

# Rechnernetze & Verteilte Systeme

Ludwig-Maximilians-Universität München  
Sommersemester 2016

Prof. Dr. D. Kranzlmüller



# Organisatorisches

**Achtung:** Keine Tutorien in KW 25 (Pfingsten)

Das aktuelle Übungsblatt geht über zwei Wochen!

# Kapitel 4: Transportschicht

(Engl.: Transport layer)



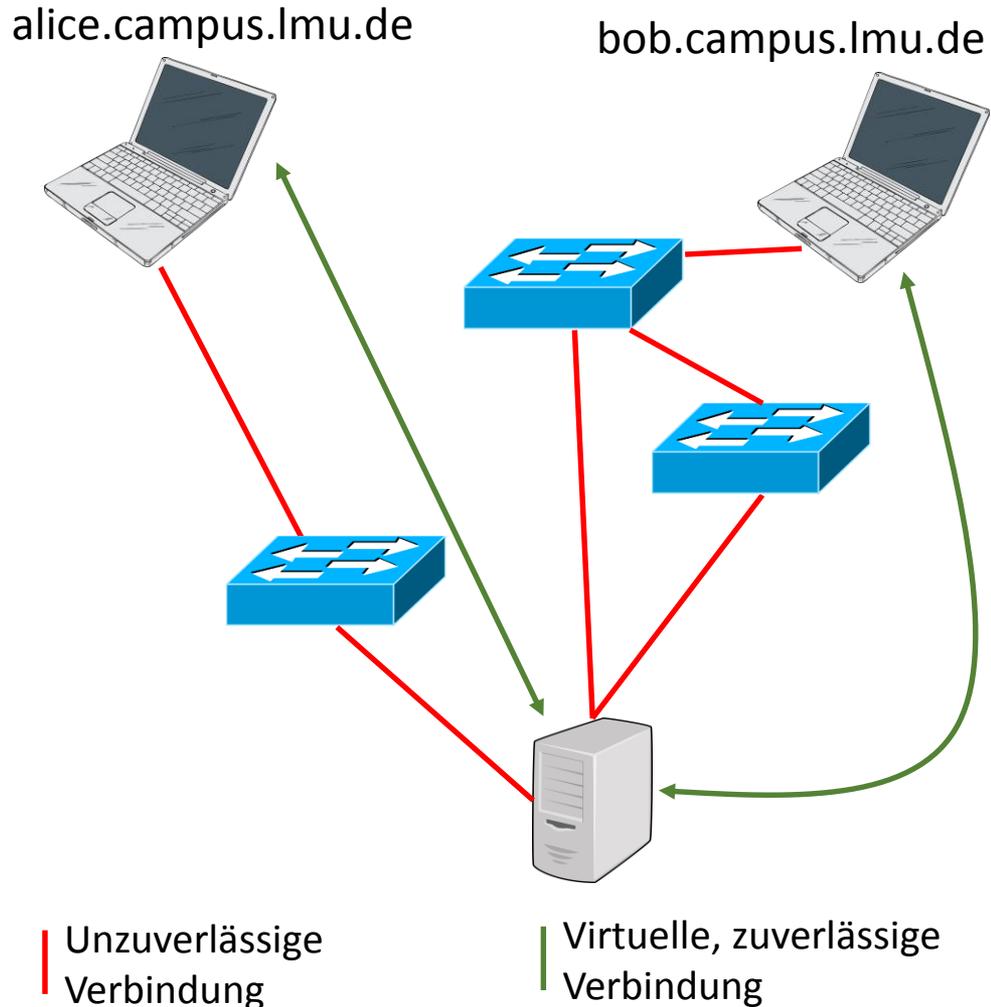
# Chat-Beispiel: Unsicherer Kanal

Bisher sind wir davon ausgegangen, dass die Nachrichten der Teilnehmer:

- in der richtigen Reihenfolge
- unverfälscht
- vollständig

übermittelt werden können.

Die Transportschicht kann diese zuverlässige, virtuelle Verbindung zwischen den Teilnehmern über unzuverlässige Kanäle bereitstellen.



# Inhalt von Kapitel 4

1. Ziele und Rahmenbedingungen des Transportdienstes
2. Techniken zur Realisierung verbindungsorientierter Ansätze
3. Flussteuerung und Staukontrolle
4. Ports und Sockets
5. Transmission Control Protocol (TCP)
6. User Datagram Protocol (UDP)

# Einordnung von Kapitel 4

- Kapitel 1: Querschnittsthema „Namen und Adressen“
  - inkl. Dienst der Anwendungsschicht - Domain Name System (DNS)
- Dienste der Anwendungsschicht benötigen
  - **zuverlässige** Übertragung von Nachrichten
  - über **unzuverlässige** Transitnetze
- Aufgabe der Transportschicht:
  - Bereitstellung der gewünschten Zuverlässigkeit unter Verwendung der Dienste der Vermittlungsschicht (Kapitel 4)

# Kapitel 4.1

## Ziele und Rahmenbedingungen des Transportdienstes

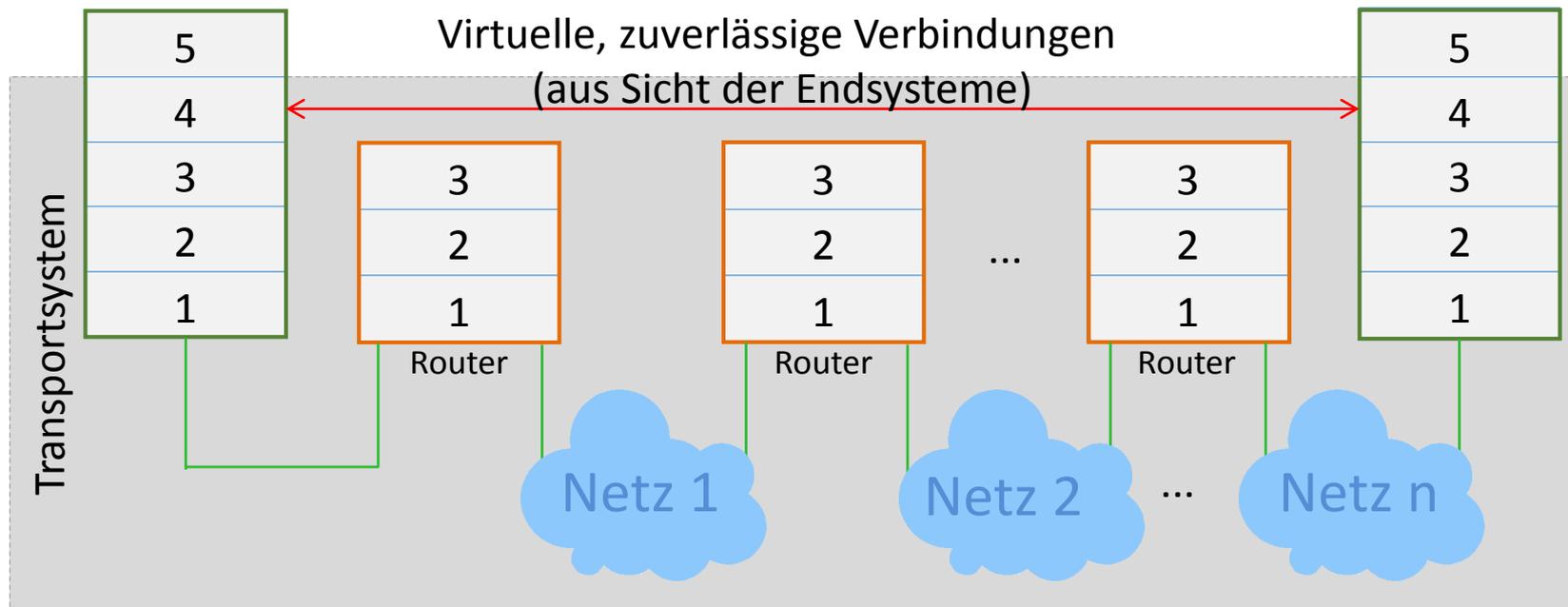
# Ende-zu-Ende Argument (Saltzer et al., 1984)

- Grundproblem:
  - Aus Sicht der Nutzer von Endsystemen: Transitsysteme allgemein außerhalb der eigenen Kontrolle und oft unzuverlässig (vgl. Kapitel 1.8: Fehlertypen)
- Ende-zu-Ende Argument:
  - um mit der Unzuverlässigkeit der Transitsysteme umzugehen → akzeptieren und umfangreiche Fehlerbehandlung in den Endsystemen.

# Ende-zu-Ende Betrachtung als Grafik (Wiederholung)

Endsystem A

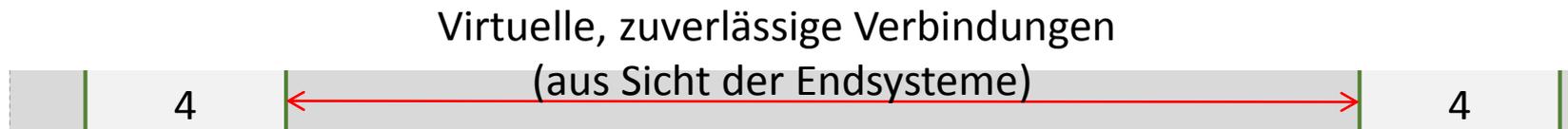
Endsystem B



# Ende-zu-Ende Betrachtung als Grafik (Wiederholung)

Endsystem A

Endsystem B



# Vermittlungsschicht als Rahmenbedingung (1/2)

- Transportschicht bietet Anwendungssystem ihre Dienste *unter Verwendung der Dienste der Vermittlungsschicht* an
- Zum Verständnis: Vorschau auf Dienste der Vermittlungsschicht:
  - Ausführliche Behandlung im Kapitel 5
  - Protokolle der Vermittlungsschicht (z.B. IP) arbeiten i.d.R. nach dem *best effort* Prinzip - nach besten Möglichkeiten wird versucht, einzelne Pakete möglichst effizient an ihr Ziel zu bringen
  - Bei Überlastung oder wenn kein Weg zum Ziel gefunden wird, können Pakete verworfen werden

# Vermittlungsschicht als Rahmenbedingung (2/2)

- Zum Verständnis: Vorschau auf Dienste der Vermittlungsschicht:
  - Nicht vorhersehbare Verzögerungen können bei der Zustellung einzelner Pakete auftreten (z.B. weil ein einzelnes Paket einen anderen Weg nimmt)
  - Aufgrund der Gegebenheiten noch tiefer liegender Schichten gibt es allgemein eine maximale Paketgröße (MTU – engl.: Maximum Transmission Unit\*), die auf einem gegebenen Pfad übertragen werden kann.
- Zusammenfassung Vermittlungsschicht:
  - Verzögerungen kommen vor
  - Pakete können nicht beliebig groß sein
  - Die Fehlerarten (Verfälschung, Verlust, Duplikate, falsche Reihenfolge) aus dem Kapitel 1 kommen vor

\* Oft auch „Maximum Transfer Unit“

# Verbindungslos oder Verbindungsorientiert?



- Grundsätzlich: Realisierung des Transportdienstes:
- **Verbindungslos:**
  - Vergangenheit irrelevant
  - geringe Möglichkeiten zur Fehlererkennung/-korrektur
  - aber (aufgrund des geringen Verarbeitungsaufwands) → hohe **Performanz**
- **Verbindungsorientiert:**
  - Vergangenheit relevant (Kontext)
  - umfangreiche Fehlerbehandlung (**Zuverlässigkeit**)
  - (notwendigerweise) höherer Verarbeitungsaufwand

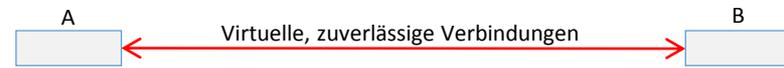
# Kapitel 4.2

## Techniken zur Realisierung verbindungsorientierter Ansätze

Sequenznummern, Quittungen, Fenstertechnik  
Drei-Wege-Handschlag

# Motivation/Einordnung

- In diesem Unterkapitel: *allgemeine Techniken zur Realisierung verbindungsorientierter Dienste*
  - kommen sowohl in verbindungsorientierten *Transportprotokollen (TCP)*
  - als auch in verbindungsorientierten Protokollen anderer Schichten zum Einsatz.
- Es handelt sich um schichtunabhängige Konzepte



