Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

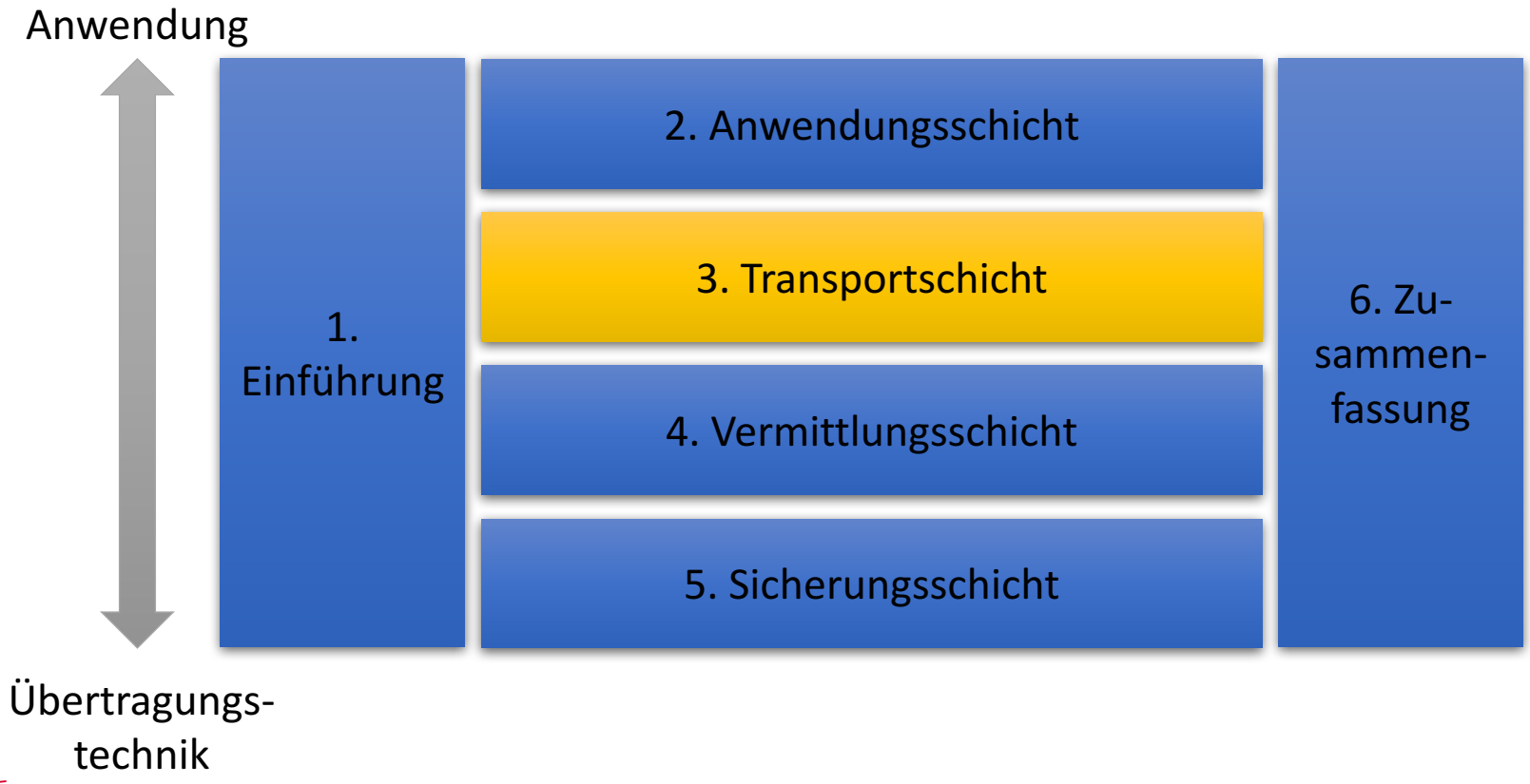
Kapitel 3

Die Transportschicht

Vorlesung Kommunikationsnetze 1
Wintersemester 2017/18

Oliver P. Waldhorst

Gliederung der Vorlesung



Ziele dieses Kapitels

Verständnis der grundlegenden Prinzipien der Dienste der Transportschicht

- Multiplexing, Demultiplexing
- Zuverlässiger Datentransport
- Flusskontrolle
- Staukontrolle

Kennenlernen der Transportschichtprotokolle im Internet

- UDP: Verbindungsloser Transport
- TCP: Verbindungsorientierter Transport

Dienste und Protokolle der Transportschicht

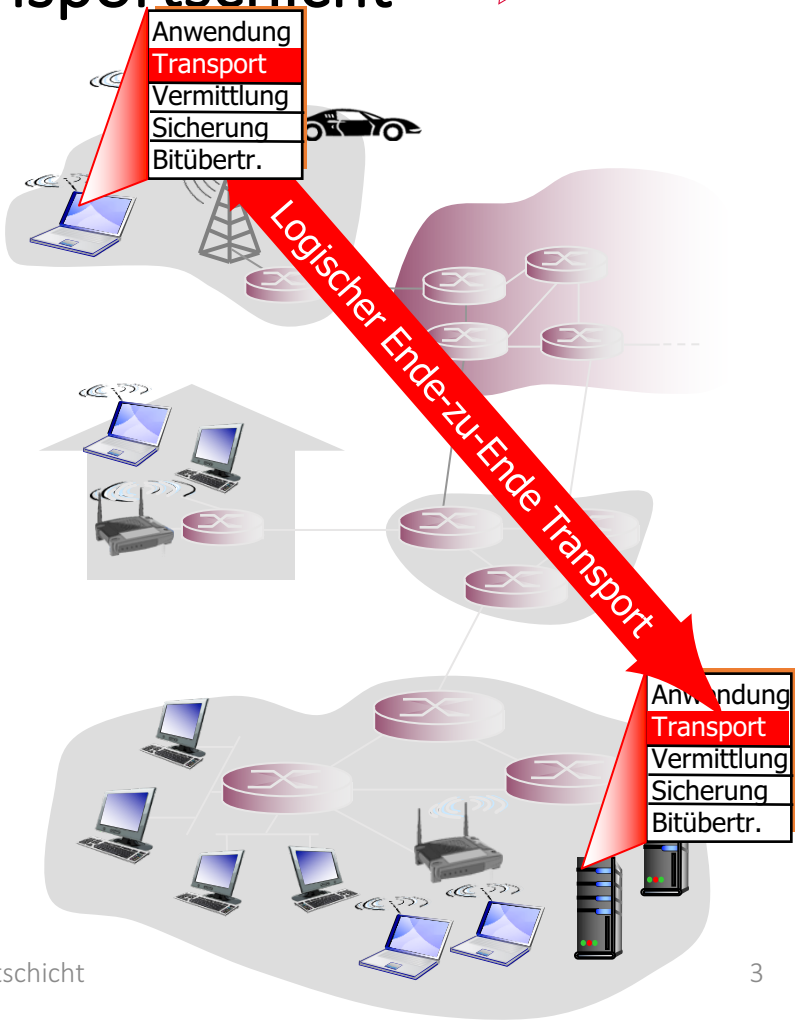
Ziel: Bereitstellung einer logischen Kommunikationsverbinddung zwischen Anwendungsprozessen auf verschiedenen Endsystemen

Transport-Protokolle laufen auf Endsystemen

- Sender-Seite: Zerlegen von Anwendungsnachrichten in Segmente, die an die Vermittlungsschicht weitergegeben werden
- Empfänger-Seite: Zusammensetzen der Segmente zu Nachrichten, die an die Anwendungsschicht weitergegeben werden

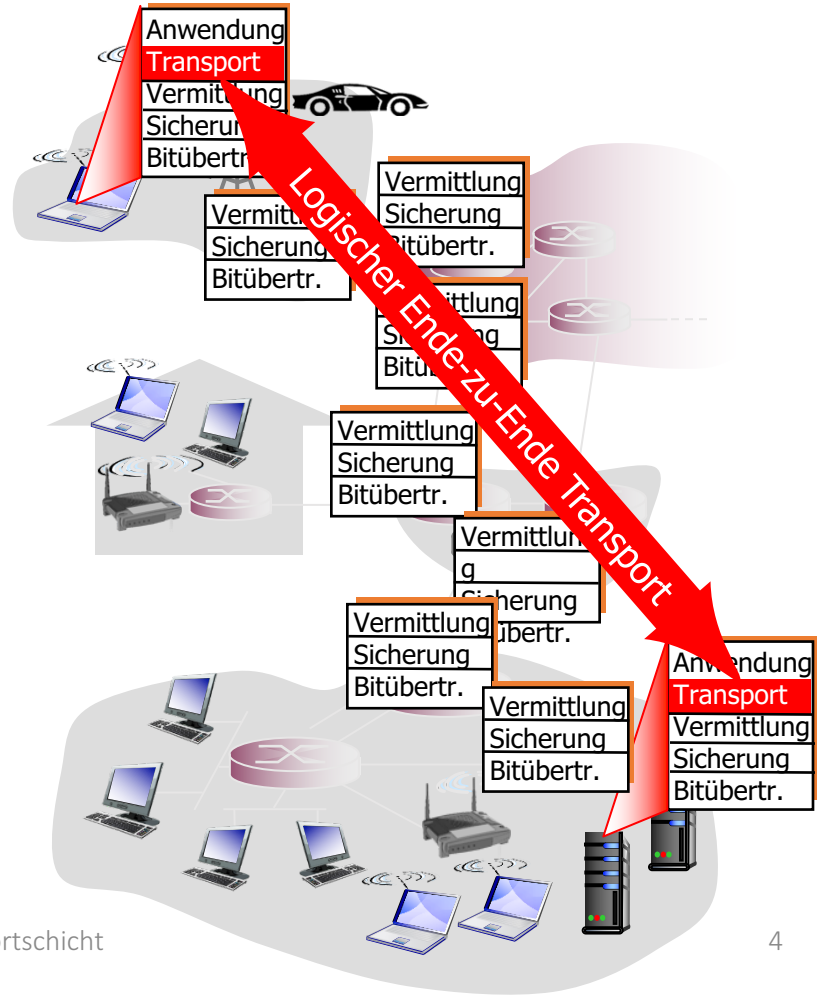
Im allgemeinen mehrere Transportprotokolle verfügbar

- Im Internet: TCP und UPD



Transportprotokolle im Internet

- Zuverlässiger, Reihenfolge-erhaltender Transport (TCP)
  - Staukontrolle
  - Flusskontrolle
  - Verbindungsauf- und -abbau
- Unzuverlässiger Transport (UDP)
  - Einfache Erweiterung des Internet-Protokolls IP
- Nicht verfügbar
  - Garantien für Verzögerungen
  - Bandbreite-Garantien



Transportschicht vs. Vermittlungsschicht

Vermittlungsschicht:
Logische Kommunikation zwischen (End-)Systemen

Transportschicht:
Logische Kommunikation zwischen Prozessen

- Basiert auf Diensten der Vermittlungsschicht (und erweitert diese sogar!)

Analogie:

12 Kinder leben im Haus von Ann und senden Briefe an 12 Kinder im Haus von Bill

- Endsysteme = Häuser
- Prozesse = Kinder
- Anwendungsnachrichten = Briefe in Umschlägen
- Transportprotokoll = Ann und Bill, die an Kinder verteilen (multiplexen)
- Vermittlungsschichtprotokoll = Zustelldienst der Post

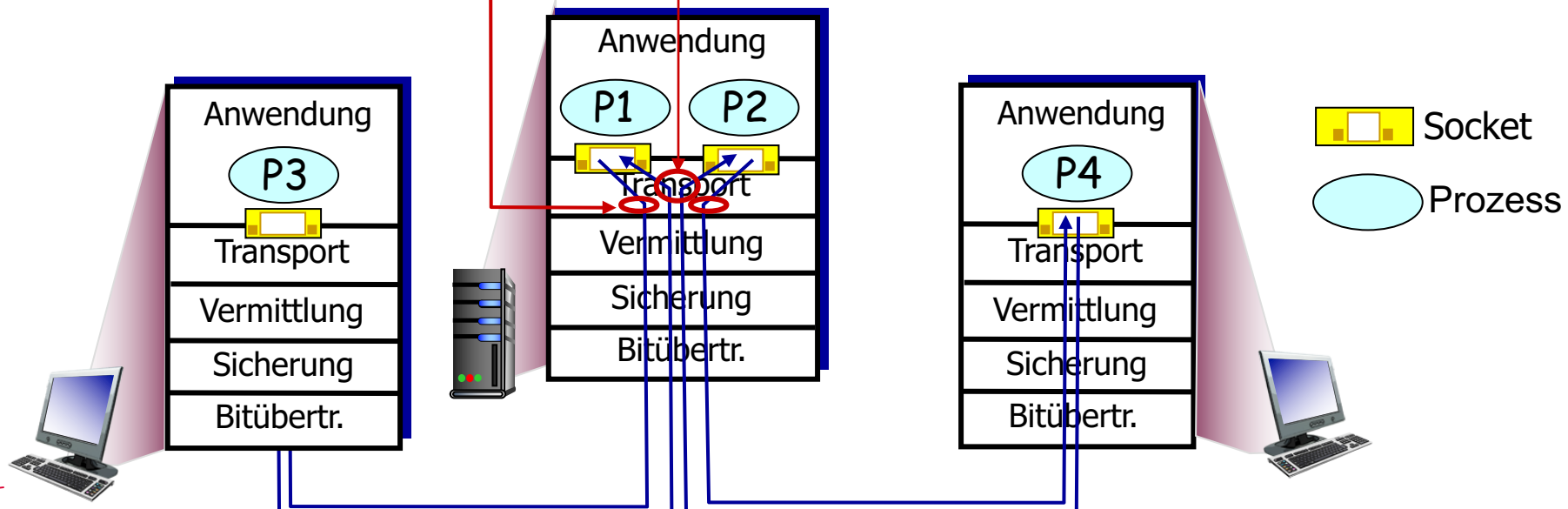
Multiplexing / Demultiplexing

Multiplexing beim Sender:

Verarbeitung der Daten von vielen Sockets, Hinzufügen von Transport-Headern (für Demultiplexing)

Demultiplexing beim Empfänger:

