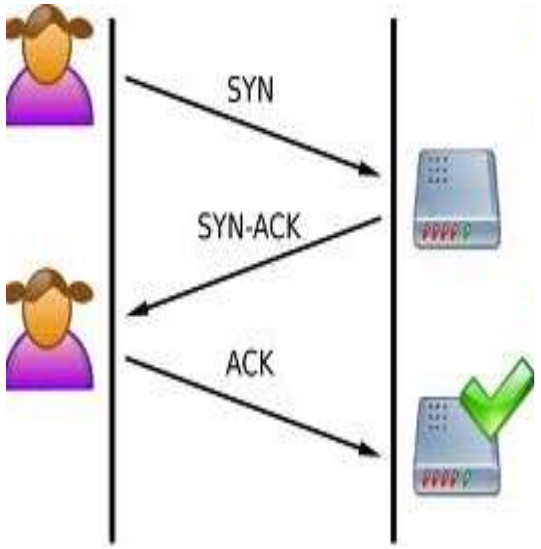
Rechnernetze

Kapitel 6: Die Transportschicht

Hochschule Ulm
Prof. Dr. F. Steiper



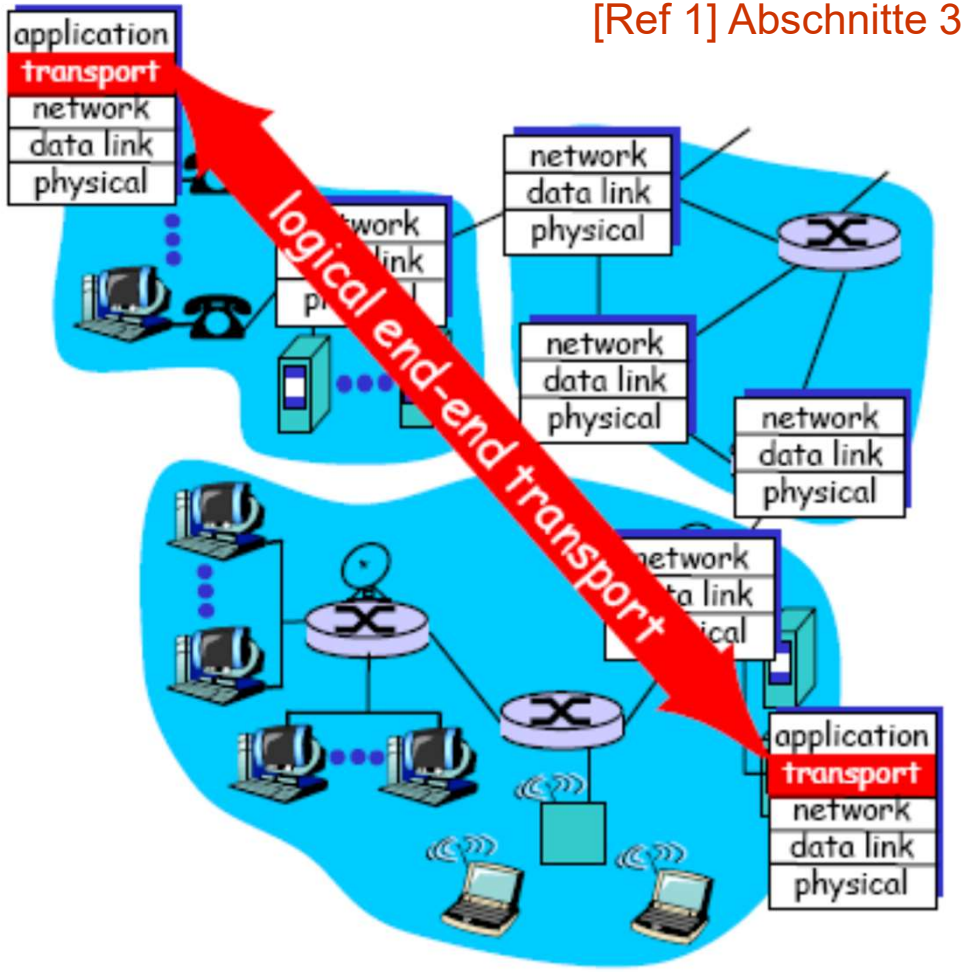
Rechnernetze, INF2, 2022

- *Urheberrechte*
  - *Die Vorlesungsmaterialien und Vorlesungsaufzeichnungen zum Kurs „Rechnernetze (INF2)" dürfen nur für private Zwecke im Rahmen Ihres Studiums an der Technischen Hochschule Ulm genutzt werden.*
  - *Eine Vervielfältigung und Weitergabe dieser Materialien in jeglicher Form an andere Personen ist untersagt.*
  - *© Copyright. Frank Steiper. 2022. All rights reserved*

