



# Systeme II

## 5. Die Transportschicht

Christian Schindelhauer

Technische Fakultät

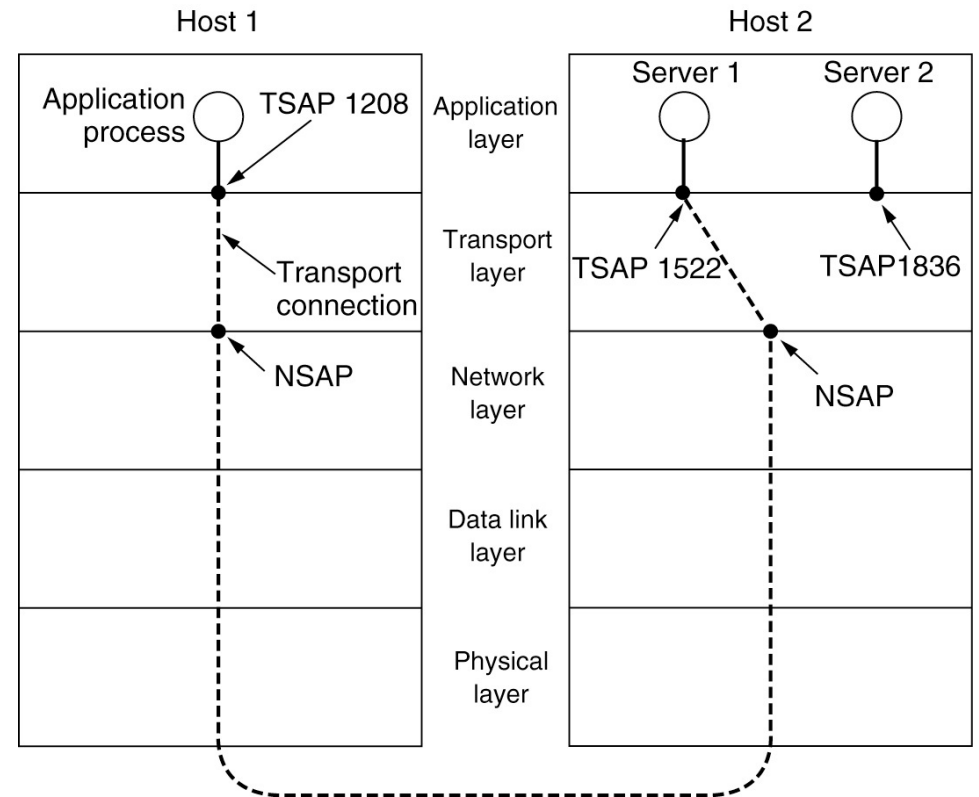
Rechnernetze und Telematik

Albert-Ludwigs-Universität Freiburg

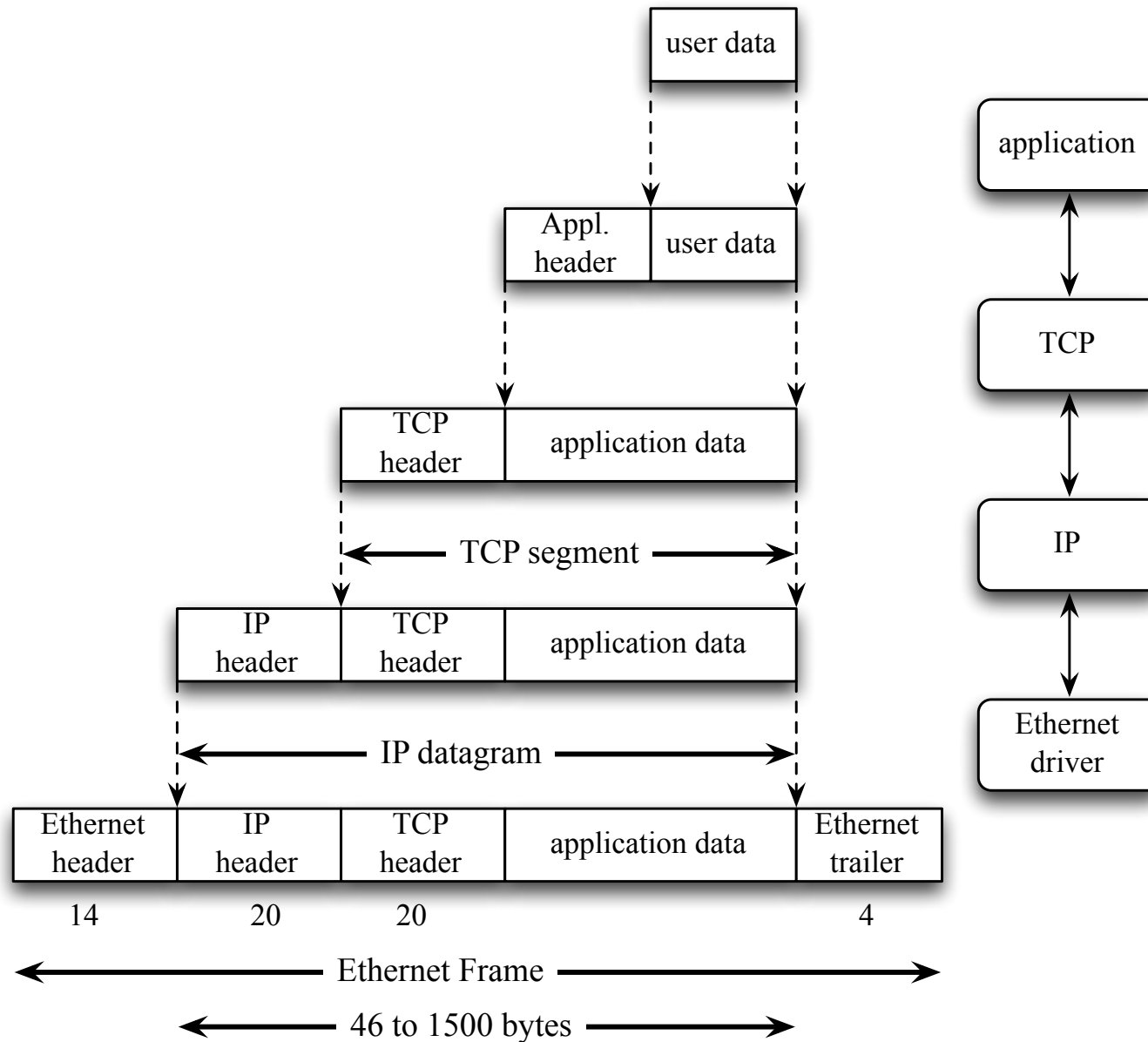
Version 02.07.2014

- Verbindungslos oder Verbindungsorientiert
  - Beachte: Sitzungsschicht im ISO/OSI-Protokoll
- Zuverlässig oder unzuverlässig
  - Best effort oder Quality of Service
  - Fehlerkontrolle
- Mit oder ohne Congestion Control
- Möglichkeit verschiedener Punkt-zu-Punktverbindungen
  - Stichwort: Demultiplexen
- Interaktionsmodelle
  - Byte-Strom, Nachrichten, „Remote Procedure Call“

- Die Netzwerkschicht leitet Daten an die Transportschicht unkontrolliert weiter
- Die Transportschicht muss sie den verschiedenen Anwendungen zuordnen:
  - z.B. Web, Mail, FTP, ssh, ...
  - In TCP/UDP durch Port-Nummern
  - z.B. Port 80 für Web-Server

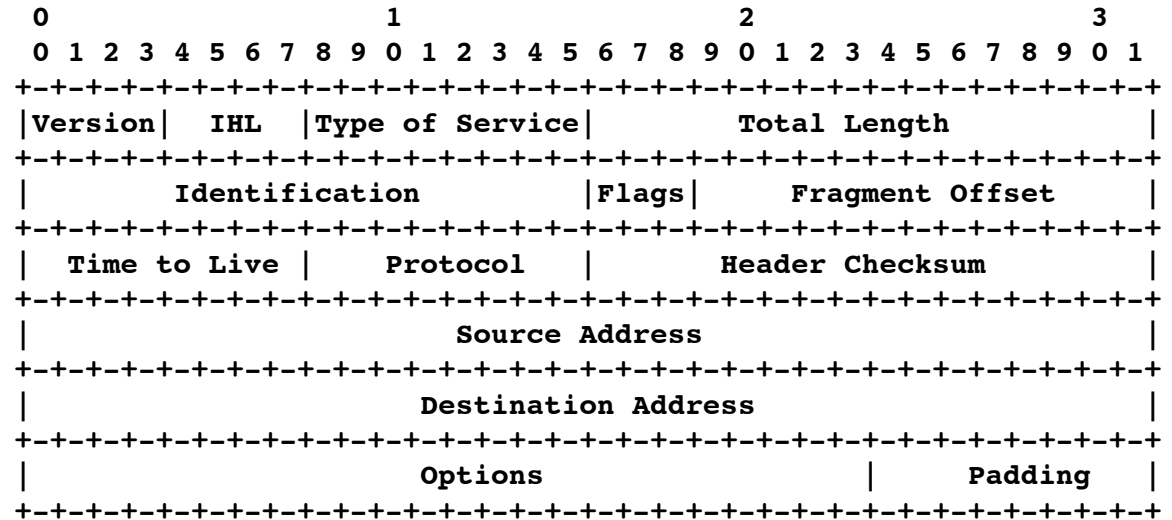


# Datenkapselung



# IP-Header (RFC 791)

- Version: 4 = IPv4
- IHL: Headerlänge
  - in 32 Bit-Wörter (>5)
- Type of Service
  - Optimierte delay, throughput, reliability, monetary cost
- Checksum (nur für IP-Header)
- Source and destination IP-address
- Protocol, identifiziert passendes Protokoll
  - Z.B. TCP, UDP, ICMP, IGMP
- Time to Live:
  - maximale Anzahl Hops



- Sequenznummer
  - Nummer des ersten Bytes im Segment
  - Jedes Datenbyte ist nummeriert modulo  $2^{32}$
- Bestätigungsnummer
  - Aktiviert durch ACK-Flag
  - Nummer des nächsten noch nicht bearbeiteten Datenbytes
    - = letzte Sequenznummer + letzte Datenmenge:

- Port-Adressen

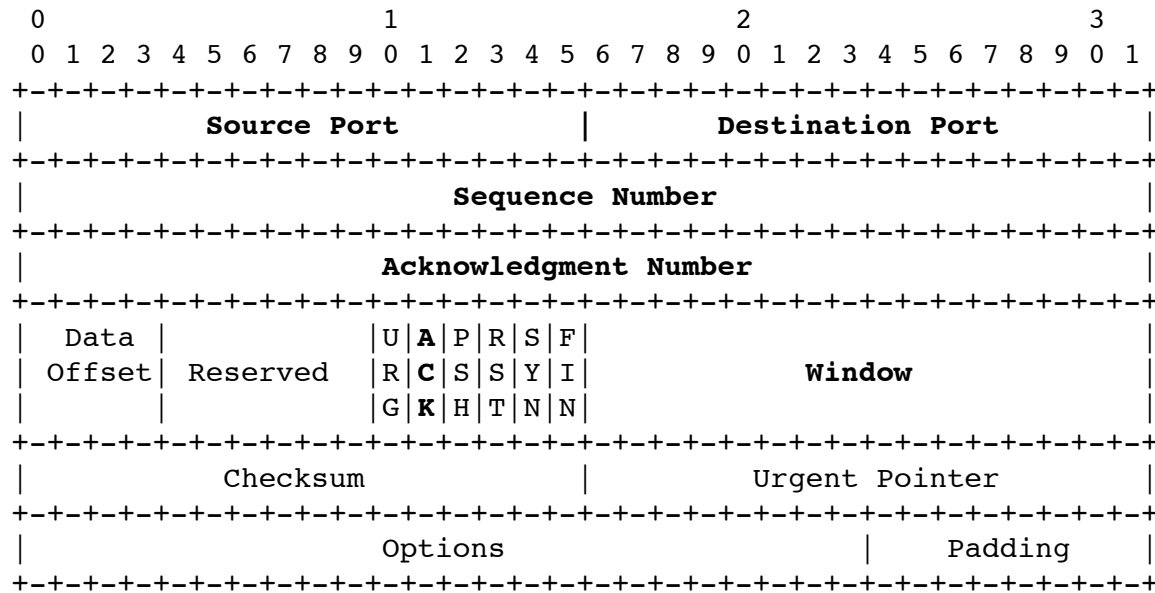
- Für parallele TCP-Verbindungen
- Ziel-Port-Nr.
- Absender-Port

- Headerlänge

- data offset

- Prüfsumme

- Für Header und Daten



- TCP (transmission control protocol)
  - Erzeugt zuverlässigen Datenfluß zwischen zwei Rechnern
  - Unterteilt Datenströme aus Anwendungsschicht in Pakete
  - Gegenseite schickt Empfangsbestätigungen (Acknowledgments)
- UDP (user datagram protocol)
  - Einfacher unzuverlässiger Dienst zum Versand von einzelnen Päckchen
  - Wandelt Eingabe in ein Datagramm um
  - Anwendungsschicht bestimmt Paketgröße
- Versand durch Netzwerkschicht
- Kein Routing: End-to-End-Protokolle

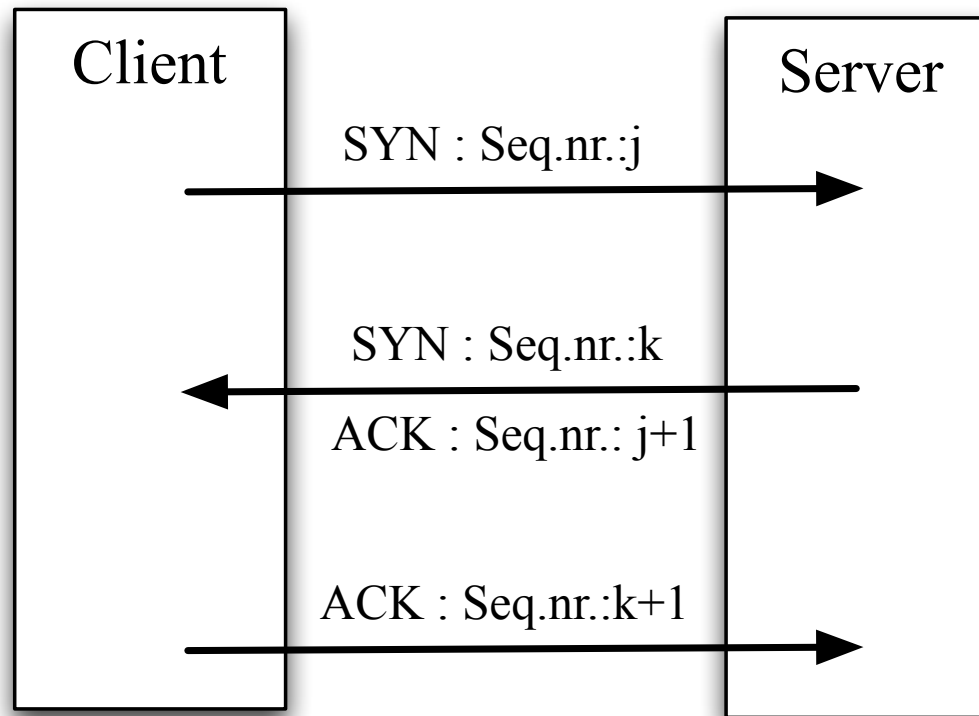
- TCP ist ein verbindungsorientierter, zuverlässiger Dienst für bidirektionale Byteströme
- TCP ist verbindungsorientiert
  - Zwei Parteien identifiziert durch Socket: IP-Adresse und Port  
(TCP-Verbindung eindeutig identifiziert durch Socketpaar)
  - Kein Broadcast oder Multicast
  - Verbindungsaufbau und Ende notwendig
  - Solange Verbindung nicht (ordentlich) beendet, ist Verbindung noch aktiv



- TCP ist ein verbindungsorientierter, zuverlässiger Dienst für bidirektionale Byteströme
- TCP ist zuverlässig
  - Jedes Datenpaket wird bestätigt (acknowledgment)
  - Erneutes Senden von unbestätigten Datenpakete
  - Checksum für TCP-Header und Daten
  - TCP nummeriert Pakete und sortiert beim Empfänger
  - Löscht duplizierte Pakete

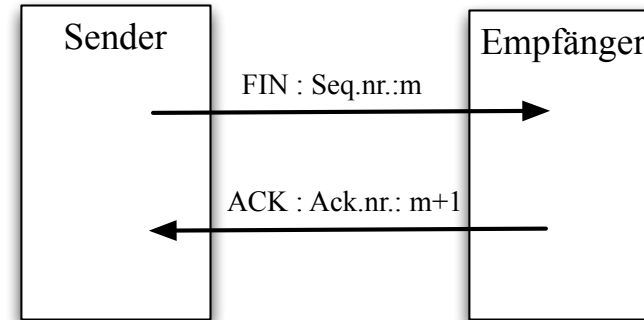
- TCP ist ein verbindungsorientierter, zuverlässiger Dienst für bidirektionale Byteströme
- TCP ist ein Dienst für bidirektionale Byteströme
  - Daten sind zwei gegenläufige Folgen aus einzelnen Bytes (=8 Bits)
  - Inhalt wird nicht interpretiert
  - Zeitverhalten der Datenfolgen kann verändert werden
  - Versucht zeitnahe Auslieferung jedes einzelnen Datenbytes
  - Versucht Übertragungsmedium effizient zu nutzen
    - = wenig Pakete

- In der Regel Client-Server-Verbindungen
  - Dann Aufbau mit drei TCP-Pakete (=Segmente)
  - Mit ersten SYN-Segment auch Übermittlung der MSS (maximum segment size)

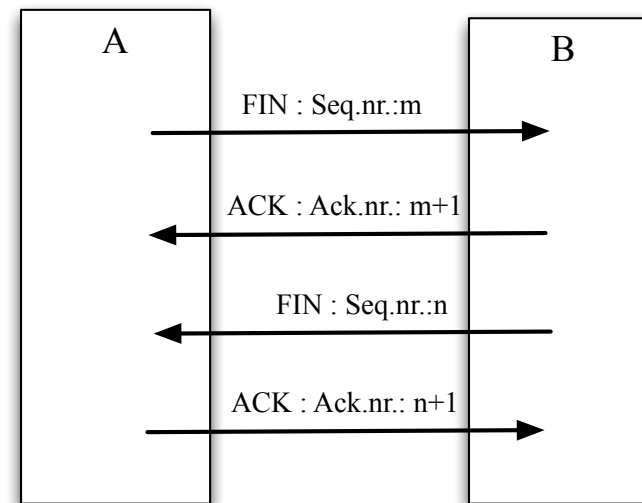


## ■ Half-Close

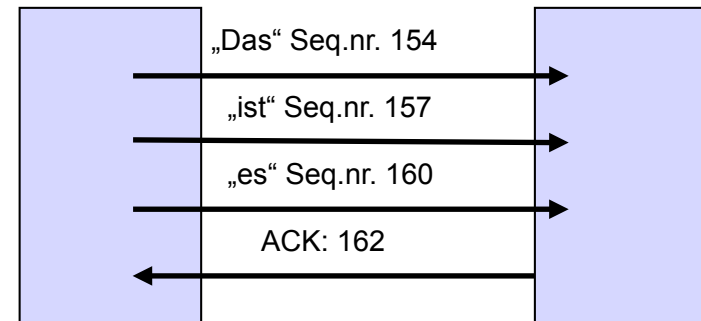
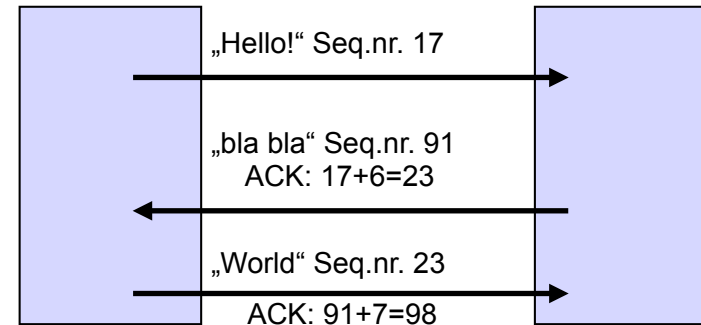
- Sender kündigt Ende mit FIN-Segment an und wartet auf Bestätigung
- In Gegenrichtung kann weitergesendet werden