Verwendet Header-Informationen zur Auslieferung an das richtige Socket

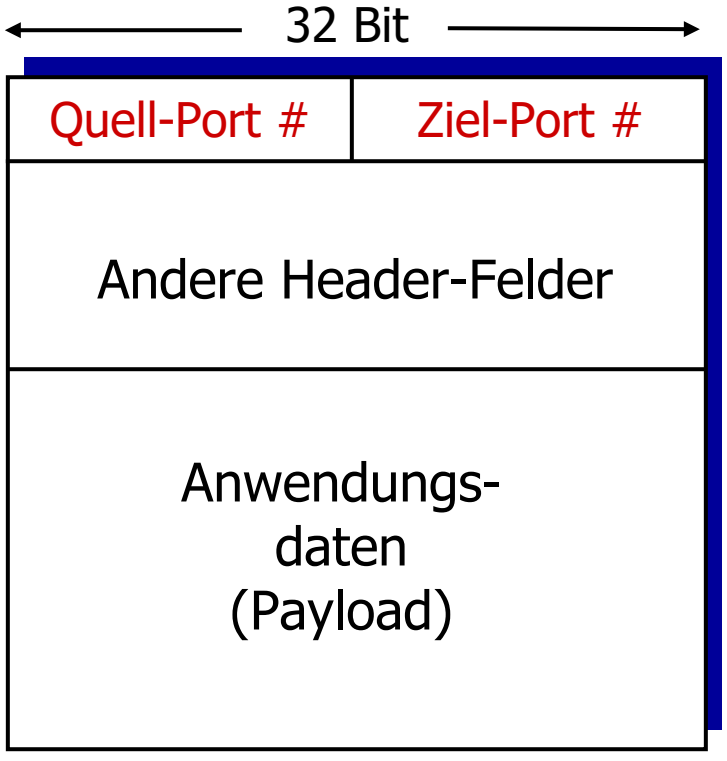


Wie Demultiplexing funktioniert

(End-)Systeme empfangen IP-Datagramme

- Jedes Datagramm hat Quell-(IP-)Adresse und Ziel-(IP-)Adresse
- Jedes Datagramm überträgt ein Transportschicht-Segment
- Jedes Segment hat Quell-Port-Nummer und Ziel-Port-Nummer

IP-Adresse und Port bestimmen das Ziel-Socket



TCP/UDP Segment Format

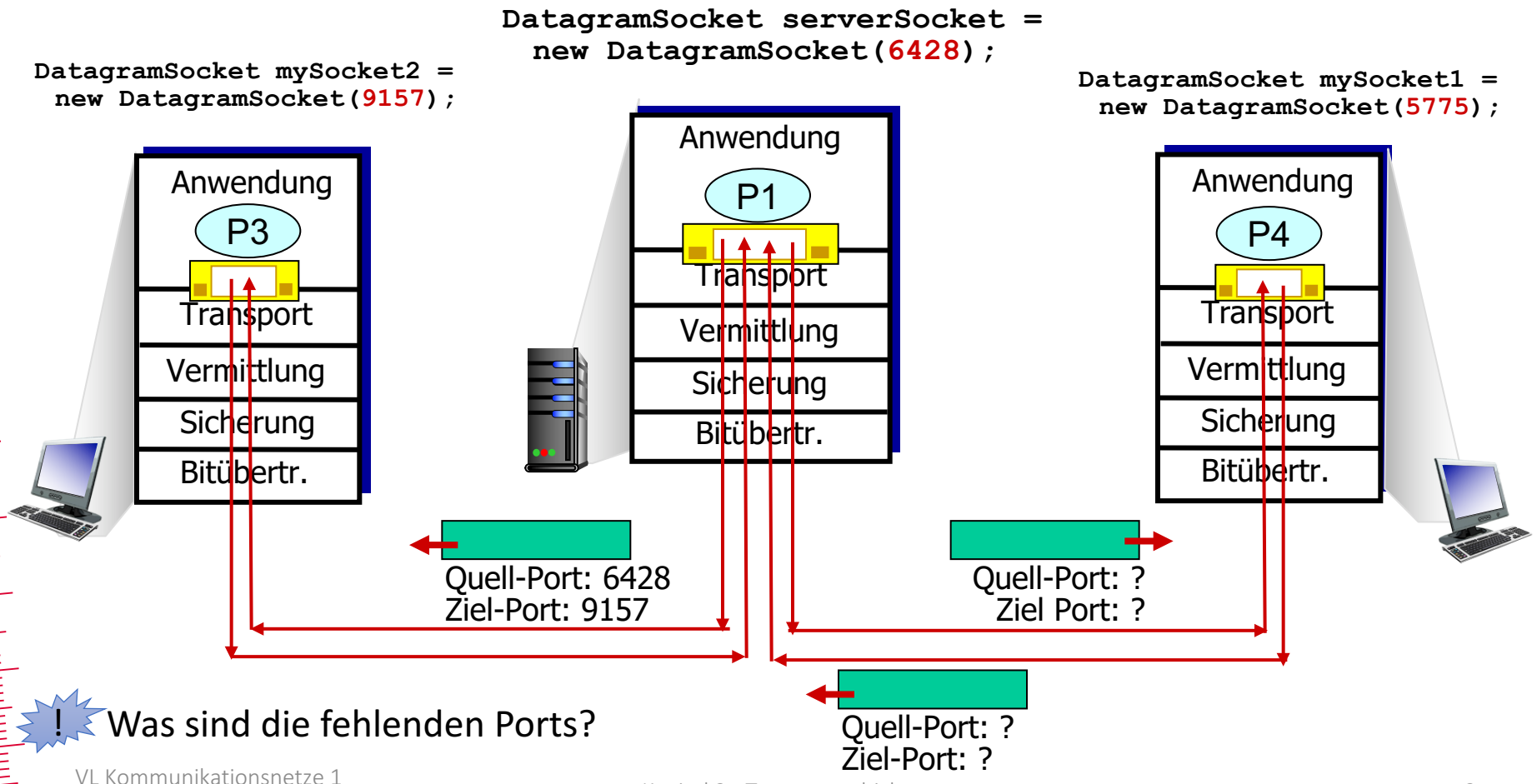
Verbindungsloses Demultiplexing (UDP)

Wenn ein Endsystem ein UDP-Segment empfängt

- ...prüft es die Ziel-Port-Nummer im Segment
- ... und leitet das Segment an das Socket mit diesem Port weiter

- D.h. (Ziel-IP-Adresse, Ziel-Port-Nummer) identifizieren das UDP-Socket eindeutig!
- IP-Datagramme mit gleicher Ziel-Port-Nummer, aber unterschiedlicher Quell-IP-Adresse werden an dasselbe Socket weitergeleitet!

Beispiel verbindungsloses Multiplexing



! Was sind die fehlenden Ports?

Verbindungsorientiertes Multiplexing (TCP)

- TCP-Socket wird durch 4-Tupel identifiziert
  - Quell-IP-Adresse
  - Quell-Port-Nummer
  - Ziel-IP-Adresse
  - Ziel-Port-Nummer
- Demultiplexing: Empfänger nutzt alle vier Werte zur Identifikation des richtigen Sockets

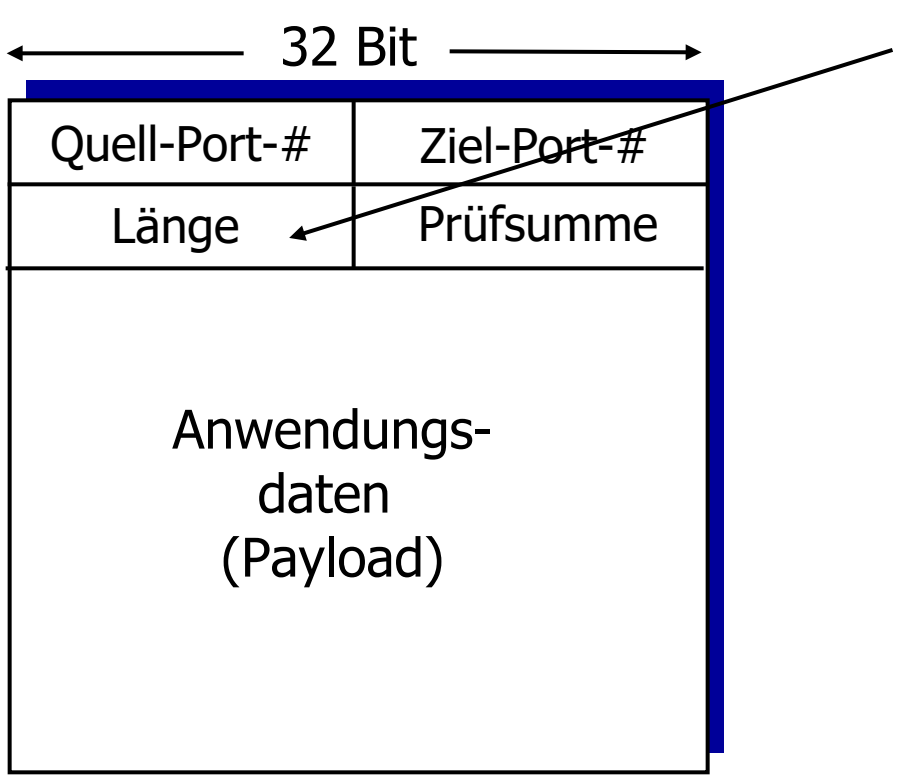
- Server-Systeme können viele gleichzeitige TCP-Sockets nutzen
  - Jedes Socket hat sein eigenes 4-Tupel
- Web-Server haben unterschiedliche Sockets für jeden verbundenen Client
  - Nicht-persistentes HTTP hat sogar unterschiedliche Sockets für jede HTTP-Anfrage eines Clients

UDP: User Datagram Protocol [RFC 768]

- Direkte Anwendung des Internet-Protokolls
- „Best effort“ Dienst, UDP-Segmente können...
  - ... verloren gehen
  - ... ohne Einhaltung der Reihenfolge an die Applikation gegeben werden
- Verbindungslos
  - Kein Verbindungsaufbau zwischen Sender und Empfänger
  - Jedes UDP Segment wird unabhängig von anderen Segmenten behandelt

- UDP Anwendungsfälle
  - Multimedia Streaming (Fehlertolerant, aber sensibel bezüglich Datenrate)
  - DNS
  - SNMP
- Zuverlässiger Transport über UDP
  - Zuverlässigkeit auf Anwendungsschicht
  - Anwendungsspezifische Fehlerbehandlung!

Aufbau eines UDP Segments



Länge des UDP Segments, inkl. Header in Bytes

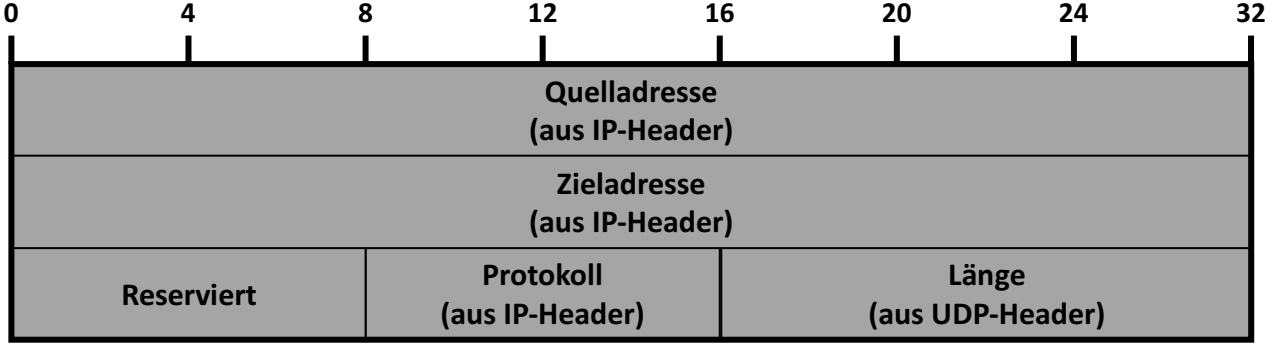
UDP Segment Format

Internet-Prüfsummen (vgl. auch [RFC 1071])

Ziel: Erkennung von Fehlern im Segment, z.B. „gekippte“ Bits

- Inhalt des Segments inkl. UDP-Header und Pseudo-IP-Header (s.u.) wird als Sequenz von 16-bit Worten interpretiert
- Prüfsummenfeld ist beim Sender zur Berechnung 0
- Prüfsumme wird als Einerkomplement der Summe aller Worte berechnet
- Sender überträgt die Prüfsumme im UDP Prüfsummenfeld

Aufbau Pseudo-Header:



Übung: UDP-Pseudo-Header

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Stellen Sie eine UDP-Pseudo-Header für das folgende UDP-Segment auf, das von

- Quell-IP-Adresse 192.168.178.25 an
- Ziel-IP-Adresse 224.0.0.1

gesendet wurde:

208	5	33	164
0	24	141	93
80	74	74	80
1	1	0	0
0	0	0	0
0	0	0	0

Die Protokoll-Nummer für UDP ist 17

Internet-Prüfsumme: Beispiel

Beispiel: Addieren von zwei 16-bit Worten

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Übertrag	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
Summe		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
Prüfsumme		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Empfänger

- Berechnet die Prüfsumme über UDP-Segment und Pseudo-IP-Header
  - Summe enthält nur 1-Bits -> Kein Fehler erkannt
  - Summe enthält 0-Bits -> Fehler

TCP: Transmission Control Protocol [RFCs 793,1122,1323, 2018, 2581]

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Punkt-zu-Punkt
- Ein Sender, ein Empfänger