6.1 Aufgaben und Dienste

- *Logische Kommunikation zwischen Anwendungen*

[Ref 1] Abschnitte 3.1, Seite 227-232



6.1 Aufgaben und Dienste

- *User Datagram Protocol*
  - *verbindungslos*
    - *Verfahren ähnlich Postzustellung*
  - *unzuverlässig*
    - *optionale Fehlererkennung*
    - *keine Fehlerbehebung*
    - *keine Erhaltung der Reihenfolge*
    - *keine Senderatenregulierung*
  - *Nachrichten orientiert*
    - *inkl. Header 2^{16} Bytes max. UDP-Segment-Länge; d.h. es exist. max. Nachrichtenlänge*
  - *Multicast-Unterstützung*
- *Transmission Control Protocol*
  - *verbindungsorientiert*
    - *Verfahren ähnlich Telefonverbindung*
  - *zuverlässig*
    - *bietet Fehlererkennung*
    - *bietet Fehlerbehebung*
    - *garantiert Erhaltung der Reihenfolge*
    - *bietet Überlastkontrolle*
  - *Byte-Strom orientiert*
    - *puffert unstrukturierte Byteströme und bildet daraus TCP-Segmente*
  - *unterstützt nur 1:1-Kommunikation*

6.1 Aufgaben und Dienste

- *Port-Nummern*
  - *Problem*
    - *Die Ziel-Adresse eines IP-Pakets bestimmt eindeutig den Zielrechner*
    - *Wie wird festgelegt, für welche Anwendung ein IP-Paket bestimmt ist?*
  - *Lösung*
    - *Einführung von Port-Nummern*
    - *Portnummern sind 16 Bit lange Integerzahlen*
      - → *Portnummern können im Bereich 0 bis 65535 liegen*
      - → *Die Portnummer 0 ist reserviert und wird nicht vergeben*
    - *Port-Nummern werden für UDP und TCP getrennt verwaltet*
      - → *UDP- und TCP-Verbindungen können zur gleichen Zeit gleiche Portnummern nutzen*

6.1 Aufgaben und Dienste

- *Registrierung von Port-Nummern*
  - *Geschieht durch die IANA (Internet Assigned Numbers Authority)*
    - *http://www.iana.org/assignments/port-numbers*
- *Festlegungen*
  - *Die Portnummern ≤1023 sind fest reserviert: "Well-Known"-Ports*
    - *Die Portnummer bestimmt das Anwendungsprotokoll*
    - *Z.B.:*

telnet:	*23/tcp*	*(TCP-Verb. Remote-Host);*
ftp-data:	*20/tcp ,ftp-control: 21/tcp*	*(File-Transfer);*
http:	*80/tcp*	*(WWW);*
smtp:	*25/tcp*	*(Email);*
dhcp-server:	*68/udp,dhcp-client 67/udp*	*(autom. IP-Konfig.);*
dns-server:	*53/tcp,udp*	*(IP-Adr. ↔ Rechnername);*

  - *Die Ports 1024 - 49151 heißen „Registered"-Ports*
    - *Hersteller von Anwendungen lassen diese unter einer Port-Nummer registrieren*
    - *Diese Ports sind damit jedoch nicht reserviert!*
    - *Diese Bekanntgabe hat nur informativen Charakter*
  - *Die Ports 49152-65535 heißen „Dynamic"- bzw. „Private"-Ports*
    - *Diese Ports sind für jeden Anwender ohne Einschränkung nutzbar*

