Rechnernetze I

SoSe 2020

Roland Wismüller
Universität Siegen
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: 8. Juli 2020

Rechnernetze I

SoSe 2020

7 Ende-zu-Ende Protokolle

Inhalt

- Ports: Adressierung von Prozessen
- UDP
- TCP
  - Bytestrom, Paketformat
  - Verbindungsaufbau und –zustände
- Sicherung der Übertragung
- Übertragungssicherung in TCP

- Peterson, Kap. 5.1, 5.2.1 – 5.2.4, 5.2.6
- CCNA, Kap. 9

Einordnung

- **Protokolle der Vermittlungsschicht:**
  - Kommunikation zwischen **Rechnern**
  - Adressierung der Rechner
  - IP: *Best Effort*, d.h. keine Garantien
- **Protokolle der Transportschicht:**
  - Kommunikation zwischen **Prozessen**
  - Adressierung von Prozessen auf einem Rechner
  - ggf. Garantien: Zustellung, Reihenfolge, ...
    - Sicherung der Übertragung notwendig
  - zwei Internet-Protokolle: UDP und TCP

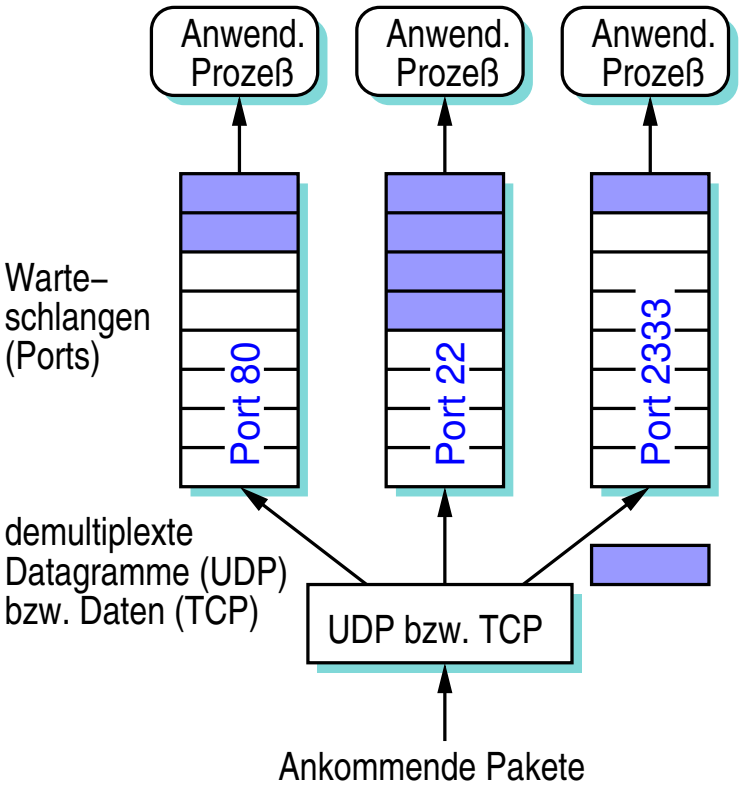
7.1 Ports: Adressierung von Prozessen

- Wie identifiziert man Prozesse?
  - **Nicht** durch die Prozeß-ID des Betriebssystems:
    - systemabhängig, „zufällig“
  - **Sondern**: indirekte Adresierung über Ports
    - 16-Bit Nummer
- Woher weiß ein Prozeß die Port-Nummer des Partners?
  - „*well known ports*“ (i.d.R. 0...1023) für Systemdienste
    - z.B.: 80 = Web-Server, 25 = Mail-Server
    - Analogie: Tel. 112 = Feuerwehr
  - Server kennt die Port-Nummer des Clients aus UDP- bzw. TCP-Header der Anfrage

7.1 Ports: Adressierung von Prozessen ...

Port-Demultiplexing

- Ports sind typischerweise durch Warteschlangen realisiert
- Bei voller Warteschlange: UDP bzw. TCP verwirft das Paket



Anmerkungen zu Folie 202:

Das Port-Multiplexing unterscheidet sich bei UDP und TCP etwas:

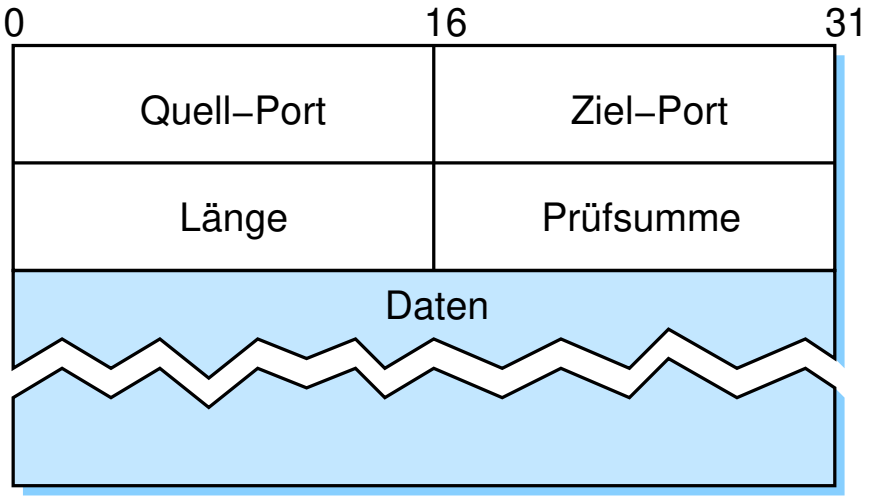
- Bei **UDP** werden unterscheidbare Datagramme in die Warteschlange eingereiht. Der Anwendungsprozess erhält beim Auslesen der Warteschlange jeweils ein Datagramm.
- Bei **TCP** ist die Warteschlange als Folge von Bytes, d.h. als Byte-Strom realisiert (siehe Folie 206). Beim Eintreffen eines TCP-Segments werden die Nutzdaten des Segments an den Byte-Strom angefügt. Der Anwendungsprozess kann jedes Mal eine völlig beliebige Zahl von Bytes aus dem Strom lesen (solange dieser nicht leer wird).