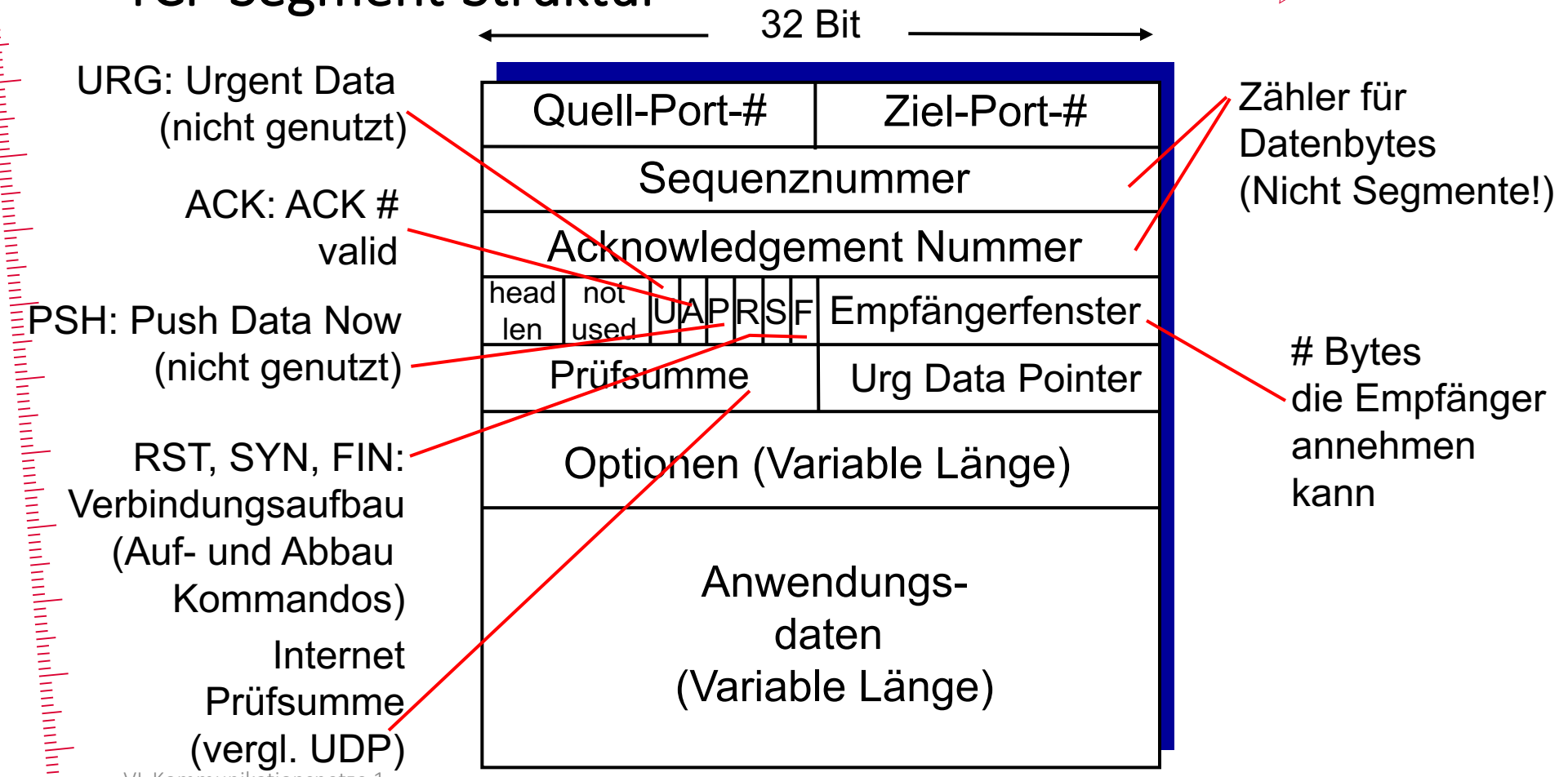
Verbindungsorientiert
- Verbindungsaufbau (Austausch von Kontrollnachrichten, Handshaking) erzeugt Zustand für Sender und Empfänger, bevor Daten übertragen werden können

Zuverlässig, Reihenfolge-erhaltend
- Segmente mit max. Maximum Segment Size (MSS) Bytes an Nutzdaten
- Sequenznummern
- Quittungen (Acknowledgements, ACKs)
- Timeouts und Übertragungswiederholungen

Mehrere Segmente gelichzeitig unterwegs
- Überlappende Übertragung (Pipelining)
- TCP Staukontrolle und TCP Flusskontrolle bestimmen Anzahl überlappend übertragener Segmente

Vollduplex-Übertragung
- Bi-direktionaler Datenfluss über dieselbe Verbindung

TCP Segment-Struktur



TCP Sequenznummer und ACKs

Sequenznummer

- „Byte-Strom-Nummer“ des ersten Daten-Bytes im Segment

Quittungen / Acknowledgements

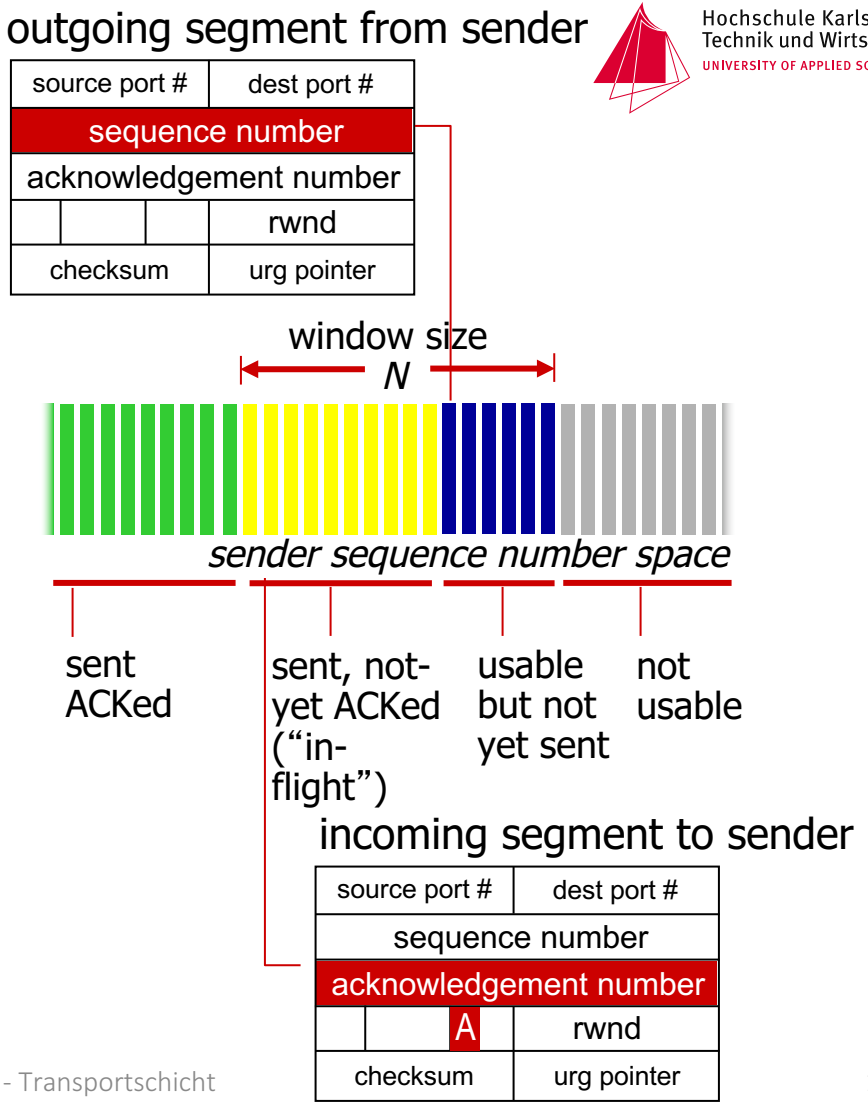
- Sequenznummer des nächsten erwarteten Bytes

Kumulative Acknowledgements

- Bei mehreren ausstehenden Segmenten wird das letzte bestätigt

TCP-Spezifikation sagt nichts zur Behandlung von Segmenten außerhalb der Reihenfolge aus

- Implementierungsspezifisch

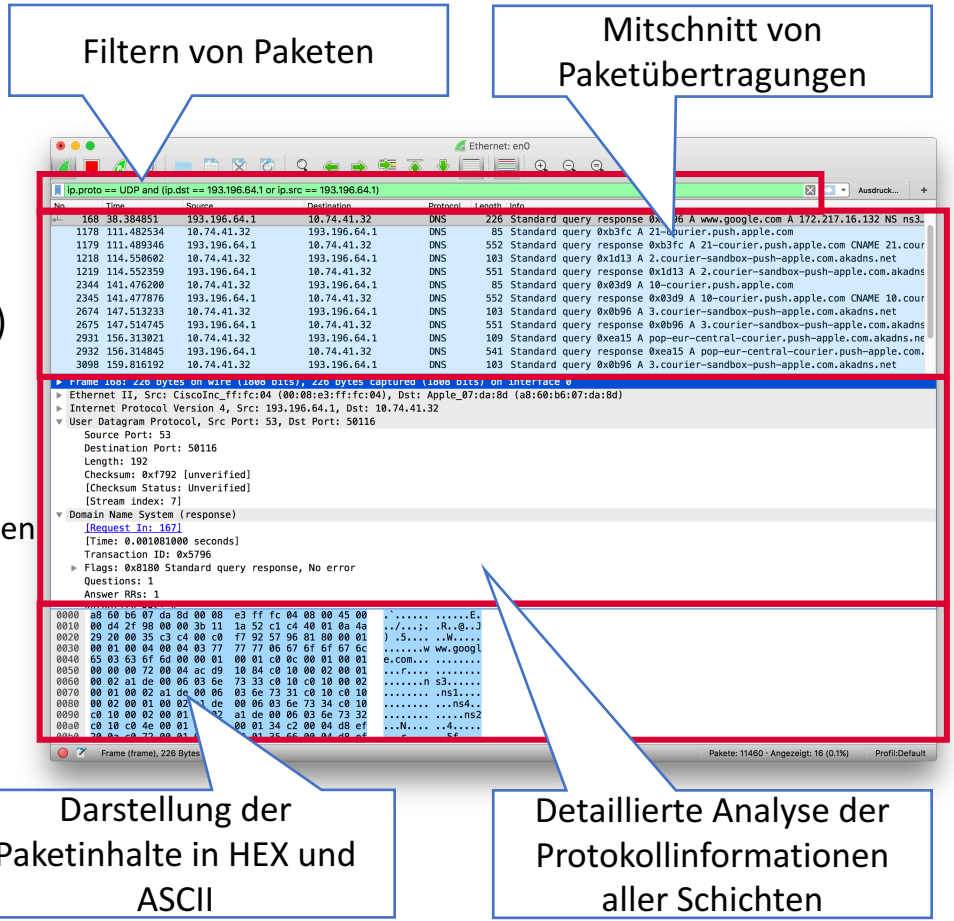


Einschub: Netzwerkanalyse mit Wireshark

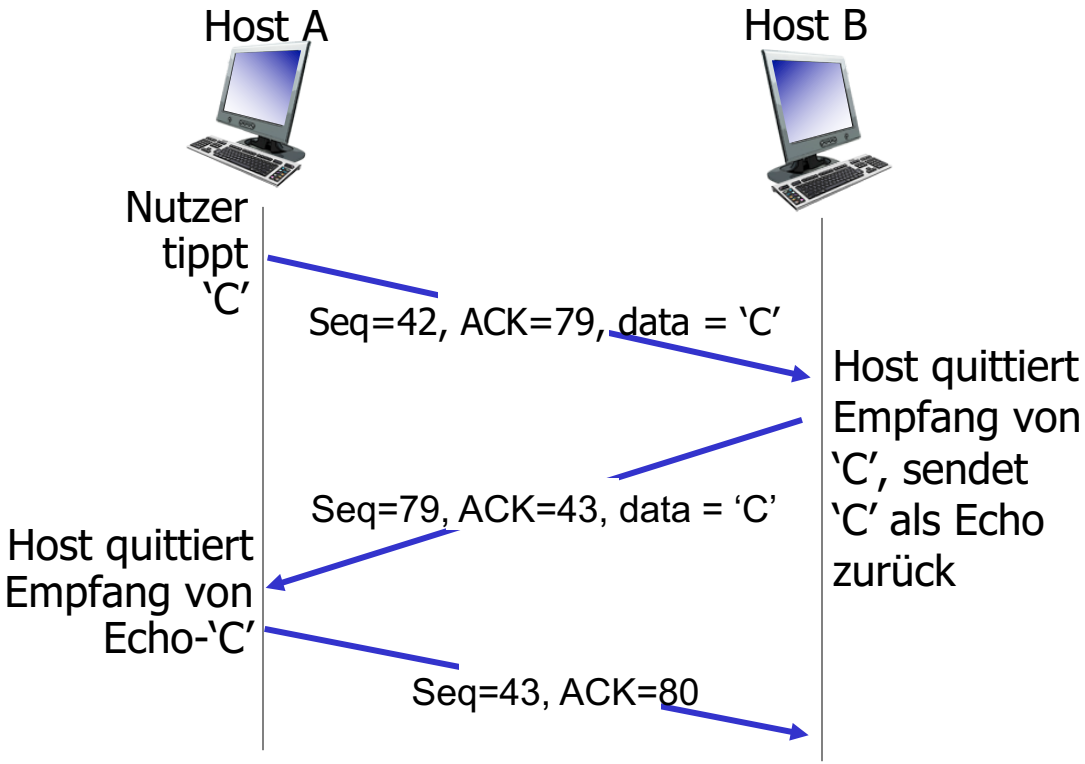
Das Werkzeug Wireshark ermöglicht Analyse von über das Netz übertragenen Paketen

- Möglichst vollständige Aufzeichnung
- Möglichst detaillierte Darstellung (inkl. Header und Informationen aller Schichten)
- Features
  - Live-Mitschnitt von Paket-Übertragungen, Speichern, Laden, Importieren von Mittschnitten
  - Darstellung detaillierter Protokollinformationen
  - Filtern von Paketen
  - Verschiedene Statistiken
  - … und vieles mehr

Siehe https://www.wireshark.org/



TCP Sequenznummer und ACKs



Einfaches Telnet Szenario

Übung: TCP-Sequenznummern

Setzen Sie den Verlauf des Beispiels für den Fall fort, in dem der Client „`ls -l\n`“ sendet, und der Server mit der selben Zeichenkette als Echo antwortet!

TCP Round Trip Time (RTT) und Timeouts

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Frage: Wie setzt man einen Timeout-Wert für fehlende Acknowledgements?