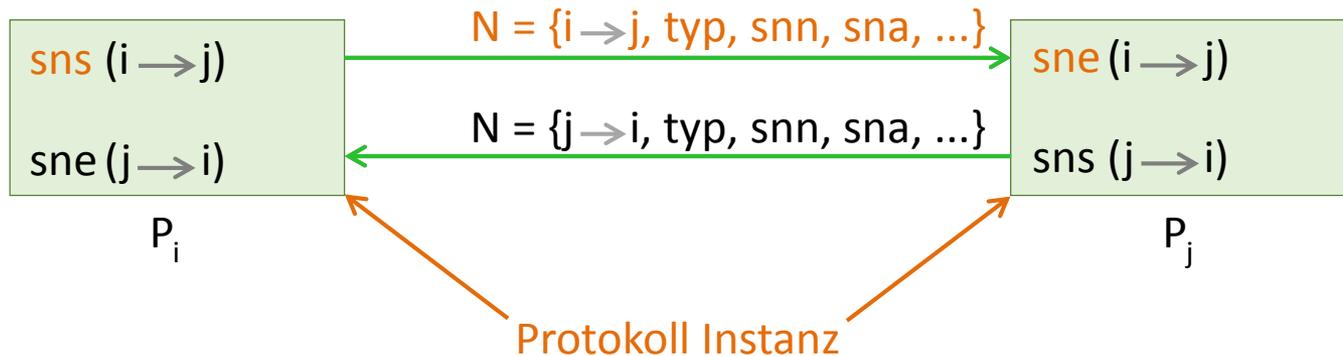
# Techniken zur Realisierung verbindungsorientierter Dienste

- Sequenznummer und Quittungen
  - Größe des Sequenznummernraumes
  - Maximale Lebensdauer von Nachrichten
  - Fenstertechnik
- Verbindungsaufbau/-abbau
- Flusssteuerung und Staukontrolle
  
- Ports und Sockets
- TCP / UDP

# Sequenznummern und Quittungen

- Verbindungsorientierter Protokolle werden oft mit Sequenznummern realisiert
- Wenn gesendete Nachrichten durchnummeriert werden, können verloren gegangene Nachrichten neu angefordert, Duplikate verworfen, und Nachrichten in falscher Reihenfolge richtig angeordnet werden.
- Wenn zusätzlich Quittungen (der Empfänger bestätigt empfangene Nachrichten) verwendet werden, kann außerdem die **Flusssteuerung** geregelt werden:
  - Der Sender schickt erst dann weitere Nachrichten, wenn ausreichend viele, bereits gesendete Nachrichten quittiert wurden.
  - So wird verhindert, dass ein langsamer Empfänger (oder auch ein langsames Transitsystem) von einem schnellen Sender überlastet wird.

# Sequenznummern Notation



$sns(i \rightarrow j)$ : Nummer der nächsten zu sendenden Nachricht

$sne(i \rightarrow j)$ : Nummer der nächsten erwarteten Nachricht

$N = \{i \rightarrow j, typ, snn, sna, \dots\}$  = Nachricht

$i \rightarrow j$ : Verbindungs- und Richtungskennung

$typ$ :  $\in \{NN, ACK, NAK, SYN, CLS, \dots\}$

mit NN normale Nachricht, ACK/NAK pos./neg. Quittung

SYN Synchronisations-/Aufbauwunsch, CLS engl.: Close/Abbauwunsch

$snn$ : Sequenznummer einer Nachricht

$sna$ : Sequenznummer der nächsten erwarteten Nachricht (=  $sne$ )

# Quittungsvarianten

- Quittungen können positiv oder negativ sein („Die PDU Nr. n ist angekommen/nicht angekommen“).
- Sammelquittung: Einzelne Quittung für mehrere aufeinander folgende PDUs („Die PDUs bis Nr. n sind angekommen“).
- Selektive Quittung: Nur eine Untermenge der Nachrichten wird quittiert.
- Quittungen werden oft in Verbindung mit Timern verwendet (erfolgt eine erwartete Quittung nicht innerhalb eines gewissen Zeitfensters, wird neu gesendet).

# Beispiel:

## Nachricht

The quick brown fox jumps over the lazy frog.

PDU = Protocol Data Unit

Sequenznummer (snn): 1 Ziffer pro PDU

UD =



1	h
2	e
3	u
4	q
5	u
6	i
⋮	



0,1,2,3,4,5,6 ✓

6 ✓

Quittung

# Größe des Sequenznummernraumes (1/2)

- Sequenznummernräume sind i.d.R. **endlich** (und haben meist eine feste Größe).
- Im Beispiel:
  - 10 unterschiedliche Sequenznummern (0,1,...,9).
  - Nachricht 11 verwendet eine Sequenznummer (=0), die bereits früher einmal verwendet wurde.
  - **Gefahr: verzögerte Nachricht** mit derselben Sequenznummer (=0) ist noch im Netz unterwegs
  - Würde die verzögerte Nachricht vor Nachricht 11 ankommen:
    - Empfänger interpretiert die falsche Nachricht als erwartete Nachricht 11 und
    - verwirft danach ankommende Nachricht 11 als Duplikat

# Größe des Sequenznummernraumes (2/2)

- *Wie lange kann eine Nachricht im Netz unterwegs sein?*
- **Gesucht:** Mechanismus, um die Dauer der maximalen Verzögerung einer Nachricht zu bestimmen
- Wähle Sequenznummernraum so, dass innerhalb der maximalen Verzögerung nie dieselbe Sequenznummer doppelt verwendet werden muss
- **Möglichkeit:** Sequenznummernraum  $>$  Senderate  $\times$  RTD (engl.: Round Trip Delay).
- Beispiel:
  - Sei RTD = 15s, Senderate 200/s
  - Benötigt: 12 Bits zur Codierung der Sequenznummern

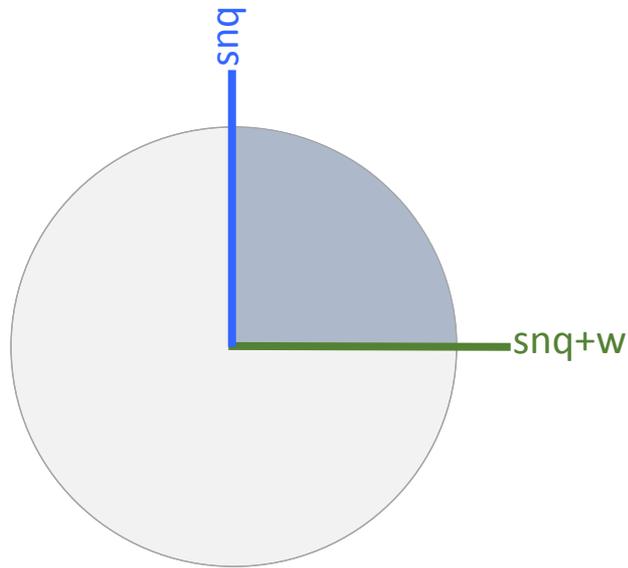
# Maximale Lebensdauer von Nachrichten

- Begrenzung der Lebensdauer verzögerter Nachrichten auf ein Maximum:
  - alle Nachrichten erhalten einen **Zeitstempel** und eine **Lebensdauer**
  - zu alte Nachrichten werden einfach verworfen
- Erfordert aufwändige Schätzung oder bestenfalls synchronisierte Uhren.
  
- Andere Möglichkeit: Zählung der Hops (“Sprünge“)

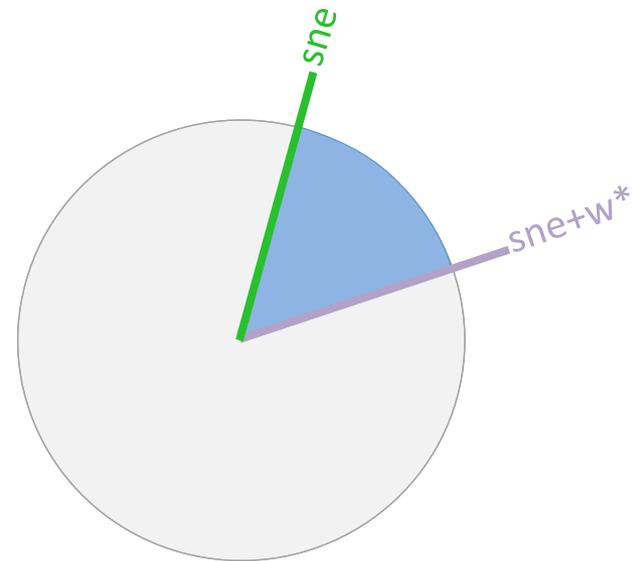
# Idee Fensterertechnik (Tanenbaum: „Schiebefensterprotokoll“)

- Festlegung eines Teilbereiches des Sequenznummernraumes als
  - Sendefenster:
    - Größe  $w$
    - Beginn Sendefenster: zuletzt (lückenlos) quittierte Sequenznummer  $snq$
    - Sender sendet nur Nachrichten mit Sequenznummern innerhalb des Sendefensters (Flusssteuerung/Staukontrolle)
  - Empfangsfenster
    - Größe  $w^*$
    - Beginn Empfangsfenster: nächste erwartete Sequenznummer ( $sne$ )
    - Nachrichten mit Sequenznummern außerhalb des Empfangsfensters werden vom Empfänger verworfen
- Fenster werden verschoben, wenn Nachrichten quittiert oder empfangen werden

# Fensterertechnik als Animation



—  $snq$   
—  $snq + w$



—  $sne$   
—  $snq + w^*$

- Lage Sendefenster und Empfangsfenster i.A. verschieden
- $snq \leq sne \leq snq + w$
- $w$  und  $w^*$  können differieren
- Fensterertechnik impliziert Flusststeuerung
- Sequenznummern i.A. schichtspezifisch eingesetzt

# Nachricht

The quick brown fox jumps over the lazy frog.

Quelle

0	T
1	h
2	e
3	└
0	q
1	u

⋮

Senke

⋮

0

3

snn

1

2

MNM  
TEAM

0

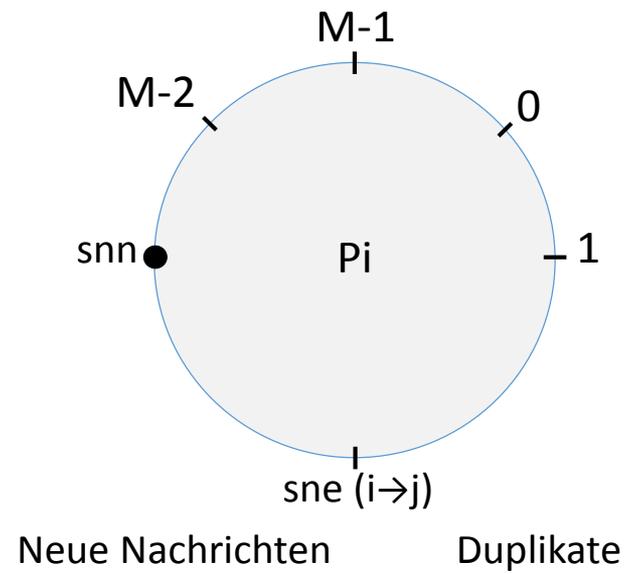
3

sne

1

2

- Problem: Ist  $s_{nn}$  neu oder Duplikat?
- Ansatz: Fenstertechnik



## Nachricht

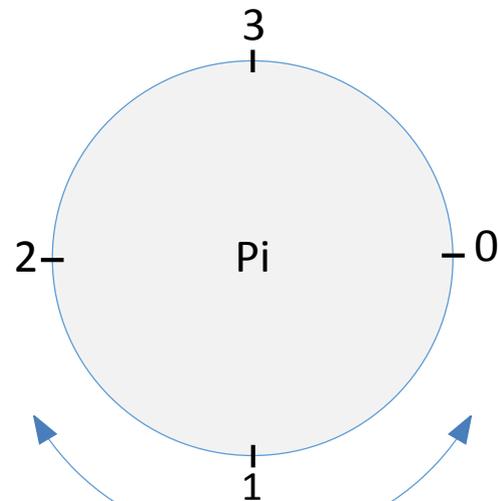
The quick brown fox jumps over the lazy frog.