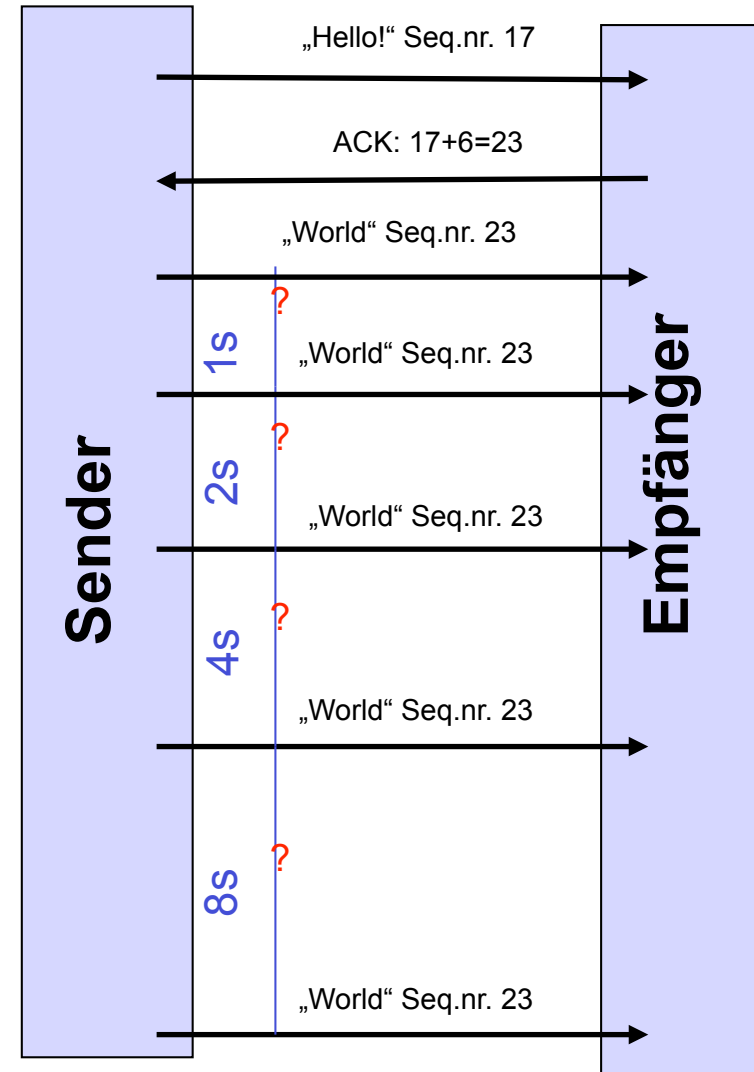
## ■ 2 Half-Close beenden TCP-Verbindung



- Huckepack-Technik
  - Bestätigungen „reiten“ auf den Datenpaket der Gegenrichtung
- Eine Bestätigungssegment kann viele Segmente bestätigen
  - Liegen keine Daten an, werden Acks verzögert



- Retransmission Timeout (RTO)
  - regelt Zeitraum zwischen Senden von Datenduplikaten, falls Bestätigung ausbleibt
- Wann wird ein TCP-Paket nicht bestätigt?
  - Wenn die Bestätigung wesentlich länger benötigt, als die durchschnittliche Umlaufzeit (RTT/round trip time)
    - 1. Problem: Messung der RTT
    - 2. Problem: Bestätigung kommt, nur spät
  - Sender
    - Wartet Zeitraum gemäß RTO
    - Sendet Paket nochmal und setzt
    - $RTO \leftarrow 2 RTO$  (bis  $RTO = 64$  Sek.)
- Neuberechnung von RTO, wenn Pakete bestätigt werden



- TCP-Paket gilt als nicht bestätigt, wenn Bestätigung „wesentlich“ länger dauert als RTO
  - RTT nicht on-line berechenbar (nur rückblickend)
  - RTT schwankt stark
- Daher: Retransmission Timeout Value aus großzügiger Schätzung:
  - RFC 793: ( $M :=$  letzte gemessene RTT)
    - $R \leftarrow \alpha R + (1 - \alpha) M$ , wobei  $\alpha = 0,9$
    - $RTO \leftarrow \beta R$ , wobei  $\beta = 2$
  - Jacobson 88: Schätzung nicht robust genug, daher
    - $A \leftarrow A + g (M - A)$ , wobei  $g = 1/8$
    - $D \leftarrow D + h (|M - A| - D)$ , wobei  $h = 1/4$
    - $RTO \leftarrow A + 4D$
- Aktualisierung nicht bei mehrfach versandten Pakete

- Wie kann man sicherstellen,
  - dass kleine Pakete zeitnah ausgeliefert werden
  - und bei vielen Daten große Pakete bevorzugt werden?
- Algorithmus von Nagle:
  - Kleine Pakete werden nicht versendet, solange Bestätigungen noch ausstehen.
    - Paket ist klein, wenn Datenlänge < MSS
  - Trifft die Bestätigung des zuvor gesendeten Pakets ein, so wird das nächste verschickt.
- Beispiel:
  - Telnet versus ftp
- Eigenschaften
  - Selbst-taktend: Schnelle Verbindung = viele kleine Pakete



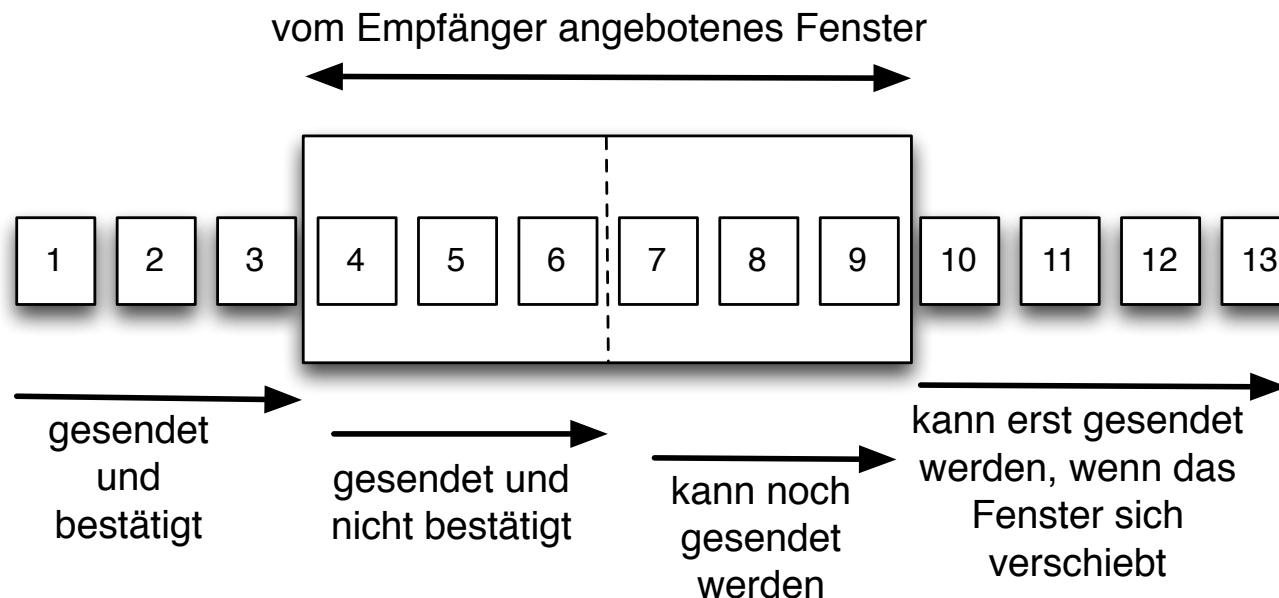
- Problem: Schneller Sender und langsamer Empfänger
  - Der Sender lässt den Empfangspuffer des Empfängers überlaufen
  - Übertragungsbandweite wird durch sinnlosen Mehrfachversand (nach Fehlerkontrolle) verschwendet
- Anpassung der Frame-Sende-Rate an dem Empfänger notwendig

Langsamer Empfänger

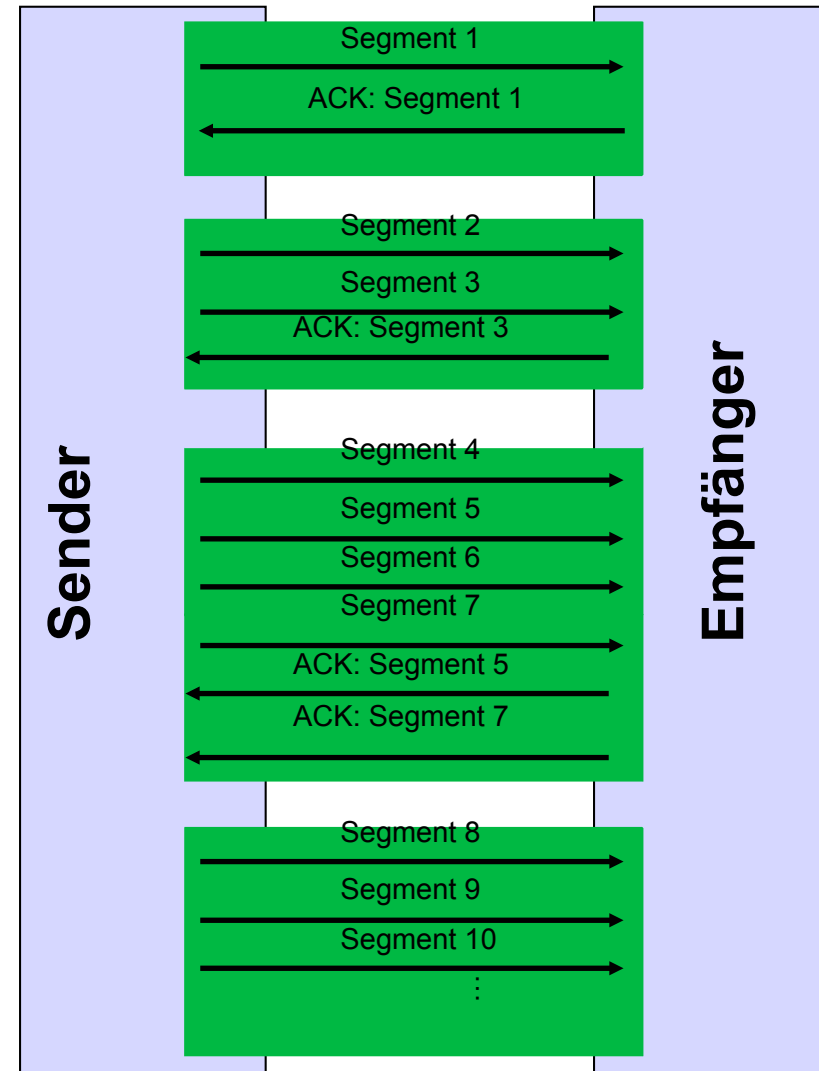


Schneller Sender

- Datenratenanpassung durch Fenster
  - Empfänger bestimmt Fenstergröße (wnd) im TCP-Header der ACK-Segmente
  - Ist Empfangspuffer des Empfängers voll, sendet er  $wnd=0$
  - Andernfalls sendet Empfänger  $wnd>0$
- Sender beachtet:
  - Anzahl unbestätigter gesender Daten  $\leq$  Fenstergröße



- Sender darf vom Empfänger angebotene Fenstergröße nicht von Anfang wahrnehmen
- 2. Fenster: Congestion-Fenster (cwnd/Congestion window)
  - Von Sender gewählt (FSK)
  - Sendefenster:  $\min \{w_{nd}, c_{wnd}\}$
  - S: Segmentgröße
  - Am Anfang:
    - $c_{wnd} \leftarrow S$
  - Für jede empfangene Bestätigung:
    - $c_{wnd} \leftarrow c_{wnd} + S$
  - Solange bis einmal Bestätigung ausbleibt
- „Slow Start“ = Exponentielles Wachstum



# Stauvermeidung in TCP Tahoe

- Jacobson 88:

**x: Anzahl Pakete pro RTT**

- Parameter: cwnd und Slow-Start-Schwellwert (ssthresh=slow start threshold)
- S = Datensegmentgröße = maximale Segmentgröße

- Verbindungsaufbau:

- $cwnd \leftarrow S$                        $ssthresh \leftarrow 65535$

$x \leftarrow 1$        $y \leftarrow \max$

- Bei Paketverlust, d.h. Bestätigungsdauer > RTO,

- multiplicatively decreasing

$$cwnd \leftarrow S \qquad ssthresh \leftarrow \max \left\{ 2 \cdot S, \frac{\min \{ cwnd, wnd \}}{2} \right\}$$

$x \leftarrow 1$        $y \leftarrow x/2$

- Werden Segmente bestätigt und  $cwnd \leq ssthresh$ , dann

- slow start:  $cwnd \leftarrow cwnd + S$

$x \leftarrow 2 \cdot x, \text{ bis } x = y$

- Werden Segmente bestätigt und  $cwnd > ssthresh$ , dann additively increasing

$$cwnd \leftarrow cwnd + S \cdot \frac{S}{cwnd}$$

$x \leftarrow x + 1$

# TCP Tahoe

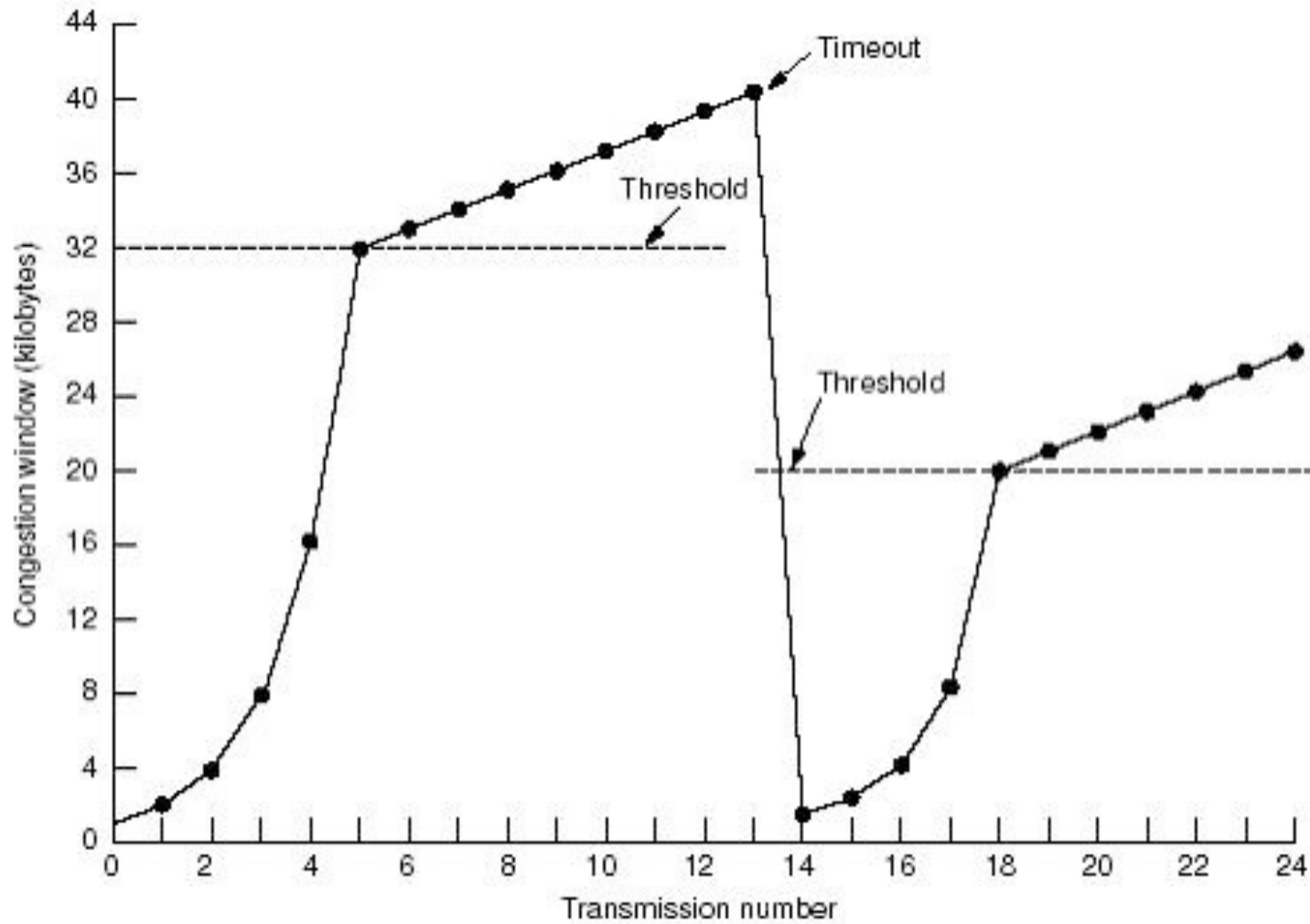


Fig3

pictures from TANENBAUM A. S. *Computer Networks 3rd edition*

- TCP Tahoe [Jacobson 1988]:
  - Geht nur ein Paket verloren, dann
    - Wiederversand Paket + Restfenster
    - Und gleichzeitig Slow Start
  - Fast retransmit
    - Nach drei Bestätigungen desselben Pakets (triple duplicate ACK),
    - sende Paket nochmal, starte mit Slow Start
- TCP Reno [Stevens 1994]
  - Nach Fast retransmit:
    - $ssthresh \leftarrow \min(wnd, cwnd)/2$
    - $cwnd \leftarrow ssthresh + 3 S$
  - Fast recovery nach Fast retransmit
    - Erhöhe Paketrate mit jeder weiteren Bestätigung
    - $cwnd \leftarrow cwnd + S$
  - Congestion avoidance: Trifft Bestätigung von P+x ein:
    - $cwnd \leftarrow ssthresh$

$y \leftarrow x/2$
$x \leftarrow y + 3$

- Kombination von TCP und Fast Recovery verhält sich im wesentlichen wie folgt:

- Verbindungsaufbau:

$$x \leftarrow 1$$

- Bei Paketverlust, MD:multiplicative decreasing

$$x \leftarrow x/2$$

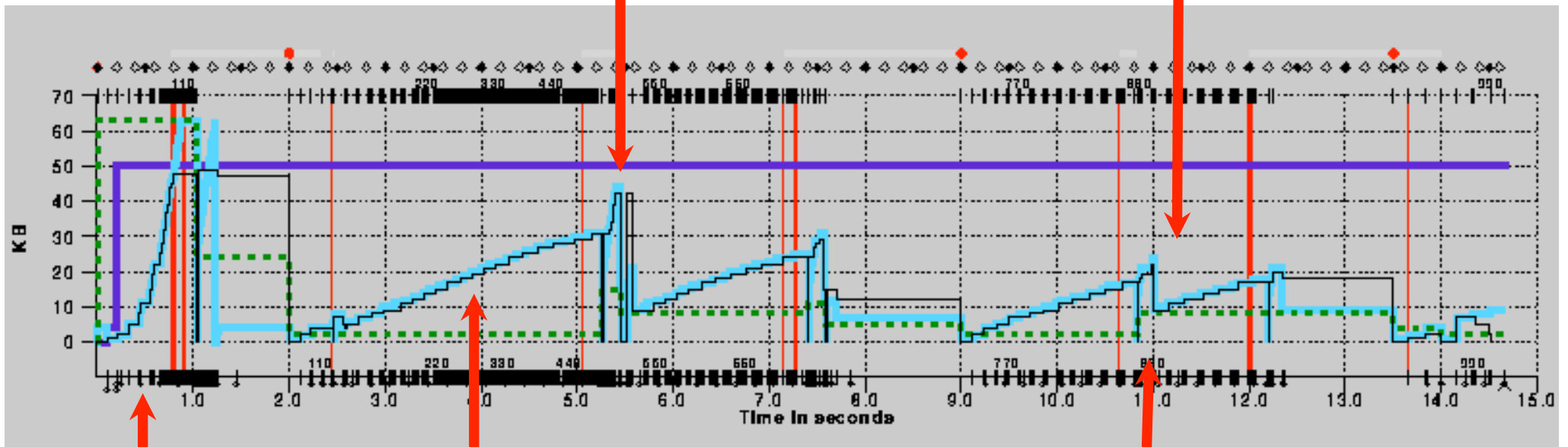
- Werden Segmente bestätigt, AI: additive increasing

$$x \leftarrow x + 1$$

# Beispiel: TCP Reno in Aktion

Fast Retransmit

Fast Recovery



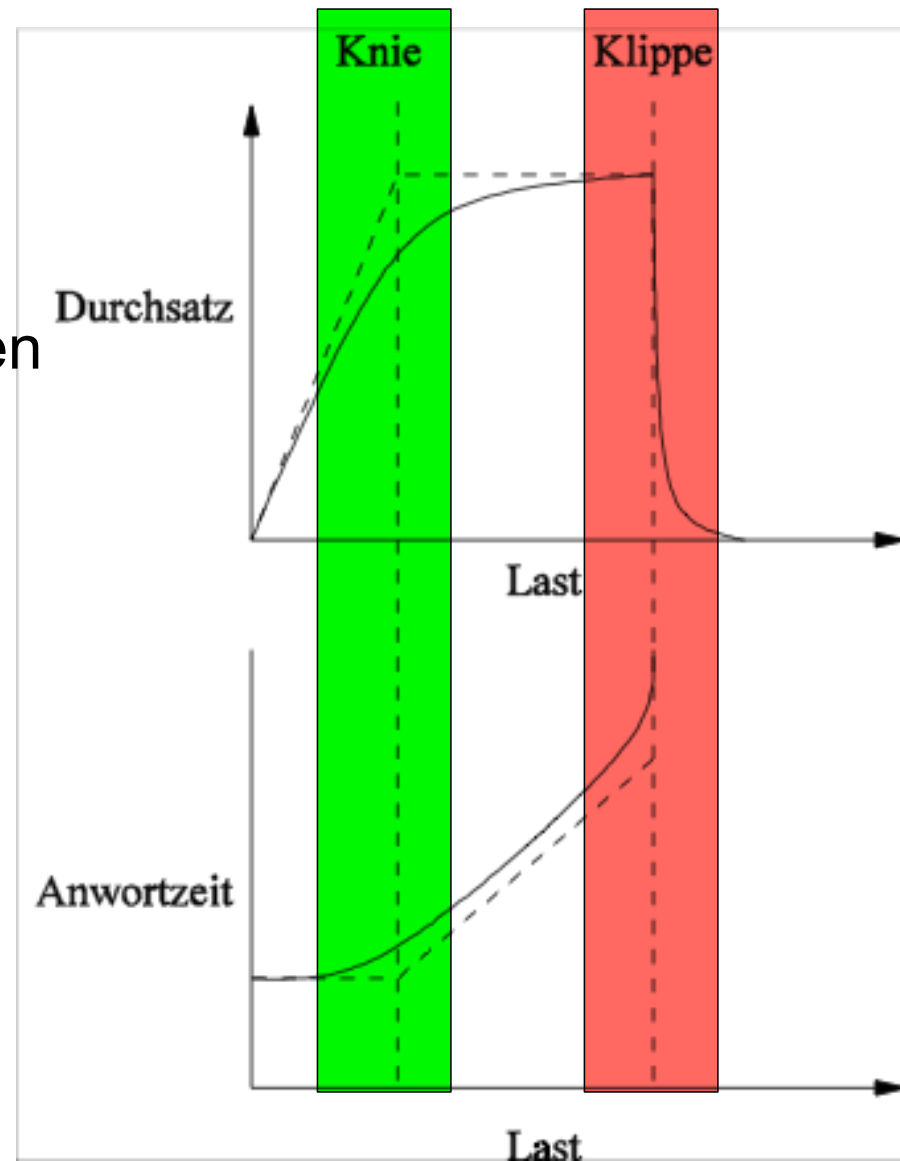
Additively Increase

Slow Start

Multiplicatively Decrease



- **Klippe:**
  - Hohe Last
  - Geringer Durchsatz
  - Praktisch alle Daten gehen verloren
- **Knie:**
  - Hohe Last
  - Hoher Durchsatz
  - Einzelne Daten gehen verloren