- Länger als RTT
  - ... aber RTT schwankt
- Zu kurz: Timeout zu früh, unnötige Übertragungs-wiederholung
- Zu lang: Langsame Reaktion auf verlorene Segmente

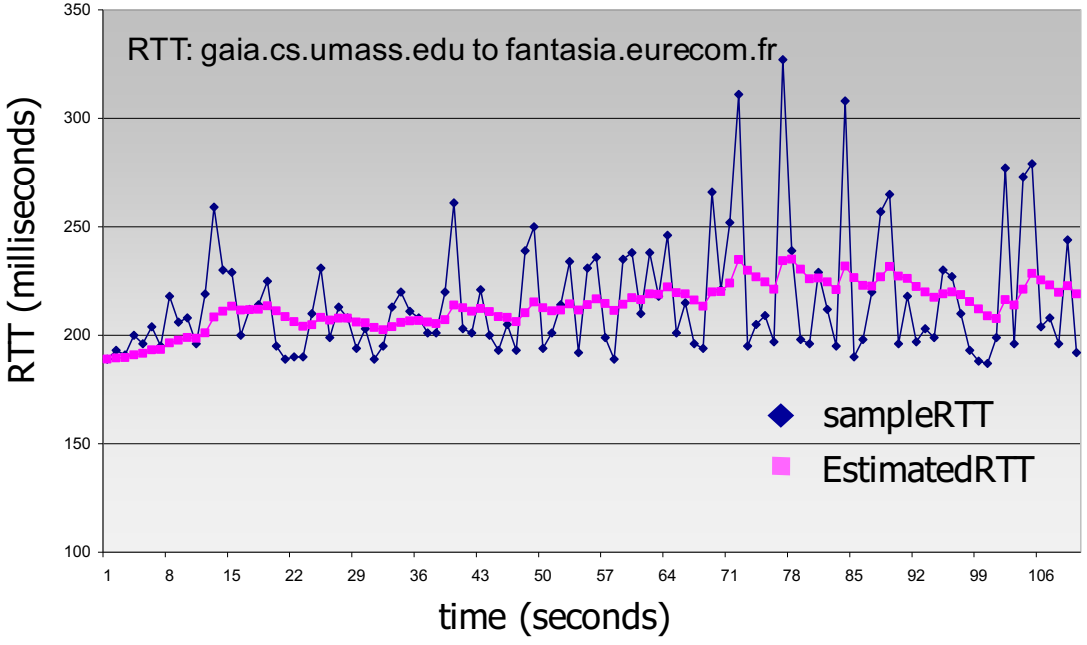
Frage: Wie schätz man die RTT?

- `SampleRTT`: Gemessene RTT zwischen Senden eines Segmentes und empfang eines ACK
  - Übertragungswiederholungen werden ignoriert
- `SampleRTT` schwankt, Schätzung muss „geglättet“ werden
  - `EstimatedRTT`: Mittelwert aus mehreren aktuellen Schätzungen, nicht nur aktuelle Messung

TCP Round Trip Time (RTT) und Timeouts

$$EstimatedRTT_{Neu} = (1- \alpha)*EstimatedRTT_{Alt} + \alpha*SampleRTT$$

- Exponentiell geglätteter Mittelwert (exponential weighted moving average)
- Einfluss von „alten“ Messungen verschwindet schnell
- Typischer Wert: $\alpha = 0.125$



TCP Round Trip Time (RTT) und Timeouts

- `TimeoutInterval`: `EstimadtedRTT` plus „Sicherheitsspanne“
  - Größere Varianz von `EstimatedRTT` → größere Sicherheitsspanne
- Ableitung von `SampleRTT`-Abweichung **`DevRTT`** aus `EstimatedRTT`

$$DevRTT_{Neu} = (1-\beta)*DevRTT_{Alt} + \beta*|SampleRTT-EstimatedRTT_{Alt}|$$

(typischerweise $\beta = 0.25$)

$$TimeoutInterval = EstimatedRTT + 4*DevRTT$$

Geschätzte RTT

“Sicherheitsspanne”

Übung: Bestimmung **TimeoutInterval**

Was ergibt sich für $\mathbf{EstimatedRTT_{Neu}}$, $\mathbf{DevRTT_{Neu}}$ und **TimeoutInterval**, wenn

- $\mathbf{EstimatedRTT_{Alt}}$ = 21,5ms,
- $\mathbf{DevRTT_{Alt}}$ = 3,25ms und
- **SampleRTT** = 31ms ist?

TCP: Zuverlässige Übertragung

- TCP implementiert eine zuverlässige Übertragung über den unzuverlässigen Service von IP
  - Überlappende (Pipelined) Übertragung von Segmenten
  - Kumulative ACKs
  - Nur ein einziger Timer für Übertragungswiederholungen
- Übertragungswiederholungen ausgelöst von
  - Timeout-Ereignissen
  - Wiederholten Acknowledgements

Wir betrachten zunächst einen vereinfachten TCP-Sender

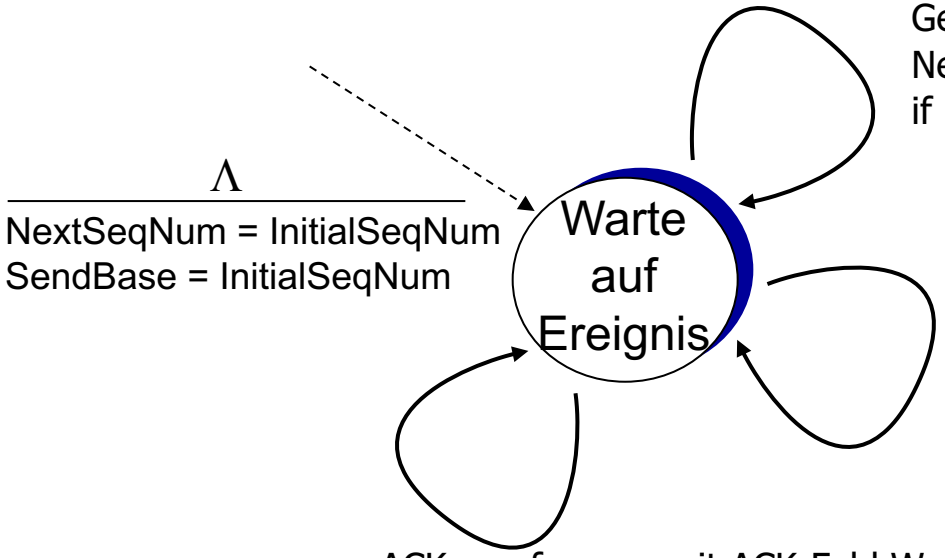
- Ignorieren von wiederholten ACKs
- Ignorieren von Fluss- und Staukontrolle

Versendern von TCP-Segmenten

Event beim Sender	TCP Sender Aktion
Daten von Anwendung empfangen	• Erzeuge und sende ein Segment mit Sequenz-nummer (Nummer des ersten Daten-Bytes im Byte-Strom) • Starte Timer, wenn dieser nicht bereits läuft • Timer gilt für ältestes nicht quittiertes Segment, Ablaufzeit: `TimeOutInterval`
Timeout	• Übertrage Segment, das Timeout erzeugte, erneut • Starte Timer neu mit verdoppelter Ablaufzeit

ACK empfangen	• Wenn ACK für bisher nicht quittiertes Segment
	  • Aktualisiere letztes bestätigtes Segment
	  • Starte Timer neu wenn noch nichtquittierte Segmente, Ablaufzeit: `TimeOutInterval`

Vereinfachter TCP Sender



Daten von Anwendung empfangen

Erzeuge Segment, SeqNr.: NextSeqNum
Gebe Segment an IP (d.h., "sende")
NextSeqNum = NextSeqNum + length(data)
if (Timer läuft gerade nicht)
    Starte Timer

Timeout

Wiederhole nichtquittiertes Segment
mit kleinster SeqNr.
Starte Timer mit doppelter Ablaufzeit

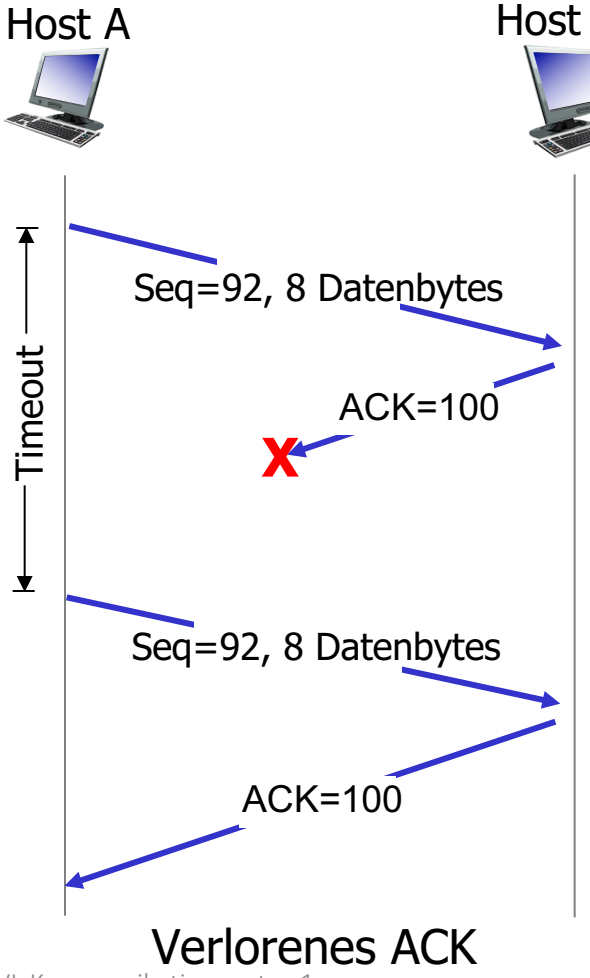
ACK empfangen, mit ACK-Feld Wert y

```
if (y > SendBase) {
    SendBase = y
    /* SendBase–1: letztes kumulativ quittiertes Byte */
    if (es gibt aktuell nichtquittierte Segmente)
        Starte Timer
    else Stoppe Timer }
```

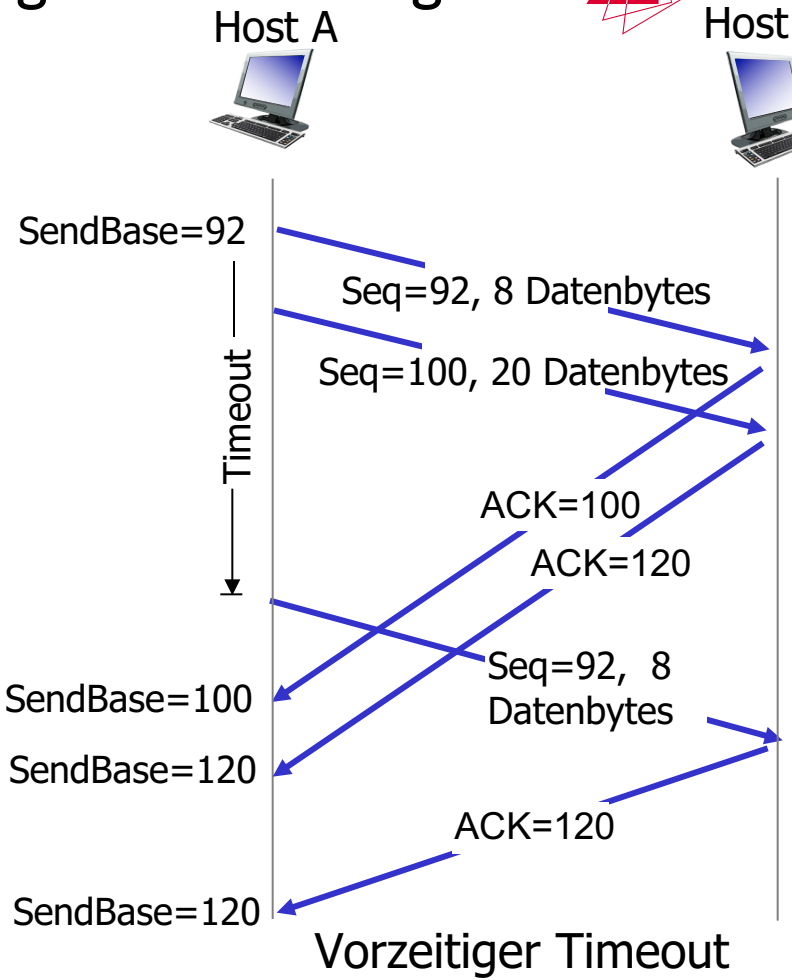
TCP ACK Erzeugung [RFC 1122, RFC 2581]

Event beim Empfänger	TCP Empfänger Aktion
Ankunft eines Segments in Reihenfolge mit erwarteter Sequenznummer.	Verzögertes ACK. Warte bis zu 500ms auf nächstes Segment. Wenn kein Segment empfangen, sende ACK.
Ankunft eines Segments in Reihenfolge mit erwarteter Sequenznummer, ein ACK für anderes Segment ausstehend.	Sofortiges Senden von kumulativem ACK, dass beide Segmente quittiert.
Ankunft eines Segments außerhalb der Reihenfolge mit Sequenznummer höher als erwartet. Lücke erkannt.	Sofortiges Senden eines wiederholten (duplicate) ACKs für Sequenznummer des nächsten erwarteten Bytes.
Ankunft eines Segments, dass eine Lücke schließt.	Sofortiges Senden eines ACK, falls das Segment am unteren Ende der Lücke beginnt.