Quelle

1	u
---	---

Senke

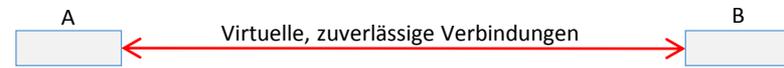
0	T
1	h
2	e
3	└
0	q



Neue Nachrichten | Duplikate

# Vorteile der Fenstertechnik

- Fenstertechnik ist sehr allgemein und kann flexibel an die Bedürfnisse konkreter Protokolle angepasst werden
- Fenstergröße = 1: „Stop-and-Wait“ Protokoll.
- Je nach Protokoll: dynamische Anpassung von Fenstergrößen, u.a. zur Flusssteuerung
- Beispiel: beim Sender oder Empfänger ist begrenzter Pufferspeicherplatz aufgebraucht



# Techniken zur Realisierung verbindungsorientierter Dienste

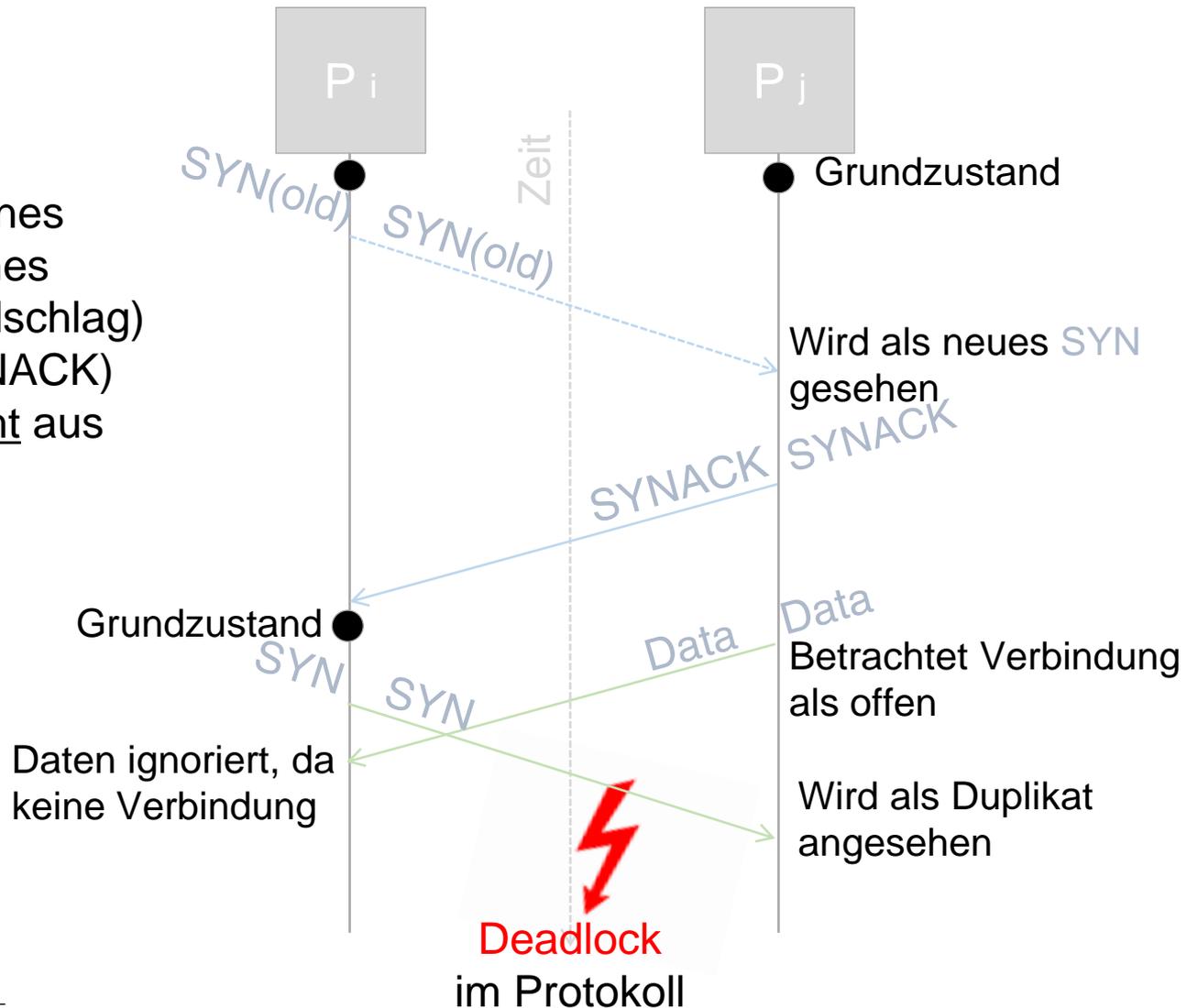
- Sequenznummer und Quittungen
  - Größe des Sequenznummernraumes
  - Maximale Lebensdauer von Nachrichten
  - Fenstertechnik
- Verbindungsaufbau/-abbau
- Flusssteuerung und Staukontrolle
  
- Ports und Sockets
- TCP / UDP

# Verbindungsaufbau

- **Sonderfall:** Synchronisation am Beginn einer Kommunikation
- *Wie wird einem Kommunikationspartner bei einer neuen Verbindung die erste verwendete Sequenznummer mitgeteilt?*
- **Problem:** auf Empfängerseite müssen verzögerte Duplikate alter Aufbauwünsche von aktuellen Aufbauwünschen unterschieden werden
- Lösungsansatz: Handschlag-Protokoll

# Zwei-Wege-Handschlag

Ein einfache Bestätigung eines Aufbauwunsches (2-Wege-Handschlag) (SYN und SYNACK) reicht u.U. nicht aus

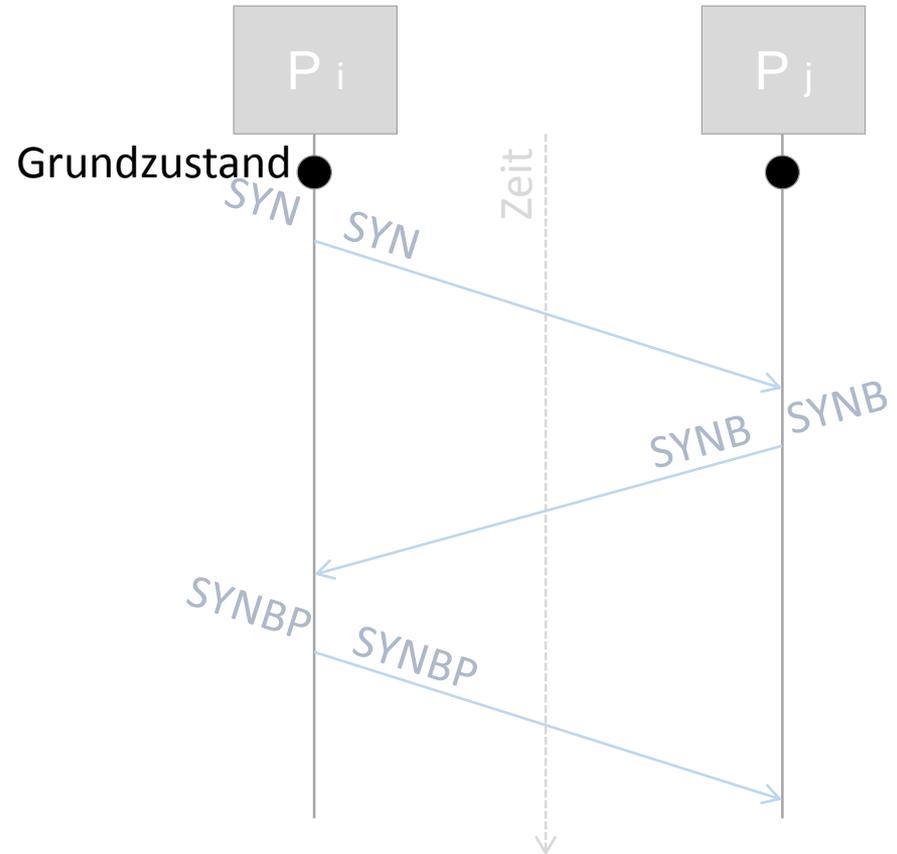


# Drei-Wege-Handschlag (1/2)

Verbesserter Ansatz:  
Drei-Wege-Handschlag

Neue PCI Typen

- **SYN**: Aufbauwunsch
- **SYNB**: Aufbauwunsch - Bestätigung
- **SYNBP/N**: pos./neg. Bestätigung der Bestätigung



# Drei-Wege-Handschlag (2/2)

## (Engl.: Three-Way-Handshake)

- In erster Nachricht (SYN) ist erste Sequenznummer für den Verkehr von  $P_i$  nach  $P_j$  enthalten
- Antwort (SYNB) bestätigt Sequenznummer und enthält erste Retoursequenznummer für den Verkehr von  $P_j$  nach  $P_i$ .
- Bestätigung (SYNBP) bestätigt Retoursequenznummer (und kann je nach Implementierung auch bereits erste Daten enthalten).
- Verzögerte Duplikate alter SYN oder SYNB Nachrichten können anhand der verwendeten Sequenznummern erkannt werden

# Verbindungsabbau

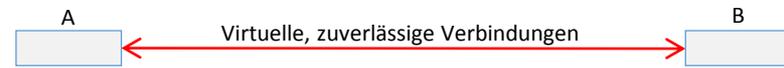
Neue PCI (Protocol Control Information) Typen:

- CLS ("close") Abbauwunsch
- CLSB Abbauwunsch-Bestätigung
- CLS und CLSB müssen von beiden Partnern gegeben werden → beide „Halbverbindungen“ werden explizit geschlossen (symmetrische Verbindungsfreigabe)
- **Übungsaufgabe:**  
Entwerfe ein vollständiges  
Zustandsübergangdiagramm einschließlich Fehlerfälle!

# Kapitel 4.3

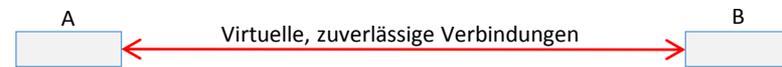
# Flusssteuerung und Staukontrolle

Begriffseinführung



# Techniken zur Realisierung verbindungsorientierter Dienste

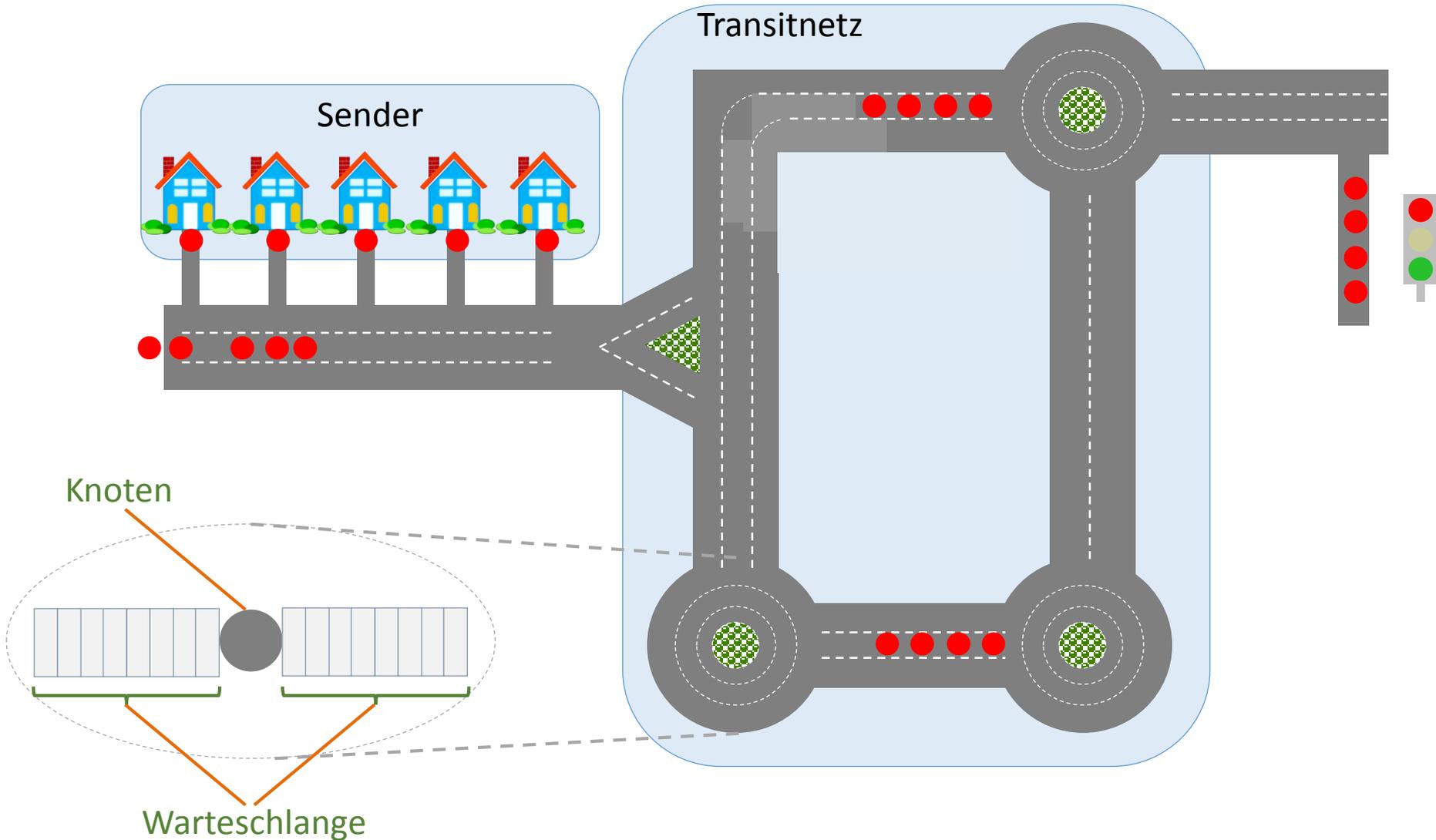
- Sequenznummer und Quittungen
  - Größe des Sequenznummernraumes
  - Maximale Lebensdauer von Nachrichten
  - Fenstertechnik
- Verbindungsaufbau/-abbau
- Flusssteuerung und Staukontrolle
  
- Ports und Sockets
- TCP / UDP



# Einordnung/Motivation:

- Warteschlangen in technischen Systemen sind endlich → Rückstaumechanismen erforderlich
  - für Verbindungen, für Schnittstellen, für Netze
- **Flusssteuerung (engl. flow control):** Maßnahmen gegen die Überlastung eines Empfängers durch den zugehörigen Sender.
- **Staukontrolle (engl. congestion control):** Maßnahmen gegen die Überlastung der Transitnetze durch eine relevante Menge individueller Kommunikationsbeziehungen.

