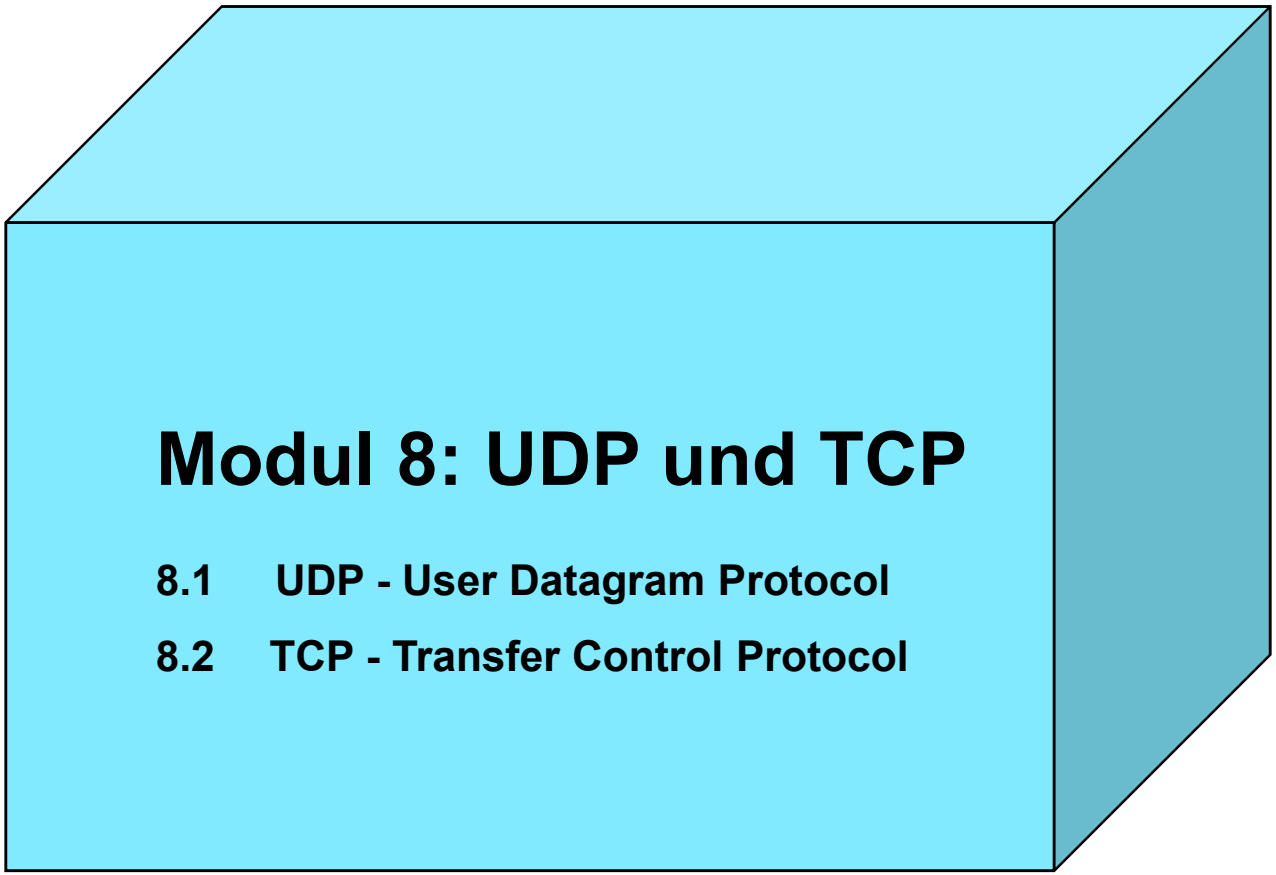
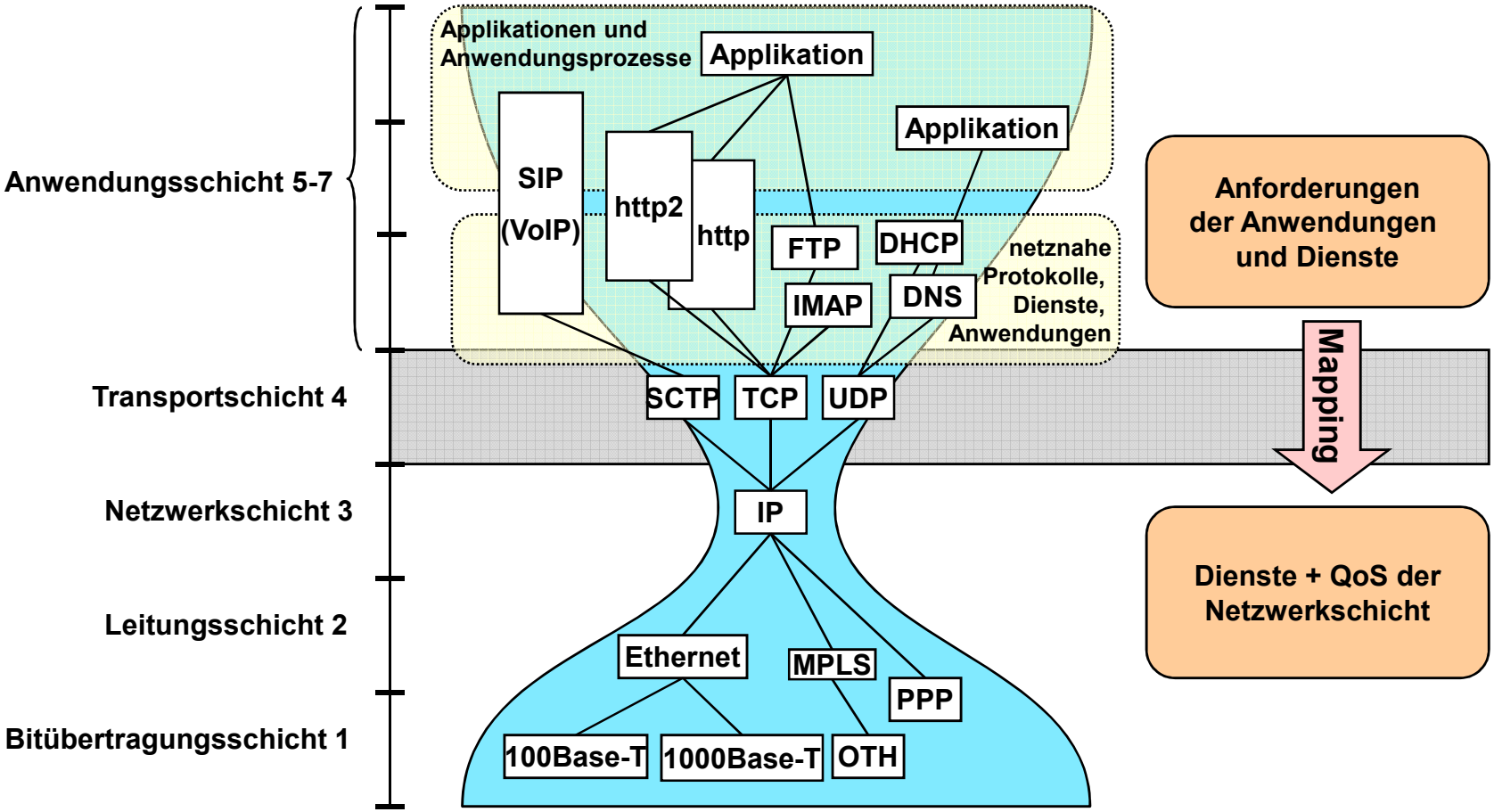
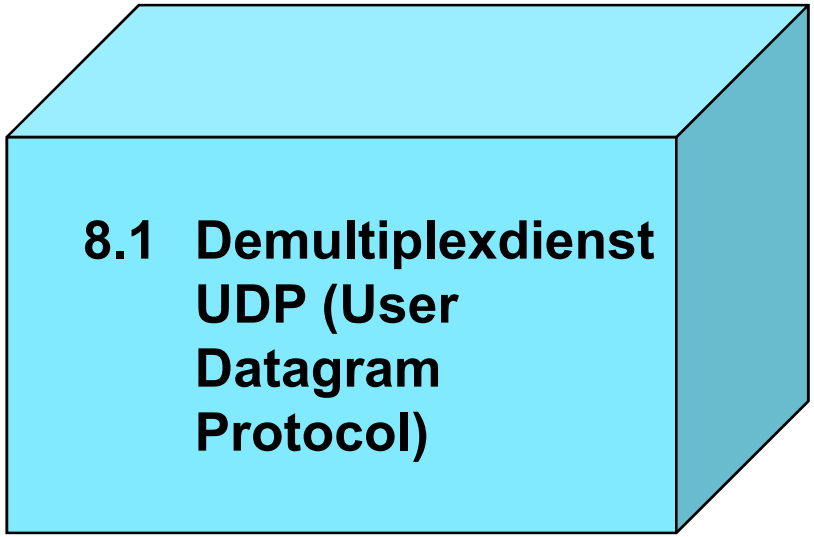

Modul 8: UDP und TCP
8.1 UDP - User Datagram Protocol
8.2 TCP - Transfer Control Protocol

Ende-zu-Ende Protokolle: Einordnung + Aufgabenstellung



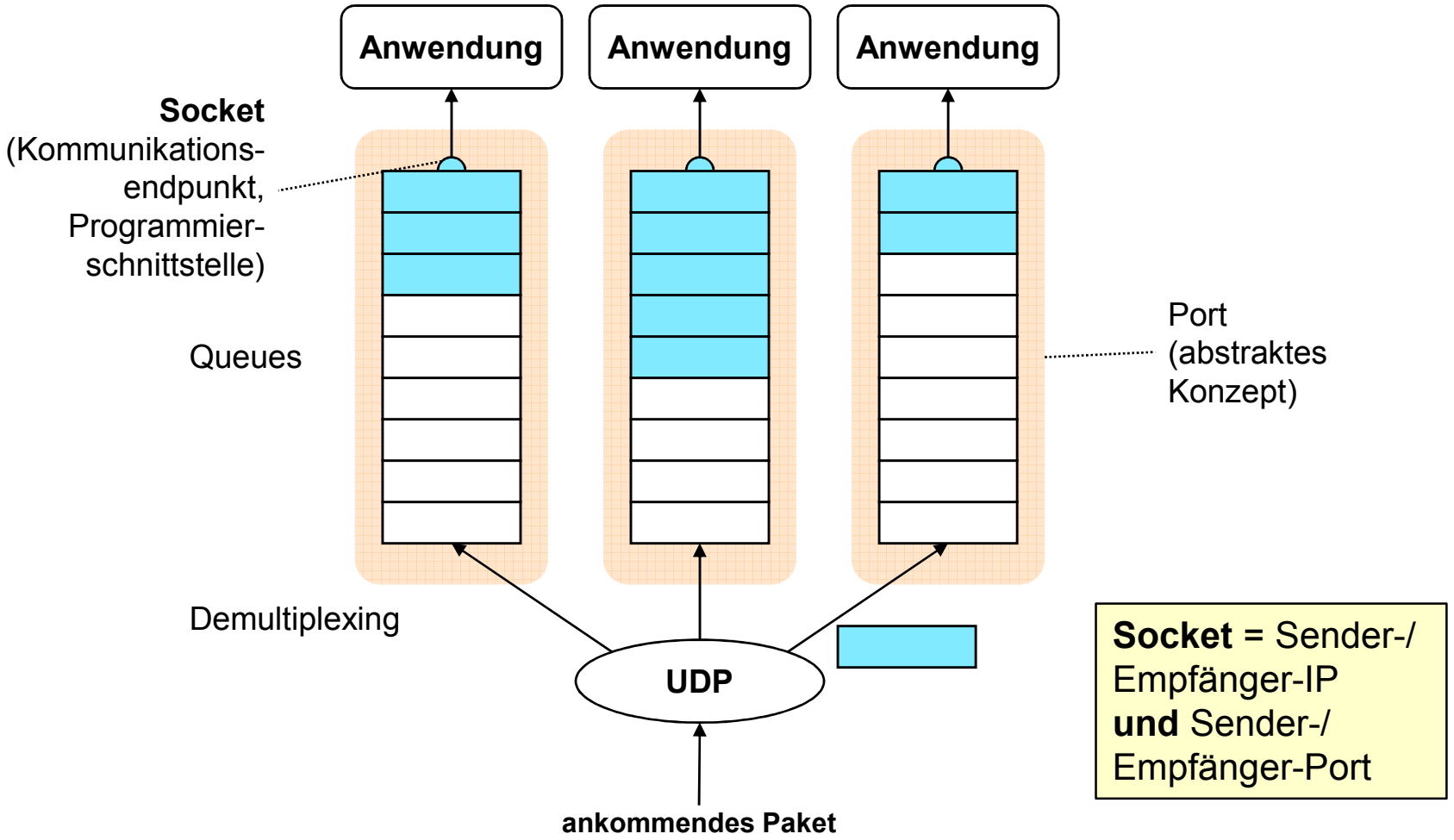


8.1 Demultiplexdienst UDP (User Datagram Protocol)

Lernziele:

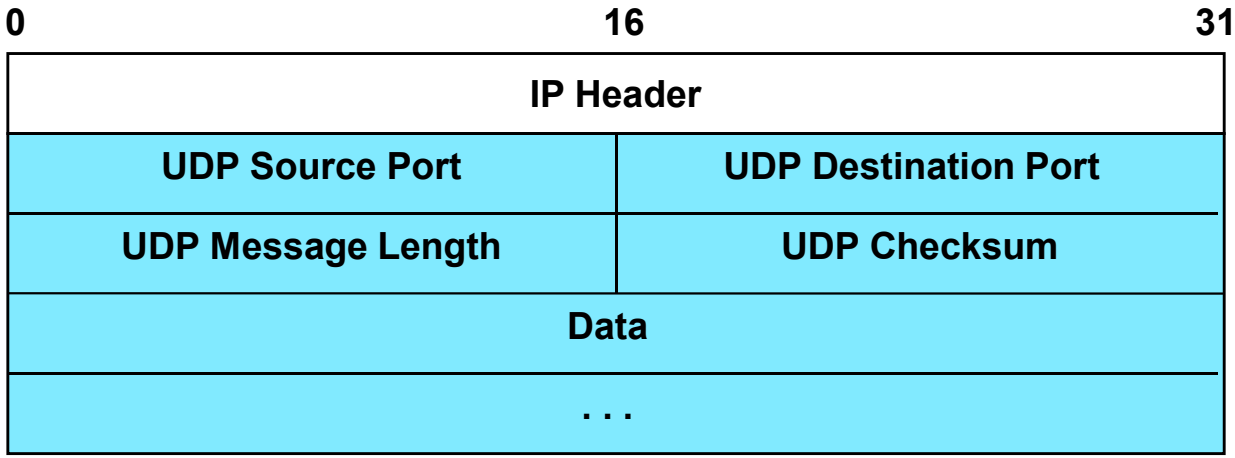
Nach Durcharbeiten dieses Teilkapitels sollen Sie das Grundkonzept von UDP darstellen können.

UDP Demultiplexing (Ports und Sockets)



Socket = Sender-/ Empfänger-IP **und** Sender-/ Empfänger-Port

UDP Header

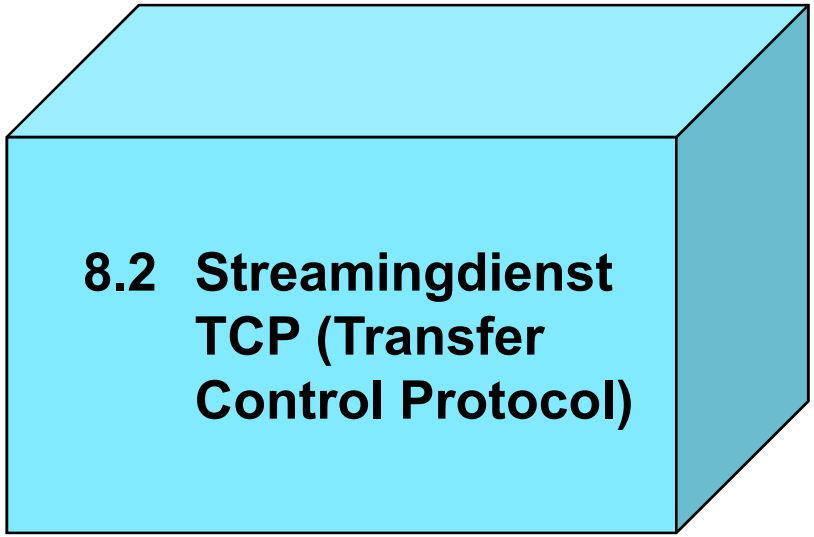


Source Port: Enthält den Port des versendenden Hosts;
optionale Angabe, d.h. wenn angegeben, ist dies der Port, an den eine evtl. Antwort gehen soll

Destination Port: Zielport, wohin das Datagramm geschickt wird

Message Length: Gibt die Gesamtlänge des Datagramms an

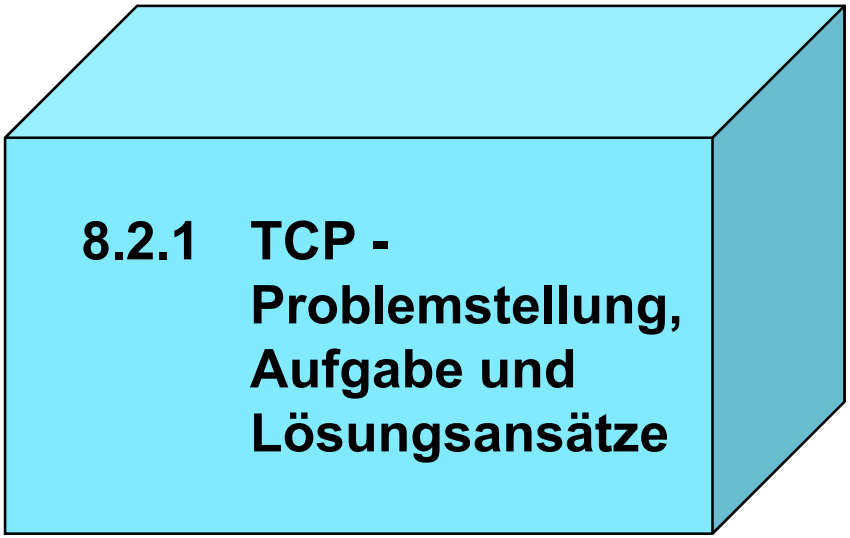
Checksum: Prüfsumme über UDP-Header sowie IP-Pseudoheader (Protokollnummer+IP-Adressen)
Stellt sicher, dass der richtige Port und der richtige Host erreicht worden sind.



8.2 Streamingdienst TCP (Transfer Control Protocol)

Lernziele:

Nach Durcharbeiten dieses Teilkapitels sollen Sie Grundelemente, Grundkonzepte und grundlegende Funktionen von TCP darstellen können.



Problemstellung: Zuverlässige Transportleistung über unzuverlässige Netzwerkschicht

Eigenschaften der IP-Netzwerkschicht:

- Aufteilung der Information in Pakete
- Maximale Paketgrößen
- Paketverluste bei der Übertragung
- Empfang fehlerhafter Pakete
- Mehrfaches Übertragen von Paketen
- Umordnung der Paketreihenfolge
- Wechselnde Paketübertragungszeiten
- Speicherfähigkeit des Netzes

IP ist (nur) paketorientiertes Best-Effort-Netz

Was soll TCP leisten?

TCP stellt einen zuverlässigen bidirektionalen Bytestrom zur Verfügung

Für TCP gibt es viel zu tun

Überblick TCP

wichtige Folie!

Grundeigenschaften:

- **Punkt-zu-Punkt-Verbindung**
- **Streaming-Schnittstelle**
  - **Byteorientiert**
- **Zuverlässige, verbindungsorientierte Übertragung**
  - **Drei Phasen: Verbindungsaufbau, Nutzdatenübertragung, Verbindungsabbau**
  - **Zuverlässiger Verbindungsaufbau, Initialisierung des Empfänger/Sender-Kontexts (→ *Three-Way-Handshake*)**
  - **Zuverlässige Nutzdatenübertragung (→ *Retransmission*)**
  - **zuverlässiger Verbindungsabbau**
- **Eine Vollduplex-Verbindung**
  - **bidirektionaler Datenfluss in derselben Verbindung**
  - **Optimale maximale Segmentgröße (512-1500 Bit) beachten**
- **Flusskontrolle und Überlastkontrolle**

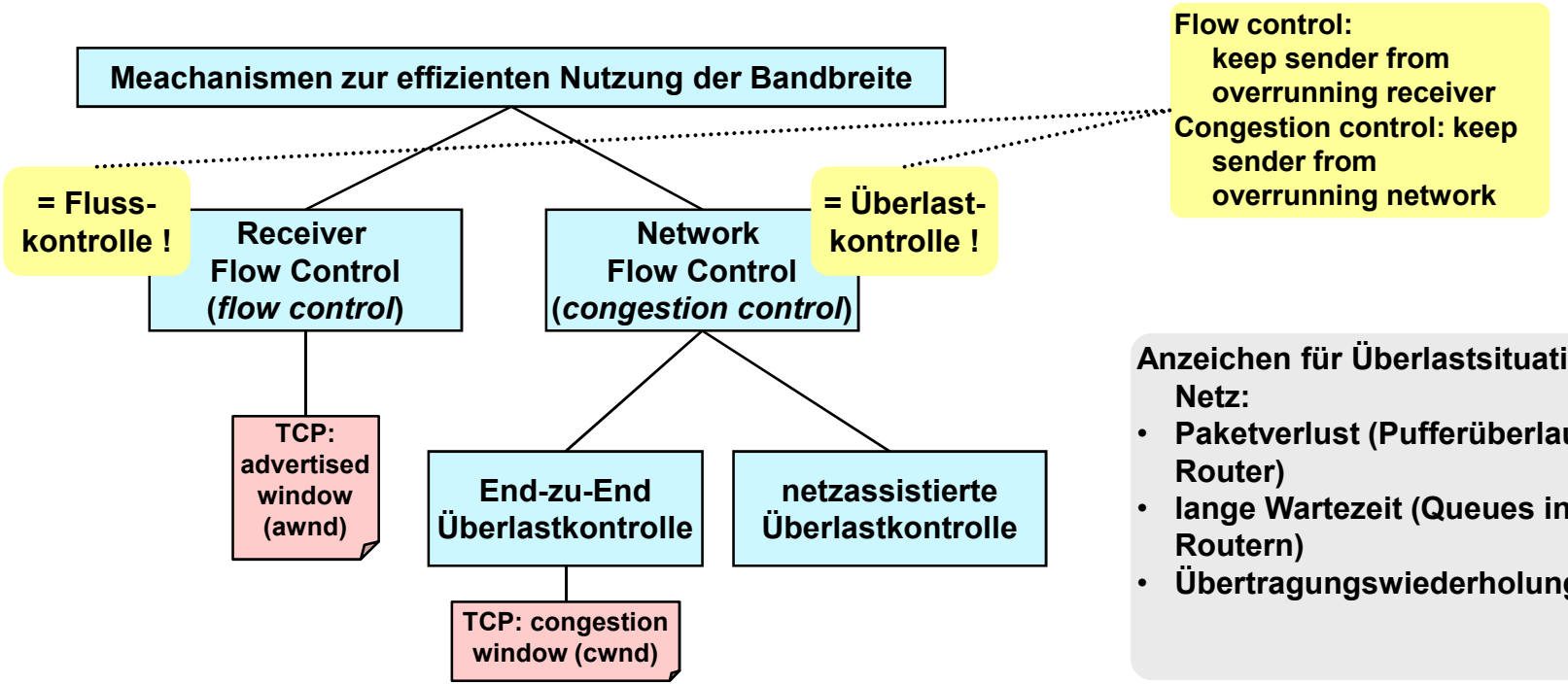
Begriffsklärung: Fluss- und Überlastkontrolle

Ziel der Kontrollmechanismen:

Effiziente Nutzung der Bandbreite
eines verbindungslosen Netzes durch Sender und Empfänger .