TCP: Szenarien für Übertragungswiederholung

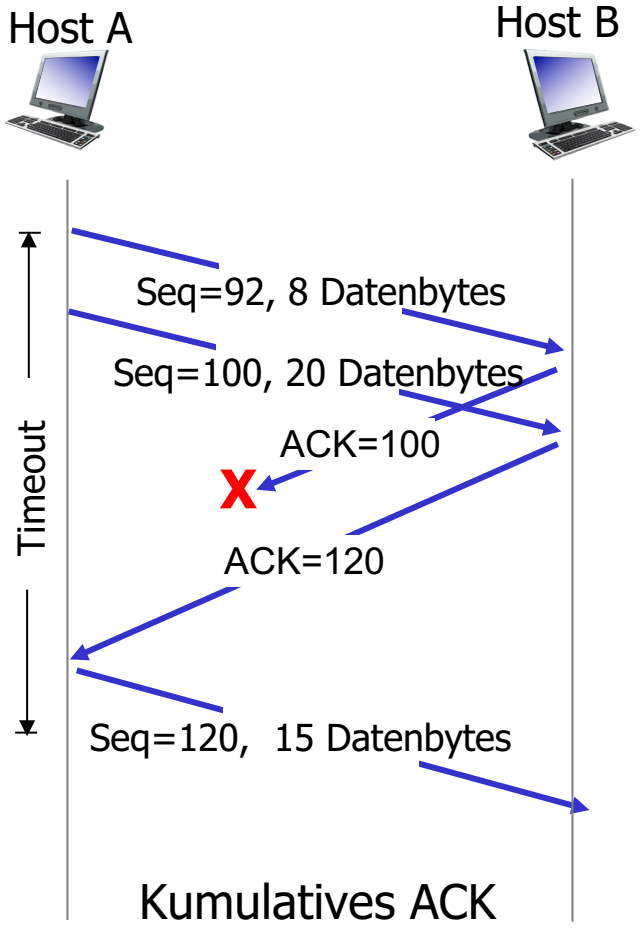


Verlorenes ACK



Vorzeitiger Timeout

TCP: Szenarien für Übertragungswiederholung



TCP Fast Retransmit [RFC 2581]

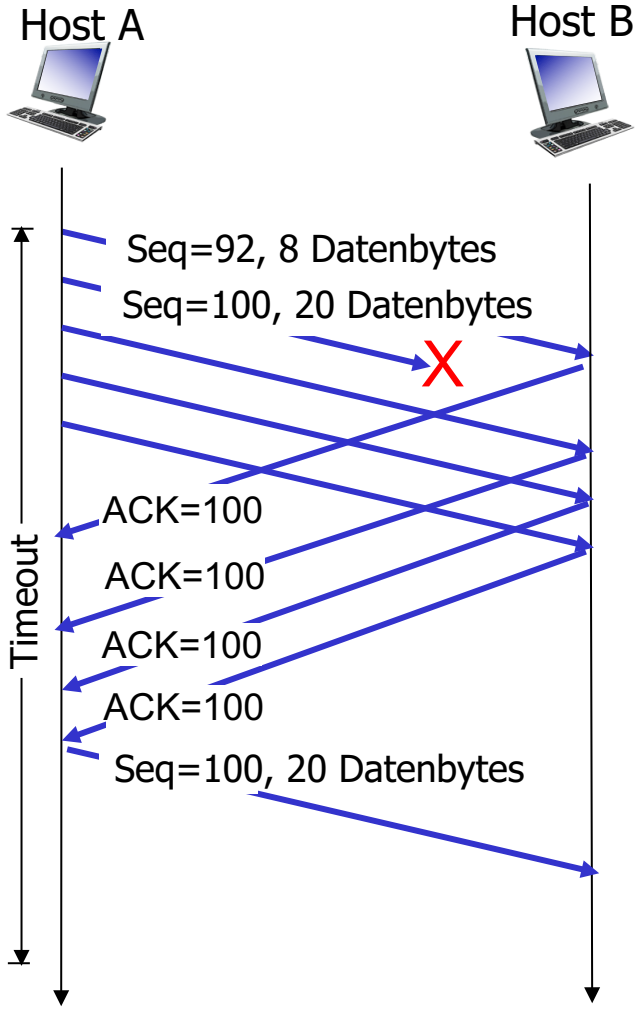
- Timeout ist häufig relativ lang
  - Lange Verzögerung bevor verlorenes Segment erneut übertragen wird
- Erkennung über wiederholte (duplicate) ACKs
  - Sender sendet häufig Segmente direkt hintereinander
  - Wenn ein Segment verloren geht, werden wahrscheinlich viele wiederholte ACKs (ausgelöst durch nächste Segmente) eintreffen

TCP Fast Retransmit

- Wenn Sender drei wiederholte ACKs für bereits bestätigtes Segment erhält („Triple duplicate ACKs“) wird nichtquittiertes Segment mit kleinster Sequenznummer erneut übertragen
  - Mit hoher Wahrscheinlichkeit ist diese Segment verloren gegangen, daher nicht auf Timeout warten

TCP Fast Retransmit

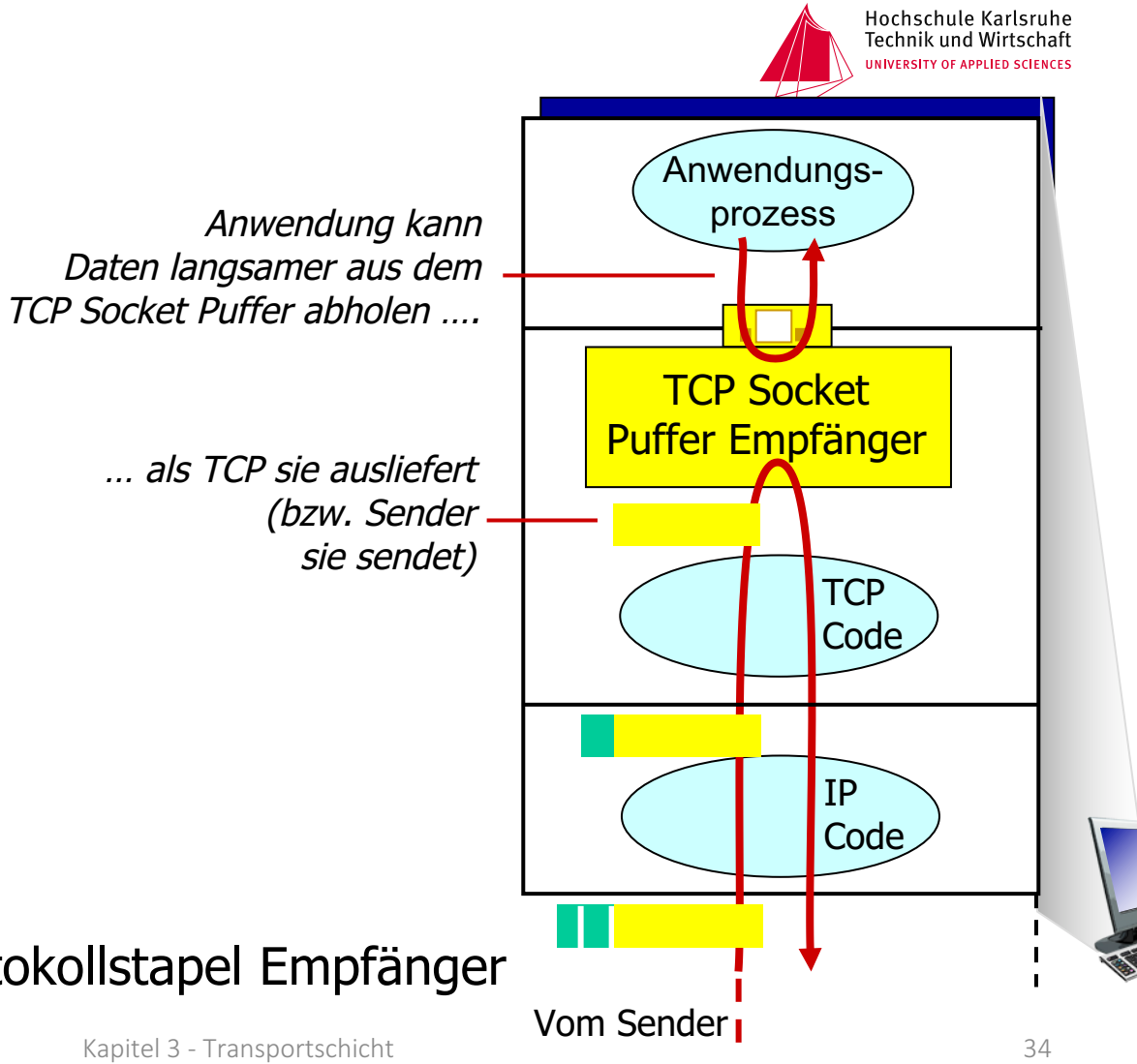
Fast Retransmit nachdem Sender dreimal wiederholtes ACK empfangen hat



TCP Flusskontrolle

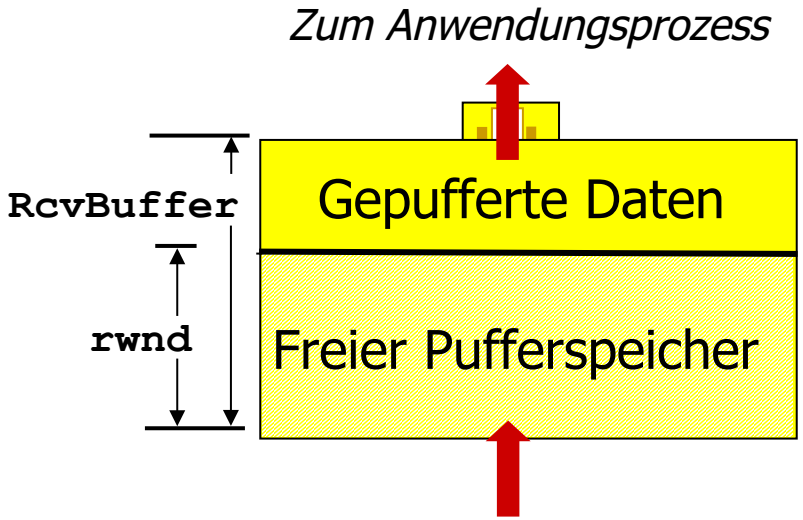
Flusskontrolle

- Empfänger kontrolliert Sender, damit dieser nicht den Empfänger-Puffer durch zu schnelles Senden „überschwemmen“ kann



TCP Flusskontrolle

- Empfänger bewirbt („advertise“) freien Pufferplatz über **`rwnd`** im TCP Header von Sender-Empfänger-Segmenten
  - **`RcvBuffer`** Größe kann über Socket-Option gesetzt werden (Standard ist 4096 Byte)
  - Viele Betriebssysteme passen **`RcvBuffer`** automatisch an
- Sender beschränkt die Menge von nichtquittierten Daten („in-flight“) auf **`rwnd`** des Empfängers
  - **`LastByteSend`** – **`LastByteAcked`** <= **`rwnd`**
- Bei Null-Fenster werden trotzdem Segmente mit eine Daten-Byte gesendet, um ACKs mit neuer Fenstergröße auszulösen
- TCP Window Scale Option: Bei Verbindungsaufbau kann ausgehandelt werden, die Werte von **`rwnd`** mit einer Konstanten von bis zu 2^{14} zu multiplizieren