- n Teilnehmer, Rundenmodell
  - Teilnehmer i hat Datenrate  $x_i(t)$
  - Anfangsdatenrate  $x_1(0), \dots, x_n(0)$  gegeben
- Feedback nach Runde t:
  - $y(t) = 0$ , falls  $\sum_{i=1}^n x_i(t) \leq K$
  - $y(t) = 1$ , falls  $\sum_{i=1}^n x_i(t) > K$
  - wobei K ist Knielast
- Jeder Teilnehmer aktualisiert in Runde t+1:
  - $x_i(t+1) = f(x_i(t), y(t))$
  - Increase-Strategie  $f_0(x) = f(x, 0)$
  - Decrease-Strategie  $f_1(x) = f(x, 1)$
- Wir betrachten lineare Funktionen:

$$f_0(x) = a_I + b_I x \quad \text{und} \quad f_1(x) = a_D + b_D x .$$

- Interessante Spezialfälle:

- AIAD: Additive Increase  
Additive Decrease

$$f_0(x) = a_I + x \quad \text{und} \quad f_1(x) = a_D + x ,$$

wobei  $a_I > 0$  und  $a_D < 0$ .

- MIMD: Multiplicative  
Increase/Multiplicative  
Decrease

$$f_0(x) = b_I x \quad \text{und} \quad f_1(x) = b_D x ,$$

wobei  $b_I > 1$  und  $b_D < 1$ .

- AIMD: Additive Increase  
Multiplicative Decrease

$$f_0(x) = a_I + x \quad \text{und} \quad f_1(x) = b_D x ,$$

wobei  $a_I > 0$  und  $b_D < 1$ .

- Effizienz

- Last: 
$$X(t) := \sum_{i=1}^n x_i(t)$$

- Maß

$$|X(t) - K|$$

- Fairness: Für  $x=(x_1, \dots, x_n)$ :

$$F(x) = \frac{(\sum_{i=1}^n x_i)^2}{n \sum_{i=1}^n (x_i)^2} \cdot$$

- $1/n \leq F(x) \leq 1$

- $F(x) = 1 \leftrightarrow$  absolute Fairness

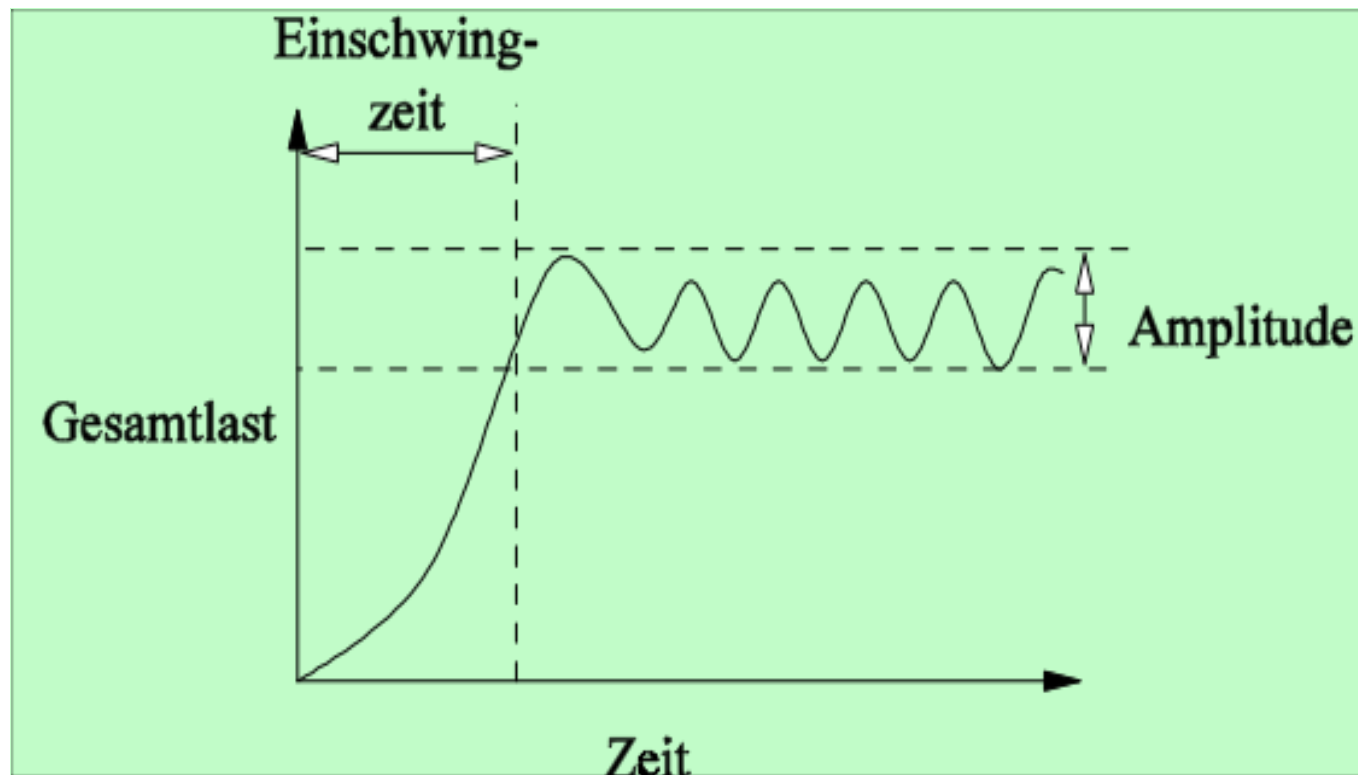
- Skalierungsunabhängig

- Kontinuierlich, stetig, differenzierbar

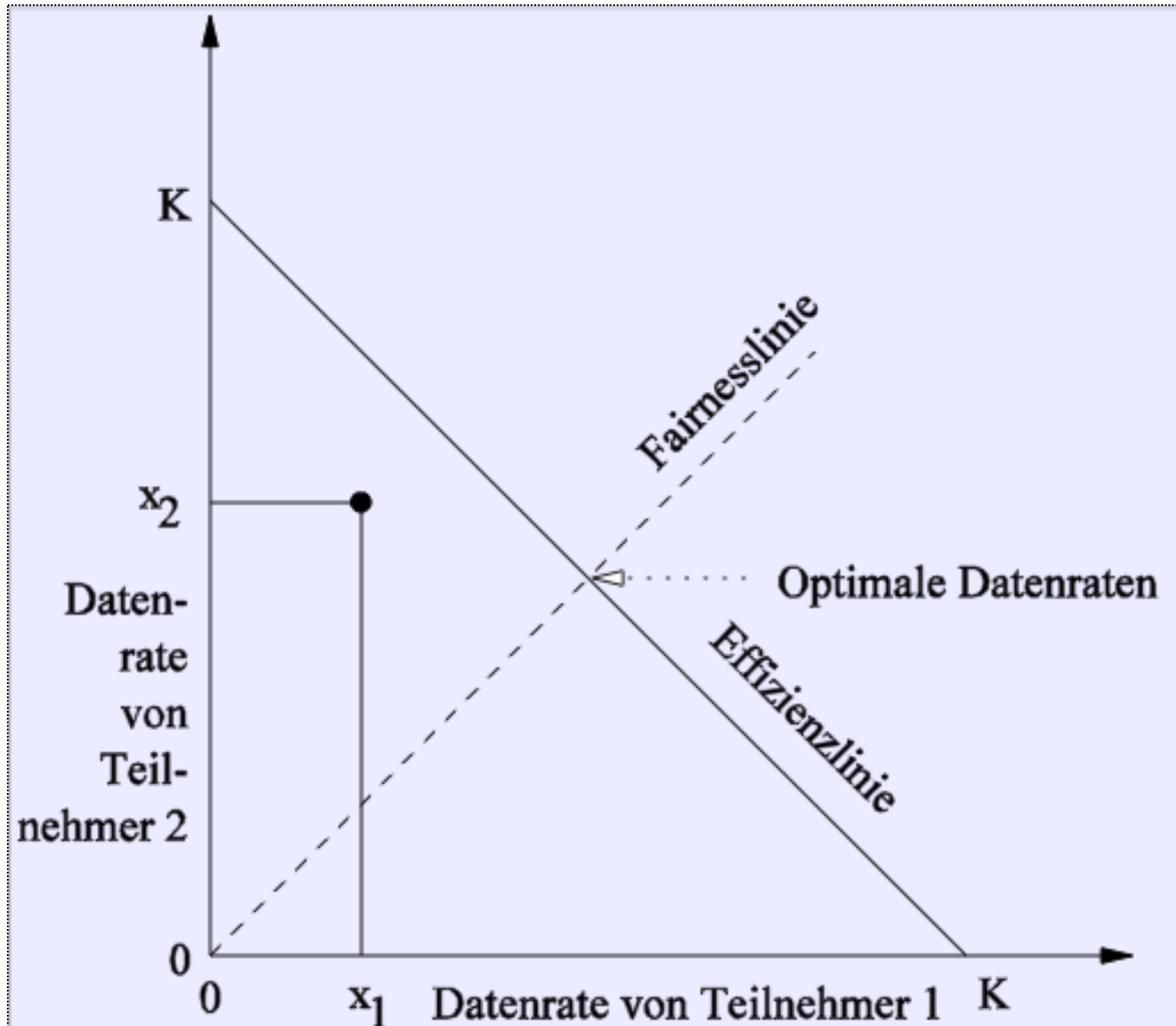
- Falls  $k$  von  $n$  fair, Rest 0, dann  $F(x) = k/n$