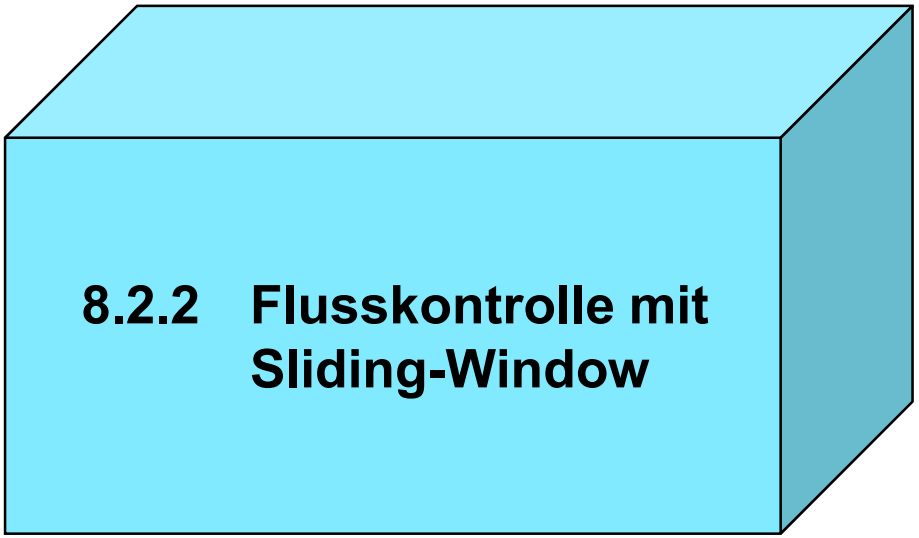
Fenster-Flusskontrolle:

Kontrollmechanismen, die auf Fenstergrößen (=Puffergrößen) beruhen.

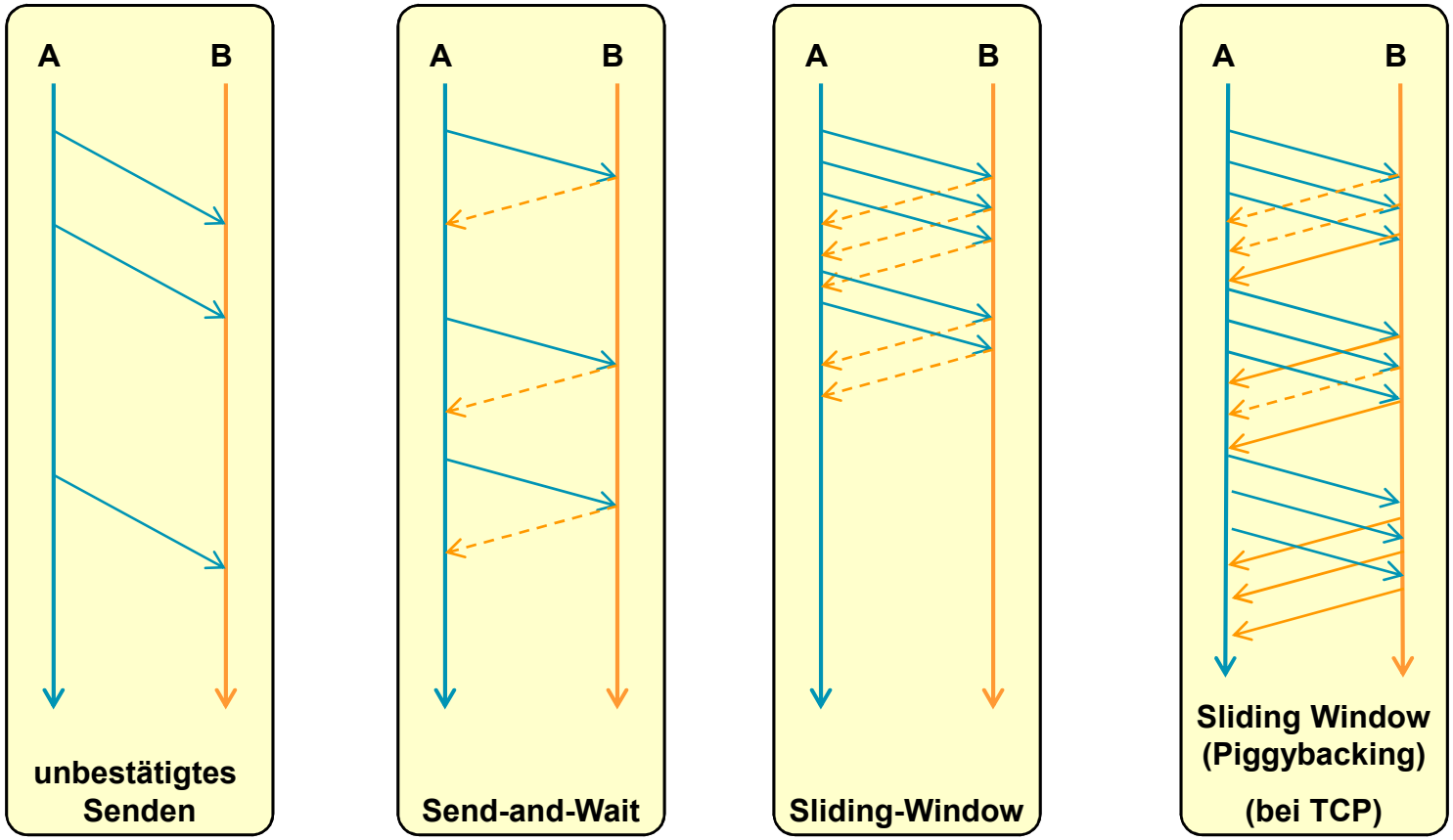


Anzeichen für Überlastsituationen im Netz:

- **Paketverlust (Pufferüberlauf Router)**
- **lange Wartezeit (Queues in den Routern)**
- **Übertragungswiederholungen**