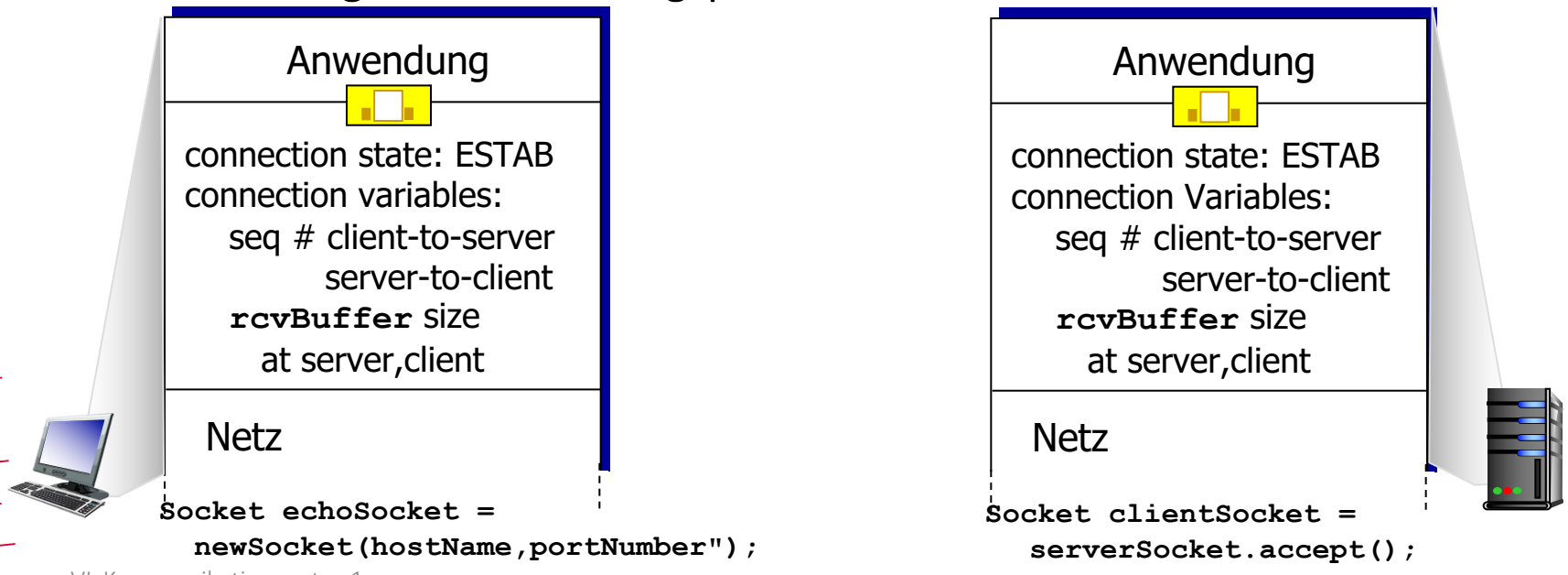


Puffer auf Empfängerseite

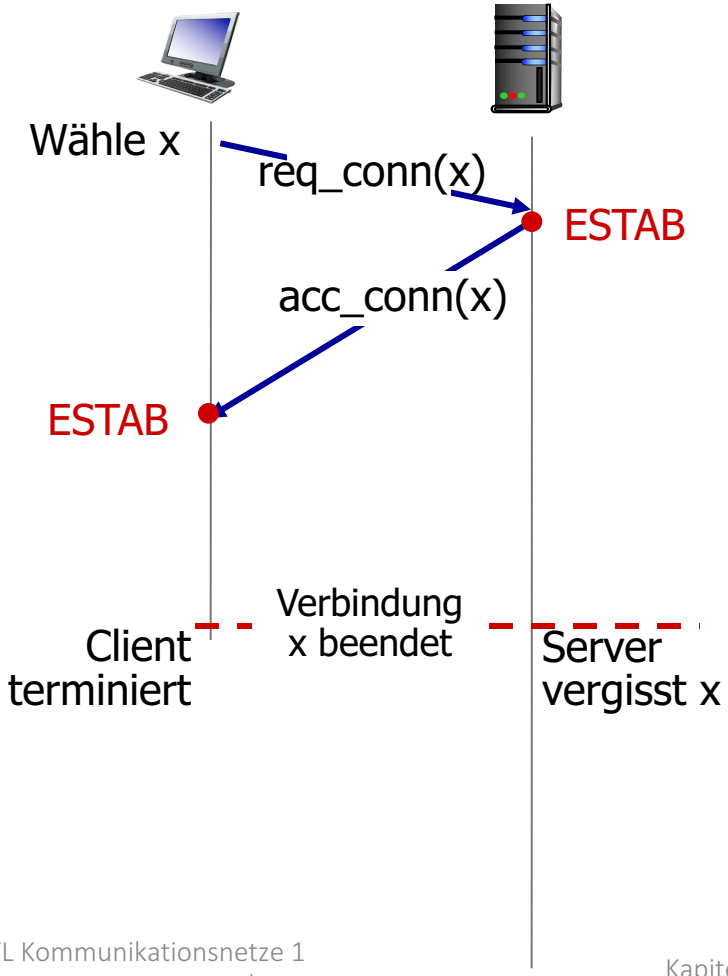
Verbindungsmanagement

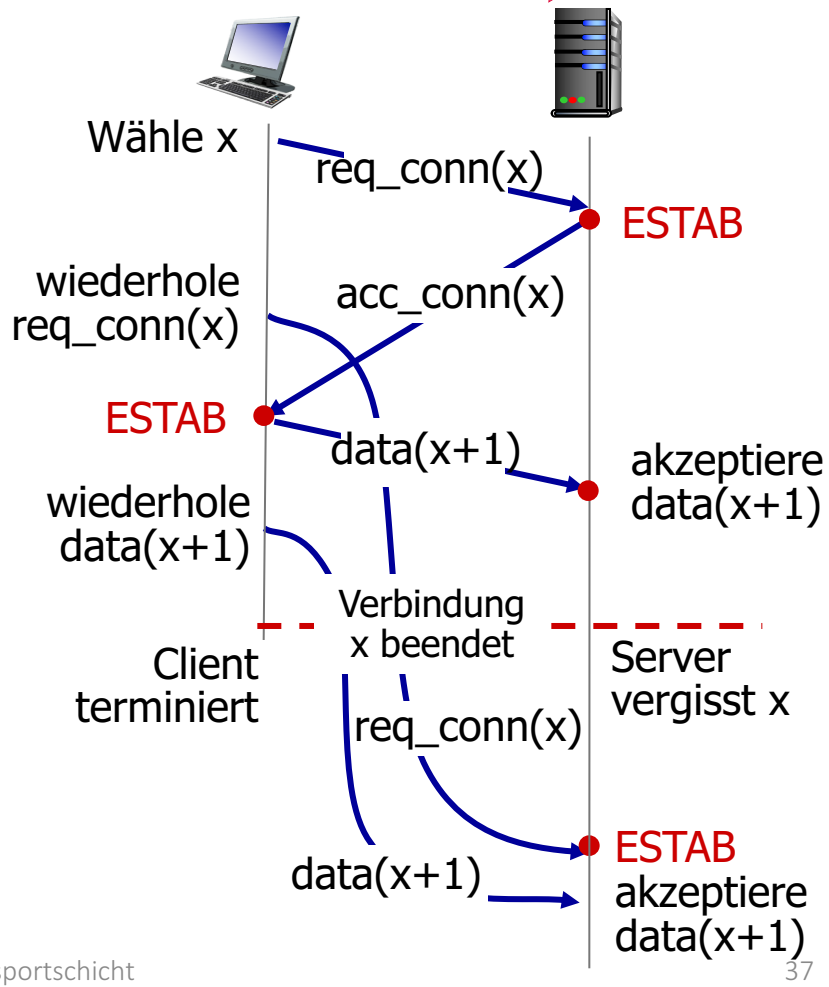
Bevor Daten ausgetauscht werden, findet ein Verbindungsaufbau („Handshake“) zwischen Sender und Empfänger statt:

- Beidseitige Zustimmung zum Verbindungsaufbau
- Aushandlung von Verbindungsparametern



2-Wege-Handshake und mögliche Fehler





TCP 3-Wege-Handshake

Client Zustand

CLOSED

Wähle initiale SeqNr., x
Sende TCP SYN Nachricht

SYN_SENT

SYNbit=1, Seq=x

Server Zustand

LISTEN

Wähle initiale SeqNr., y
Sende TCP SYNACK
msg, die SYN quittiert

SYN_RCVD

SYNbit=1, Seq=y
ACKbit=1; ACKnum=x+1

ESTAB

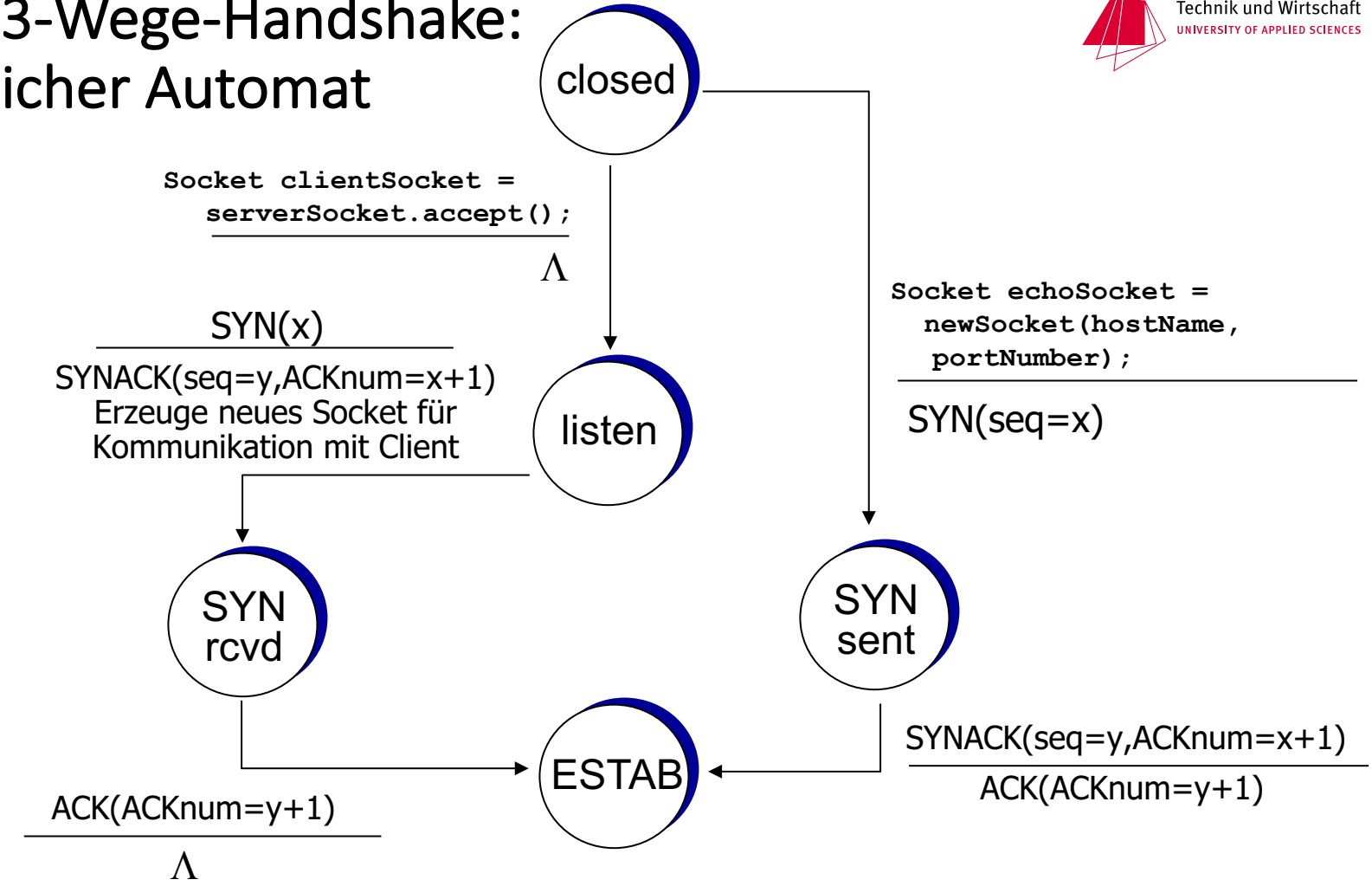
Empfangenes SYNACK(x)
zeigt Server-Verfügbarkeit;
Sende ACK für SYNACK;
Dieses Segment darf Client-
zu-Server-Daten enthalten

ACKbit=1, ACKnum=y+1

Empfangenes ACK(y)
zeigt Client-Verfügbarkeit

ESTAB

TCP 3-Wege-Handshake: Endlicher Automat



TCP Verbindungsabbau

Client oder Server schließen ihre Seite der Verbindung

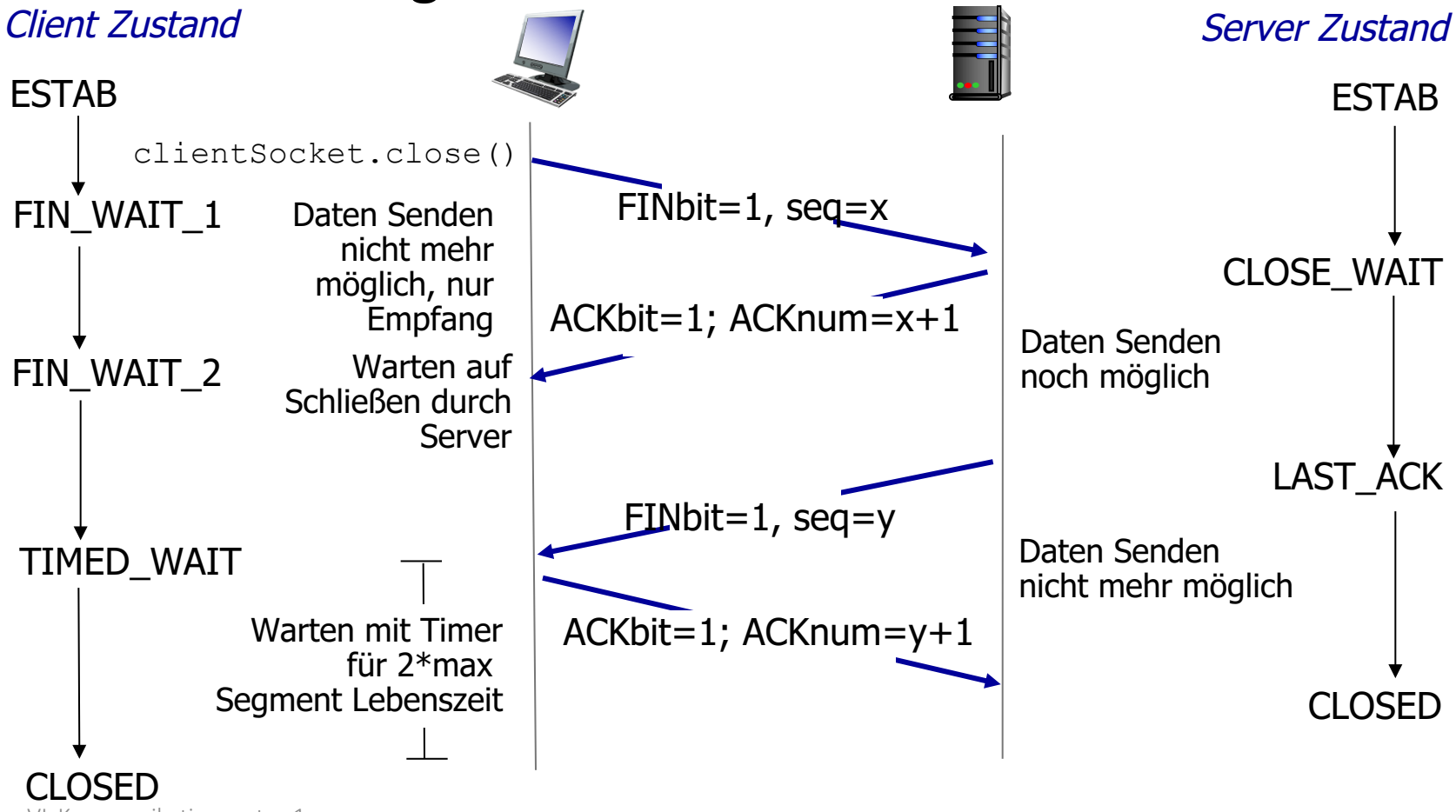
- TCP-Segment mit FIN-Bit = 1

FIN wird mit ACK beantwortet

- Ggf. kann ACK mit eigenem FIN beantwortet werden

Gleichzeitiger Austausch von FIN-Nachrichten möglich

TCP Verbindungsabbau



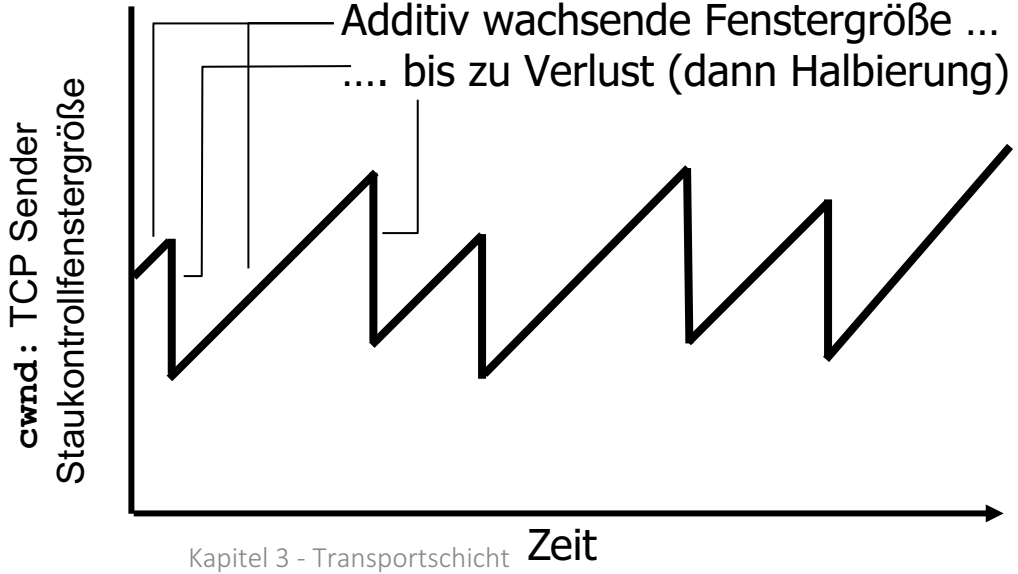
TCP Staukontrolle (auch: Überlastkontrolle)