7.2 UDP

UDP: *User Datagram Protocol*

- Dienstemodell von UDP:
  - Übertragung von Datagrammen zwischen Prozessen
    - unzuverlässiger Dienst
- „Mehrwert“ im Vergleich zu IP:
  - Kommunikation zwischen Prozessen
    - ein Prozess wird identifiziert durch das Paar (Host-IP-Adresse, Port-Nummer)
    - UDP übernimmt das Demultiplexen (☞ **7.1**)
      - d.h. Zustellung an Zielprozess auf dem Zielrechner
  - Prüfsumme über Header und Nutzdaten

Aufbau eines UDP-Pakets



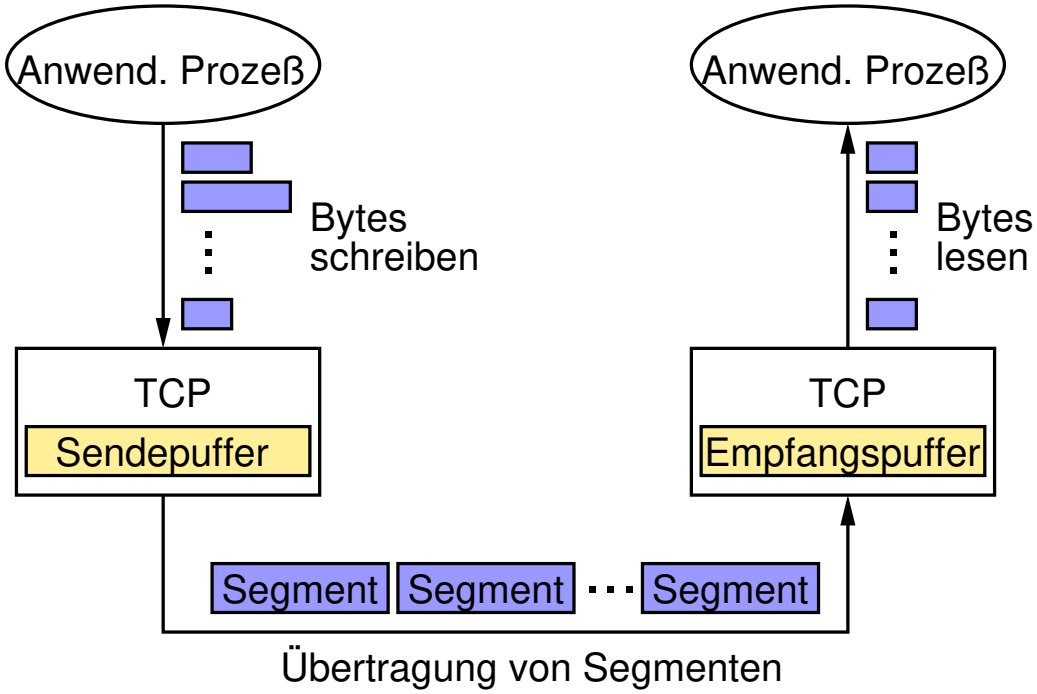
- (Das UDP-Paket ist im Nutzdatenteil eines IP-Pakets!)

7.3 TCP

TCP: *Transmission Control Protocol*

- Dienstemodell von TCP:
  - zuverlässige Übertragung von Daten**strömen** zw. Prozessen
    - verbindungsorientiert
- Meist verwendetes Internet-Protokoll
  - befreit Anwendungen von Sicherung der Übertragung
- TCP realisiert:
  - Port-Demultiplexing (☞ **7.1**)
  - Vollduplex-Verbindungen
  - Flußkontrolle
  - Überlastkontrolle

TCP überträgt Daten (Byteströme) segmentweise

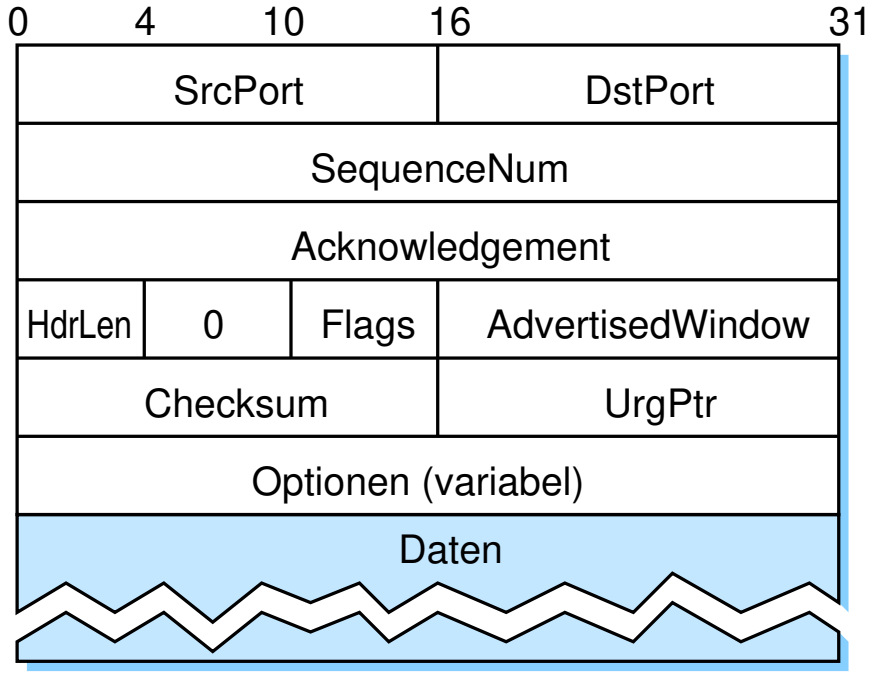


Wann wird ein Segment gesendet?