Prinzip des Sliding-Window: Zuverlässigkeit + Effizienz



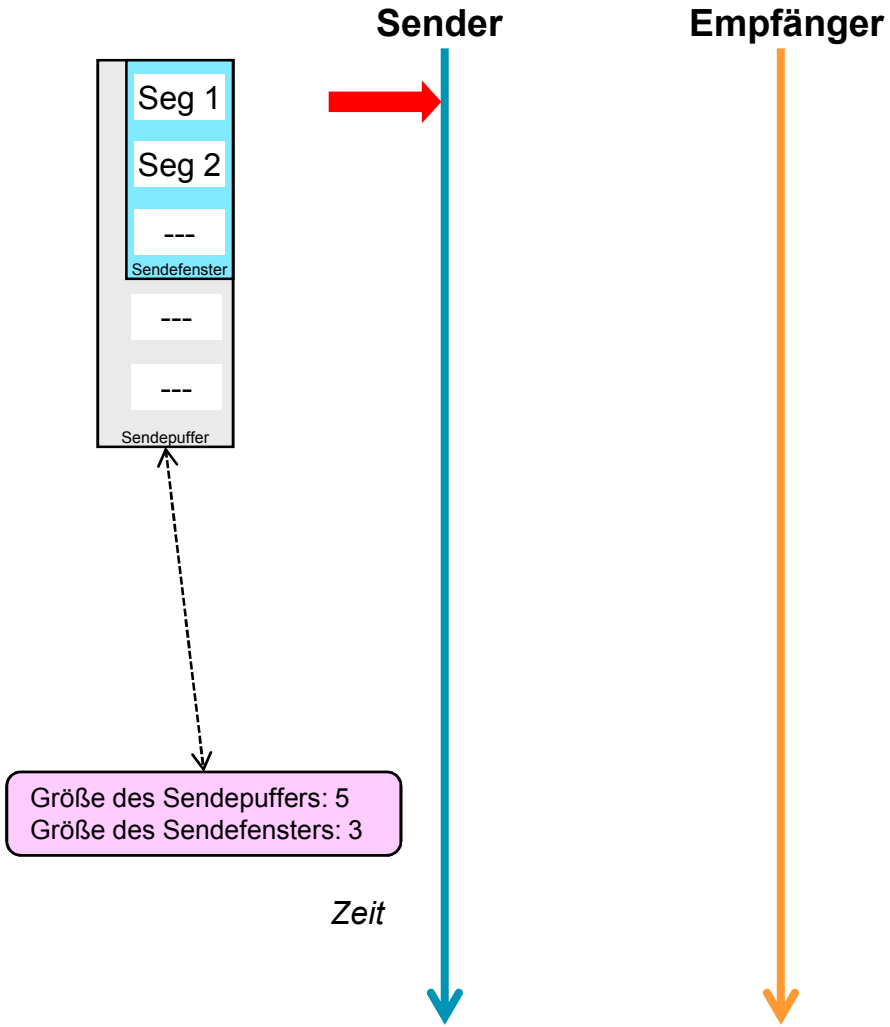
unzuverlässig effizient

zuverlässig ineffizient

zuverlässig effizient

zuverlässig effizient

Sliding-Window - detaillierter

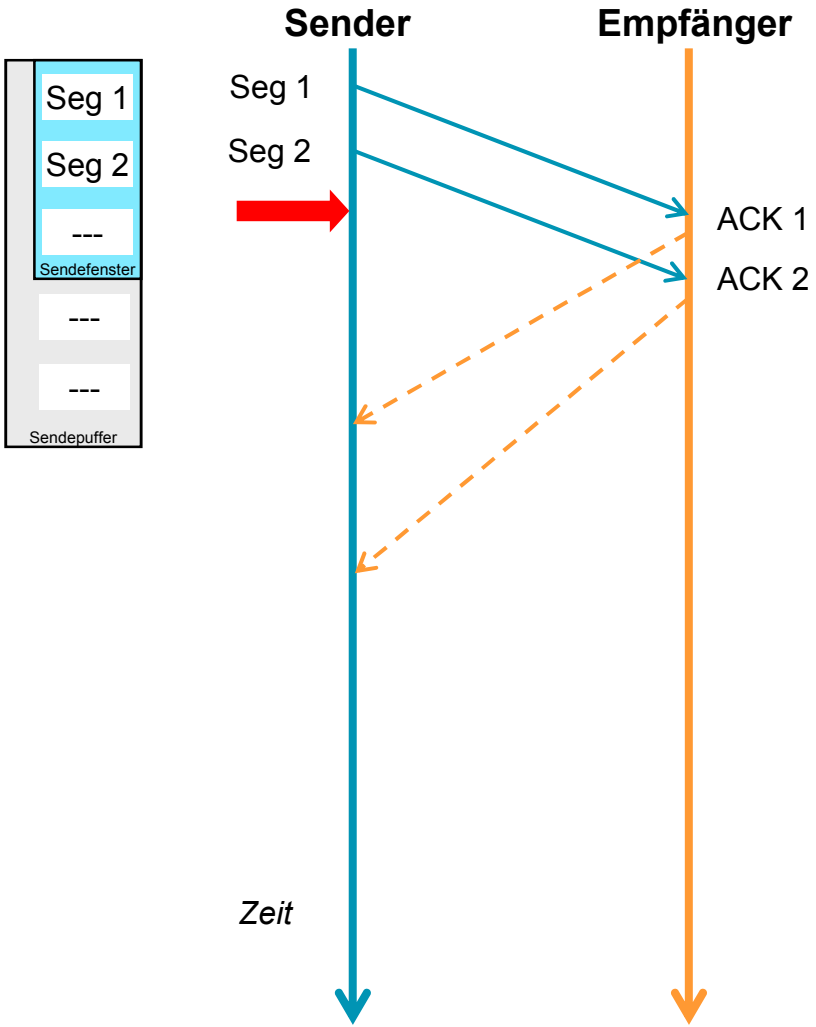


- Anwendung produziert 2 Segmente
- Diese Segmente liegen im Sendepuffer und im Sendefenster

Was ist der Unterschied zwischen Sendepuffer und Sendefenster?

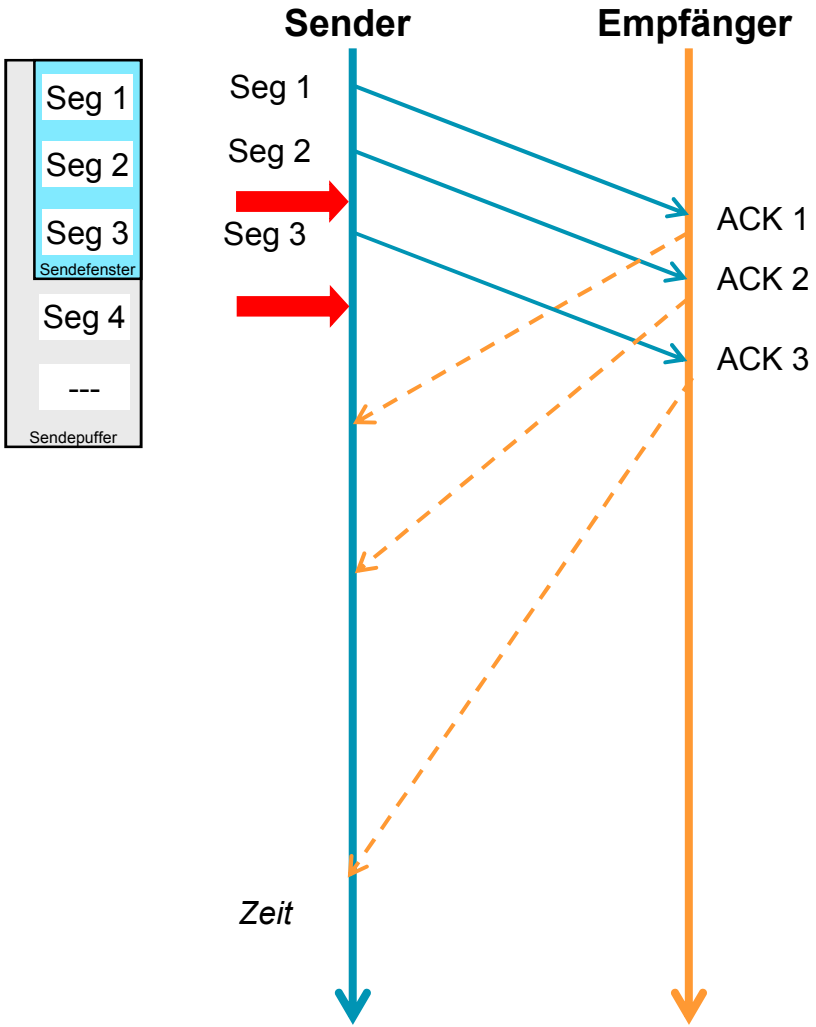
Was ist die Aufgabe des Sendefensters?

Sliding-Window - detaillierter



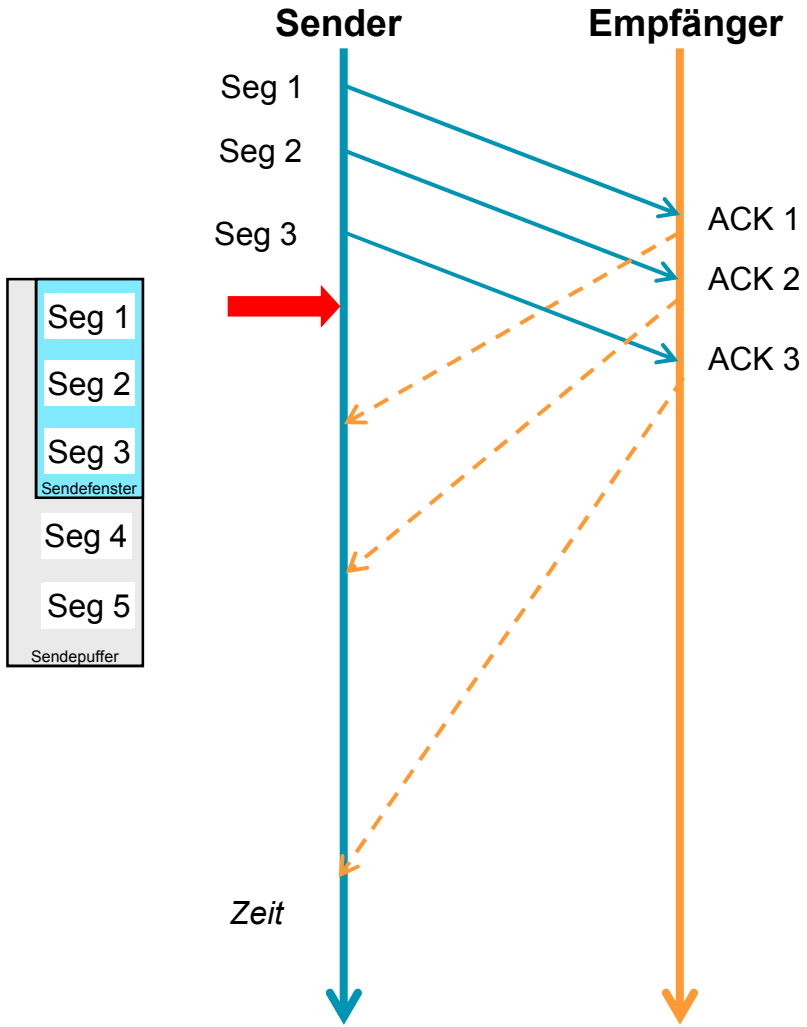
- **Was passiert jetzt??**
- **Beide Segmente können gesendet werden**
- **Auswirkungen auf den Sendepuffer und das Sendefenster?**

Sliding-Window - detaillierter



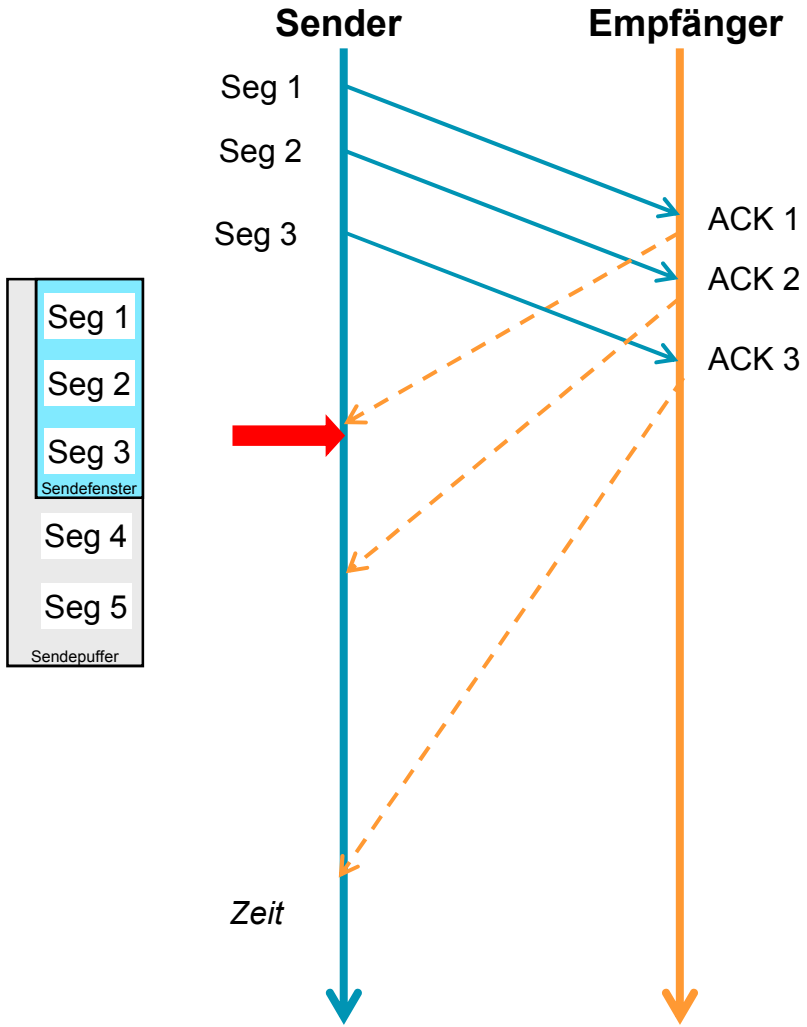
- **Anwendung produziert zwei weitere Segmente**
- **Was macht der Sender?**