-

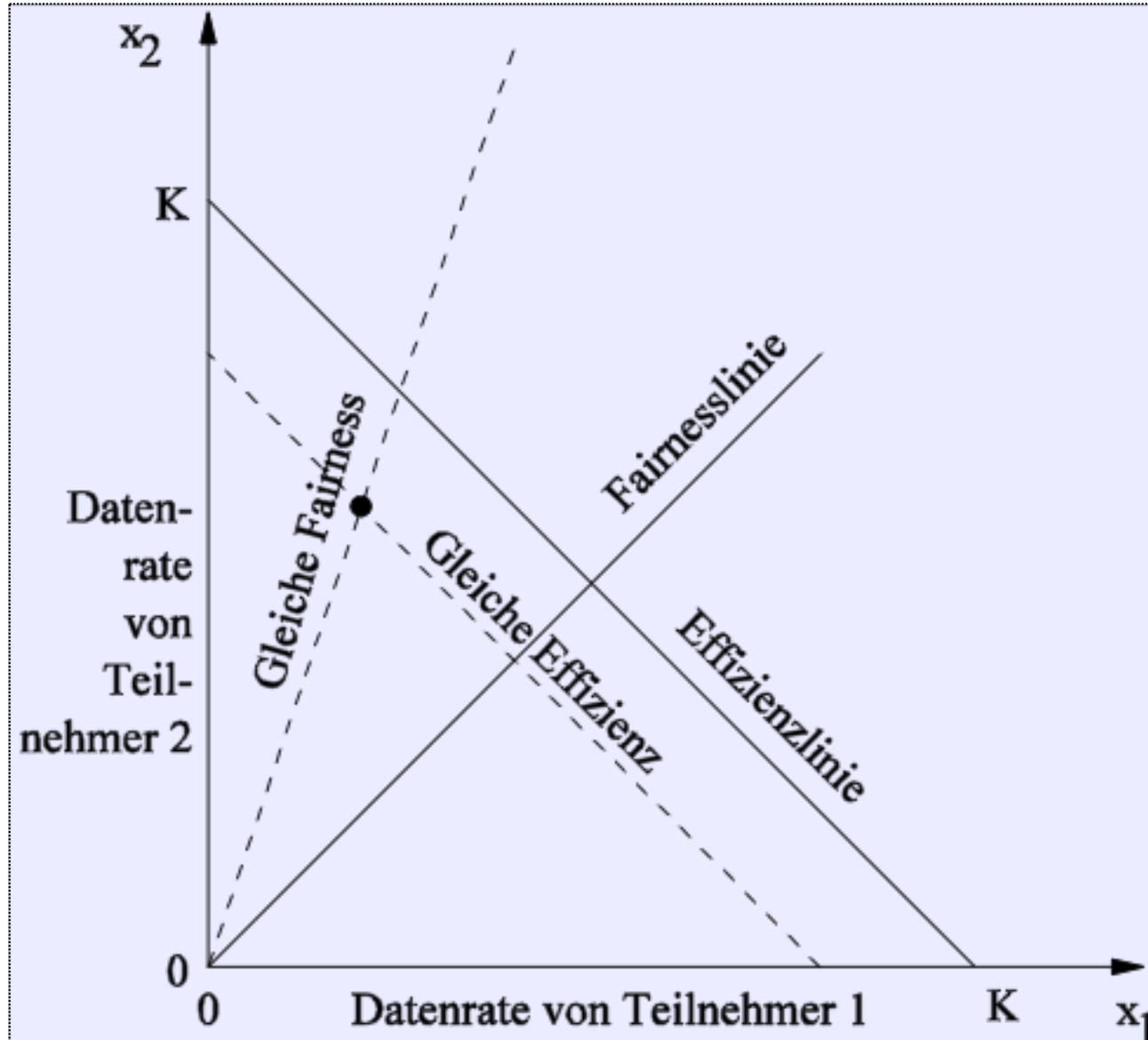
- Konvergenz unmöglich
- Bestenfalls Oszillation um Optimalwert
  - Oszillationsamplitude  $A$
  - Einschwingzeit  $T$



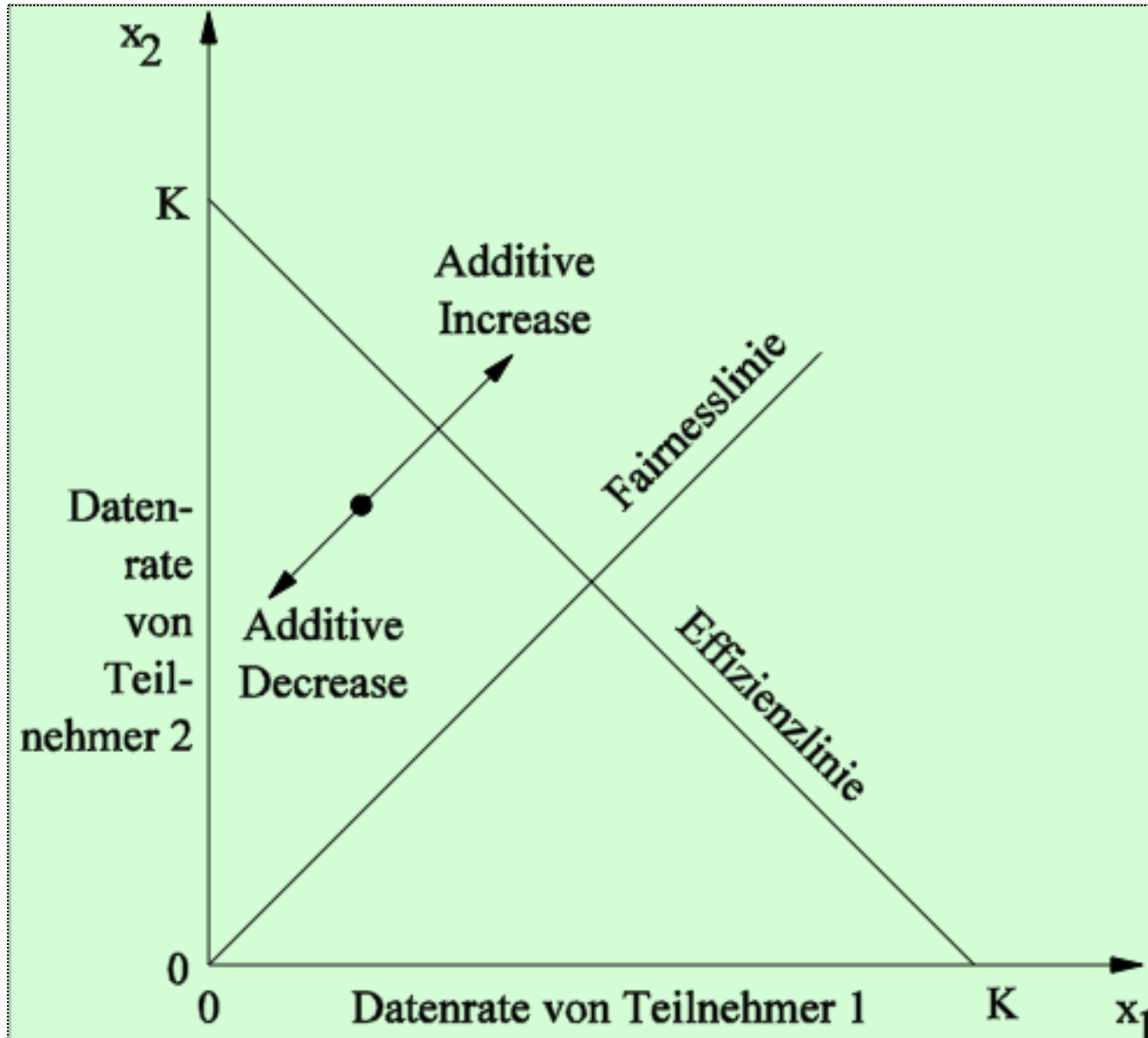
# Vektordarstellung (I)