# Warteschlangen (Grafik)



# Lösungsansätze Staukontrolle

- virtuelle Kanäle: sicher, aber Puffervergeudung und Aufbauzeit
- Begrenzung des Nachrichtenflusses in das Netz
  - Begrenzung der Anzahl von Verbindungen pro Host/Prozess
  - Begrenzung der Nachrichtenrate (traffic contract, leaky bucket)
- Sicherstellung des Abflusses aus dem Netz
  - garantierte Abnahmerate durch Empfänger
  - Reassembly im Host
- Wegwurf von Paketen bei Überlast (z.B. im Internet)
- Konstante Last im Netz (Taxi-Verfahren, nicht optimal)
- Füllungsregelung pro Verbindung

# Lösungsansätze Flussteuerung

- Sender verlangt Pufferreservierung vor Sendung, Senden nach Bestätigung → zusätzliche Nachrichten und Verzögerung
- Empfänger stellt Credit über Allokationsnachrichten (z.B. TCP) → Verfahren nicht robust
- Fenstertechnik (mit festem oder dynamischem Fenster) → Problem bei Fensterverkleinerung
- Stop and Go–Technik → Oszillierendes Verhalten, schlecht bei großem RTD
- Zeitrasterabhängiges Senden

# Kapitel 4.5

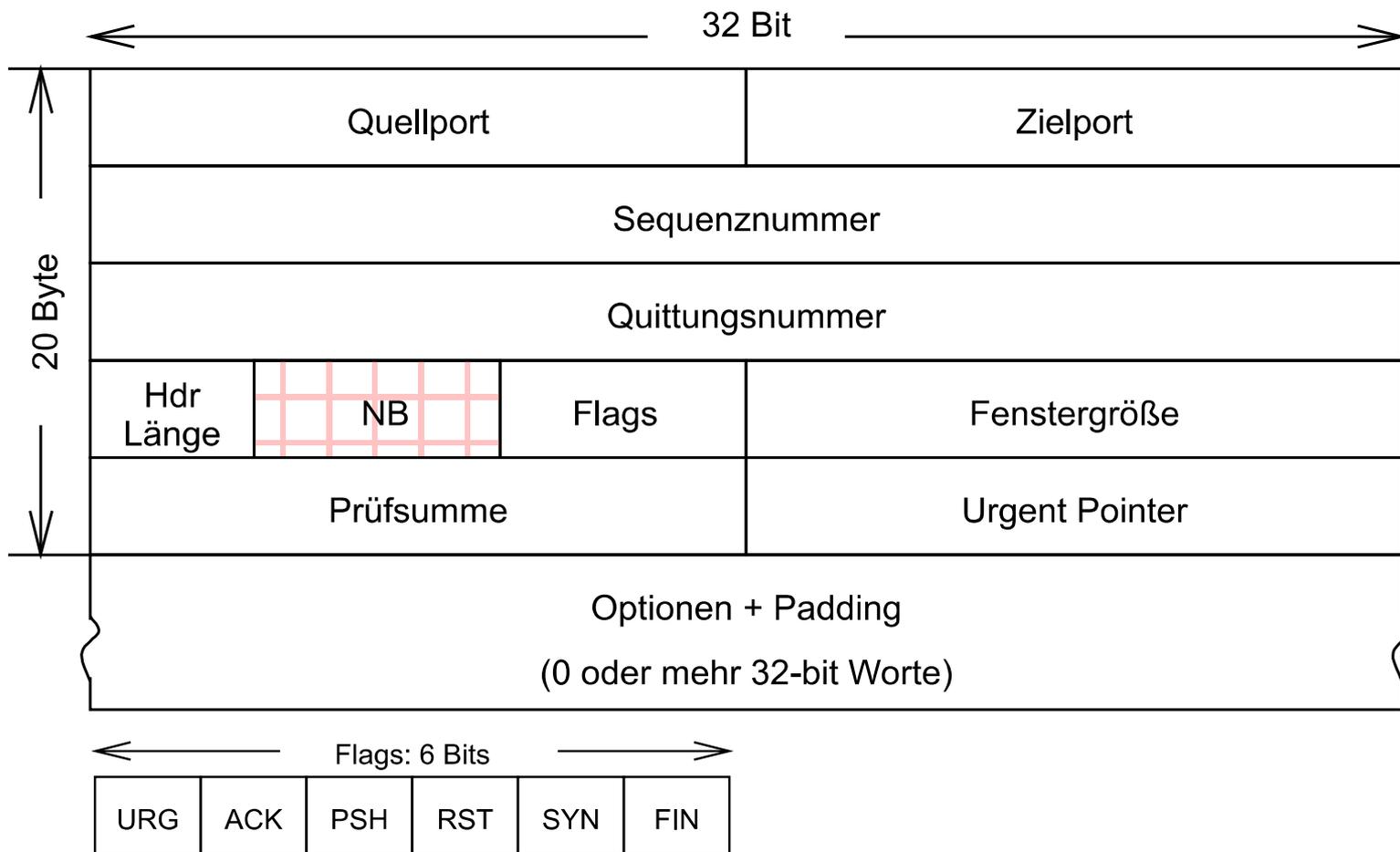
# Transmission Control Protocol (TCP)

Das verbindungsorientierte Transportprotokoll des Internets

# Überblick TCP

- Zuverlässiges Internet Transportprotokoll (z.B.: RFC 793)
- unterstützt Ende-zu-Ende-Transportverbindungen
  - Vollduplex (gleichzeitiges Senden und Empfangen in beide Richtungen)
  - mit Fehlerbehandlung (Verlust, Duplikate, usw.)
  - mit Flusssteuerung und Staukontrolle
- Byte-Strom-orientiert: Sequenz- und Quittungsnummern beziehen sich auf Bytes.
- Verbindungsorientiert: Aufbau per Drei-Wege-Handschlag
- unterstützt das Multiplexen zwischen Anwendungen und Vermittlungsdienst.
- TCP-Nutzer (Anwendungen) sind über Sockets adressierbar.
- Dienstdaten werden zwischengespeichert.
- Eine TCP-PDU wird als **Segment** bezeichnet.

# TCP Header (PCI) (1/3)



# TCP Header (PCI) (2/3)

- **Quellport/Zielport** (je 16 Bit): Endpunkte der Verbindung
- **Sequenznummer** (32 Bit): Bytestromnummer des ersten Nutzdaten-Bytes des Segments.
- **Quittungsnummer** (32 Bit): Bytestromnummer des nächsten erwarteten Nutzdaten-Bytes – nur lückenlos fehlerfrei empfangene Daten werden quittiert.
- **Header Länge** (4 Bit): Anzahl der 32 Bit Wörter im Header.
- **NB** (6 Bit): werden nicht benutzt.
- **Flags** (je 1 Bit):
  - **URG**: *Urgent Pointer* verwendet (Interrupt Data)
  - **ACK**: *Acknowledged* (Quittungsnummer gültig)
  - **PSH**: *Pushed Data* (nicht puffern, sondern sofort zustellen)
  - **RST**: *Reset Connection* (Host abgestürzt, Aufbauwunsch abgelehnt, etc.)
  - **SYN**: *Synchronize* (SYN=1, ACK=0 für Aufbauwunsch, beides 1 zur Bestätigung)
  - **FIN**: *Finished* (Abbauwunsch/keine weiteren Daten zu senden)

# Der TCP Header (PCI) (3/3)

- **Fenstergröße** (16 Bit): Anzahl der Bytes, die ab letzter Quittung gesendet werden dürfen.
- **Prüfsumme** (16 Bit): Einerkomplement der Summe aller 16-Bit-Worte über Pseudoheader und TCP-Segment (Der Pseudoheader enthält die folgenden Felder aus dem IP-Header: Quell- und Zieladresse, Protokoll, und Länge des TCP-Segments).
- **Urgent Pointer** (16 Bit): Zeigt auf letztes Byte in einer Kette von Daten (engl.: out-of-band data).
- **Optionen** ( $n * 32$  Bit): Wählbare Eigenschaften, wie z.B. maximale Segmentgröße und Timestamp-Option.

# TCP Ports

- 16 Bit Binärzahl (65536 Ports)
- Portnummern werden von der IANA (engl.: Internet Assigned Numbers Authority) verwaltet.
- *Well Known* Ports (0-1023)
  - TCP-Anwendungen (1-255)
  - UNIX-Anwendungen (256-1023)
- *Registered* Ports (1024-49151): meist herstellerspezifisch
- *Dynamic and/or Private* Ports (49152-65535): frei nutzbar

Beispiele für Well Known Ports	
21	FTP
22	Secure Shell (SSH)
23	telnet
25	SMTP
80	HTTP
143	IMAP

Beispiele für Registered Ports	
2628	DICT
2481	Oracle GIOP
6000	X Windows

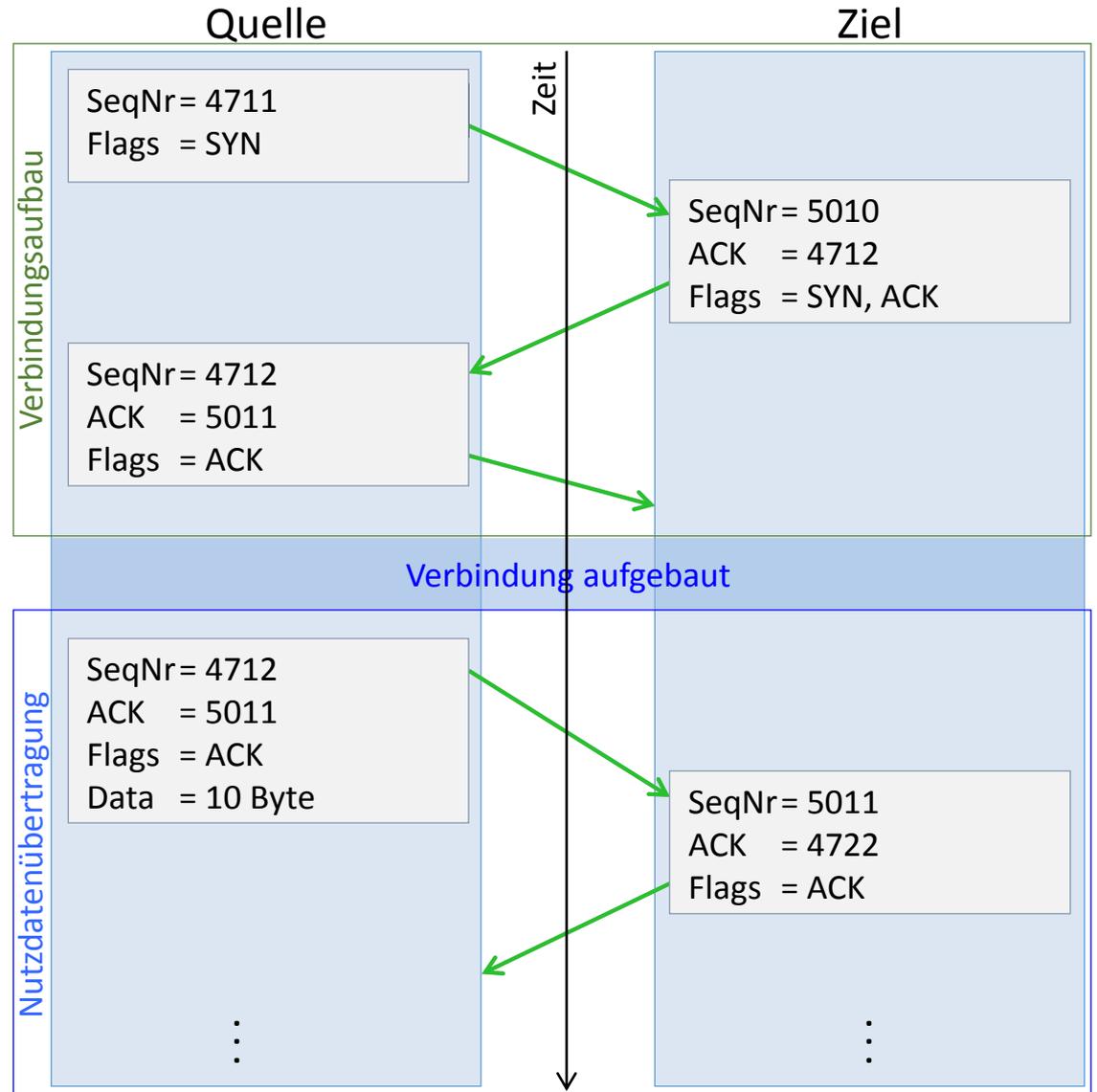
# Verbindungsaufbau: Drei-Wege-Handschlag