Sliding-Window - detaillierter



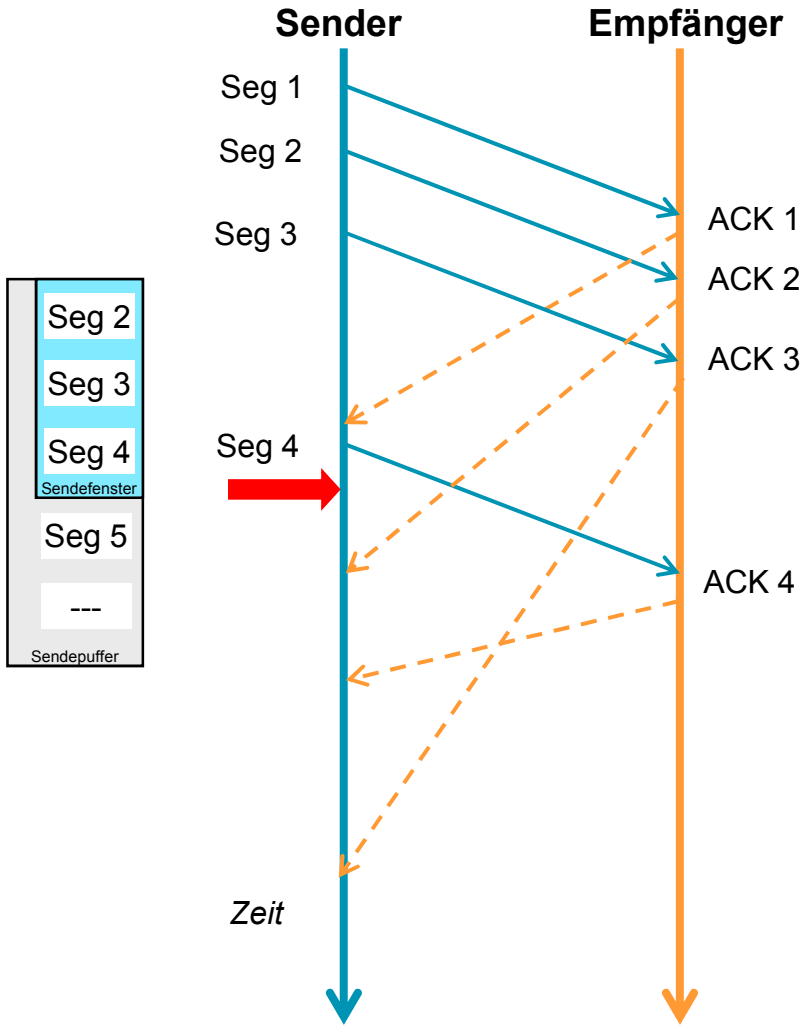
- **Anwendung produziert ein weiteres Segment**
- **Was passiert?**

Sliding-Window - detaillierter



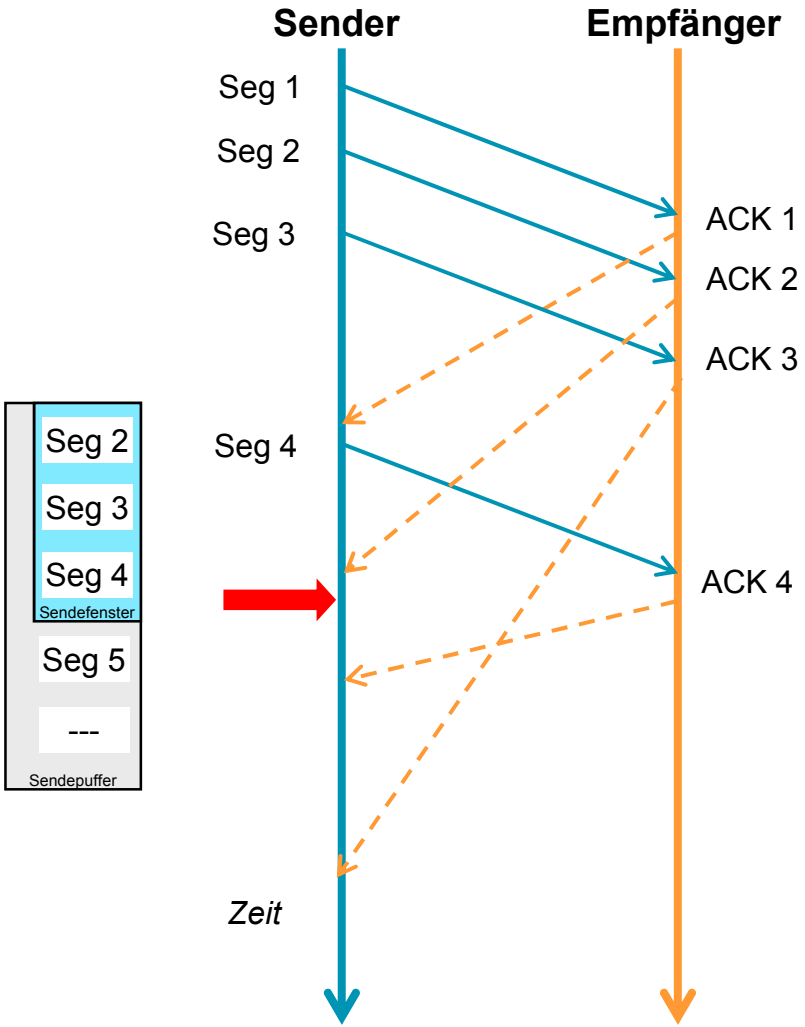
- **ACK 1 kommt nun beim Sender an**
- **Was passiert?**

Sliding-Window - detaillierter



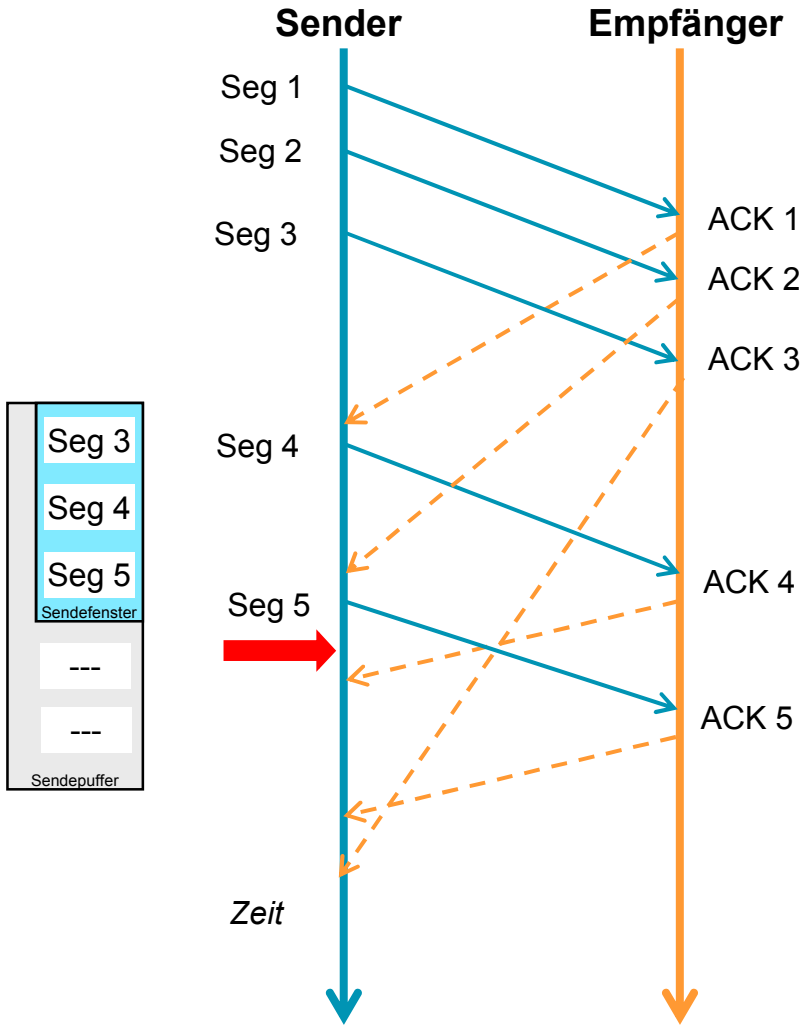
- **Schieben des Fensters**
- **Versenden von Segment 4**

Sliding-Window - detaillierter



- Nun kommt ACK 2
- Was passiert?

Sliding-Window - detaillierter



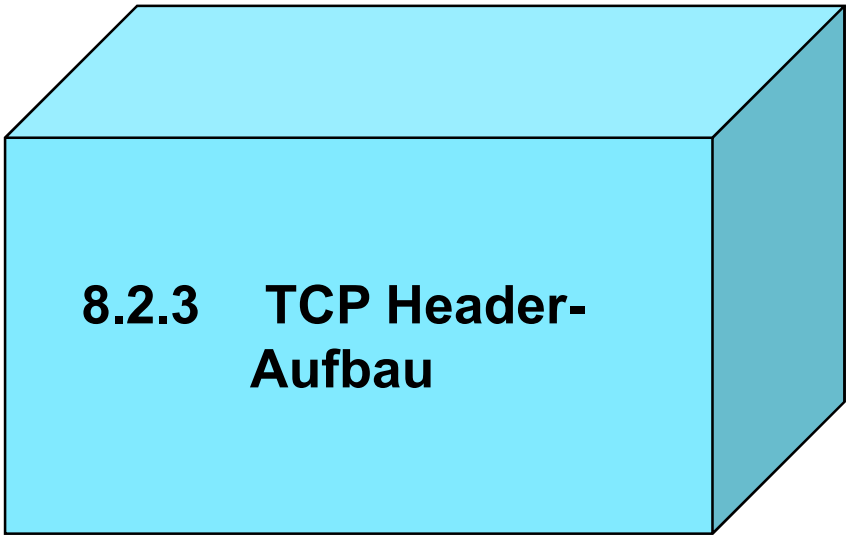
- **Schieben des Fensters**
- **Versenden von Seg 5**

Für Fortgeschrittene:

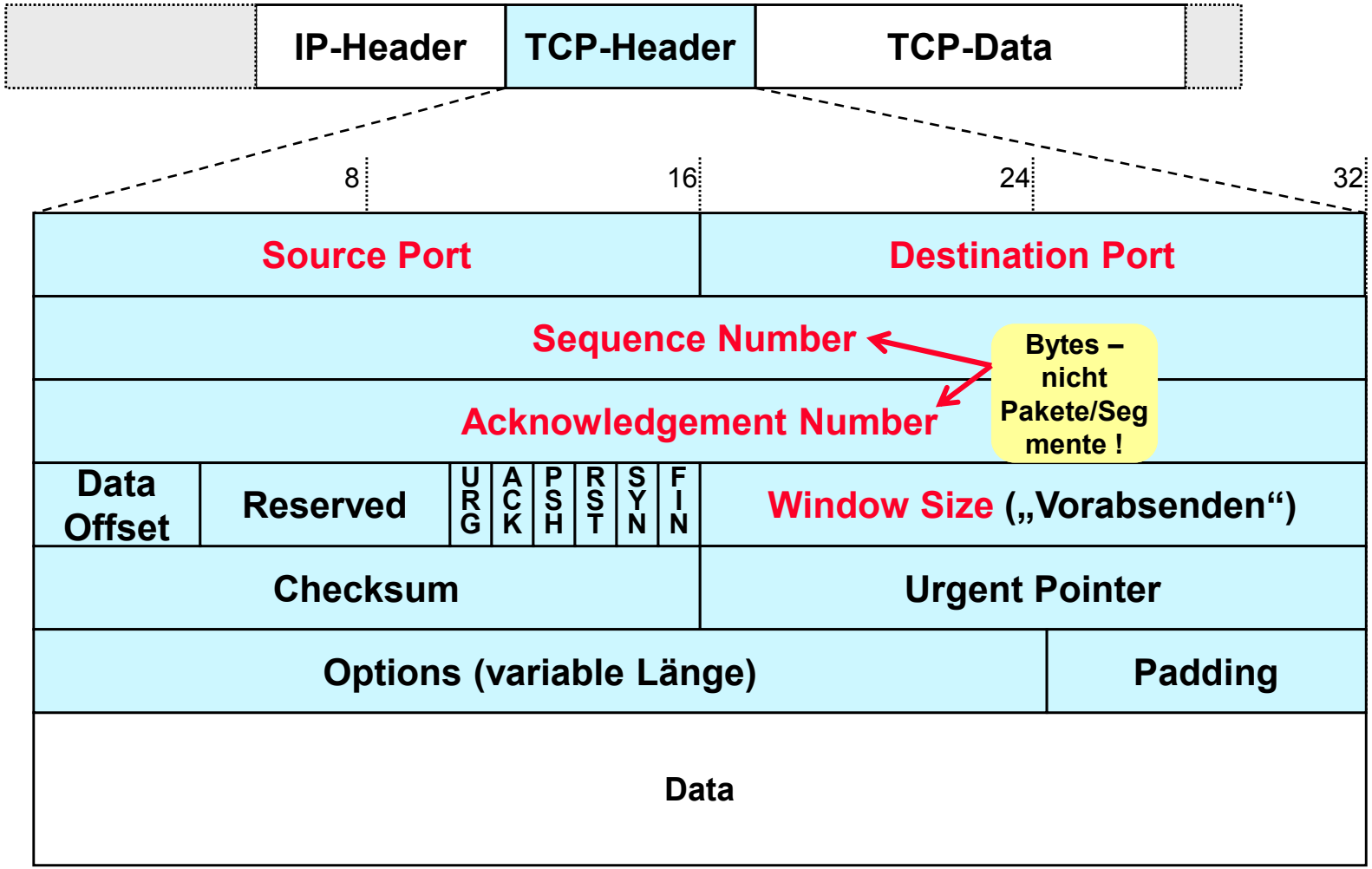
- **Was passiert, wenn ACK 4 und ACK 5 vor ACK 3 kommen?**