# Vektordarstellung (II)

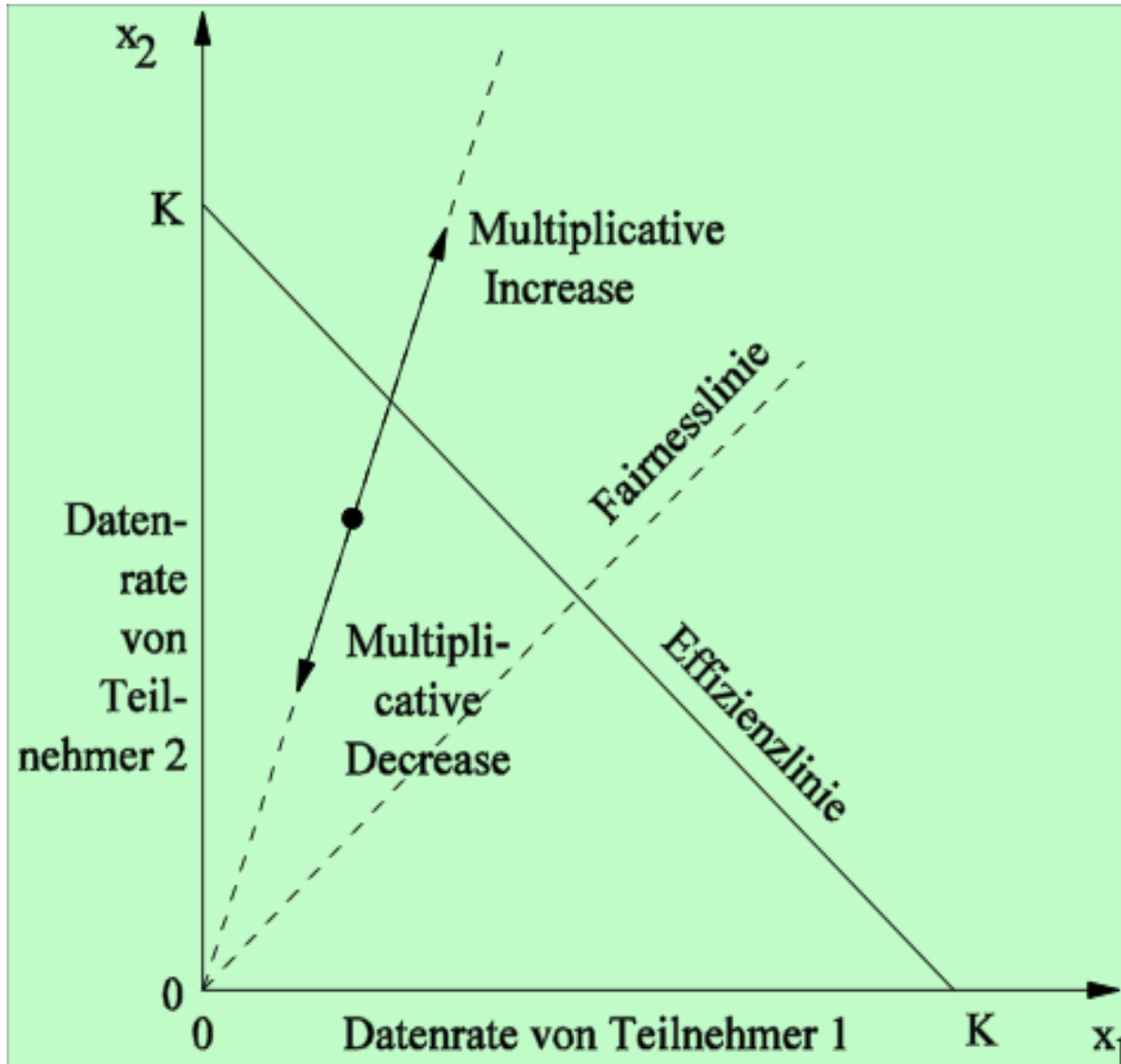


# AIAD Additive Increase/ Additive Decrease

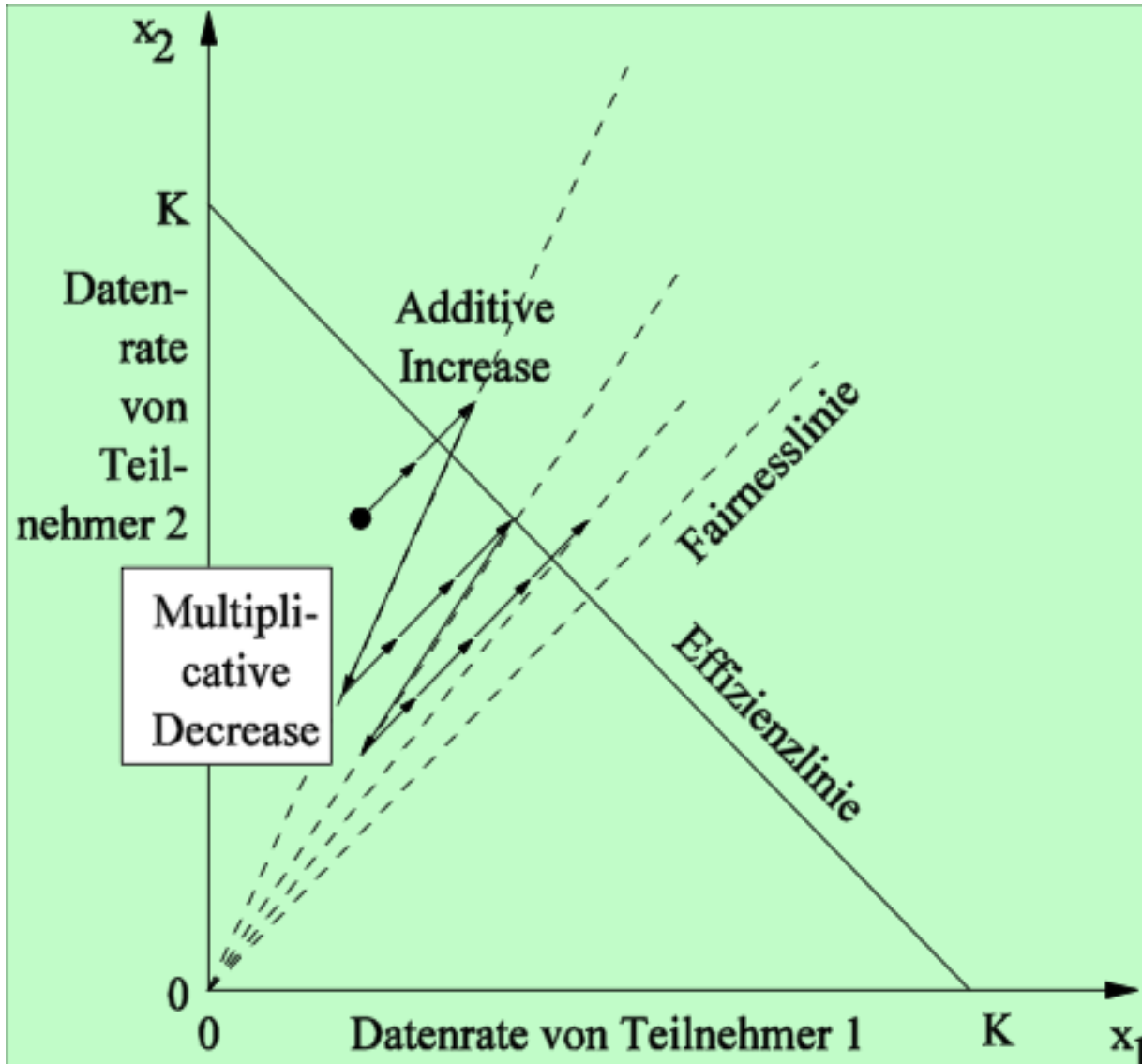




# MIMD: Multiplicative Incr./ Multiplicative Decrease



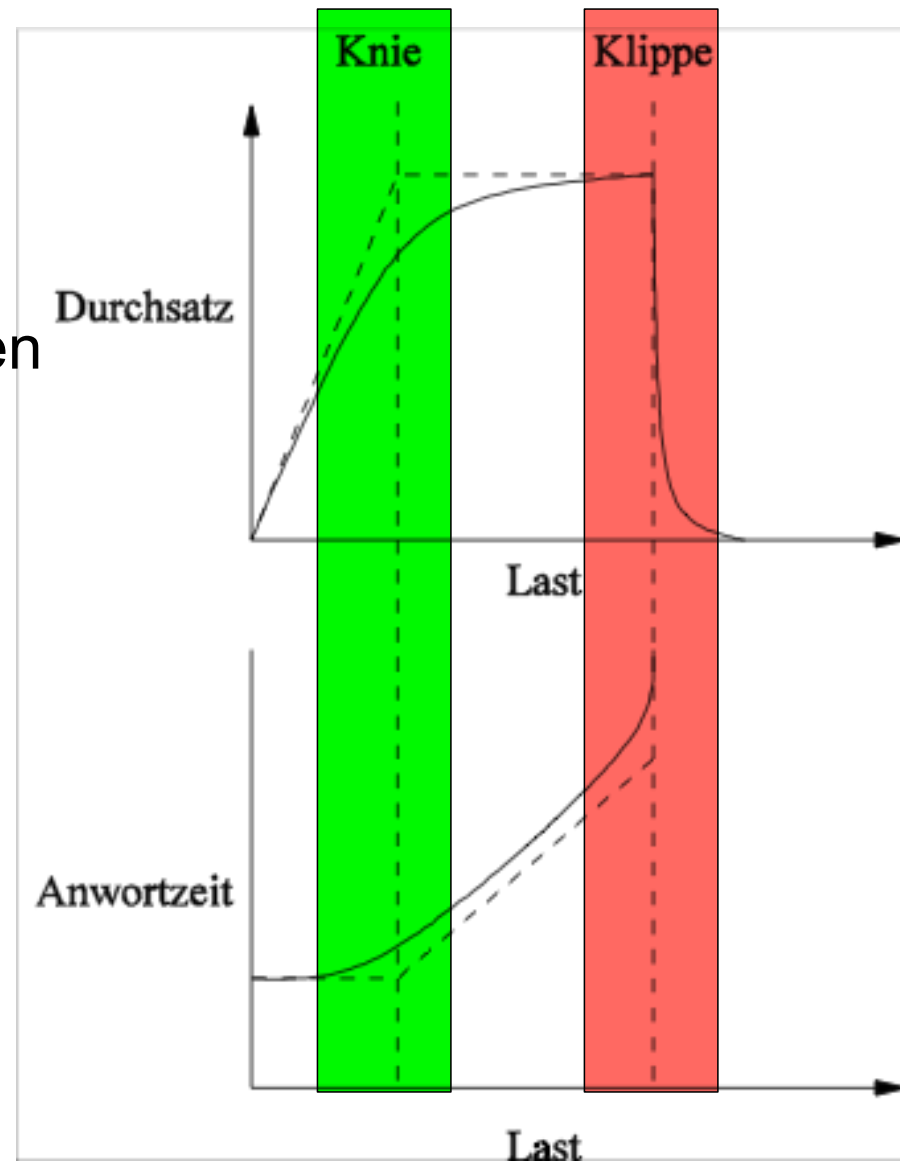
# AIMD: Additively Increase/ Multiplicatively Decrease



- Verbindungen mit großer RTT werden diskriminiert
- Warum?
  - Auf jeden Router konkurrieren TCP-Verbindungen
  - Paketverluste halbieren Umsatz (MD)
  - Wer viele Router hat, endet mit sehr kleinen Congestion-Window
- Außerdem:
  - Kleinere RTT ist schnellere Update-Zeit
  - Daher steigt die Rate (AI) auf kurzen Verbindungen schneller
  - Mögliche Lösung:
    - konstante Datenratenanpassung statt Fenster-basierte Anpassung

- RTT-basiertes Protokoll als Nachfolger von TCP Reno
  - “L. Brakmo and L. Peterson, “TCP Vegas: End-to-End Congestion Avoidance on a Global Internet”, IEEE Journal on Selected Areas of Communications, vol. 13, no. 8, October 1995, pp. 1465–1480.
- Bessere Effizienz
- Geringere Paketverluste
- Aber:
  - TCP Vegas und TCP Reno gegeneinander unfair