- Wenn die maximale Segmentgröße erreicht ist
  - ***Maximum Segment Size*** (**MSS**)
  - i.d.R. an maximale Frame-Größe (**MTU**, ***Maximum Transmission Unit***) des lokalen Netzes angepaßt:
    - MSS = MTU – Größe(TCP-Header) – Größe(IP-Header)
    - verhindert, daß das Segment von IP sofort wieder fragmentiert werden muß (☞ **5.4**)
- Wenn der Sender es ausdrücklich fordert
  - *Push*-Operation (*Flush*)
- Nach Ablauf eines periodischen Timers

7.3.2 TCP Header

Aufbau eines TCP Segments



- (Das TCP-Segment ist im Nutzdatenteil eines IP-Pakets!)

7.3.2 TCP Header ...

- **SequenceNum, Acknowledgement, AdvertisedWindow:**
  - für *Sliding-Window*-Algorithmus (siehe später, **7.5**)
- **Flags**:
  - SYN Verbindungsaufbau
  - FIN Verbindungsabbau
  - ACK **Acknowledgement**-Feld ist gültig
  - URG Dringende Daten (*out of band data*)
    UrgPtr zeigt Länge der dringenden Daten an
  - PSH Anwendung hat *Push*-Operation ausgeführt
  - RST Abbruch der Verbindung (nach Fehler)

  Es können mehrere Flags gleichzeitig gesetzt sein

7.3.3 Verbindungsauf- und -abbau

Verbindungsaufbau

- Asymmetrisch:
  - Client (rufender Teilnehmer): **aktives Öffnen**
    - sende Verbindungswunsch zum Server
  - Server (gerufener Teilnehmer): **passives Öffnen**
    - warte auf eingehende Verbindungswünsche
    - akzeptiere ggf. einen Verbindungswunsch

Verbindungsabbau

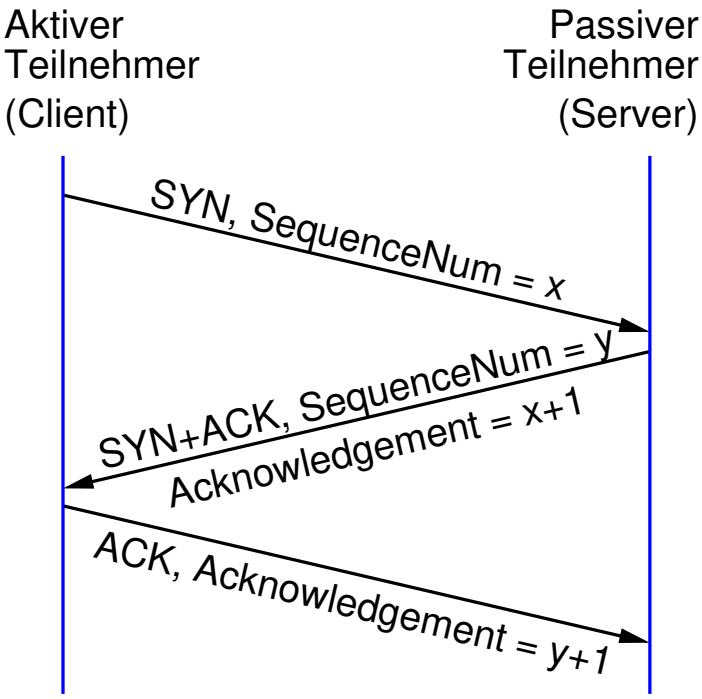
- Symmetrisch:
  - beide Seiten müssen die Verbindung schließen