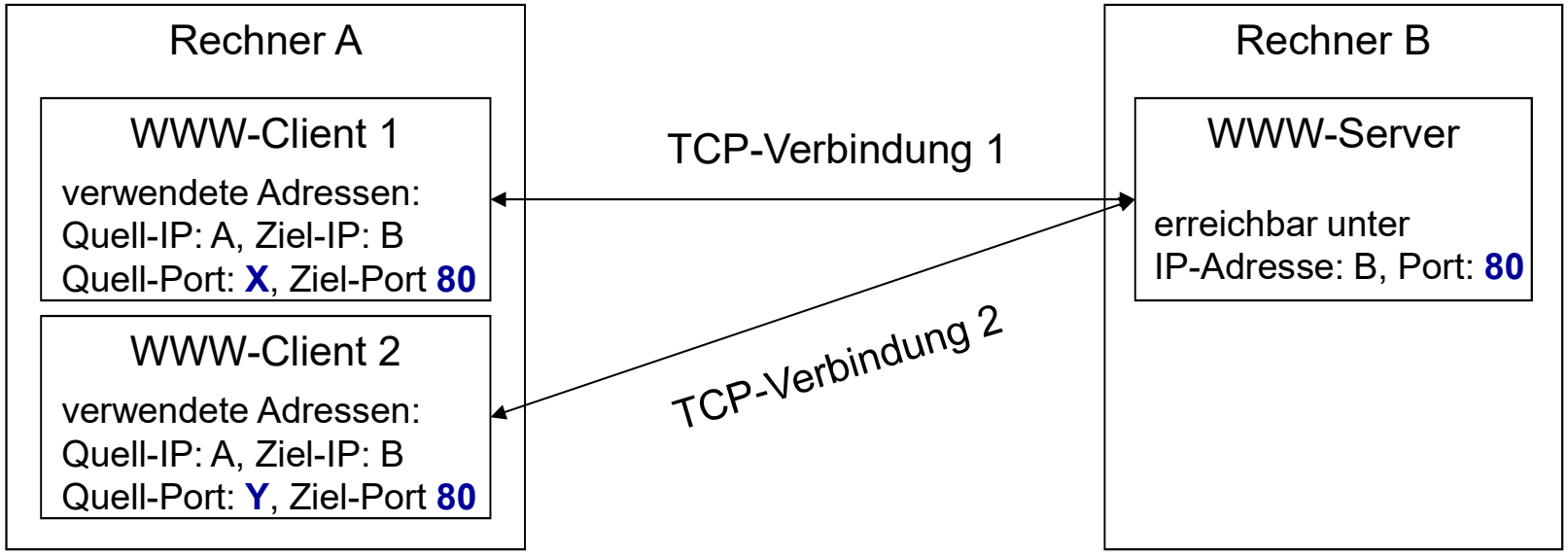
6.1 Aufgaben und Dienste

- *Port-Verwaltung bei Linux/Windows-Betriebssystemen*
  - *Ports 1-1023 reserviert für „root“ (Superuser): “privilegierte” Ports*
    - *Nur “root” kann z.B. einen WEB-Service an Port 80 binden*
    - *Unter Windows gibt es das Konzept privilegierter Ports nicht*
  - *Ephemeral (dynamic) port addresses*
    - *Betriebssystem ordnet einer Applikation eine Port-Nummer temporär aus diesem Portnummern-Bereich zu*
    - *Viele Linux-Varianten nutzen den Bereich 32768–60999*

- *Beispiele zur Verwendung von Port-Nummern*
  - *TCP-Verbindung mit DNS-Server über Zielport 53 öffnen*
    - *„telnet <dns-server-name/ip-addr> 53“*
    - *Test, ob DNS-Server unter Port 53 erreichbar ist*
  - *Email über smtp (port 25, tcp) an einen Mail-Server senden*
    - *„telnet <smtp-server-name/ip-addr> 25“*
    - *ASCII orientiertes Protokoll, kann per Hand-Eingabe emuliert werden*

- *Beispiel: Nutzung von Portnummern*
  - *Zwei WEB-Clients laufen auf gleichem Rechner*
    - *Aus Quell-Port-Nummer kann WWW-Server die Client-Applikation identifizieren*

      → *An diese Quell-Portnummer und zugehörige Quell-IP-Adresse schickt WWW-Server seine Antworten*