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

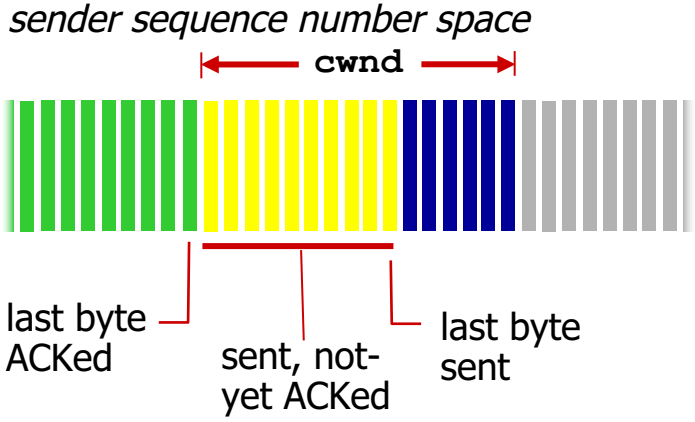
Überlastvermeidung (Congestion Avoidance, CA):

- Vermeidet Überlastung des Netzes!
- Verwaltung eines Staukontrollfensters **`cwnd`**
- Additive Vergrößerung (Additive Increase, AI)
  - **`cwnd`** wird pro **`RTT`** um **`MSS`** erhöht bis ein Paketverlust erkannt wird
- Multiplikative Verkleinerung (Multiplicative Decrease, MD)
  - **`cwnd`** wird nach Paketverlust halbiert bis minimal 1 **`MSS`**

AIMD „Sägezahn"-Verhalten: Verfügbare Bandbreite testen



Details zur Staukontrolle



- Sender begrenzt die Übertragung:

$$\texttt{LastByteSent} - \texttt{LastByteAcked} \leq \texttt{cwnd}$$

- **cwnd** hängt dynamisch von der aktuellen Netzlast ab

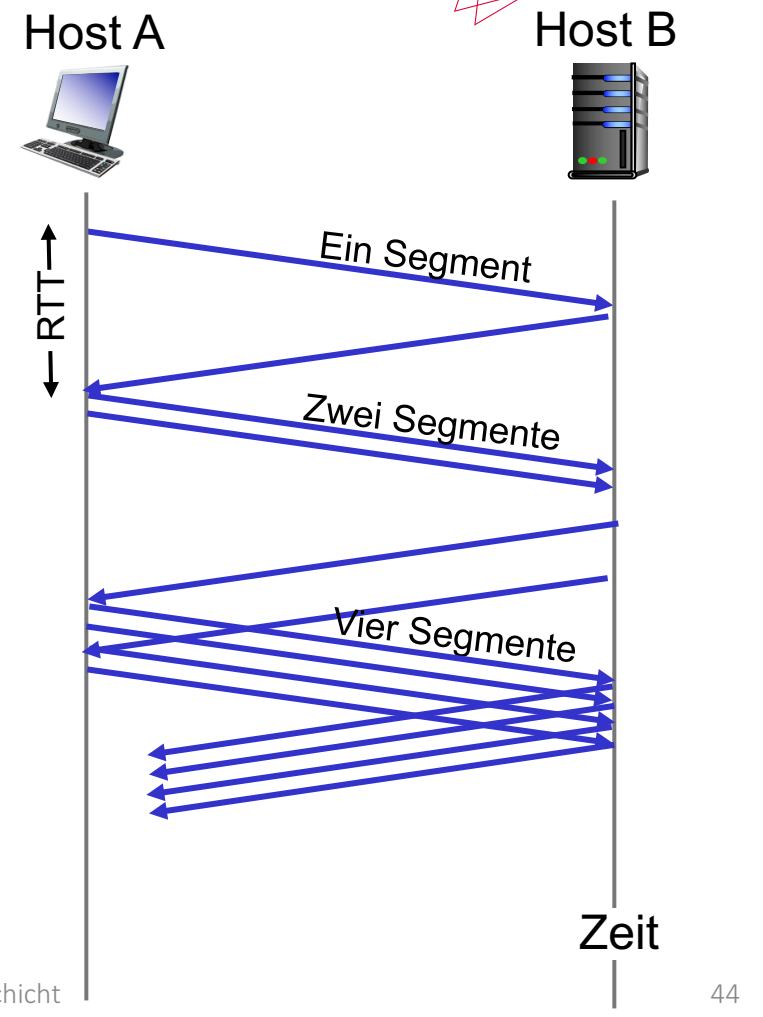
TCP Übertragungsrate:

- In erster Näherung: TCP sendet **cwnd** Bytes, wartet **RTT** auf Quittungen, sendet dann mehr Bytes

$$\text{Rate} \approx \frac{\texttt{cwnd}}{\texttt{RTT}} \quad \text{Byte/s}$$

TCP Slow Start

- Senderate nach Verbindungsaufbau exponentiell steigern bis erster Paketverlust auftritt
  - Initiales **`cwnd`** = 1 **`MSS`**
  - Verdopplung von **`cwnd`** jede **`RTT`**
  - Umgesetzt durch Erhöhung von **`cwnd`** bei jedem ACK
- Senderate am Anfang niedrig, aber wächst exponentiell



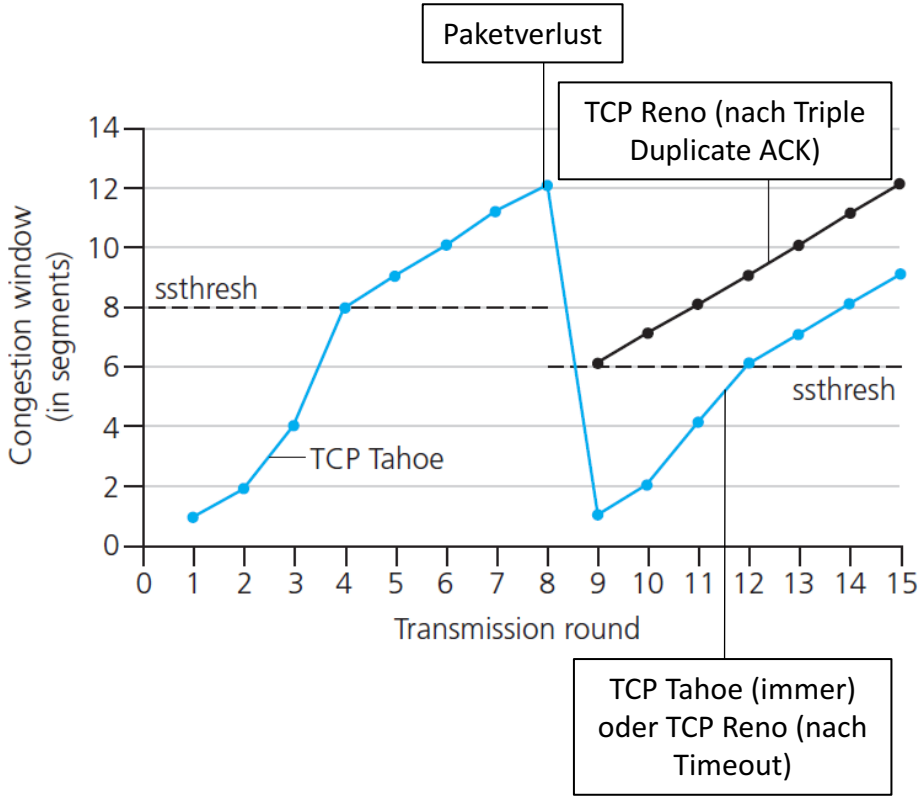
TCP Verlusterkennung und -behandlung

Bemerken von Paketverlust durch Timeout

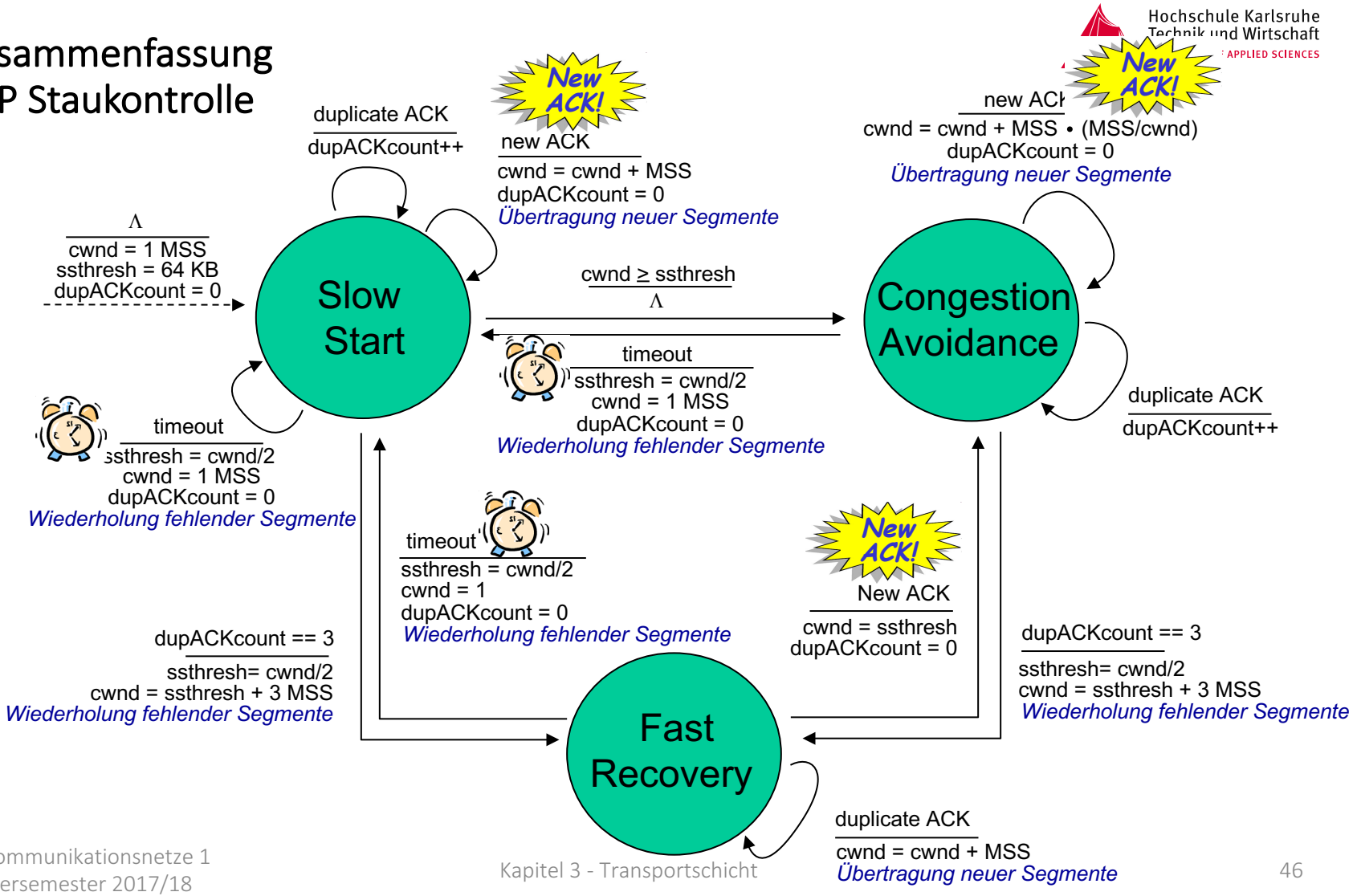
- **`ssthresh`** = **`cwnd`** / 2 (s.u.)
- **`cwnd`** wird auf 1 **`MMS`** gesetzt
- **`cwnd`** wächst danach exponentiell bis **`ssthresh`** (Slow Start), dann linear (Congestion Avoidance)

Bemerken von Paketverlust durch 3 wiederholte ACKs

- TCP Reno: Netz nicht überlastet, da ACKs nur als Reaktion auf ankommende Datenpakete gesendet; **`cwnd`** wird halbiert, dann lineares Wachstum (Congestion Avoidance)
- TCP Tahoe: **`cwnd`** immer auf 1 **`MMS`** und setzen Slow Start wie bei Paketverlust



Zusammenfassung TCP Staukontrolle



Übung: TCP Slow Start und Congestion Avoidance

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Eine TCP-Verbindung mit TCP Reno wird mit den Parametern

- **`cwnd`** = 1 und
- **`ssthresh`** = 16

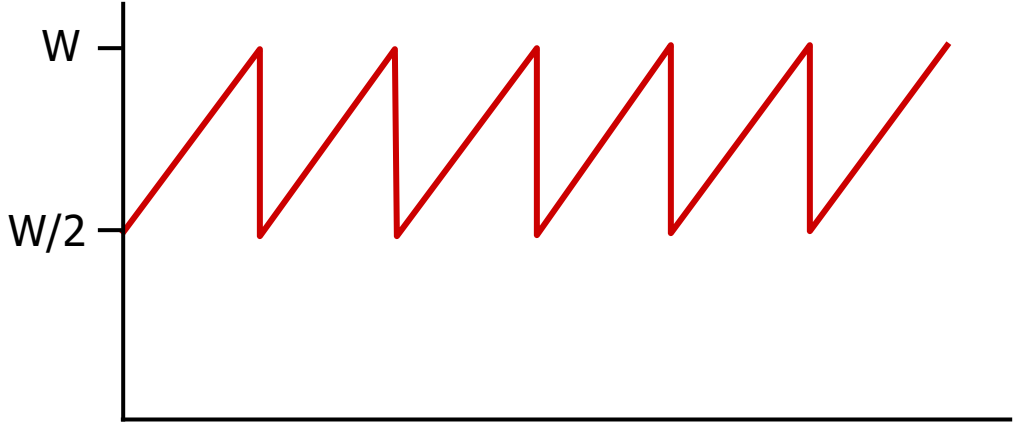
gestartet. Wie ist der Verlauf von **`cwnd`** (vgl. Folie 46) wenn in Übertragungsrunde 7 ein Paketverlust

1. durch einen Timeout
2. durch ein Triple Duplicate Acknowledgement

Bemerkt wird. Wie wird jeweils **`ssthresh`** nach dem Paketverlust gesetzt?

