bereits übertragene Stücke

⇒ Entweder Puffer begrenzen
oder Stücke mit HTTP-Range: Header holen, z.B.

HTTP Anfrage

```
GET ...
Range: bytes=0...1023
  ⋮
```

HTTP Antwort

```
HTTP/1.1 206 Partial Content
Content-Range: 0-1023/14651
  ⋮
```

Dynamic Adaptive Streaming over HTTP (DASH)

Hochschule Karlsruhe Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Motivation: Optimale Rate r für unterschiedliche Clients unterschiedlich!

Aufgabe Server:

- Video in Stücke gleicher Länge (Chunks) zerlegen
- Chunks liegen in unterschiedlichen Kodierungen vor (8 - 10 x MPEG 4)
- Manifest: Liste der URLs der kodierten Chunks

DASH (2)

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Aufgabe des Clients:

- Bandbreite messen / Pufferfüllstand beobachten
- Auswahl Chunks aus Manifest!
  - -> Beste Kodierung für gegebene Bandbreite
  - -> Wechsel Kodierung pro Chunk möglich
- Intelligenz im Client:
  - Wann Chunk? Welche Rate?

Content Delivery Networks CDN

- Viele Videos, viele Nutzer => zentraler Server nicht möglich

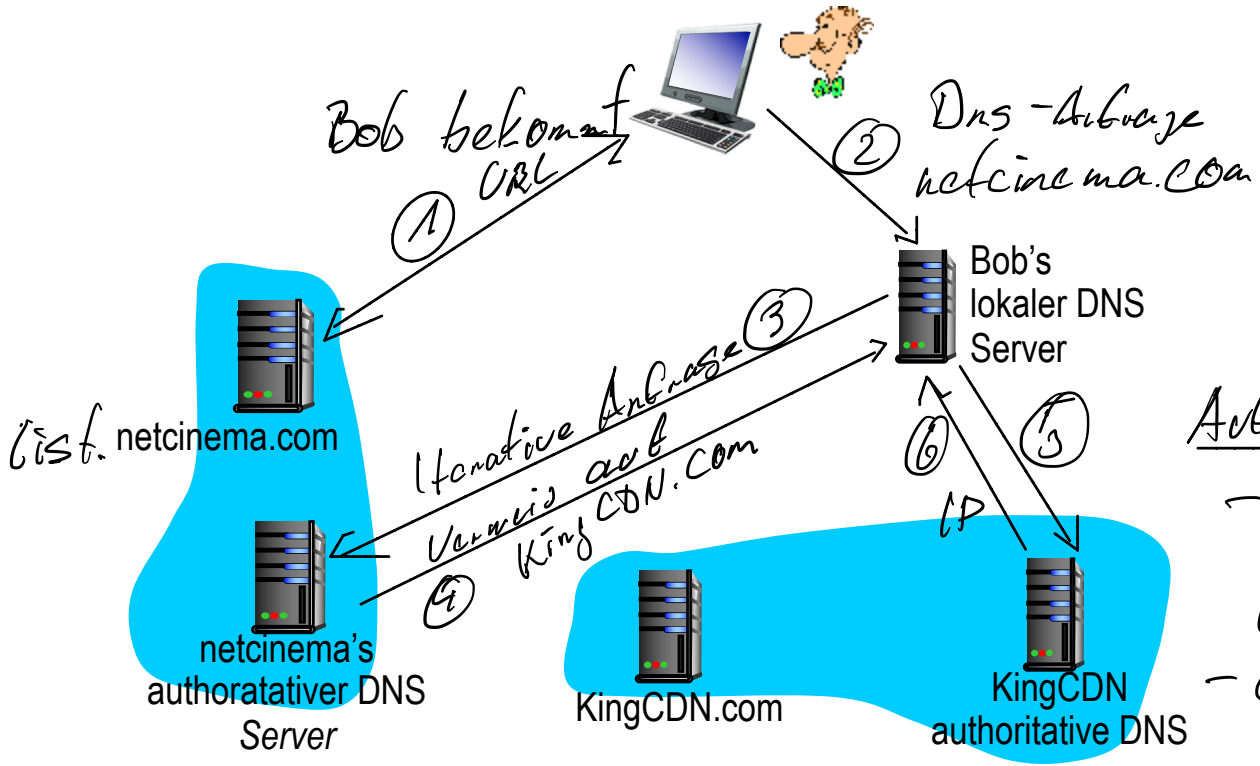
- CDN-Betreiber bietet viele „kleinere" Server, geografisch verteilt!