## Initiale Sequenznummern

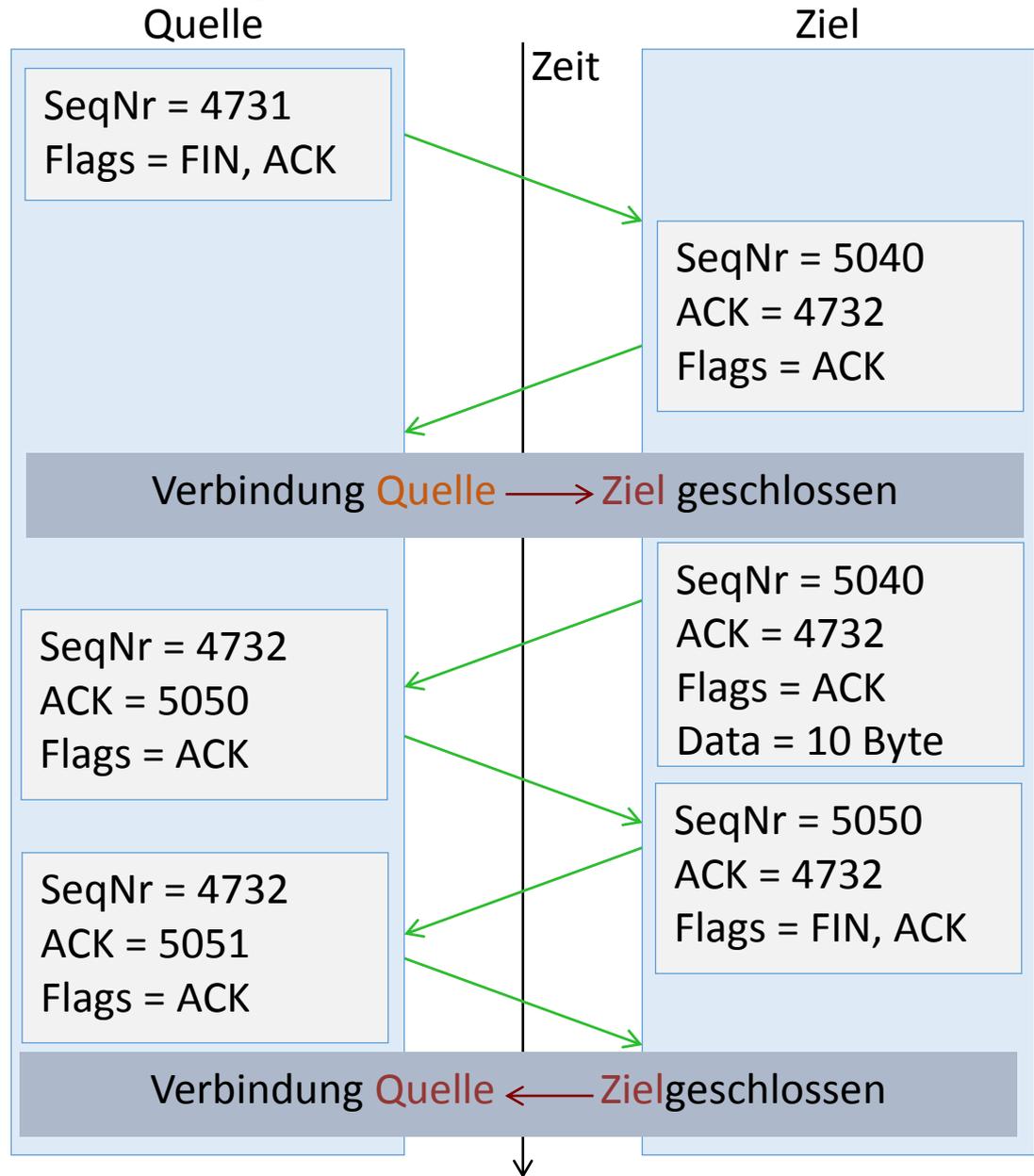
- gewählt aufgrund Zeitgeber
- Eindeutigkeit bei oft auf-/abgebauten Verbindungen

## Sequenznummern werden erhöht aufgrund von

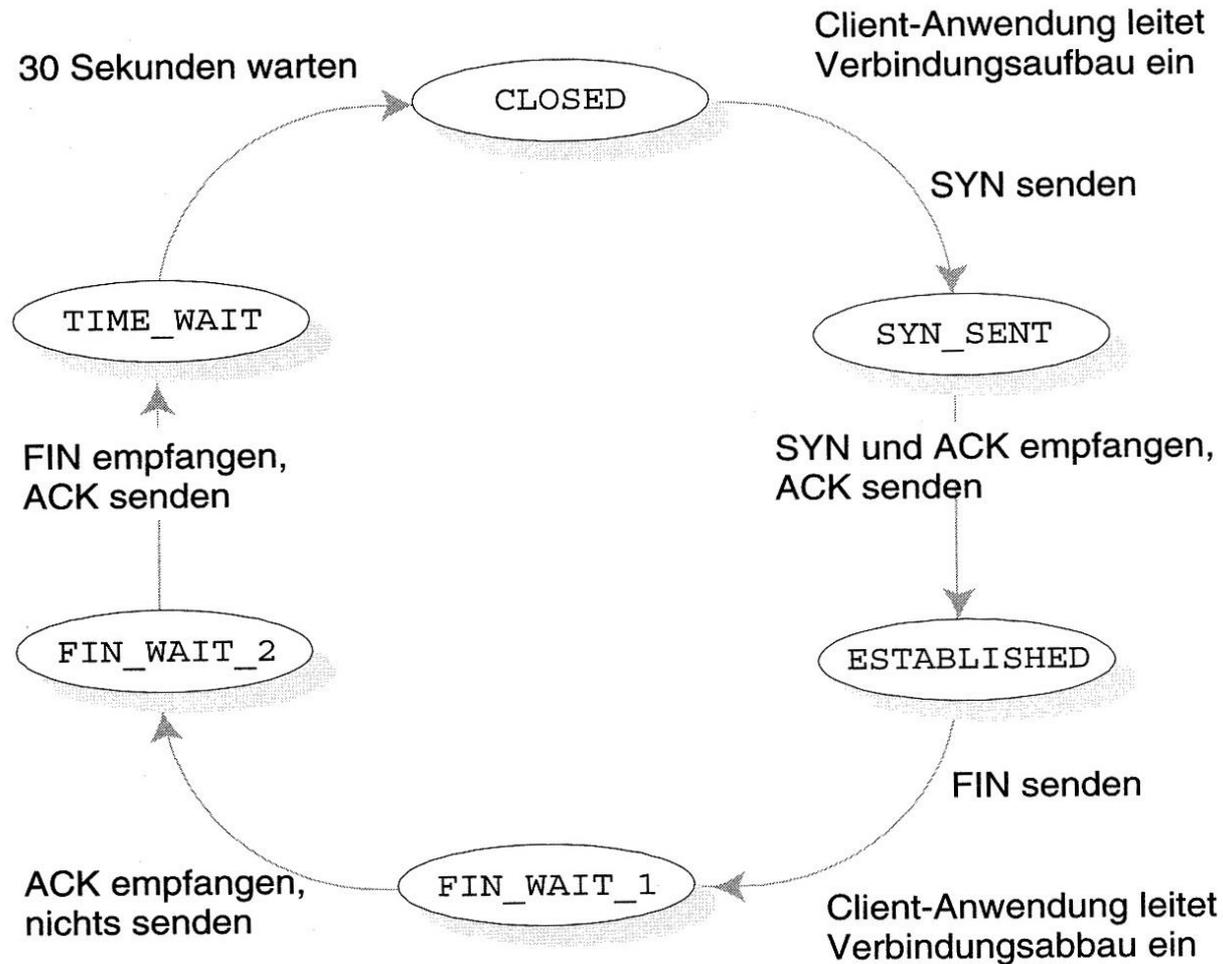
- Segmenten mit Nutzdaten (um 1 pro Byte)
- SYN und FIN (Schutz des Verbindungsmanagement)
- sonst: „leere“ Segmente -> „alte“ Seq. Nr. (z.B. Quittungen)