Es gilt: **X,Y > 1023; X≠Y**

6.2 User Datagram Protocol

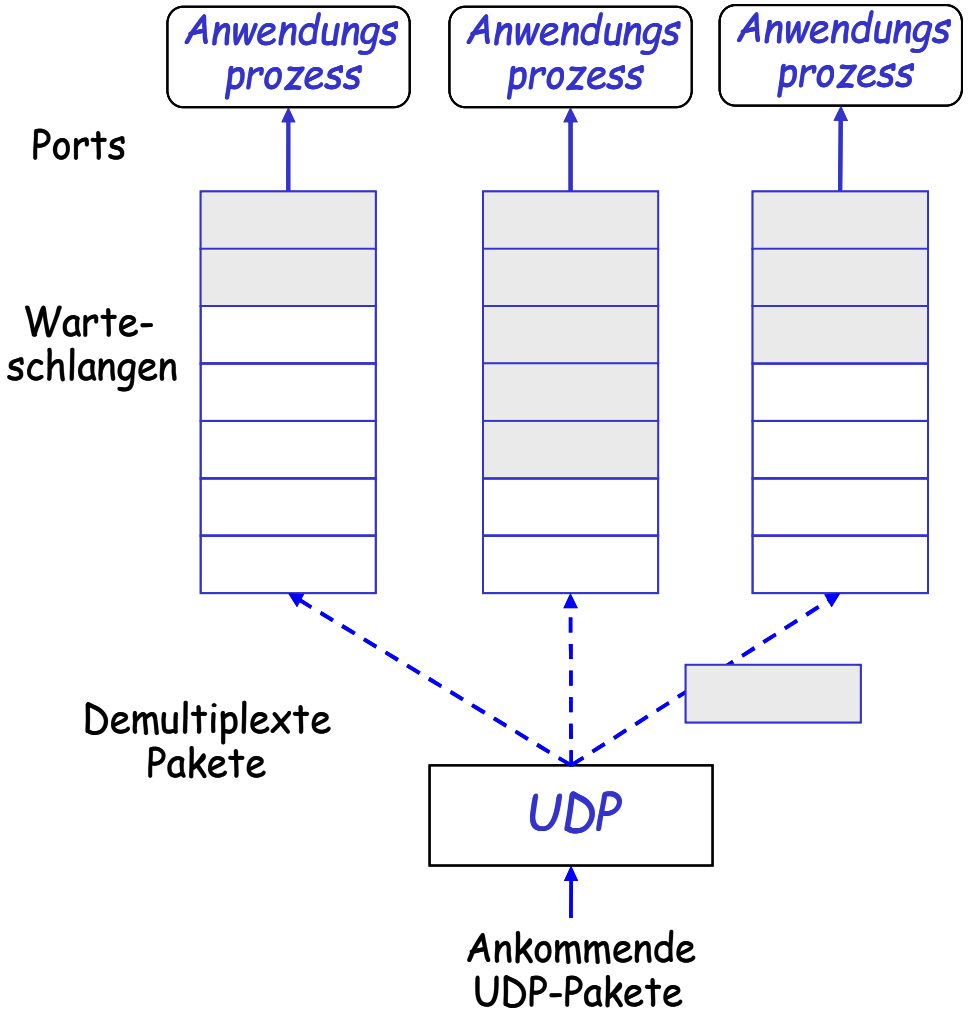
- *UDP (User Datagram Protocol)*

[Ref 1] Abschnitt 3.3, Seite 239-245
[Ref 2] Abschnitt 6.4, Seite 616-618

  - *UDP liefert einfachen Demultiplexer-Dienst auf Basis von Port-Nummern*
    - *Erweitert „Host-zu-Host“-Kommunikation der Vermittlungsschicht zur „Prozess-zu-Prozess“-Kommunikation*
  - *Nutzt optionale Prüfsumme zur Fehlererkennung*
    - *Teile des IP-Headers gehen mit in die Prüfsumme ein*
      - → *Wird als UDP-Pseudoheader bezeichnet*
      - → *Keine real übertragenen Daten im UDP-Header!*
    - *Zweck*
      - → *Falsch-Zustellung von IP-Paketen verhindern, falls ein Router fehlerhaft arbeitet (z.B. IP-Adressen verfälscht)*
      - → *Unerkannte Datenverluste erkennen, da die IP-Segmentlänge im Pseudo-Header steht*
      - → *Anm.: Für TCP existiert analog ein TCP-Pseudo-Header*