7.3.3 Verbindungsauf- und -abbau ...

Zum Begriff der TCP-Verbindung

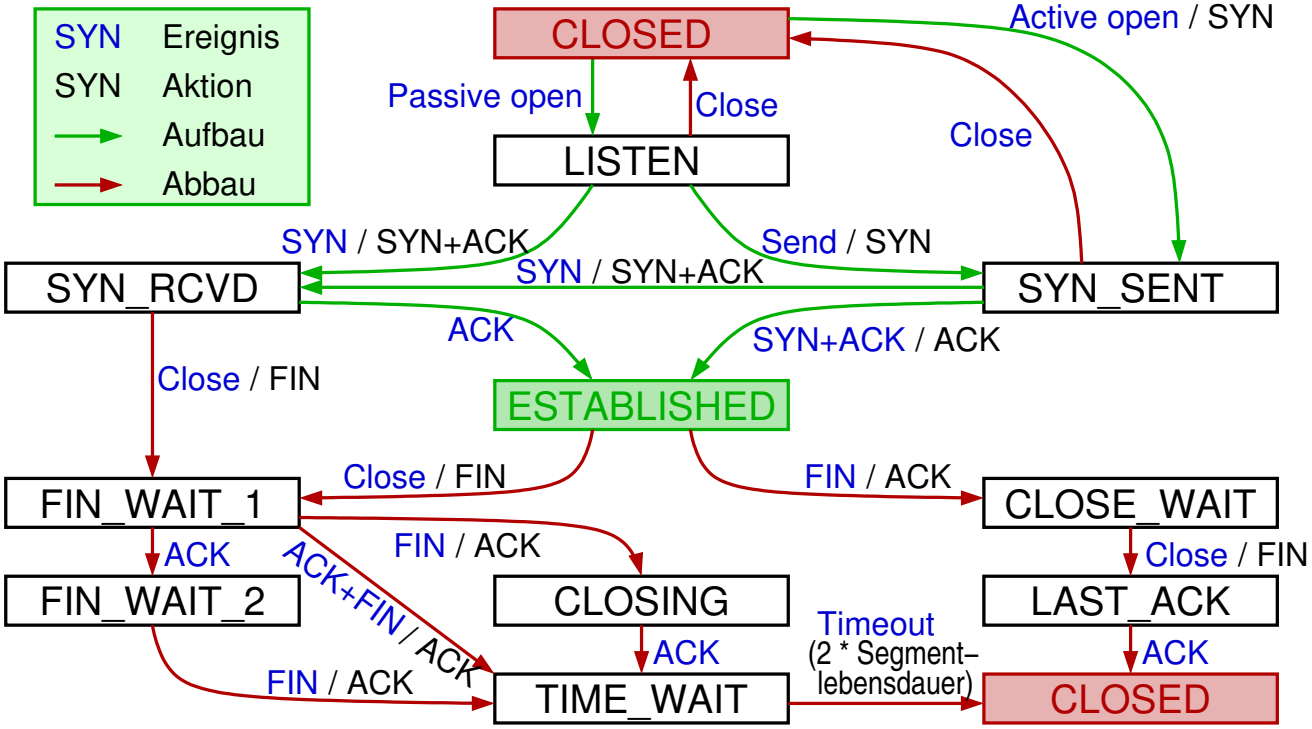
- Das Tupel (Quell-IP-Adresse, Quell-Port, Ziel-IP-Adresse, Ziel-Port) kennzeichnet eine TCP-Verbindung eindeutig
  - Nutzung als Demultiplex-Schlüssel
- Nach Abbau einer Verbindung und Wiederaufbau mit denselben IP-Adressen und Port-Nummern:
  - neue **Inkarnation** derselben Verbindung

Verbindungsaufbau: *Three-Way Handshake*



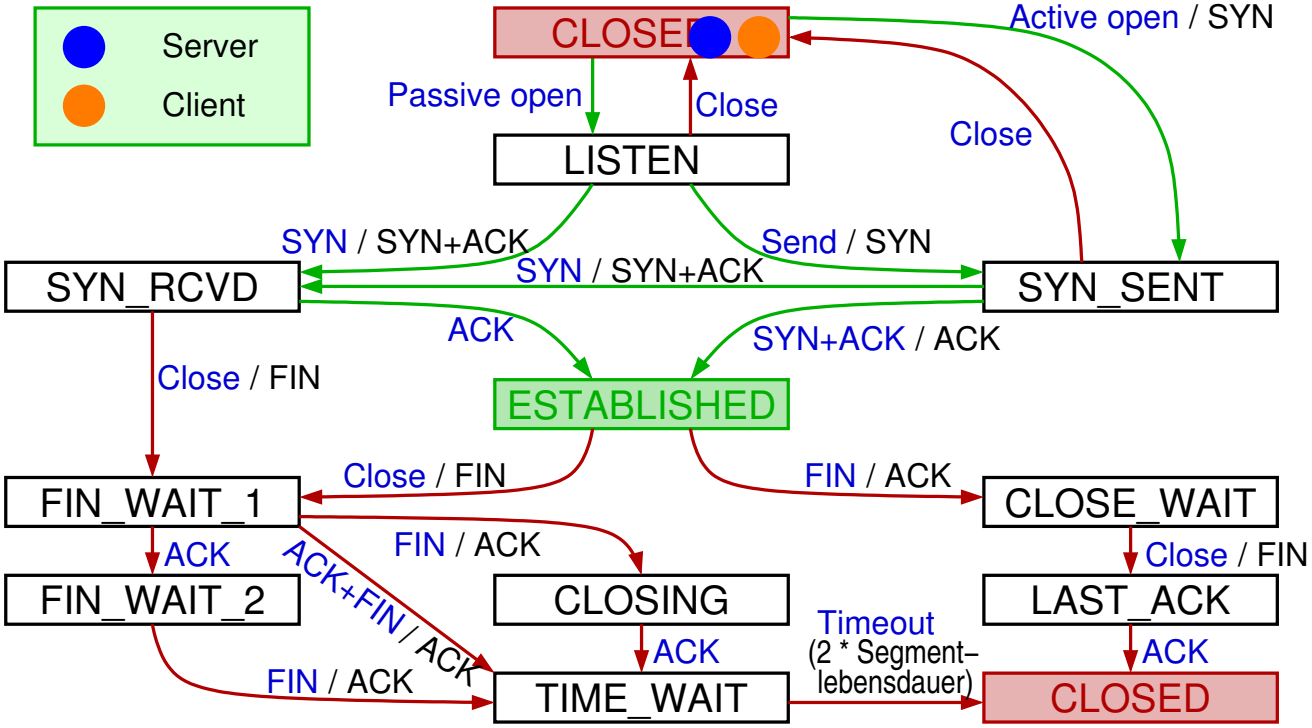
- Austausch von Sequenznummern
- „Zufälliger“ Startwert
  - jede Inkarnation nimmt andere Nummern
- Acknowledgement: nächste erwartete Sequenznummer

Zustände einer TCP-Verbindung



(Animierte Folie)

Zustände einer TCP-Verbindung ...



Anmerkungen zu Folie 214:

Im Beispiel kann der Client seinen Port nach dem Abbau der Verbindung längere Zeit nicht wiederverwenden (Zustand TIME_WAIT). Damit wird verhindert, daß zu schnell eine neue Inkarnation derselben Verbindung aufgebaut wird, da dies zu Problemen führen könnte, wenn noch Segmente aus der alten Inkarnation unterwegs sind.