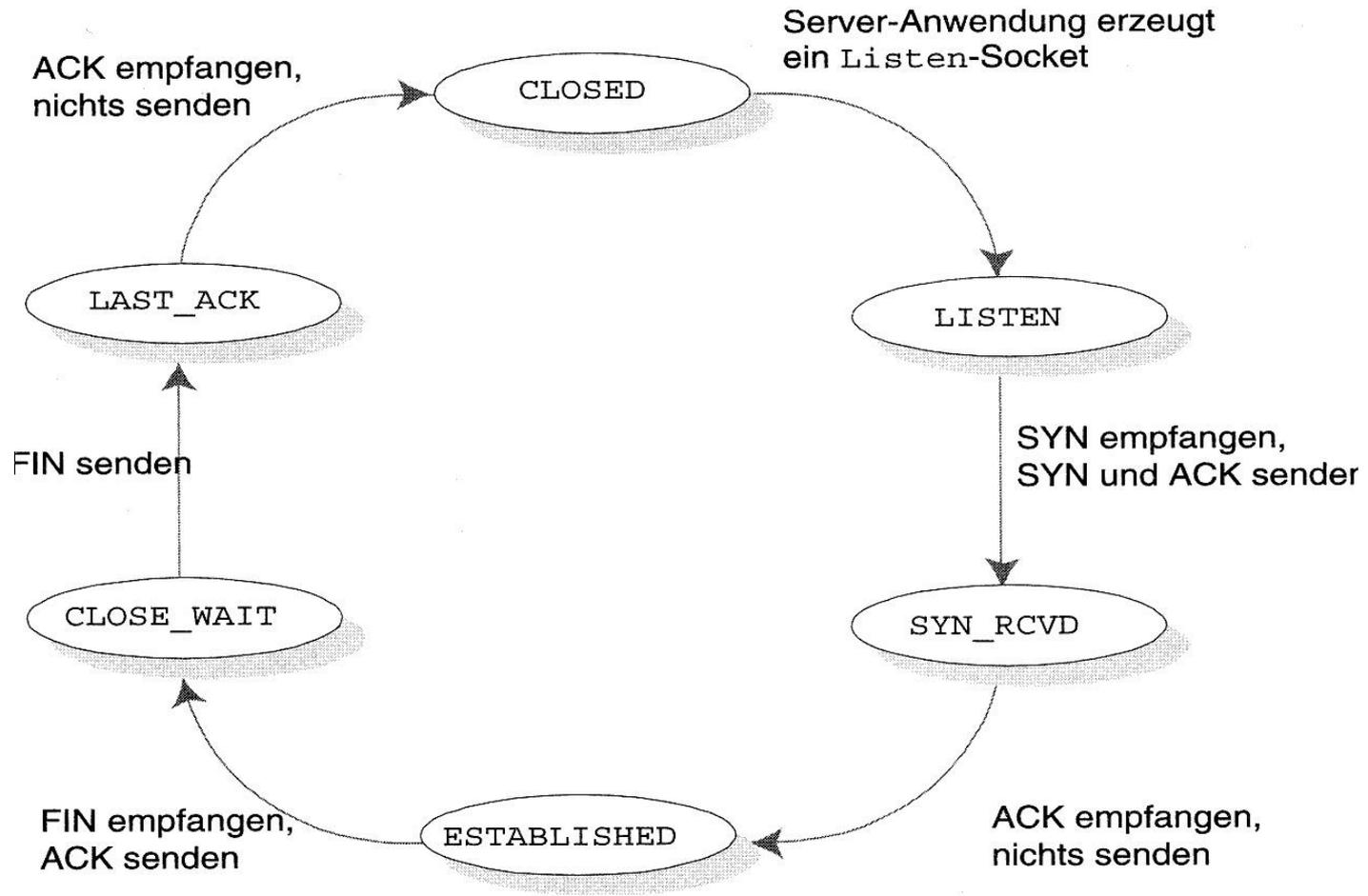
# Beidseitiger Verbindungsabbau



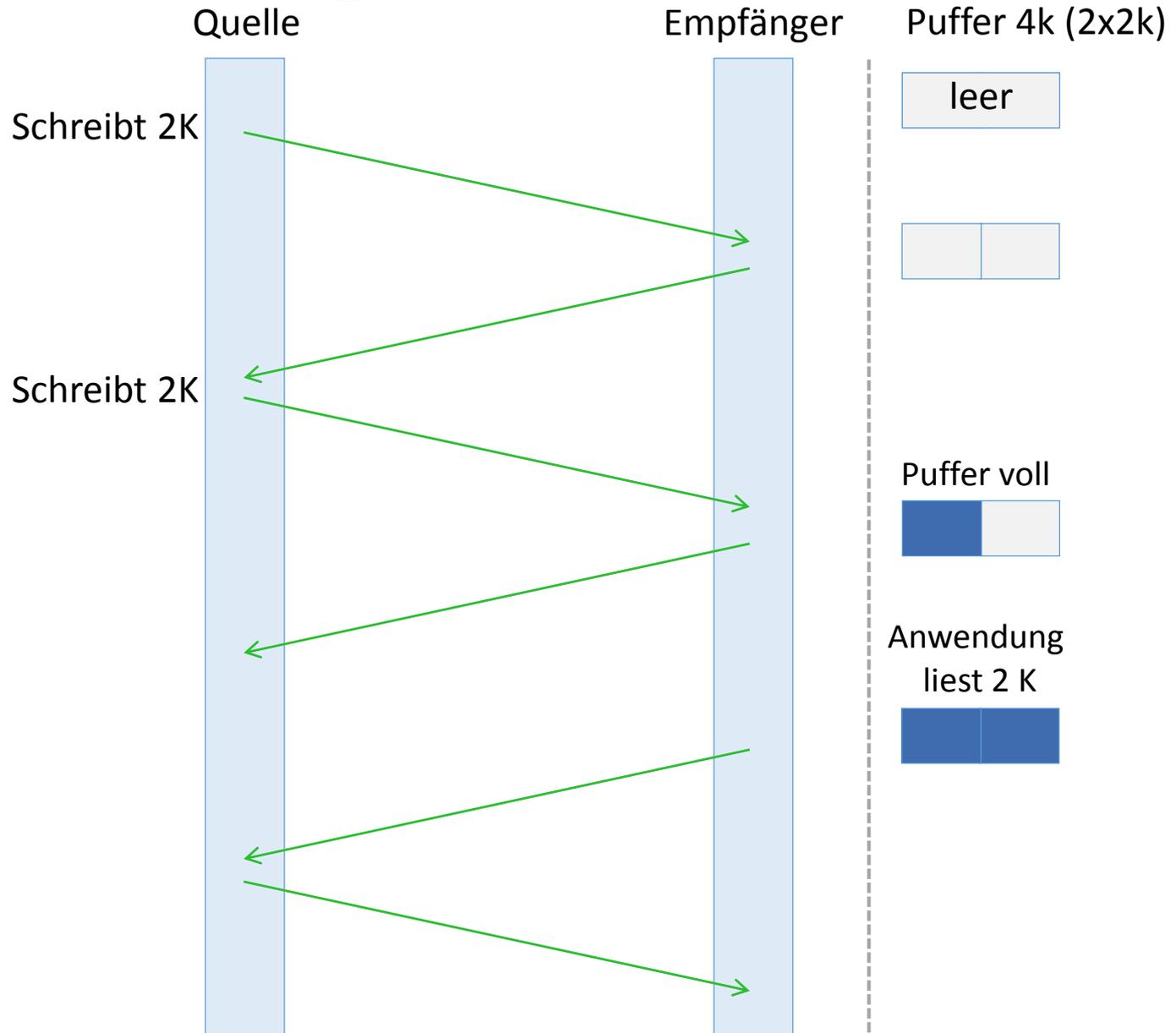
# Typische TCP-Zustände im Client



# Typische TCP-Zustände im Server



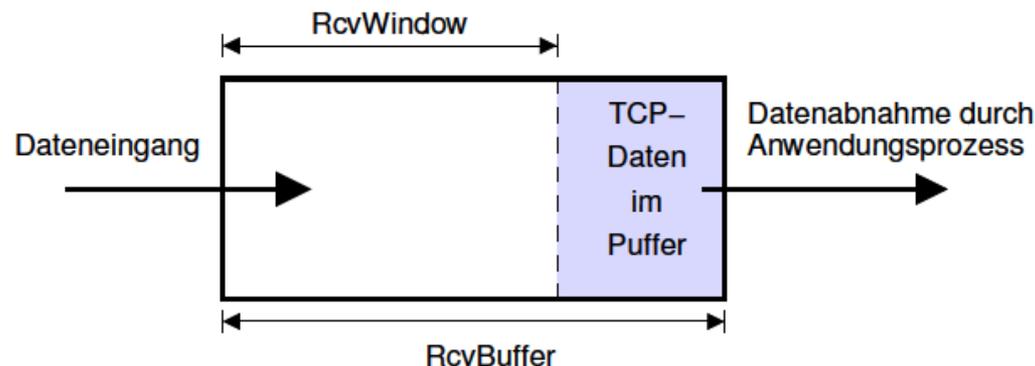
# Flussteuerung (1/2)



# Flussteuerung (2/2)

- Variablen des Senders:

- RcvWindow
- LastByteSent
- LastByteAcked



- Variablen des Empfängers:

- RcvBuffer
- LastByteRcvd: angekommen, aber noch nicht gelesen (im Puffer)
- LastByteRead: bereits vom Anwendungsprozess gelesen

- $RcvWindow = RcvBuffer - (LastByteRcvd - LastByteRead)$

- $LastByteSent - LastByteAcked \leq RcvWindow$

- Summenquittungen über mehrere Segmente

# Einordnung Staukontrolle (Überlastungskontrolle)

- Vergleich:
  - Flussteuerung: Empfänger nicht überlasten
  - Staukontrolle (Überlastungskontrolle): Netze nicht überlasten
- **Überlastung der Netze:** wenn viele Transportinstanzen auf vielen Endsystemen gleichzeitig zu viele Pakete zu schnell ins Netz einspeisen
  - ➔ Überlastung der Router auf der Vermittlungsschicht
- Fragen:
  - Wie kann auf Transportschicht festgestellt werden, dass eine Überlastung der Transitnetze besteht?
  - Wie kann man das Sendeverhalten einzelner Transportinstanzen anpassen, damit keine Überlastung der Transitnetze entsteht?

# Überlastungserkennung auf der Transportschicht

- Die Möglichkeiten, die der Transportschicht zur Verfügung stehen, um eine Netzüberlastung zu erkennen, können in zwei Gruppen eingeteilt werden.
- Entweder es wird ein Protokoll verwendet, mit dessen Hilfe überlastete Router den Endpunkten explizit mitteilen können, dass sie überlastet sind (z.B.: XCP – engl.: eXplicit Congestion Protocol)
- Oder die Transportinstanzen stellen die Netzüberlastung anhand indirekter Indikatoren wie Paketverlust fest.

# Überlastungskontrolle (Benötigte Variablen)

- Ansatz: Einführung eines Überlastfensters (engl.: congestion window)
- Analog dem Empfangsfenster (engl.: receive window) zur Flusssteuerung.

# Überlastungskontrolle (Benötigte Variablen)

- Beide TCP Instanzen benötigen folgende Variablen:
  - CongWindow (congestion window): steuert die Einspeisung ins Netz wie folgt:  
 $\text{LastByteSent} - \text{LastByteAcked} \leq \text{CongWindow}$
  - Effektives Fenster:  $\min(\text{RcvWindow}, \text{CongWindow})$
  - MSS (maximum segment size): maximale Datenmenge pro Segment.
  - Threshold: obere Schranke für das schnelle Wachstum des Überlastfensters (CongWindow).
  - RT (retransmission timer): nach Ablauf des Timer werden nicht bestätigte Segmente neu gesendet.

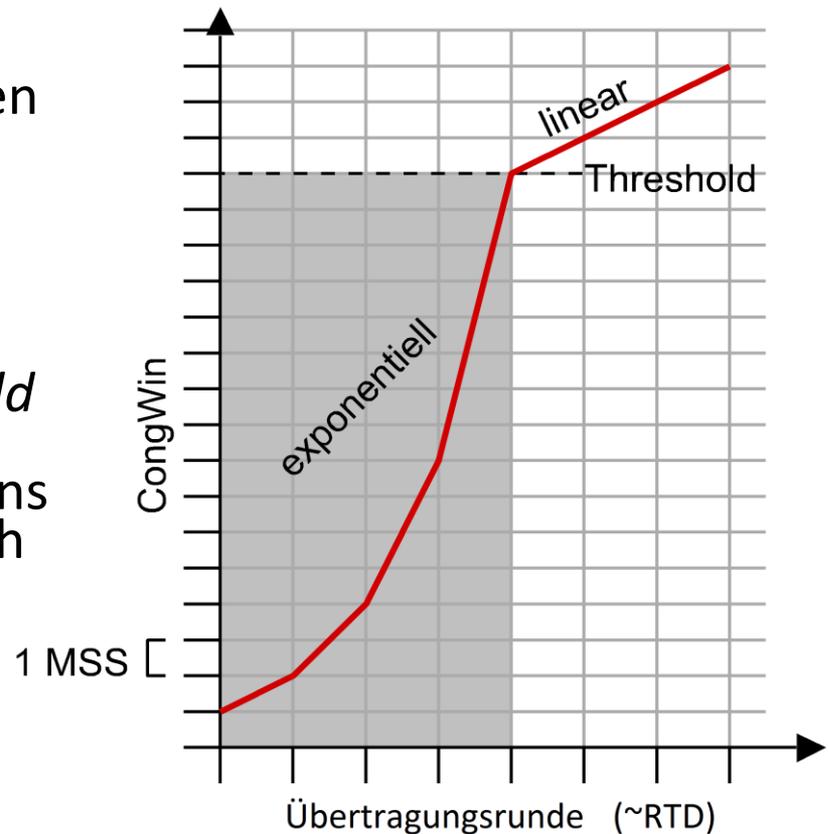
# Senderate nach Tahoe-Algorithmus (bei Netz ohne Last) (1/2)

## Slow Start:

- zum Start der Übertragung fangen wir mit einem kleinen Überlastfenster an (CongWin = 1 MSS).