TCP Durchsatz

- Durchschnittlicher Durchsatz von TCP als Funktion der Fenstergröße und RTT?
- W: Fenstergröße in Bytes bei Paketverlust
  - Durchschnittliche Fenstergröße ist ¾ W
  - Durchschnittliche Durchsatz ist ¾ W pro RTT



Durchschn. TCP Durchsatz =

$$\frac{3}{4}\,\frac{W}{RTT}$$

Zukunft von TCP:
Hohe Datenraten bei großer RTT

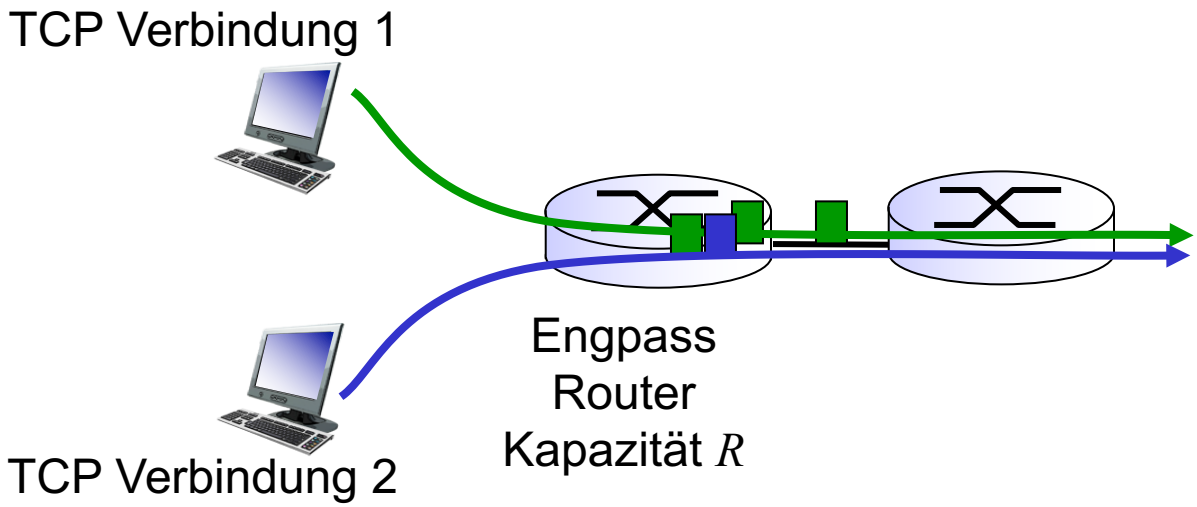
- Beispiel: 1500 Byte Segmente, 100ms RTT, angestrebter Durchsatz 10 Gb/S
  - Benötigt 83.333 parallel übertragene Segmente [RFC 3649]
  - Durchsatz als Funktion der Verlustwahrscheinlichkeit L

$$TCP\ throughput = \frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

- Wie hoch darf die Verlustrate sein, um einen Durchsatz von 10 Gb/s zu erreichen?
- Neue TCP-Versionen für Hochgeschwindigkeitsnetze benötigt!

TCP Fairness

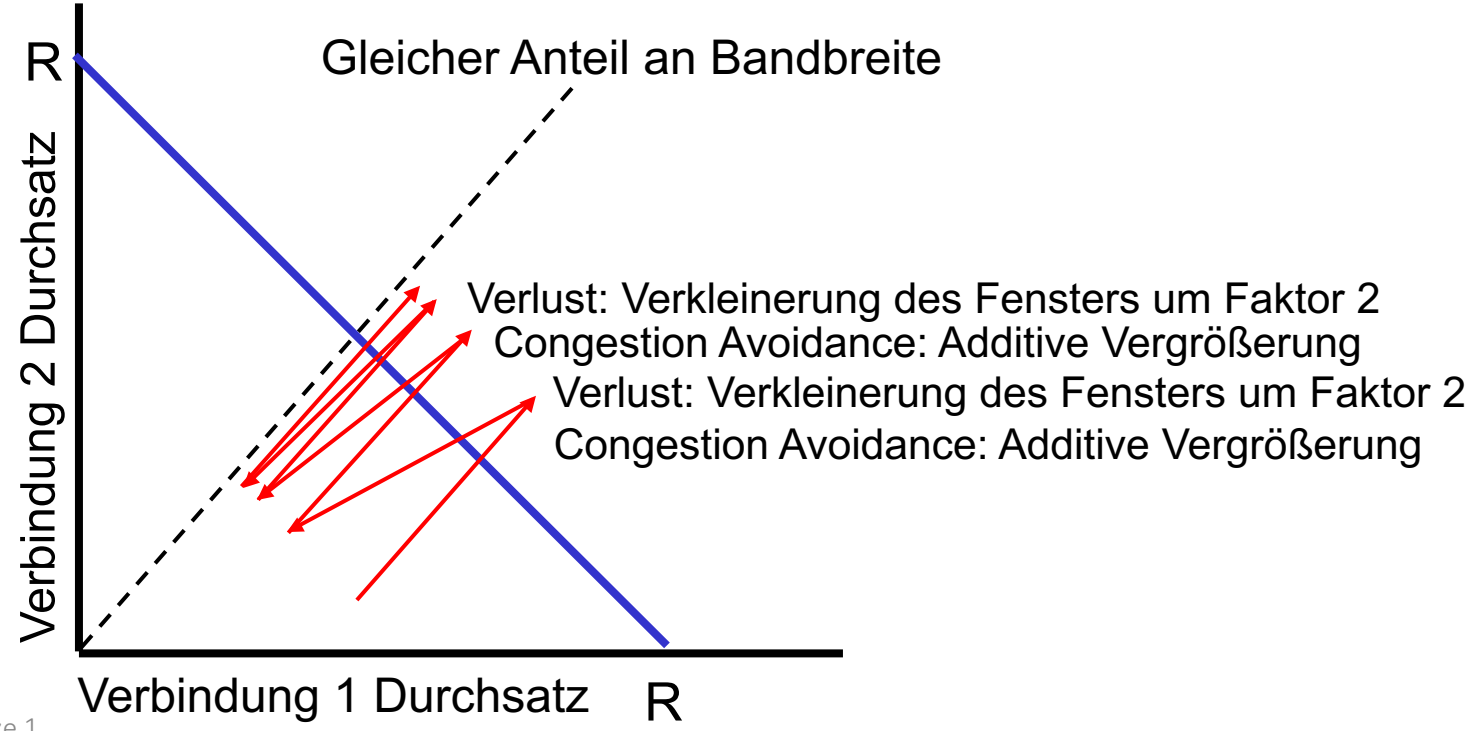
Ziel: Wenn K TCP Verbindungen den gleichen Engpass mit Bandbreite R teilen, sollte jeder einen Durchsatz von R/K erreichen



Warum ist TCP fair?

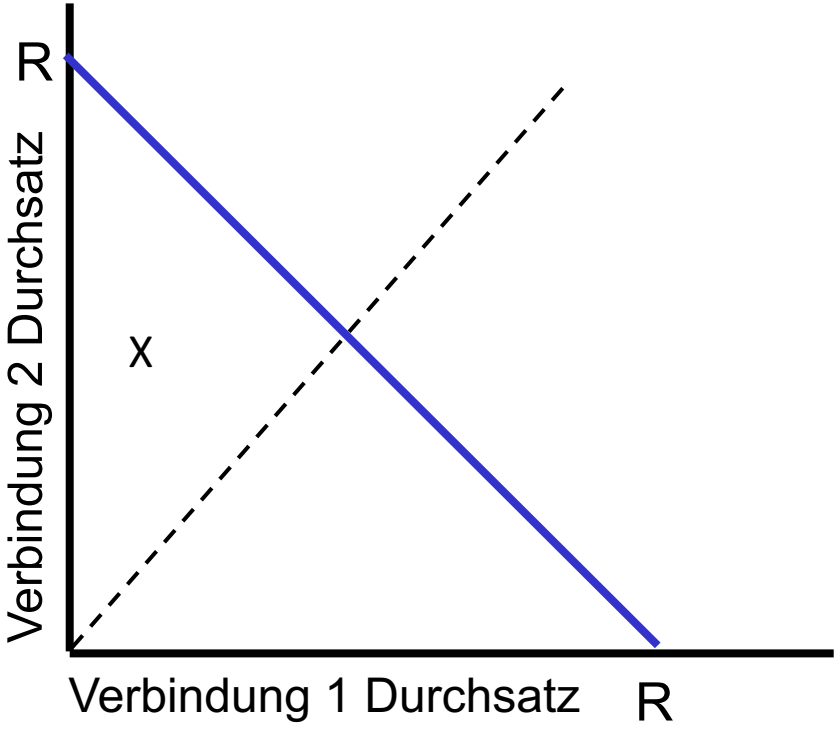
Zwei konkurrierende TCP-Verbindungen:

- Additive Vergrößerung ergibt Steigung von 1 wenn Durchsatz steigt
- Multiplikative Verringerung verringert Durchsatz proportional



Übung: TCP Fairness

Zeichnen Sie den Verlauf des Durchsatzes ausgehend von dem X ein!



Mehr zur TCP Fairness

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Fairness und UDP

- Multimedia-Anwendungen benutzen häufig kein TCP
  - Rate wird nicht durch Staukontrolle reduziert
- Stattdessen UDP
  - Audio/Video-Daten mit konstanter Rate, Paketverluste werden toleriert

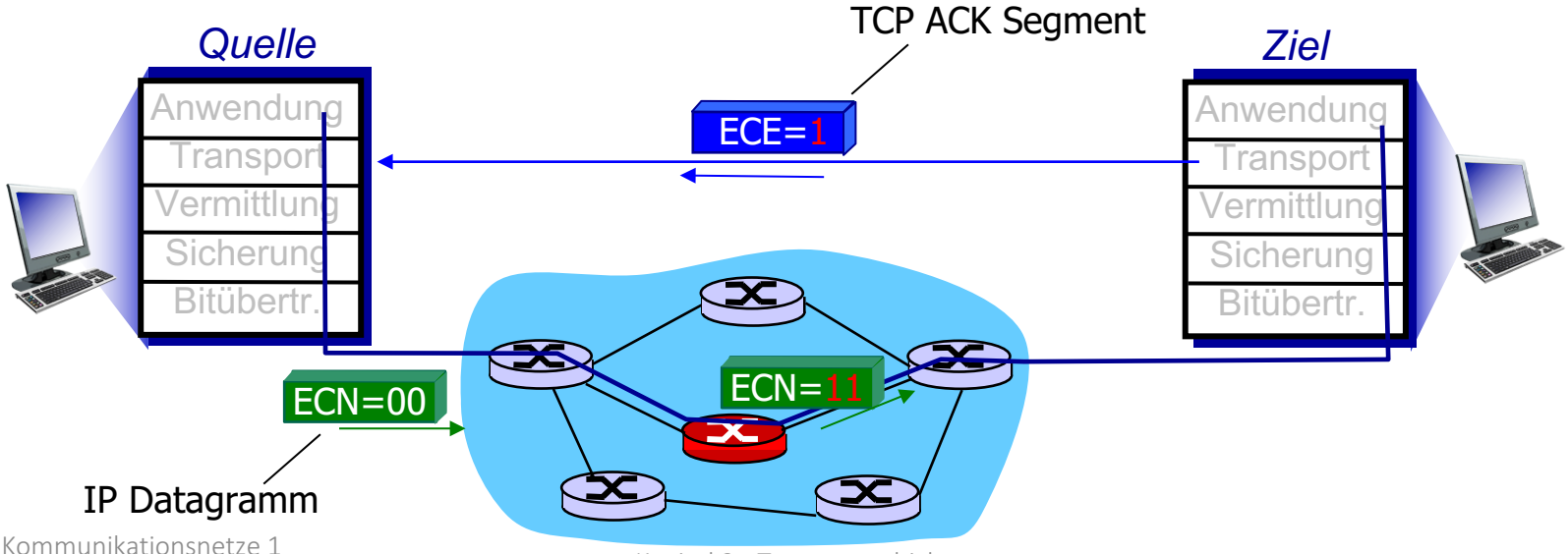
Fairness bei parallelen TCP-Verbindungen

- Anwendungen können parallele TCP-Verbindungen öffnen
- Web-Browser tun dies
- Z.B. Link mit Rate R und 9 bestehenden Verbindungen
  - Neue Anwendung nutz 1 TCP-Verbindung, bekommt Rate R/10
  - Neue Anwendung nutzt 11 TCP-Verbindungen, bekommt Rate > R/2

Explizit Congestion Notification (ECN) [RFC 3168]

Netz-unterstützte Staukontrolle

- Zwei Bits im IP-Header (ToS Feld) werden vom Router gesetzt, um Überlast zu signalisieren
- Überlast-Signal wird zum TCP-Empfänger getragen
- Empfänger (der Überlast-Signal im IP-Datagramm sieht) setzt ECE Bit (ECN Echo) in ACK an Sender um Überlast zu signalisieren



Zusammenfassung Kapitel 3

Wir haben in diesem Kapitel Wissen über die beiden wichtigsten Transportprotokolle im Internet erworben

- User Datagram Protocol UDP
  - „Unspannend“: IP + Ende-zu-Ende-Checksumme
- Transmission Control Protocol TCP
  - Zuverlässige Übertragung durch Sequenznummern, Quittungen und Timeouts
  - 3-Wege-Handschake für Verbindungsaufbau
  - Flusskontrolle über Empfänger-Fenster
  - Staukontrolle mit Slow Start, Congestion Avoidance, Fast Recovery