Problem:

- Bei der Übertragung eines Segments bzw. Frames können Fehler auftreten
  - Empfänger kann Fehler erkennen, aber i.a. nicht korrigieren
  - Segmente bzw. Frames können auch ganz verloren gehen
    - z.B. durch überlasteten Router bzw. Switch
    - oder bei Verlust der Frame-Synchronisation (☞ **3.4**)
- Segmente bzw. Frames* müssen deshalb ggf. neu übertragen werden

* Zur Vereinfachung wird im Folgenden nur der Begriff „Frame“ verwendet!

Anmerkungen zu Folie 215:

Das Problem der Übertragunssicherung tritt sowohl auf der OSI-Schicht 4 als auch auf Schicht 2 auf, wenn ein zuverlässiger Dienst angeboten wird.

Die Tatsache, daß im Fehlerfall nur ein Teil der Daten (nämlich nur die betroffenen Segmente bzw. Frames) neu übertragen werden muß, ist eine wichtige Motivation dafür, daß die Daten zur Übertragung in kleinere Einheiten (Segmente bzw. Frames) aufgeteilt werden. Eine weitere Motivation ist die Möglichkeit des Multiplexings mehrerer Datenströme über eine gemeinsame Leitung.

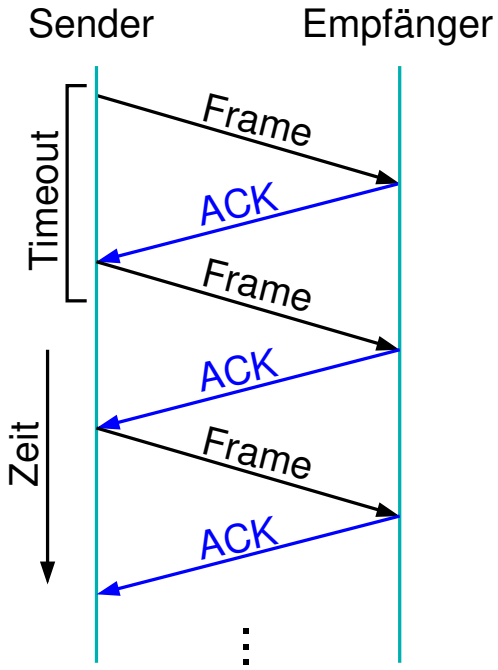
Basismechanismen zur Lösung:

- **Bestätigungen** (***Acknowledgements***, ACK)
  - spezielle Kontrollinformationen, die an Sender zurückgesandt werden
  - bei Duplex-Verbindung (wie z.B. bei TCP) auch **Huckepack-verfahren** (***Piggyback***):
    - Bestätigung wird im Header eines normalen Frames übertragen
- Senderseitige Zwischenspeicherung unbestätigter Frames
- ***Timeouts***
  - wenn nach einer bestimmten Zeit kein ACK eintrifft, überträgt der Sender den Frame erneut

7.4.1 *Stop-and-Wait*-Algorithmus

Ablauf bei fehlerfreier Übertragung

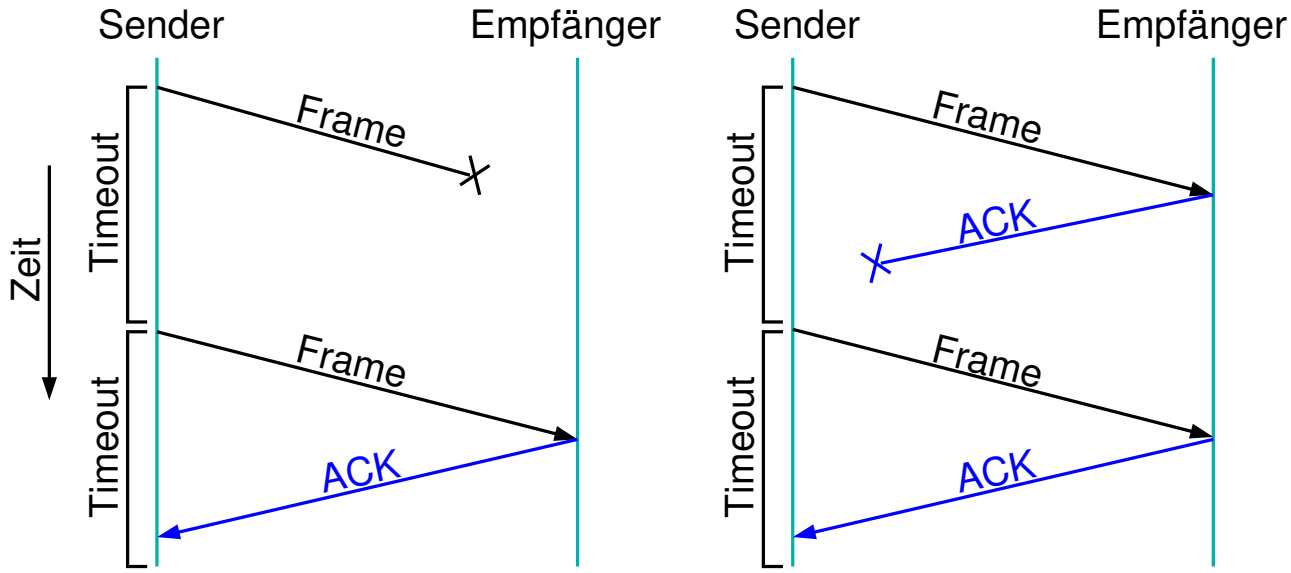
- Sender wartet nach der Übertragung eines Frames, bis ACK eintrifft
- Erst danach wird der nächste Frame gesendet



(Animierte Folie)