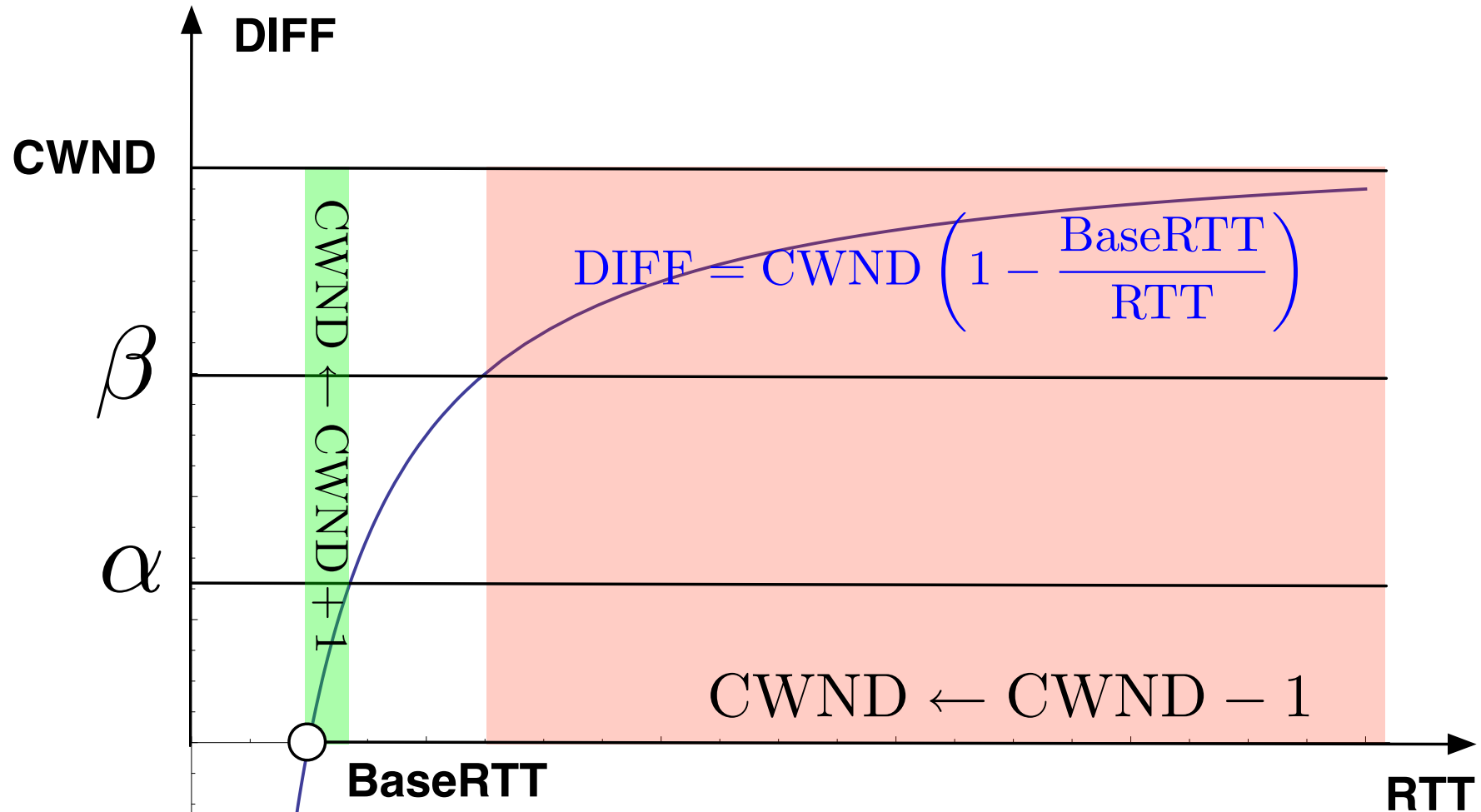
- **Klippe:**
  - Hohe Last
  - Geringer Durchsatz
  - Praktisch alle Daten gehen verloren
- **Knie:**
  - Hohe Last
  - Hoher Durchsatz
  - Einzelne Daten gehen verloren



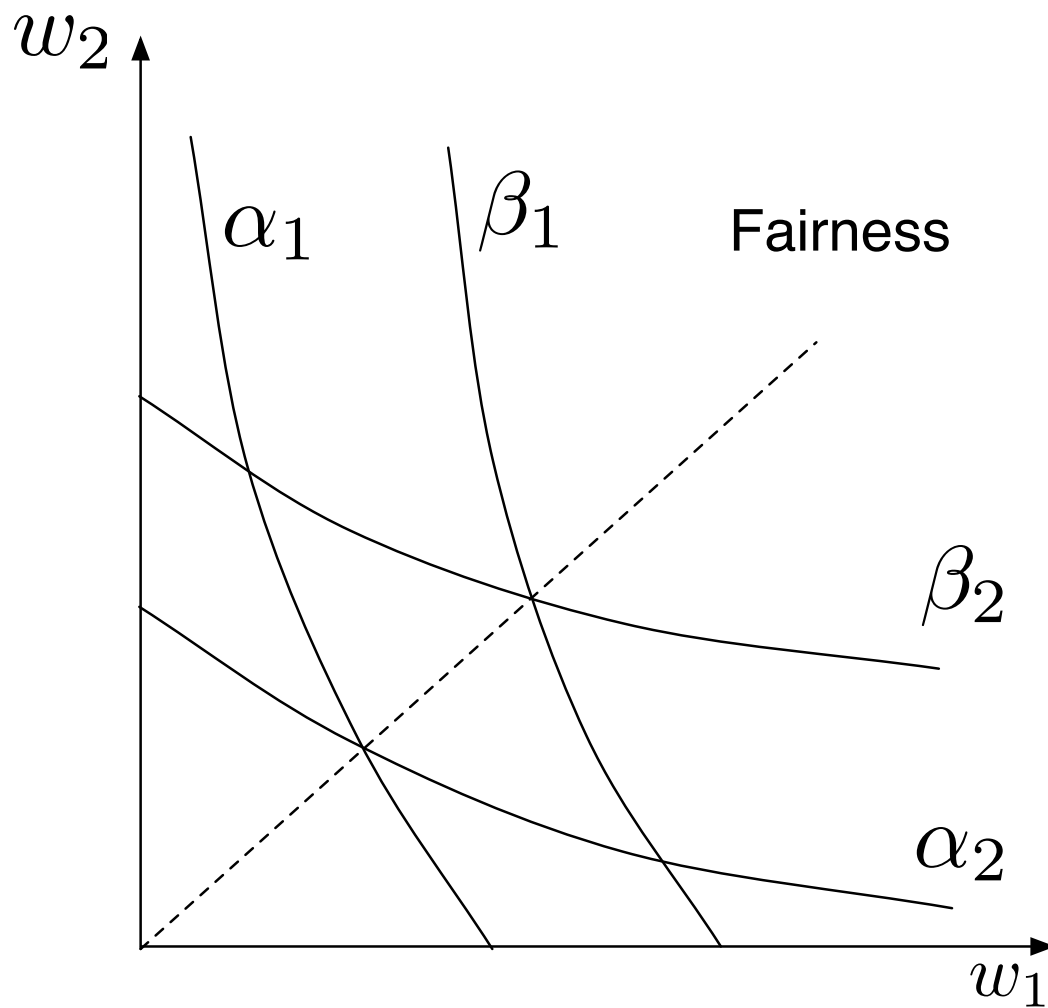


- Parameter
  - geschätzte Umlaufzeit:  $RTT$
  - minimale Umlaufzeit:  $BaseRTT$
  - wirkliche Datenrate:  $Actual = CWND/RTT$
  - erwartete Datenrate:  $Expected = CWND/BaseRTT$
  - $Diff = (Expected - Actual) BaseRTT$
  - **Programmparameter:  $0 \leq \alpha < \beta$**
- Wenn  $Diff \leq \alpha$  (d.h.  $Actual \approx Expected$ )
  - Last ist gering
  - $CWND \leftarrow CWND + 1$
- Wenn  $Diff > \beta$ , (d.h.  $Actual \ll Expected$ )
  - Last ist zu hoch
  - $CWND \leftarrow CWND - 1$
- Sonst keine Aktion:  $CWND \leftarrow CWND$

# TCP Vegas - Abhängigkeit von RTT

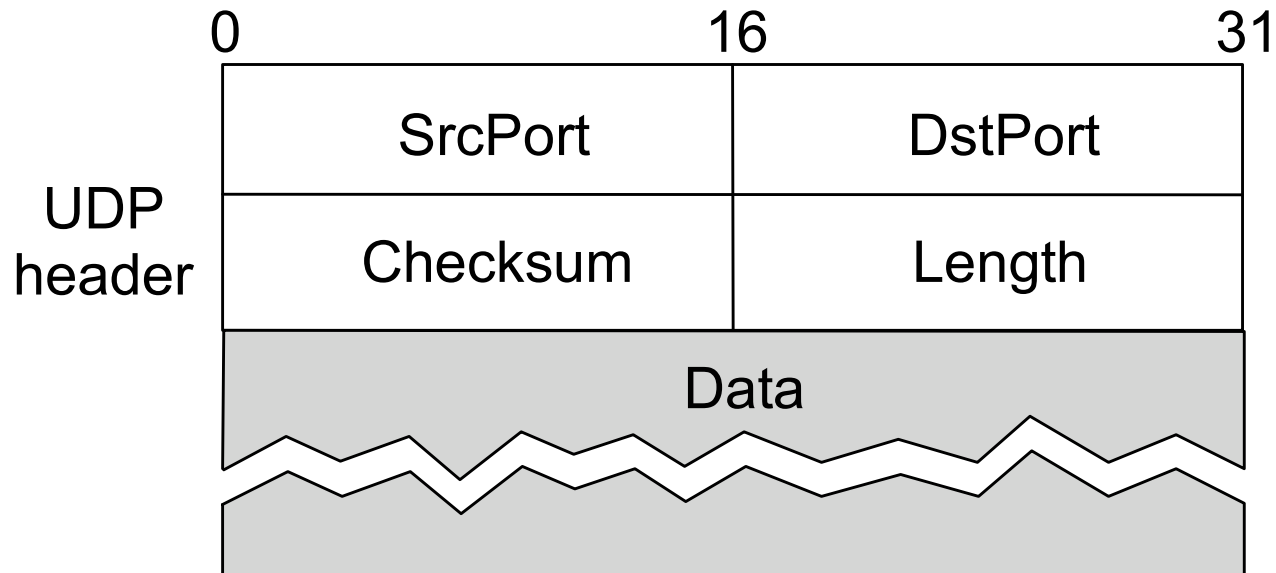






- TCP
  - reagiert dynamisch auf die zur Verfügung stehende Bandweite
  - Faire Aufteilung der Bandweite
    - Im Idealfall:  $n$  TCP-Verbindungen erhalten einen Anteil von  $1/n$
- Zusammenspiel mit anderen Protokollen
  - Reaktion hängt von der Last anderer Transportprotokolle ab
    - z.B. UDP hat keine Congestion Control
  - Andere Protokolle können jeder Zeit eingesetzt werden
  - UDP und andere Protokoll können TCP Verbindungen unterdrücken
- Schlussfolgerung
  - Transport-Protokolle müssen TCP-kompatibel sein (TCP friendly)

- User Datagram Protocol (UDP)
  - ist ein unzuverlässiges, verbindungsloses Transportprotokoll für Pakete
- Hauptfunktion:
  - Demultiplexing von Paketen aus der Vermittlungsschicht
- Zusätzlich (optional):
  - Checksum aus UDP Header + Daten



- TCP erzeugt zuverlässigen Byte-Strom
  - Fehlerkontrolle durch “GoBack-N”
- Congestion control
  - Fensterbasiert
  - AIMD, Slow start, *Congestion Threshold*
  - Flusskontrolle durch *Window*
  - Verbindungsaufbau
  - Algorithmus von Nagle

# Systeme II

## 5. Die Transportschicht

Christian Schindelhauer

Technische Fakultät

Rechnernetze und Telematik

Albert-Ludwigs-Universität Freiburg