  z.B. Akamai in Zugangsnetzen 1700 Server
  „Enter Deep" Strategie → Cover!
  Limelight 10-100 Server Pop Tier 1 BB
  „Bring Home" Strategie → höhere Latenz

Wie finde ich den passenden Server

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Bob (client) fragt nach Video http://netcinema.com/6Y7B23V



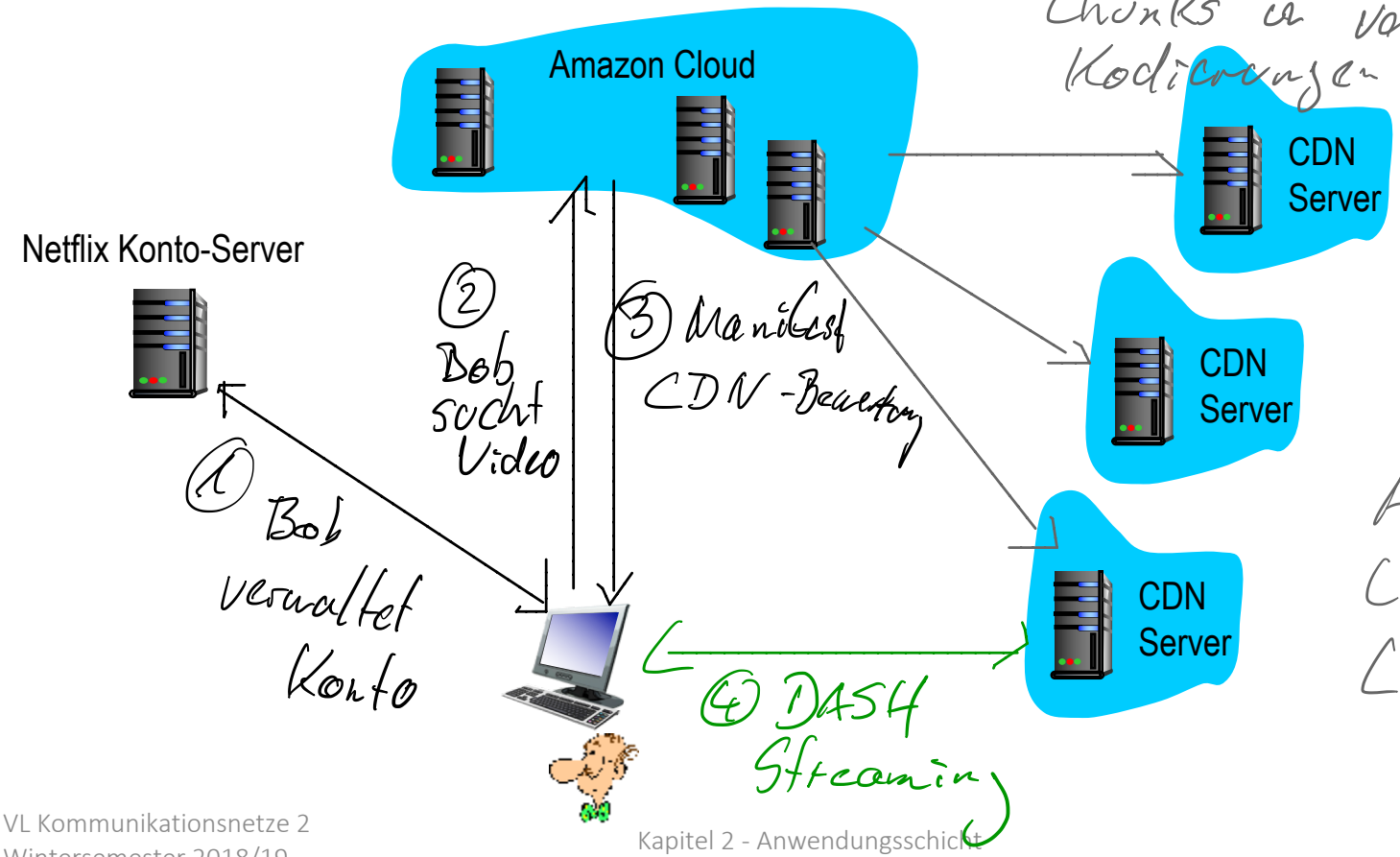
Auflösung bei KingCDN:

- geografische Nähe (z.B. IP lokalen DNS-S)
- Last auf Servern
- Anycast (über BGP)

Case Study Netflix

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Netflix erzeugt Chunks in verschiedenen Kodierungen (vorher)



Mehrere CDNs

Akamai, Limelight Level-3 (2012)

Zusammenfassung Streaming gespeichertes A/V

- Schwankende Übertragungsverzögerung
  → Puffer + Wdg.-Verzögerung
- Schwankende / unterschiedliche Verbindungsqualität
  → Video chunks mit unterschiedl. Qualität
  DASH
- Million verteilter Nutzer
  → Content Delivery Networks

Dialogorientiertes A/V

(am Beispiel Voice-over-IP, VoIP)

Motivation

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Bsp.: 8000 B/s, Blöcke 20 ms

⇒ alle 20 ms mit 160 Byte über UDP

Verzögerungen

- < 150 ms ab 400 ms richtig schlecht

Verzögerungsschwankungen (Jitter)

- Abstände sehr unterschiedlich!

Paketverluste

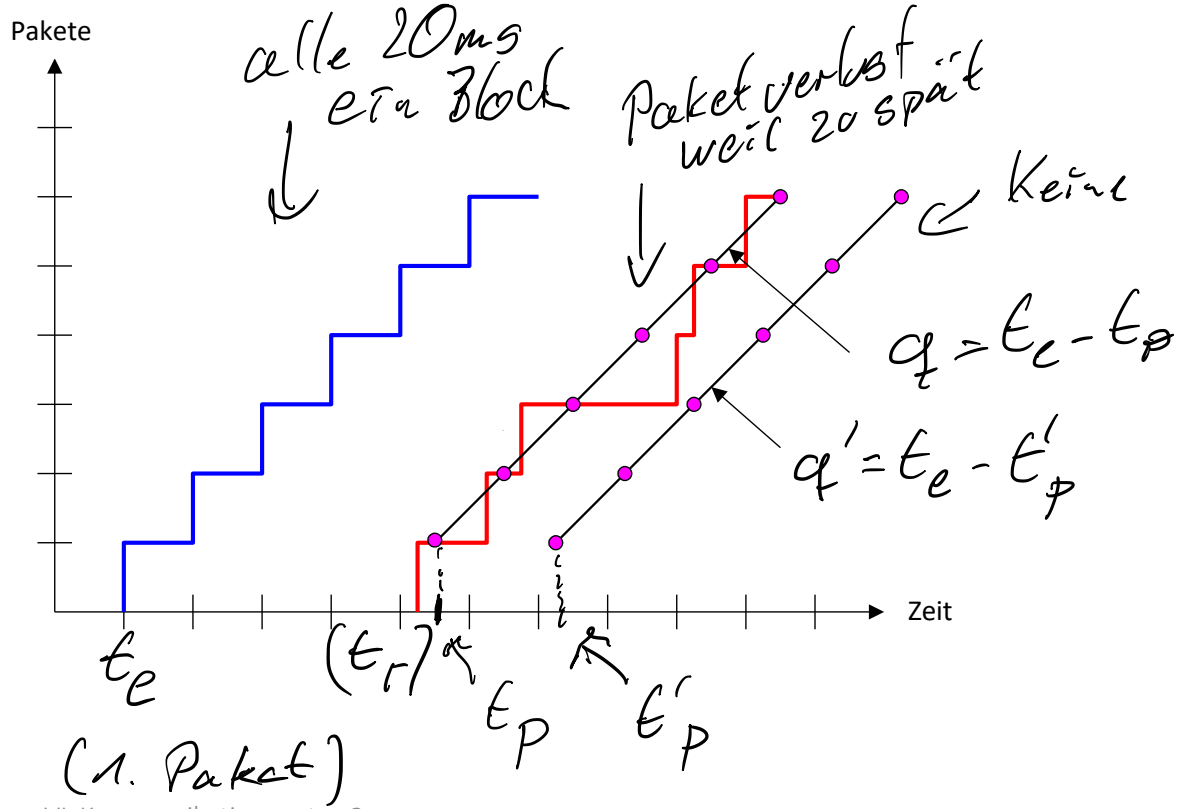
- < 10% geht noch ⇒ Vorwärtsfehlerkorrektur!