## Exponentielles Wachstum:

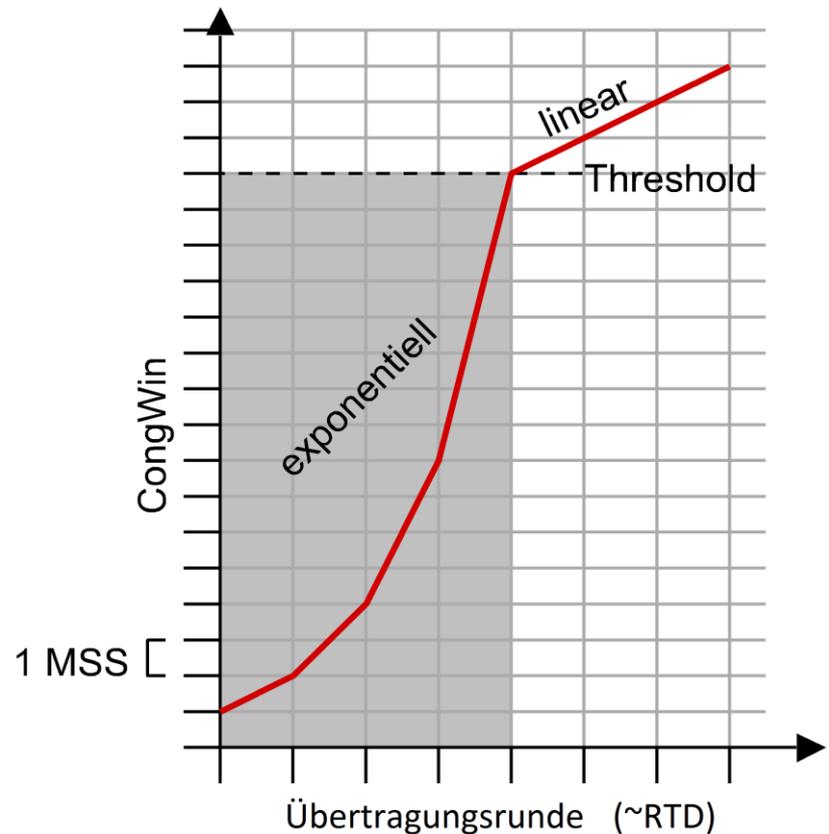
- Bis zum Erreichen eines *Threshold* inkrementieren wir CongWin bei jeder erhaltenen Quittung um eins → CongWin verdoppelt sich nach jeder Übertragungsrunde (Es werden jeweils doppelt so viele Nachrichten wie in der Runde davor quittiert).
- Eine Übertragungsrunde entspricht in etwa einer RTD



# Senderate nach Tahoe-Algorithmus (bei Netz ohne Last) (2/2)

## Lineares Wachstum:

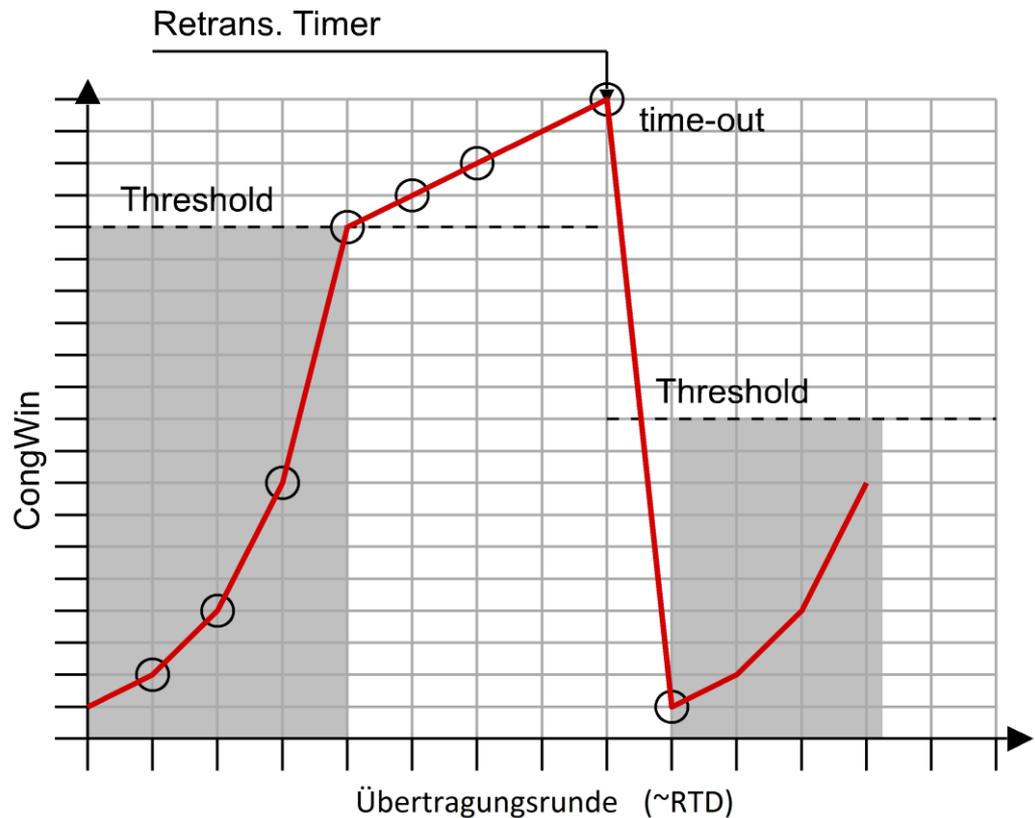
- Nach Erreichen des *Threshold* wächst CongWin pro Übertragungsrunde (daher wenn alle Segmente der Runde erfolgreich quittiert wurden) um eins.
- Diese Phase geht so lange weiter bis ein Segment verloren geht (Timeout) → lineares Wachstum solange das Netz nicht überlastet ist.



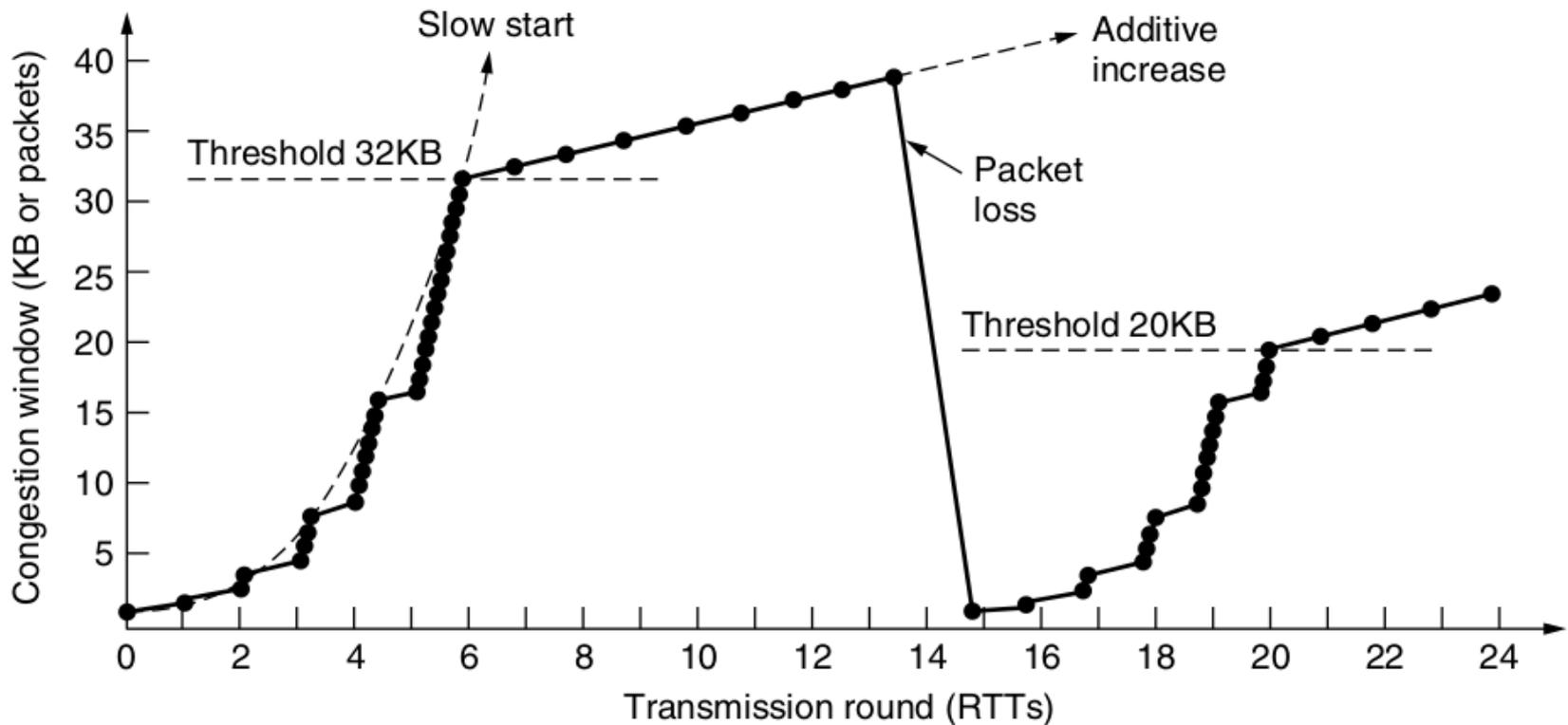
# Senderate nach Tahoe-Algorithmus (bei Netzüberlastung)

Bei Timeout  
(Retransmission Timer)  
Reduzierung der  
Senderate wie folgt:

- Reduzierung des Threshold auf die Hälfte der derzeitigen Fenstergröße:  
 $\text{Threshold} = \text{CongWin} / 2$
- Neustart mit Slow Start:  
 $\text{CongWin} = 1 \text{ MSS}$



# Senderate nach Tahoe-Algorithmus (Alternative Grafik)



Quelle: Tanenbaum, Computer Networks 5th Edition

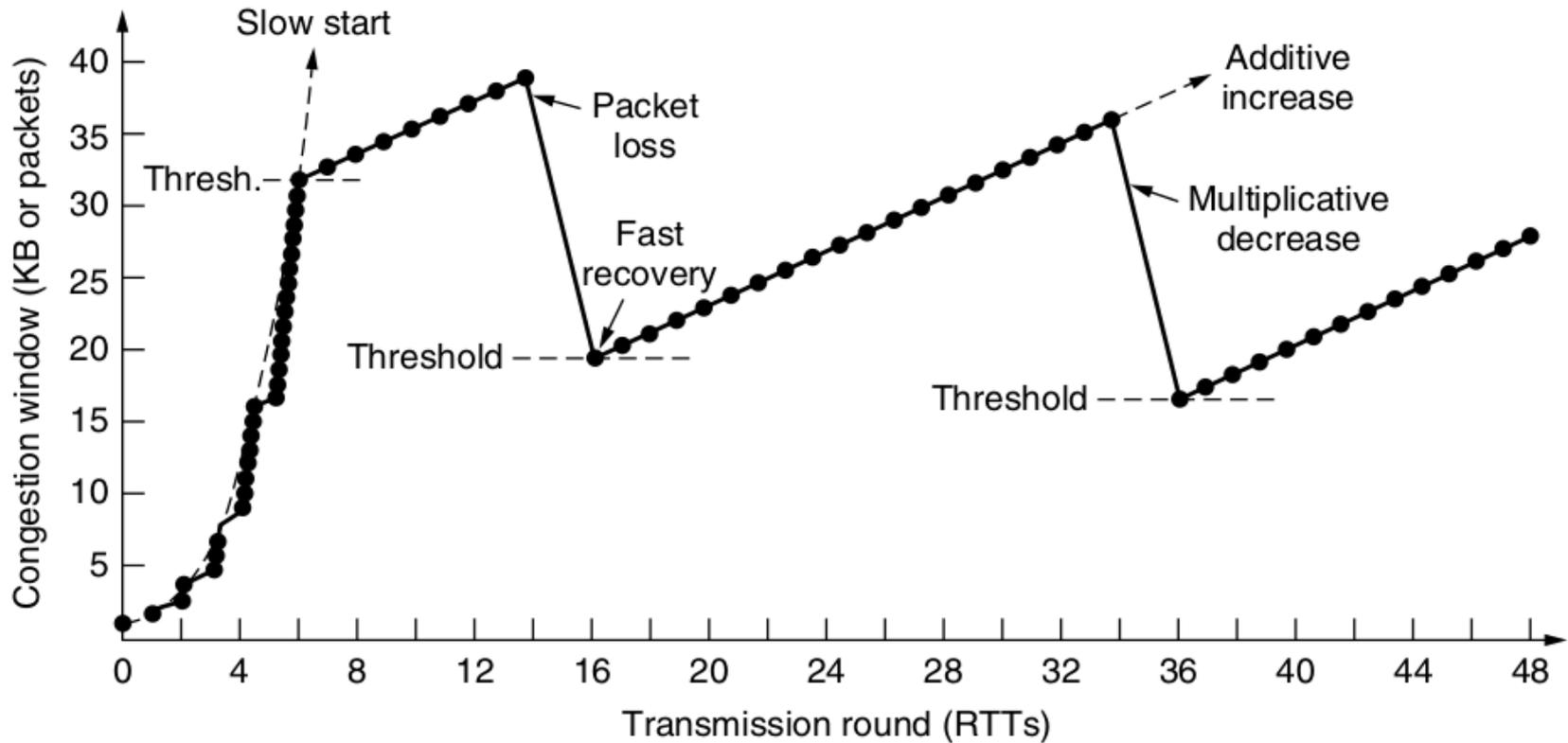
# Optimierung bei Verlust (RFC 5681)

- Grundlage:  
Geht ein Segment verloren, so werden darauf folgende, empfangene Segmente alle mit der gleichen Quittungsnummer (ACK-Nummer) quittiert (Byte-Strom-Orientierung).
- **Quittungsduplikate** weisen also frühzeitig (noch vor dem Timeout) auf den Verlust oder den Empfang eines Segments außer der Reihe hin.