Ablauf bei Übertragungsfehler

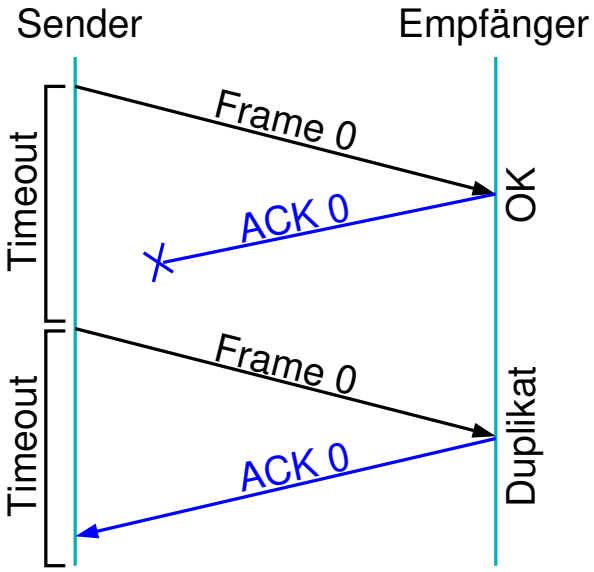
- Falls ACK nicht innerhalb der Timout-Zeit eintrifft:
  - Wiederholung des gesendeten Frames



Was passiert, wenn ACK verloren geht oder zu spät eintrifft?

- Der Empfänger erhält den Frame mehrfach
- Er muß dies erkennen können!

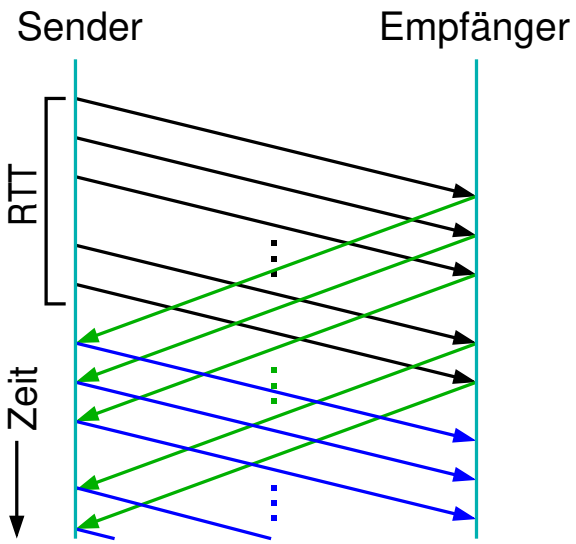
- Daher: Frames und ACKs erhalten eine **Sequenznummer**
- Bei Stop-and-Wait reicht eine 1 Bit lange Sequenznummer
  - d.h. abwechselnd 0 und 1



(Animierte Folie)

Motivation

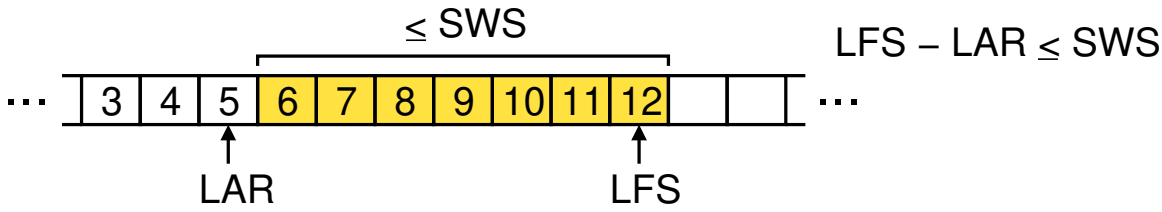
- Problem bei *Stop-and-Wait*:
  - Leitung wird nicht ausgelastet, da nur ein Frame pro RTT übertragen werden kann
- Um Leitung auszulasten:
  - Sender sollte die Datenmenge senden, die dem Verzögerungs(RTT)-Bandbreiten-Produkt entspricht, bevor er auf das erste ACK wartet
  - dann mit jedem ACK einen neuen Frame senden



(Animierte Folie)

Funktionsweise

- Jeder Frame erhält eine Sequenznummer
- Der **Sender** besitzt ein „Schiebefenster“ (*Sliding Window*):

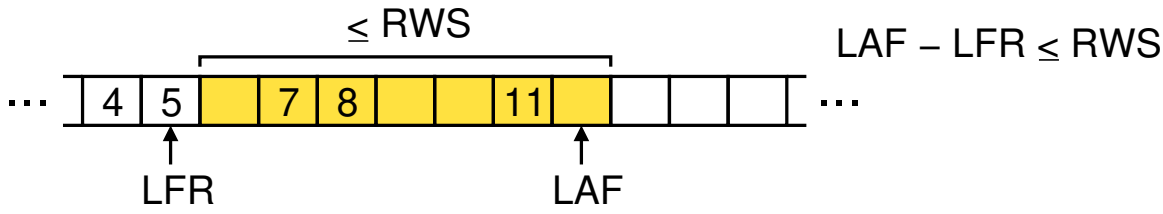


  - Jeder Eintrag steht für einen gesendeten Frame
  - LAR: *Last Acknowledgement Received*
    - bis zu diesem Frame (incl.) wurden alle quittiert
  - LFS: *Last Frame Sent*
  - SWS: *Sender Window Size*
    - max. SWS Frames werden ohne ACK abgeschickt

(Animierte Folie)

Funktionsweise ...