Frage an alle:

- **Wie groß soll das Sendefenster sein?**
- **Wer legt das Sendefenster fest?**



TCP-Header

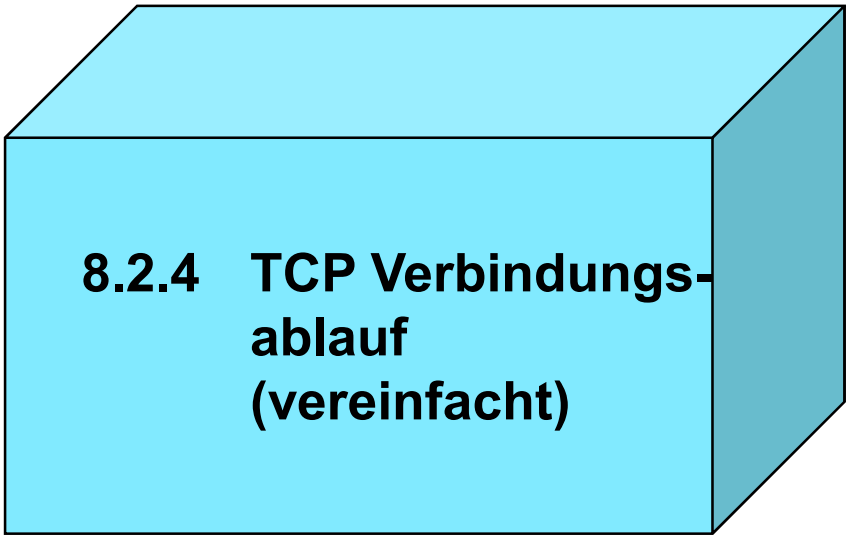

IP-Header
TCP-Header
TCP-Data
8
16
24
32
Source Port
Destination Port
Sequence Number
Acknowledgement Number
Bytes – nicht Pakete/Segmente !
Data Offset
Reserved
URG
ACK
PSH
RST
SYN
FIN
Window Size („Vorabsenden“)
Checksum
Urgent Pointer
Options (variable Länge)
Padding
Data

Aufbau des TCP-Headers

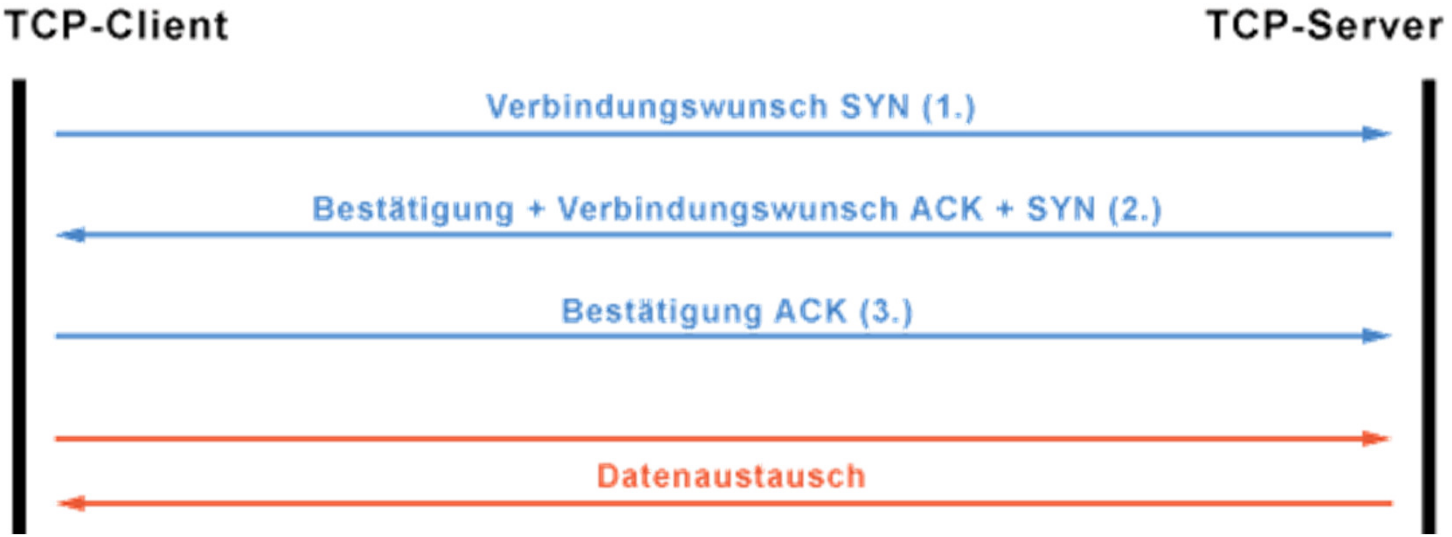
- **Source Port:** 16-Bit-Feld für lokalen TCP-User (Applikation)
- **Destination Port:** 16-Bit-Feld für remote TCP-User (Applikation)
- **Sequence number:** Position des aktuellen Blocks im laufenden Strom. Zeigt auf das erste neue Byte.
- **Acknowledgment Number:** Nächste Sequenznummer, die erwartet wird. Vorhergehende Nummern wurden korrekt empfangen!
- **Data Offset:** Länge des TCP-Headers in 32 Bit Blöcken
- **Flags:**
  - **URG** Urgent-Pointer-Flag (für Vorrang-Daten)
  - **ACK** Acknowledgement-Flag (Bestätigung)
  - **PSH** Push-Flag (direkte Weiterleitung an Anwendung)
  - **RST** Reset-Flag (Beenden der Verbindung aufgrund eines Fehlers)
  - **SYN** Synchronization-Flag (Synchronisationsvorgang)
  - **FIN** Final-Flag (ordentlicher Verbindungsabbau)

Aufbau des TCP-Haeders (Fortsetzung)

- **Window Size:** die noch verfügbare Puffergröße auf Empfangsseite, wichtige Info für den Sender, dient der Flusssteuerung
- **Urgent Pointer:** Zeiger auf Ende von dringlichen Daten
- **Ckecksum:** Prüfsumme über Header, Daten und einen Pseudoheader zum Schutz gegen fehlgeleitete Segmente
- **Options:** MSS (Maximum Segment Size) bei Verbindungsaufbau
- **Padding:** Fülldaten für 32 Bit-Grenze

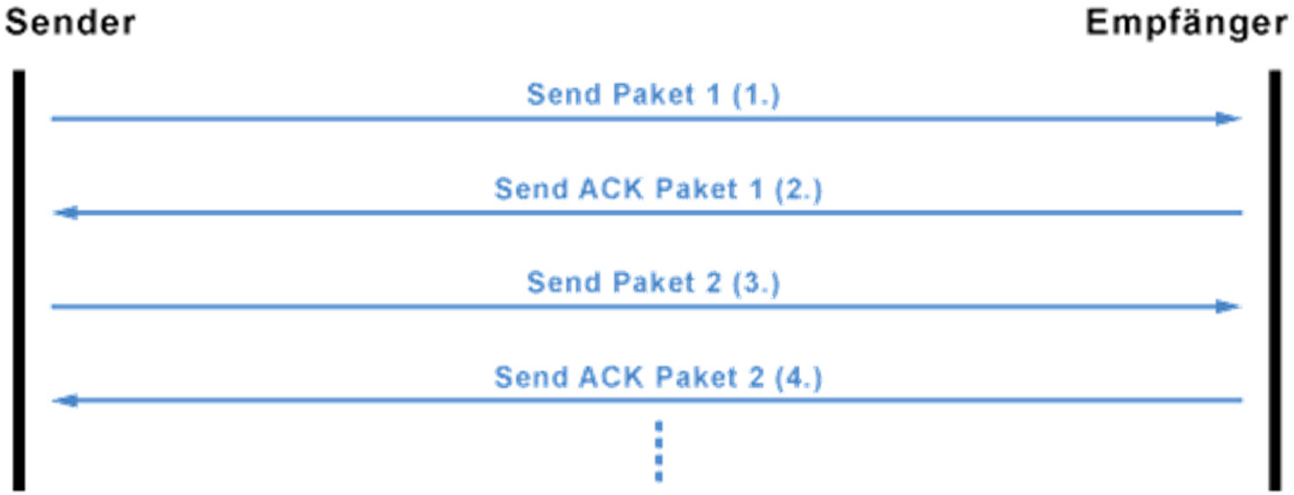


TCP-Verbindungsaufbau (stark vereinfacht)

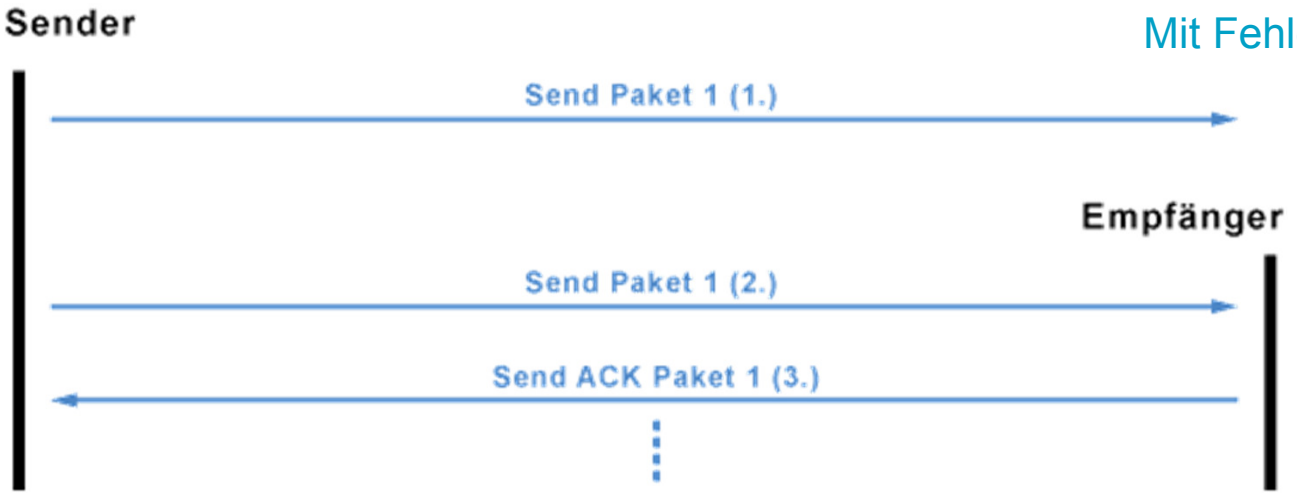


Quelle: https://www.elektronik-kompendium.de/sites/net/2009211.htm

TCP-Datenaustausch (stark vereinfacht)