Umgang mit Jitter

- Jeder Block des Audio-Stroms hat Zeitstempel t_e
- Beim Empfänger wird Block zu Zeitpunkt t_p
- Wiedergabeverzögerung $d = t_p - t_e$ (Achtung: anders als für gesp. A/V)

$\Rightarrow$ Empfänger spielt jeden Block genau Zeit d nach Erzeugung!

Feste Abspielverzögerung



festes q schlecht!

Adaptive Wiedergabeverzögerung

Ziel: Kleine Wdg.-Verz. + Verlustrate

Ansatz:
- Schätzen der Empfangsverzögerung über Zeitpunkt t_r
- q vor jeder Sprechphase einstellen bzw. in Stillephasen anpassen

Schätzung Übertragungszeit durch exponentiell geglätteter Mittelwert (→ vgl. TCP)

$$d_i = (1-\alpha)\, d_{i-1} + \alpha \underbrace{(t_{r,i} - t_{e,i})}_{\text{Übertragungszeit von Paket } i}$$

(α z.B. 0,01)

d_i: Schätzung nach Paket i

d_{i-1}: Schätzung vor Paket i

Adaptive Wiedergabeverzögerung (2)

- Berechnung der Standardabweichung (-> vgl. TCP)

$$v_i = (1-\beta)v_{i-1} + \beta \, |\underbrace{t_{r,i} - t_{e,i}}_{\text{i-2. akt. Paket}} - d_i|$$

- d_i, v_i werden ständig berechnet, aber nur am Anfang jeder Sprechphase!

$$a_f = d_i + 2v_i \qquad \text{(z.B. für } k=4\text{)}$$

- danach geht es periodisch weiter, z.B. alle 20ms
- Erkennung von Stille auf Senderseite (Signalpegel)
  -> Erkennung auf Empfängerseite beim Puffer
  Seq-Nummern, Aufnahmezeiten

Umgang mit Paketverlusten

→ darf Verzögerung nicht erhöhen!
(Ack / NACK braucht RTT!)

→ Alternative Lösung: Vorwärtsfehlerkorrektur
(Forward Error Correction FEC)