# Optimierung bei Verlust (RFC 5681)

- **Schnelle Neuübertragung** (engl.: fast retransmit):  
Der Sender wiederholt nach Empfang eines Quittungsduplikats das auf die ACK-Nummer folgende Segment (noch vor dem Timeout).
- **Schnelle Wiederherstellung** (engl.: fast recovery):  
Regelt Senderverhalten nach der schnellen Neuübertragung:
  - Pro empfangenem Quittungsduplikat wird ein Segment neuer Daten gesendet.
  - CongWin wird respektiert aber nicht verändert.
  - ab drittem Segment: Neuberechnung CongWin (ohne Slow Start)

# Senderate mit Schneller Wiederherstellung (Grafik)



Quelle: Tanenbaum, Computer Networks 5th Edition

# Retransmission Timer

- Timer sollte größer als RTT (engl.: round trip time) sein.
- Wir leiten den Timer aus der Zeit zwischen dem Versand eines Segments und dem Erhalt der Empfangsbestätigung ab (SampledRTT):

EstimatedRTT = 3; //typische Initialisierung (in Sekunden)

x = 0.125; //typischer Startwert

EstimatedRTT = (1-x) \* EstimatedRTT + x \* SampledRTT;

Abweichung = (1-x) \* Abweichung + x \* |SampledRTT - EstimatedRTT|

Timer = EstimatedRTT + 4 \* Abweichung;

# RTD- und MTU-Ermittlung

Kern-Algorithmus: Verbesserung der RTD-Schätzung

- Ziel: Anpassung des Timerwertes RTT (Round Trip Time)
- Wiederholung eines Segments → RTT wird nicht aktualisiert
- Fehlgeschlagene Übertragung (d.h. Timeout) → RTT wird verdoppelt

Bestimmung der kleinsten MTU (engl.: maximum transmissible unit) auf dem tatsächlich verwendeten (Vermittlungsschicht) Pfad → Anpassung der Segmentgröße zum Verhindern von IP-Paketfragmentierung.

# Optimierung bei kurzen Segmenten

Nagle-Algorithmus: Zusammenfassung (Blocking) mehrerer kurzer TCP-Segmente.

- Problem: Datenübergabe an TCP in kleinen Portionen (z.B. telnet, remote shell, secure shell) → viele, kleine Segmente, sehr großer Overhead (z.B. 1 Byte/Segment → 4000%)
- Lösung: Speklatives Warten, bis „genug“ Daten, dann geblockt übertragen
- Nachteil: Wartezeit auch bei insgesamt kleiner zu übertragenden Datenmenge
- TCP CORK: gesonderter Dienstaufwurf bei manchen Implementierungen, um geringe Datenmenge zu übergeben

# Kapitel 4.6

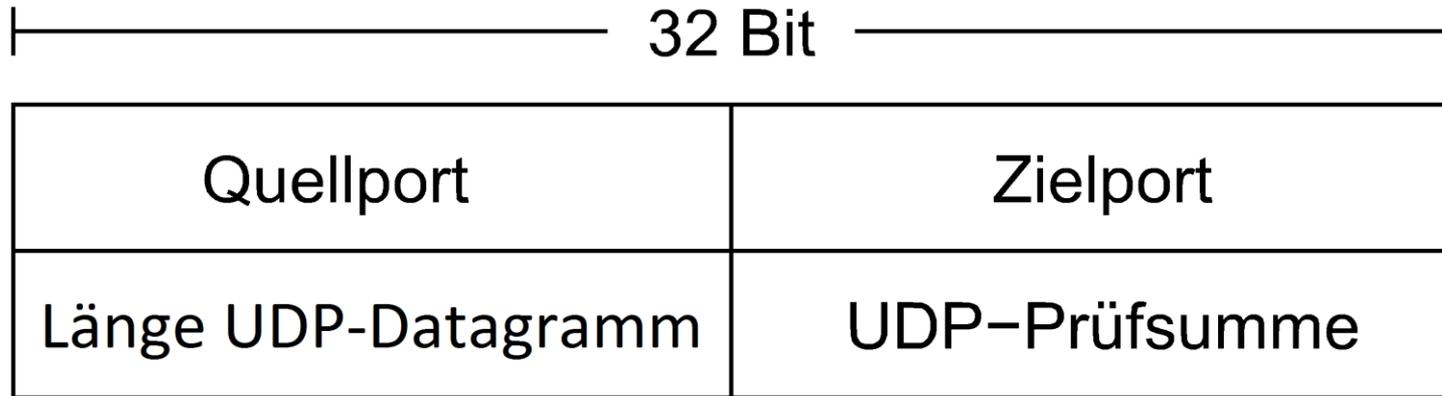
# User Datagram Protocol (UDP)

Das verbindungslose Transportprotokoll des Internets

# Überblick UDP

- Verbindungsloses, unzuverlässiges Internet Transportprotokoll (RFC 768)
  - kein Verbindungsaufbau, -abbau, oder -status
  - unregulierte Senderate
  - keine Sequenznummern
    - ➔ keine Reihenfolgesicherung, Duplikats- oder Verlusterkennung
- 8 Byte Header (vgl. mit mindestens 20 Byte bei TCP)
- Unterstützt das Multiplexen über Anwendungen mit Hilfe von UDP-Ports (wie bei TCP)
- Bsp. Protokolle, die UDP verwenden: TFTP, DNS, RPC, SNMP
- UDP-Nutzer (Anwendungen) sind über UDP-Sockets adressierbar.

# Der UDP Header (PCI)

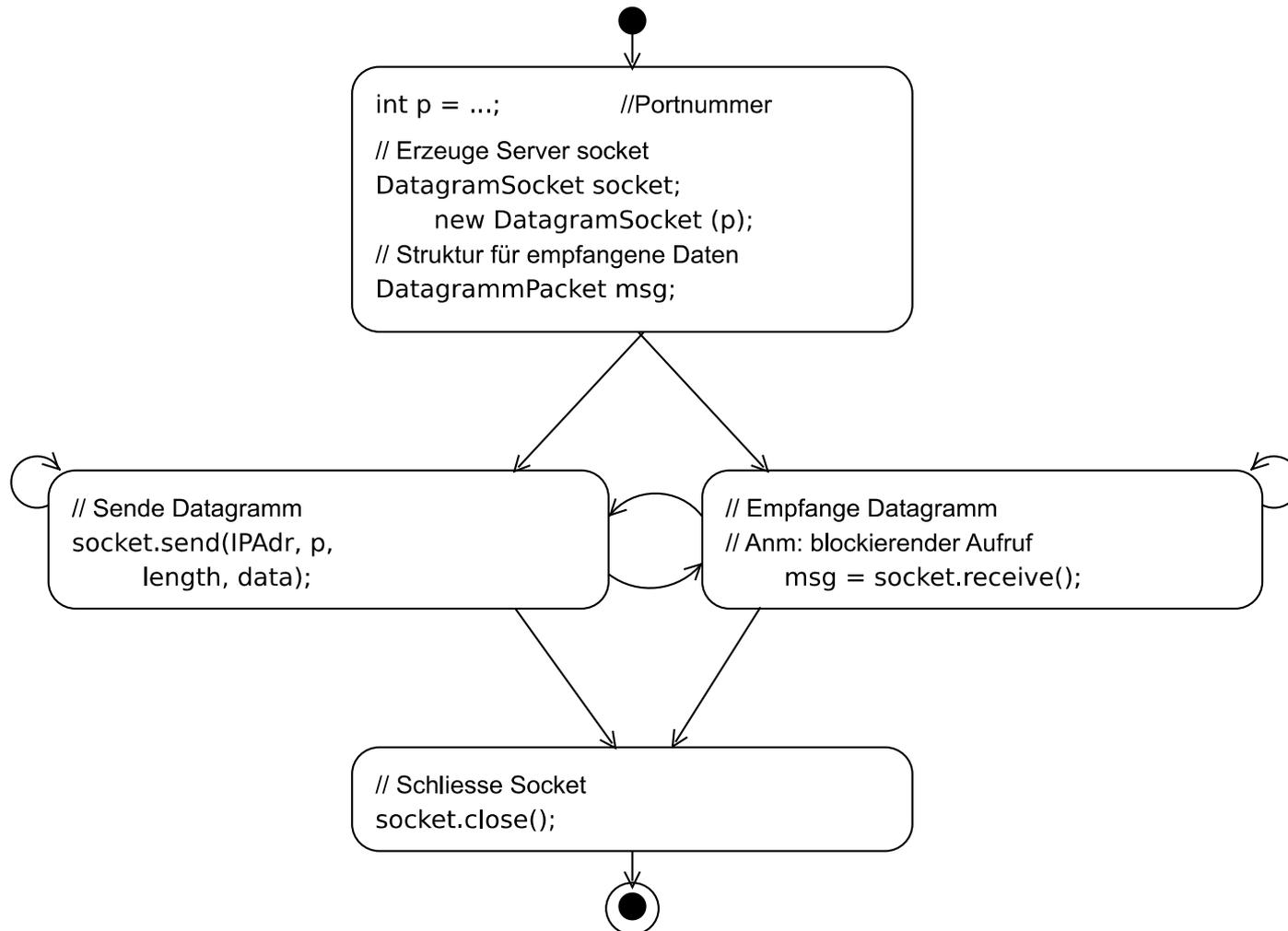


- **Quellport/Zielport** (je 16 Bit): zur Adressierung
- **Datagrammlänge** (16 Bit): Maximallänge von 65515 Bytes (bei Verwendung von IPv4)
- **Prüfsumme** (16 Bit): (optionale) Prüfsumme über IP-Pseudoheader und UDP-Datagramm

# Verwendung von UDP

- Programmierung mit UDP Sockets
  - Senden/Empfangen isolierter Datagramme
  - Sicht auf lokales Socket
  - Senden/Empfangen zu/von mehreren Hosts (IP-Adresse und Port des Empfängers werden i.d.R. beim Senden jedes einzelnen Datagramms angegeben)
- Reihenfolgesicherung, Verlust- und Duplikatsbehandlung, sowie die Steuerung der Senderate obliegen dem Anwendungsprogramm.

# Beispiel: UDP-Socket-Programm



# Zusammenfassung

## Transportschicht

- Netzunabhängiger Transport von Nachrichten zwischen Endsystemen
- Verschattung der Wege durchs Netz
- Fehlerbehandlung Ende-zu-Ende
- Anpassung der Übertragungsqualitäten
- Splitting/Multiplex über Anwendungen/Prozesse
- Verbindungsloser oder -orientierter Dienst
- z.B. im Internet: TCP (verbindungsorientiert), UDP (verbindungslos)

# Fragen zu Kapitel 4 (1/2)

- Was sind Einflussgrößen für die Größe des Sequenznummernraumes?
- Wozu kann die Fenstertechnik (Schiebefensterprotokoll) eingesetzt werden?
- Wie wird Eindeutigkeit der Sequenznummern sichergestellt?
- Wie wirken sich zu kleine Fenster aus, wie zu kleine Sequenznummernräume?
- Unter welchen Voraussetzungen genügt ein Zwei-Wege-Handschlag für einen Verbindungsaufbau?
- Warum ist für einen sicheren Verb.-Aufbau auf einer Transportschicht i.A. ein Drei-Wege-Handschlag erforderlich?
- Warum sollte ein Verbindungsabbau i.A. beidseitig geschehen?
- Was ist der Unterschied zwischen Flusssteuerung und Staukontrolle?

# Fragen zu Kapitel 4 (2/2)

- Wann ist es für eine Anwendung sinnvoll UDP zu verwenden?
- Wie sichert man für eine Anwendung, die über UDP läuft, trotzdem einen zuverlässigen Datentransfer?
- A sendet 2 TCP-Segmente (mit Sequenznummern 90 und 110) an B.
  - Wie viele Daten enthält das erste Segment? Wie lang ist es insgesamt?
  - Wie lautet die Quittungsnummer in der Bestätigung von B nach A, wenn nur das erste Segment verloren geht?
- Welche Mechanismen bietet TCP für eine zuverlässige Ende-zu-Ende Verbindung?
- Welches Transportprotokoll unterstützt ein Multiplexen von Anwendungen?