Mit Fehlerbehandlung:



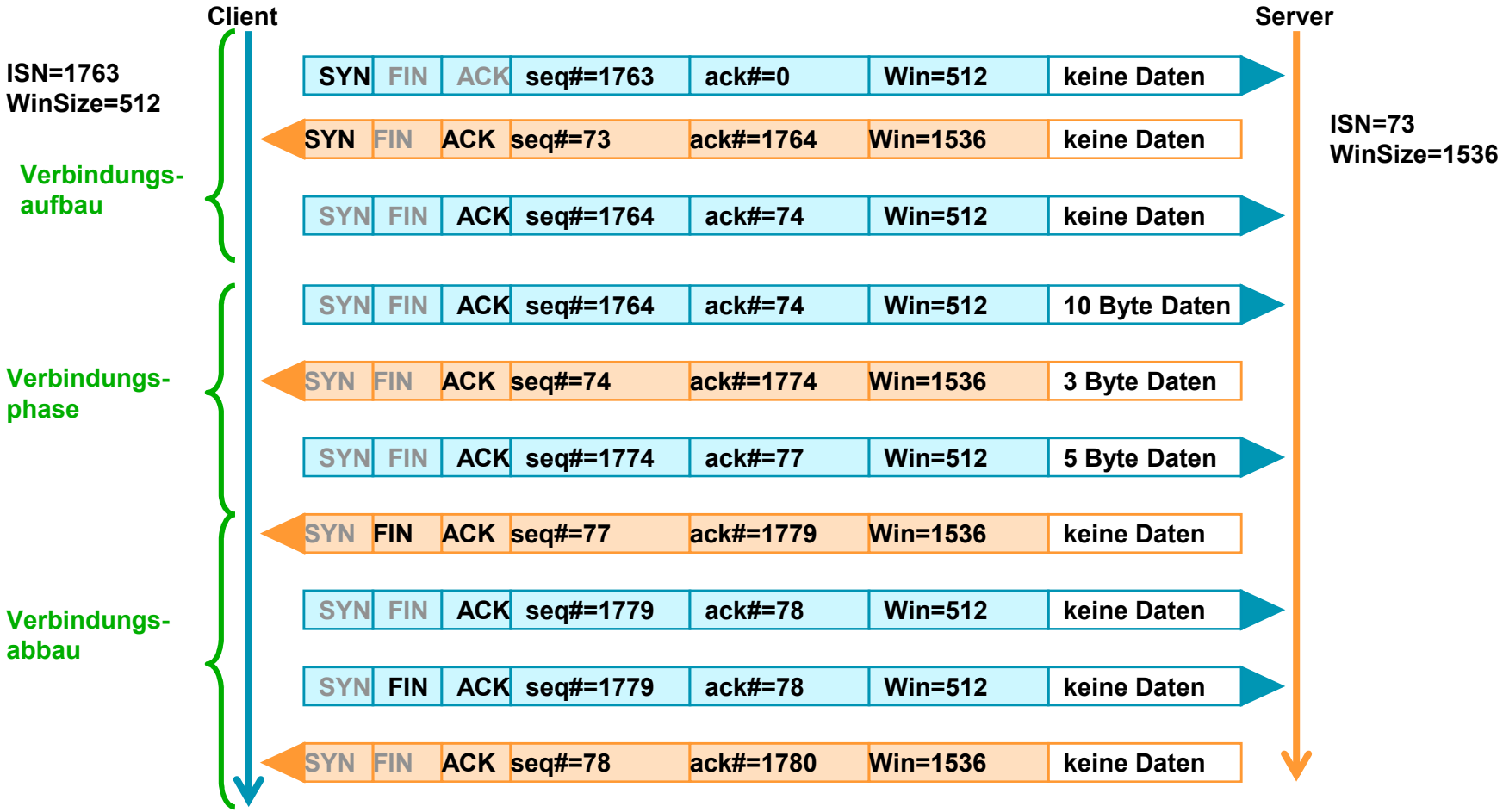
Quelle:
https://www.elektronik-kompendium.de/sites/net/2009211.htm

TCP-Verbindungsabbau (stark vereinfacht)

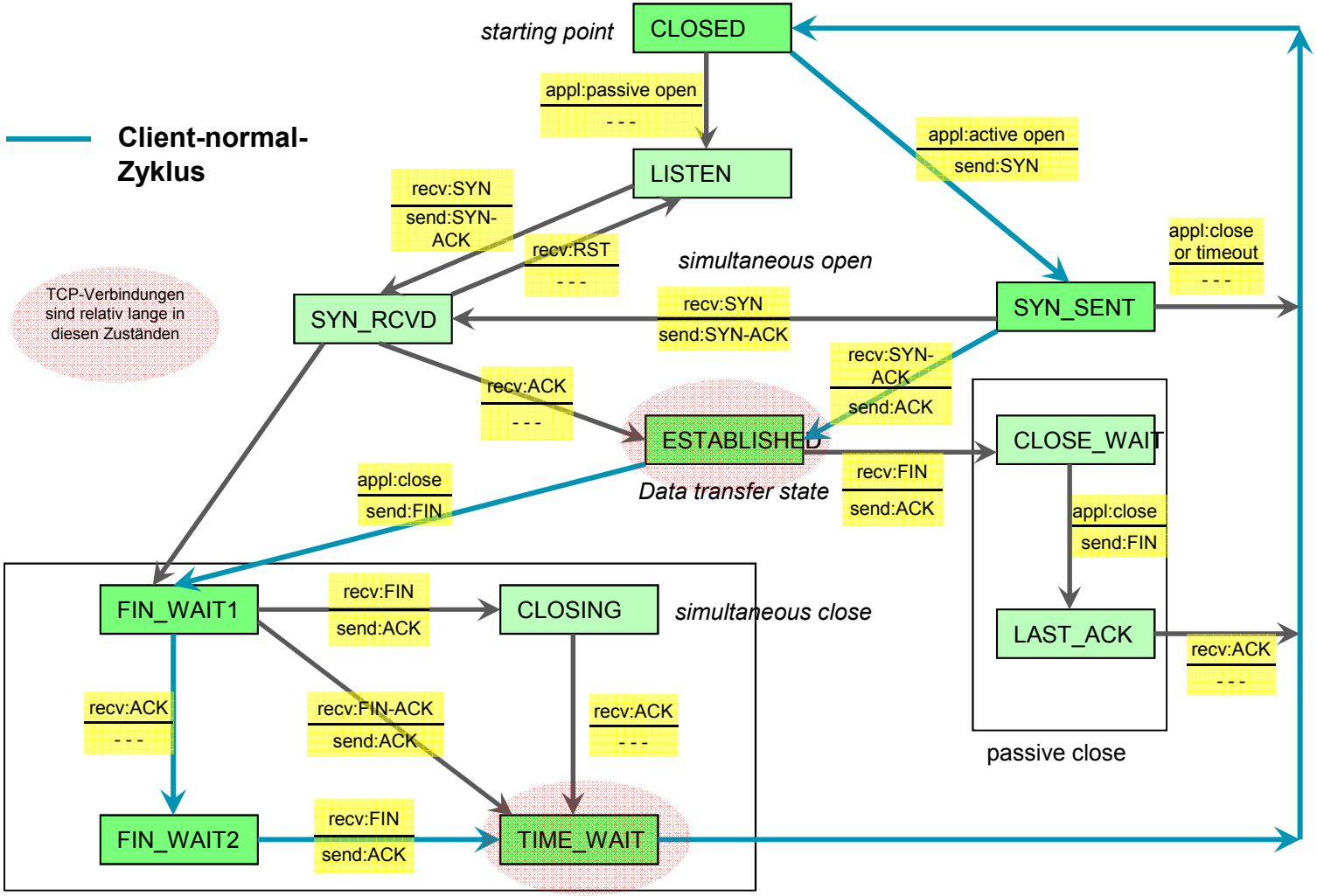
Quelle: https://www.elektronik-kompendium.de/sites/net/2009211.htm

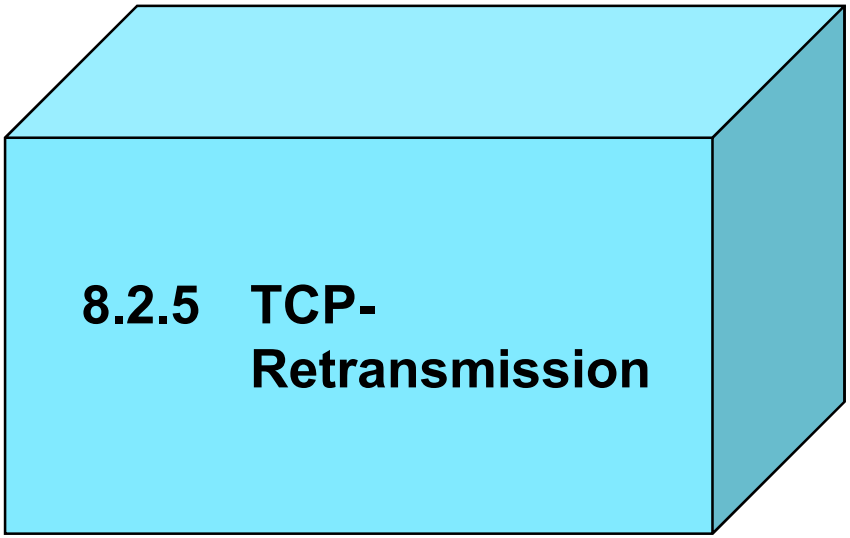
TCP-Verbindungsablauf (Normalfall, vereinfacht)