- Der **Empfänger** hat ebenfalls ein *Sliding Window*:



  - Jeder Eintrag steht für einen empfangenen Frame
  - LFR: *Last Frame Received*
    - alle Frames n mit $n \leq$ LFR wurden korrekt empfangen und quittiert
  - LAF: *Largest Acceptable Frame*
    - Frame n wird nur akzeptiert, wenn LFR $< n \leq$ LAF
  - RWS: *Receiver Window Size*
    - Anzahl der Pufferplätze beim Empfänger

Quittierung von Frames

- **Akkumulatives Acknowledgement**:
  - ACK für Frame n gilt auch für alle Frames $\leq n$
- Zusätzlich **negative Acknowledgements** möglich:
  - Wenn Frame n empfangen wird,
    aber Frame m mit $m < n$ noch aussteht,
    wird für Frame m ein NACK geschickt
- Alternative: **selektives Acknowledgement**:
  - ACK für Frame n gilt nur für diesen Frame

Problem in der Praxis

- Begrenzte Anzahl von Bits für die Sequenznummer im Frame-Header
  - z.B. bei 3 Bits nur Nummern 0 ... 7 möglich
- Reicht ein endlicher Bereich an Sequenznummern aus?
  - ja, abhängig von SWS und RWS:
    - falls RWS = 1: NSeqNum $\geq$ SWS + 1
    - falls RWS = SWS: NSeqNum $\geq 2 \cdot$ SWS

    (NSeqNum = Anzahl von Sequenznummern)
  - aber nur, wenn die Reihenfolge der Frames bei der Übertragung nicht verändert werden kann!

7.5 Übertragungssicherung in TCP

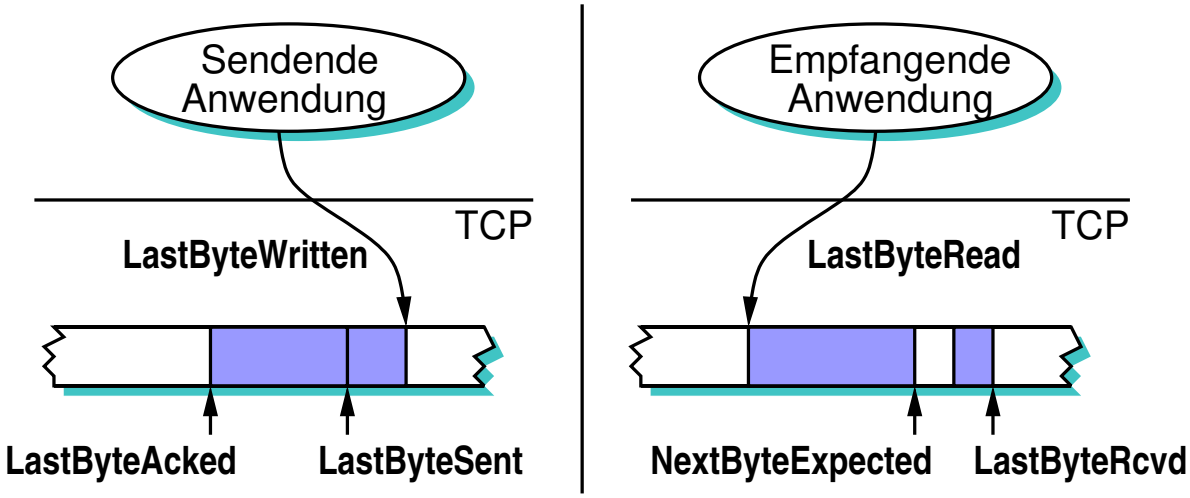
- TCP nutzt den *Sliding-Window*-Algorithmus
- Prinzipiell wie in **7.4.2** vorgestellt, aber Unterschiede:
  - Sequenznummer zählt Bytes, nicht Segmente
  - TCP benötigt Verbindungsaufbau und -abbau
    - Austausch der *Sliding-Window* Parameter
  - Netzwerk (IP) kann Pakete umordnen
    - TCP toleriert bis zu 120 Sekunden alte Pakete
  - Keine feste Fenstergröße
    - Sendefenstergröße angepasst an Puffer des Empfängers bzw. Lastsituation im Netz
  - RTT ist nicht konstant, sondern ändert sich laufend
    - Timeout muß adaptiv sein

Aufgaben des Sliding-Window-Algorithmus in TCP

- Zuverlässige Übertragung
- Sicherstellung der richtigen Reihenfolge der Segmente
  - TCP gibt Segmente nur dann an obere Schicht weiter, wenn alle vorherigen Segmente bestätigt wurden
- Flußkontrolle
  - keine feste Sendefenstergröße
  - Empfänger teilt dem Sender den freien Pufferplatz mit (**AdvertisedWindow**)
  - Sender passt Sendefenstergröße entsprechend an
- Überlastkontrolle
  - Sendefenstergröße wird dynamisch an Netzlast angepasst

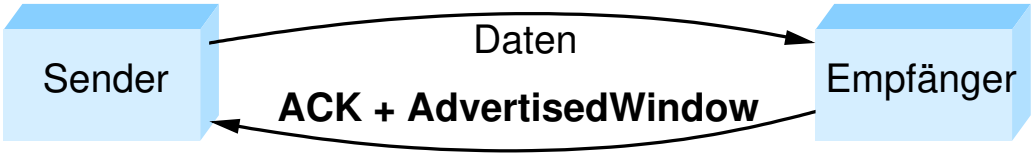
Zuverlässige und geordnete Übertragung

- Algorithmus arbeitet auf Byte-Ebene
  - Sequenznummern werden um die Anzahl gesendeter bzw. empfangener Bytes erhöht



Flußkontrolle

- Empfänger teilt Sender die Größe des freien Puffers mit:



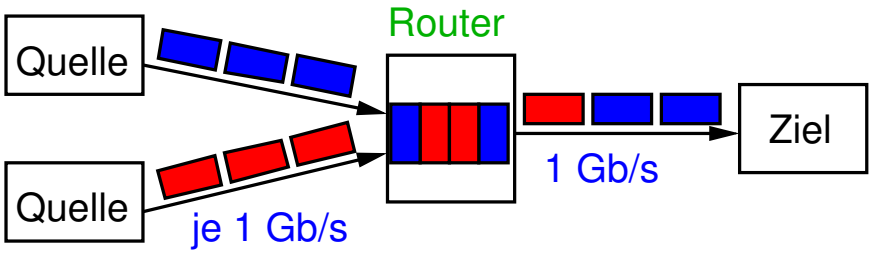
  - **AdvertisedWindow** $=$ **MaxRcvBuffer** $-$ **(LastByteRcvd** $-$ **LastByteRead)**
- Sender muß sicherstellen, daß jederzeit gilt:
  - **LastByteSent** $-$ **LastByteAcked** $\leq$ **AdvertisedWindow**
- Differenz: Datenmenge, die der Sender noch senden kann
- Sendende Anwendung wird blockiert, wenn Daten (**y** Bytes) nicht mehr in Sendepuffer passen, d.h. wenn
  - **LastByteWritten** $-$ **LastByteAcked** $+$ **y** $>$ **MaxSendBuffer**

Anmerkungen zu Folie 228:

Die Datenmenge, die der Sender noch senden kann, ist
EffectiveWindow $=$ **AdvertisedWindow** $-$ **(LastByteSent** $-$ **LastByteAcked)**

Überlastkontrolle

- **Flußkontrolle** verhindert, daß **ein Sender** seinen **Empfänger** überlastet
- **Überlastkontrolle** verhindert, daß **mehrere Sender** einen Teil des **Netzwerks** überlasten (durch Konkurrenz um Bandbreite):



  - bei unzureichender Bandbreite: Puffern der Pakete im Router
  - bei Pufferüberlauf: Router muß Pakete verwerfen
- Ein Netzwerk mit (häufigem) Pufferüberlauf heißt **überlastet** (*congested*)

Überlastkontrolle ...

- Analog zum **AdvertisedWindow** wird ein **CongestionWindow** eingeführt
  - Sender kann noch so viele Bytes senden, ohne das Netzwerk zu überlasten
- TCP beobachtet das Verhalten des Netzes (Lastsituation)
  - Paketverlust
  - ggf. auch RTT und Durchsatz
- Größe des **CongestionWindow** wird an Lastsituation angepasst
  - Fenster ist groß (bzw. wird vergrößert) bei geringer Last
  - Fenster wird (drastisch) verkleinert bei Anzeichen einer hohen Last (z.B. bei Paketverlust)

Sequenznummern-Überlauf

- Erinnerung an **7.4.2**: endlicher Seqenznummernbereich nur möglich, wenn Netzwerk die Reihenfolge erhält
- TCP-Header: 32-Bit Feld für Sequenznummern
- Pakete können bis zu 120 Sekunden alt werden

Bandbreite	Zeit bis zum Überlauf
10 MBit/s (Ethernet)	57 Minuten
100 MBit/s (FDDI)	6 Minuten
155 MBit/s (OC–3)	4 Minuten
1,2 GBit/s (OC–24)	28 Sekunden
9,95 GBit/s (OC–192)	3,4 Sekunden

- ⇒ TCP-Erweiterung: Zeitstempel als Überlaufschutz

Größe des AdvertisedWindow

- TCP-Header sieht 16-Bit vor, d.h. max. 64 KBytes
- Nötige Sendefenster-Größe, um Kanal gefüllt zu halten, bei RTT = 100 ms (z.B. Transatlantik-Verbindung):

Bandbreite	RTT * Bandbreite
10 MBit/s (Ethernet)	122 KByte
100 MBit/s (FDDI)	1,2 MByte
155 MBit/s (OC–3)	1,8 MByte
1,2 GBit/s (OC–24)	14,8 MByte
9,95 GBit/s (OC–192)	119 MByte

- ⇒ TCP-Erweiterung: Skalierungsfaktor für **AdvertisedWindow**

Adaptive Neuübertragung

- Timeout für Neuübertragung muß abhängig von RTT gewählt werden
- Im Internet: RTT ist unterschiedlich und veränderlich
- Daher: adaptive Bestimmung des Timeouts nötig
  - ursprünglich:
    - Messung der durchschnittlichen RTT (Zeit zwischen Senden eines Segments und Ankunft des ACK)
    - Timeout = 2 · durchschnittliche RTT
  - Problem:
    - Varianz der RTT-Meßwerte nicht berücksichtigt
    - bei hoher Varianz sollte der Timeout deutlich über dem Mittelwert liegen

Adaptive Neuübertragung: Jacobson/Karels-Algorithmus

- Berechne gleitenden Mittelwert und (approximierte) Standard-abweichung der RTT:
  - $\textbf{Deviation} = \delta \cdot |\textbf{SampleRTT} - \textbf{EstimatedRTT}| + (1-\delta) \cdot \textbf{Deviation}$
  - $\textbf{EstimatedRTT} = \delta \cdot \textbf{SampleRTT} + (1-\delta) \cdot \textbf{EstimatedRTT}$
- Berücksichtige Standardabweichung bei Timeout-Berechnung:
  - $\textbf{TimeOut} = \mu \cdot \textbf{EstimatedRTT} + \Phi \cdot \textbf{Deviation}$
- Typisch: μ =1, Φ = 4, δ = 0,125

7.6 Zusammenfassung

- Ende-zu-Ende Protokolle: Kommunikation zwischen Prozessen
- UDP: unzuverlässige Übertragung von Datagrammen
- TCP: zuverlässige Übertragung von Byte-Strömen
  - Verbindungsaufbau
- Sicherung der Übertragung allgemein
  - *Stop-and-Wait*, *Sliding-Window*
- Übertragungssicherung in TCP (inkl. Fluß- und Überlastkontrolle)
  - *Sliding-Window*-Algorithms, adaptive Neuübertragung

Nächste Lektion:

- Datendarstellung