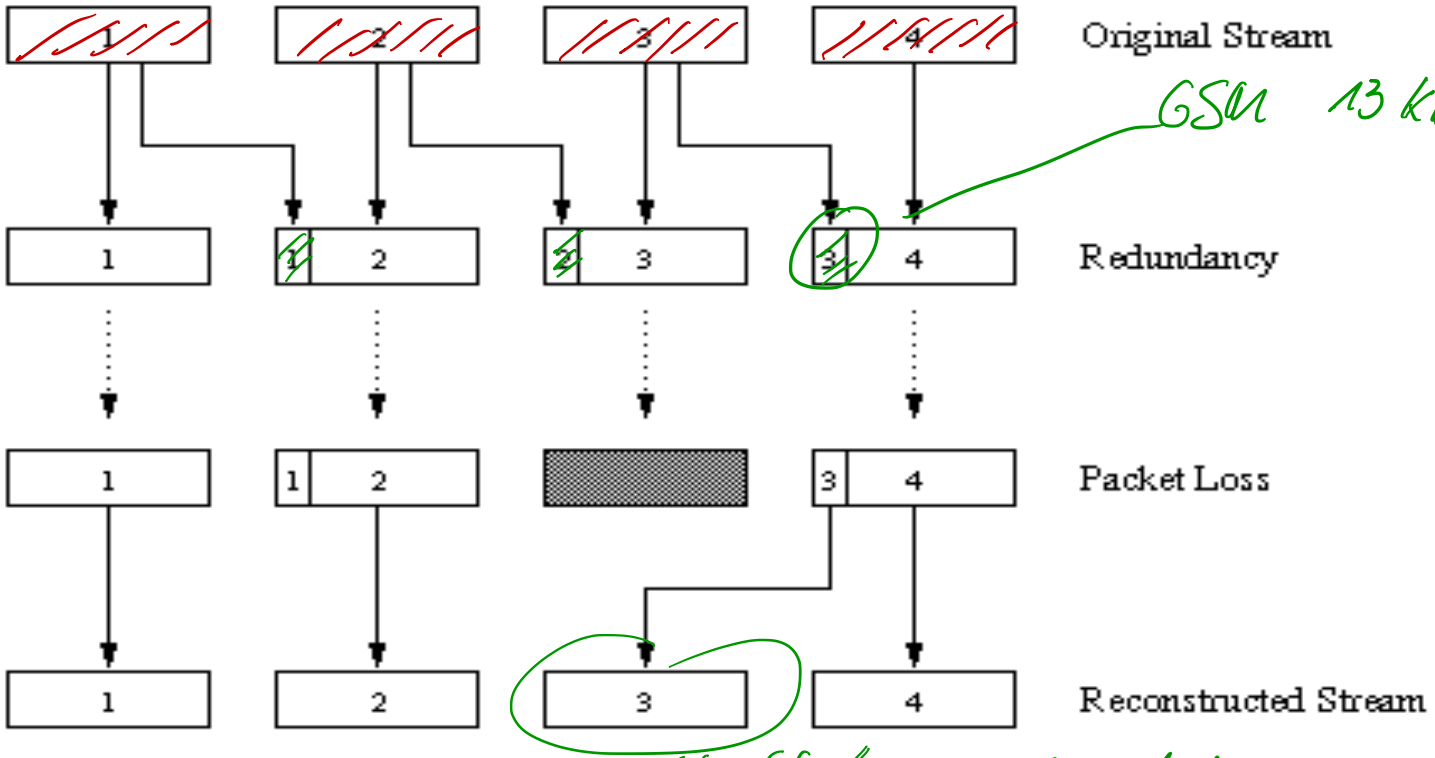
Bsp.

→ n Pakete, bilden von XOR-Verknüpfung
⇒ n+1 Pakete, Bandbreitenbedarf steigt um $\frac{n+1}{n}$

→ n Originalpakete lassen sich wieder herstellen, wenn max. 1 Paket verloren geht

Komplexe FEC für VoIP (1)