ISN = Initial Sequence Number

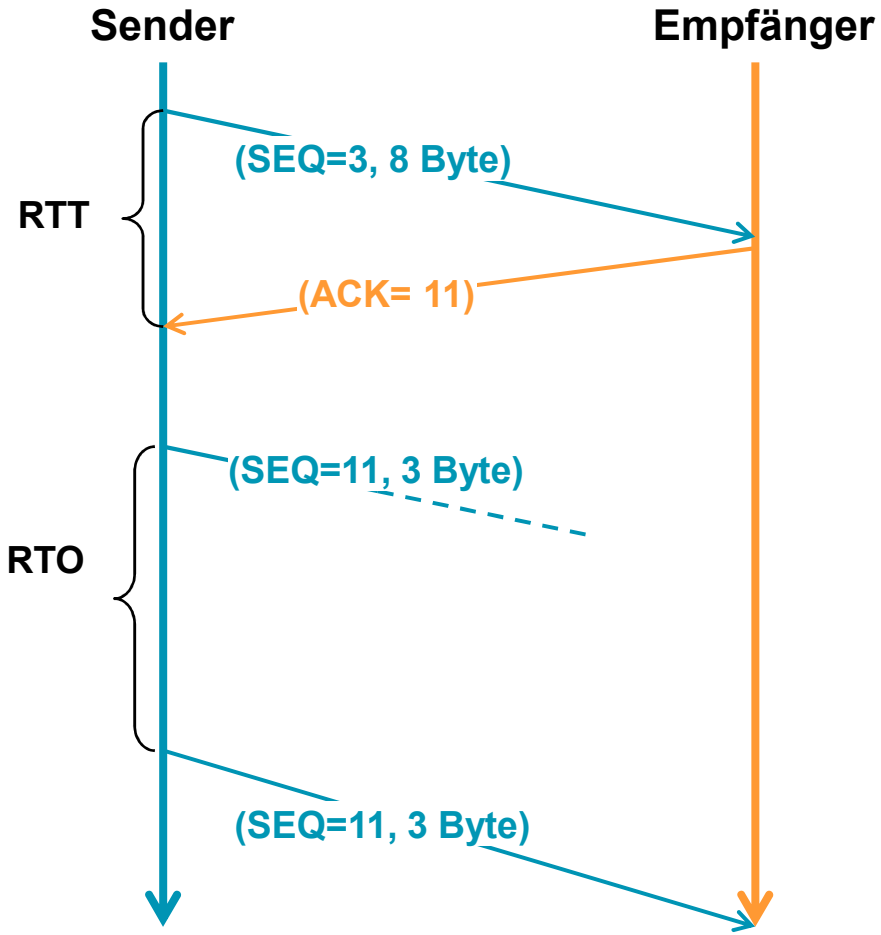


TCP Verbindungsmanagement (mit blauem Client-Life-Cycle)





TCP Retransmission - Aufgabenstellung



- **Faustregel:**
  RTO etwas größer als erwartete RTT
- **RTO zu groß:**
  schlechter Durchsatz
- **RTO zu klein:**
  unnötige Übertragungswiederholung

Also: Für effizientes TCP gute Schätzung der RTT notwendig

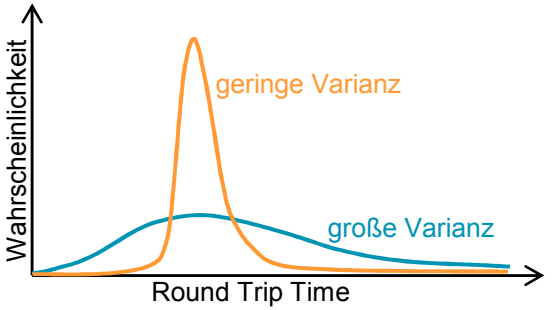
Fragen:

- **Anfangswert für RTO ?**
- **Wie soll zeitliche Änderung der RTT berücksichtigt werden ?**
- **Wie soll Timer nach Ablauf des RTOs gesetzt werden ?**

RTO-Timer nach Jacobson (1988)

Schlüsselbeobachtung:

- **Bei großer Last (starke Schwankung der RTT) ist größere Sicherheitszugabe für RTO notwendig**
- **Bei hoher RTT und geringer Varianz der RTT ist eine geringe Sicherheitszugabe notwendig**



Lösungsidee:

- **Mache den Wert des RTOs abhängig von der Standard-Abweichung**

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2}$$

zu kompliziert !

Problem:

- **Berechnung der Standardabweichung ist recht komplex**
  –> nehme „mittlere“ Abweichung

RTO-Timer nach RFC 6298 (Juni 2011)

Anfangswerte (nach erstem gesendeten Paket, initial RTO = 1s) :

$$SRTT = RTT, \quad RTTVAR = 0{,}5 * RTT, \quad RTO = RTT + 4 * RTTVAR$$

Schritt 1: Kontinuierliche Anpassung von RTTVAR und SRTT :

$$RTTVAR_{new} = (1-\beta) * RTTVAR_{old} + \beta * |SRTT_{old} - RTT_{akt}|$$

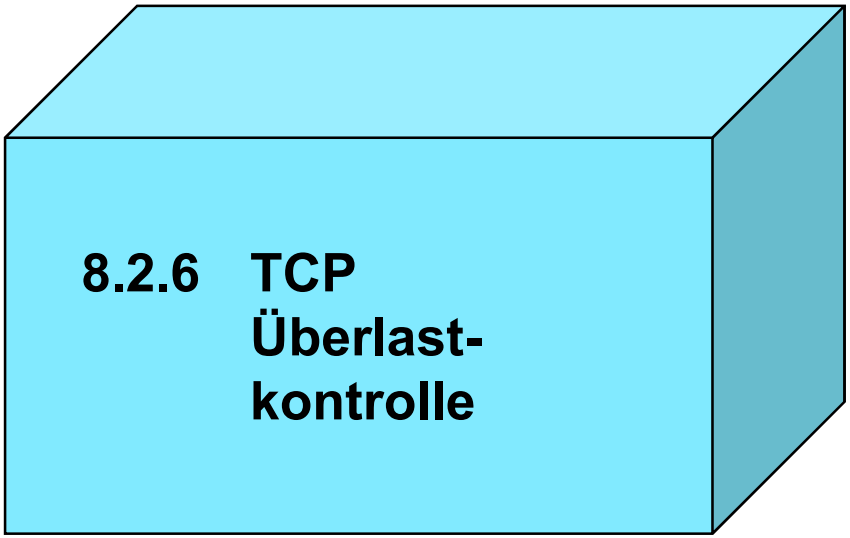
$$SRTT_{new} = (1-\alpha) * SRTT_{old} + \alpha * RTT_{akt}$$

SRTT	Smoothed Round Trip Time
α	Smoothing Faktor (typisch: 1/8)
β	Varianzfaktor (typisch 1/4)

Schritt 2: Limits beachten:

$$RTO = \min(\text{o-grenz}, \max(\text{u-grenz}, RTT + 4 * RTTVAR))$$

u-grenz	minimales RTO (z.B. 1 sec)
o-grenz	maximales RTO (z.B. 10 sec)



Phasen der Überlastkontrolle

Start der TCP-Übertragung

- **Slow Start (Aufbau eines Gleichgewichts (= Equilibrium))**
- **→ genauere Erklärung: siehe nächste Folie**

Erhalt des Gleichgewichts

- **Optimale Einstellung des RTOs (Jacobson/Karel Algorithmus)**
- **Maximierung der Auslastung (–> Additive Increase, Überlastfenster (= spezielle Fenstergröße für Überlastkontrolle) wächst linear)**

Reaktion auf Überlastungsanzeichen

- **Stabilisierung der Situation (–> Multiplicative Decrease, Überlastfenster wird halbiert)**
- **Sicherstellung einer kontinuierlichen, unterbrechungsfreien Datenflusses**

Slow Start

Ziel von Slow Start:

- Rasches Heranführen der TCP-Verbindung an das Gleichgewicht (ausgehend von kleinem Wert => Verdopplung des Sendefenster nach jedem ACK)

Strategie:

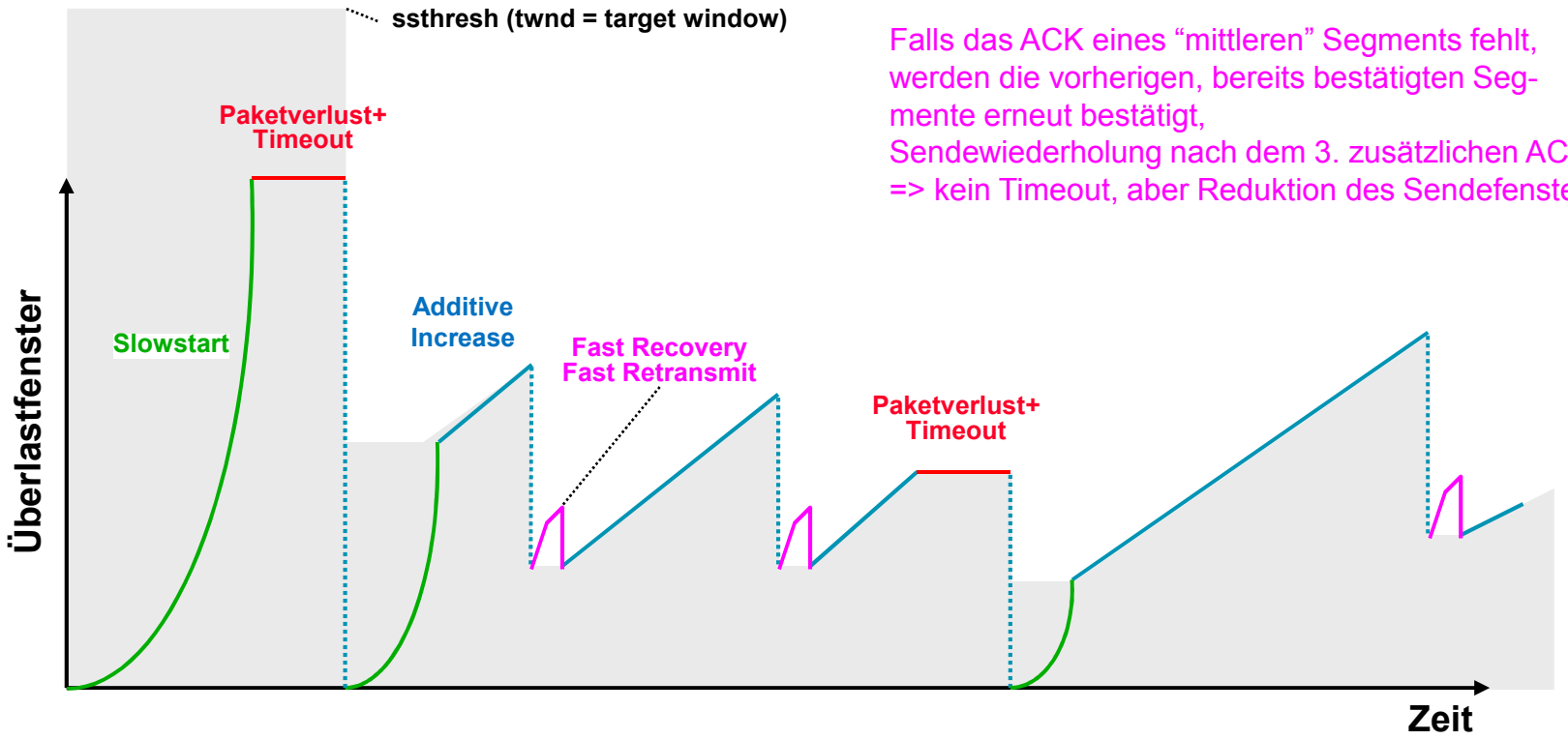
- Progressives Testen der Belastbarkeit der Strecke Sender-Empfänger (bis in die Überlast hinein)
- Paketverluste (–> Retransmission Time Out) werden als Kennzeichen für eine Überlastsituation gewertet.
  (Dies ist kritisch für mobile TCP –> niedrige Übertragungsrate)
- Beruhigung des Netzes nach dem Eintreten einer Überlastsituation (Leeren der Router-Queues am Engpass)

Einsatz:

- Beim Start einer Verbindung
- nach Ablauf des Retransmission-Timers
- nach langer passiver TCP-Phase

TCP Sägezahnverhalten

Startwert des Überlastfensters oft **MSS**
(= Maximum Segment Size mit 1460 Byte
≠ MTU = Max. Transmission Unit)



Fast Recovery, Fast Retransmit:

Falls das ACK eines “mittleren” Segments fehlt, werden die vorherigen, bereits bestätigten Segmente erneut bestätigt,
Sendewiederholung nach dem 3. zusätzlichen ACK
=> kein Timeout, aber Reduktion des Sendefensters