PCM
64 kb/s



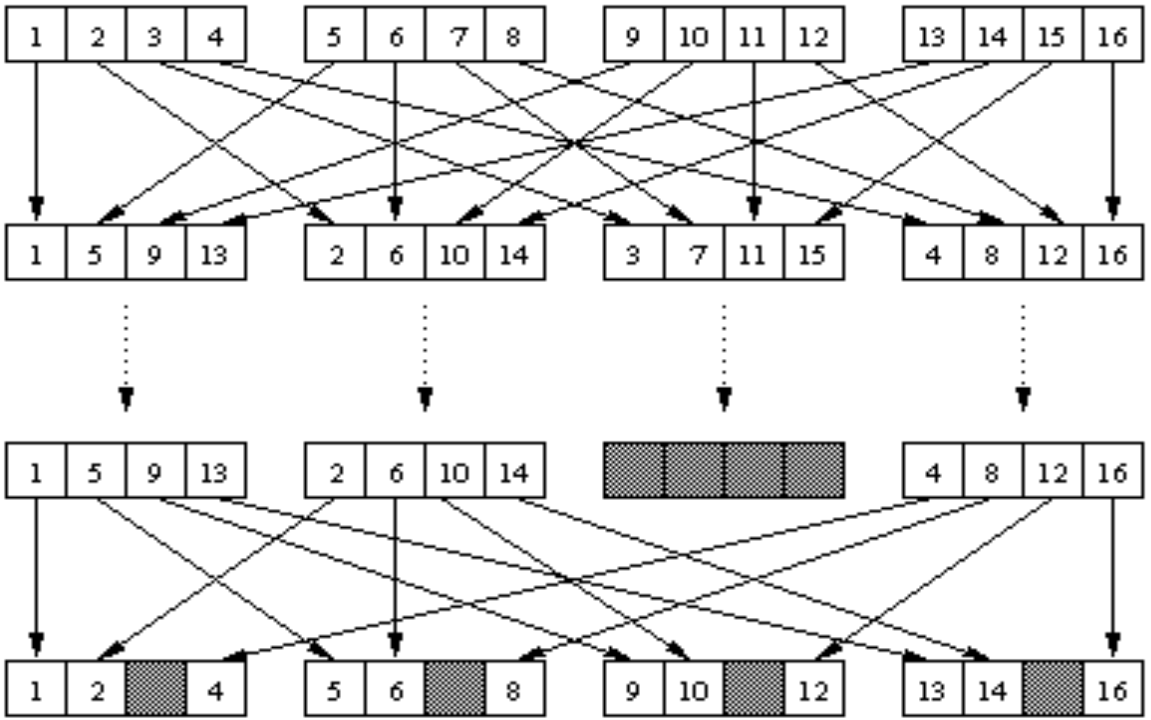
„Schlechte" Qualität!

Nicht-aufeinander folgende Verluste ausgleichbar!

Komplexe FEC für VoIP (2)

Aufteilung in kleinere Stücke (z.B. 4 x 5ms)

Block 20ms



Verwürfelung der Stücke versch. Blöcke!

↳ wir bekommen fast alles

Nachteil: größere Wiedergabeverzögerung!

Offene Protokolle: Realtime Protocol (RTP)

[RFC 3550]

- Struktur für Pakete mit A/V-Daten
- Läuft auf UDP (→ Ports, IP-Adressen, ...)
  → RTP verpackt in UDP

- Wenn zwei Clients RTP beherrschen, können sie <u>vielleicht</u> kommunizieren
  ... wenn Codecs vorhanden!

RTP-Struktur

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Sequenznr. (Reihenfolge / Verluste)

Source-ID (audio/Video-Quelle) ⇒ Ströme trennen!

7	16	32	32	...

Payload-Type (PCM, ...)

Zeitstempel (angepasst an Sampling-Rate)

(Zahlen sind Byte)

Bemerkungen RTP

- RTP läuft in Anwendungsschicht!
  - Router sehen nur UDP (sgt. NAT-Problem)
  - Keine Netzunterstützung (Garantien, ...)
- Wir sind auf Senderseite "blind"

  ⇒ Statusinformationen zwischen Sender(n) und Empfänger(n) über

  Realtime Control Protocol RTCP [RFC3550]

  Statistiken: Gesendete Pakete, Verzögerung, Verluste, ...)

  < 5% des Datastroms!