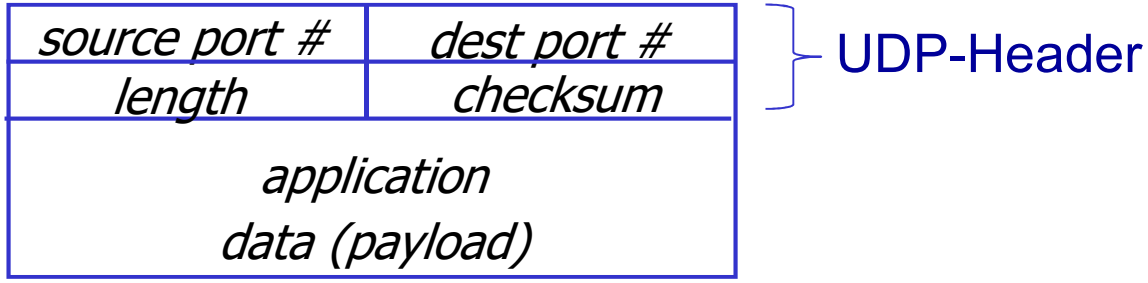
6.2 User Datagram Protocol

- *Ports werden durch eine Nachrichtenwarteschlange implementiert*
  - *Kommt ein UDP-Segment an, wird es entsprechend der Zielportnummer in die Warteschlange eingestellt*
  - *Ist die Warteschlange voll, wird das Segment verworfen (keine Flusskontrolle)*
  - *Will ein Anwendungsprozess eine Nachricht empfangen, wird diese vom Kopf der Warteschlange gelesen*
  - *Ist eine Warteschlange leer, blockiert der zugeordnete Anwendungsprozess, bis ein neues Segment verfügbar ist*

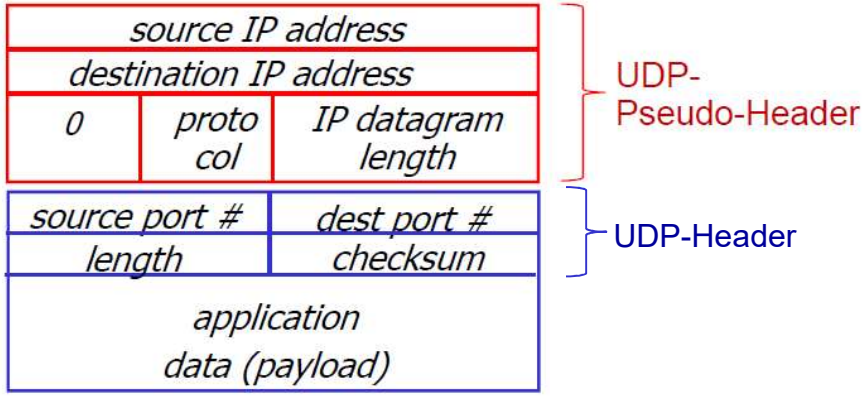


6.2 User Datagram Protocol

- *Aufbau eines UDP-Pakets*



- *Teile des IP-Headers werden als UDP- Pseudo-Header bezeichnet:*
  - *Seine Einträge gehen in die UDP-Prüfsumme ein*
  - *Der Pseudo-Header wird jedoch nicht real übertragen*



6.2 User Datagram Protocol

- *Vorteile von UDP*
  - *Kein expliziter Verbindungsaufbau*
    - *Keine Verzögerung beim Verbindungsaufbau (wichtig z.B. bei DNS)*
  - *Kein Verbindungszustand*
    - *Große Anzahl aktiver Clients pro Server möglich (keine Reservierung von Sende- und Empfangspuffern pro Verbindung, keine Sequenz- und Bestätigungsnummern...)*
  - *Minimaler Overhead*
    - *Wenig Header-Daten pro Segment*
  - *Unregulierte Senderate*
    - *Die Senderate nur durch Fähigkeiten des Senders und durch die Bitrate des Netzes begrenzt (wichtig für Echtzeit-Anwendungen)*
    - *Zu beachten: Es können jedoch Daten verloren gehen, eine Netzüberlast kann die Empfangsrate stark beeinträchtigen*

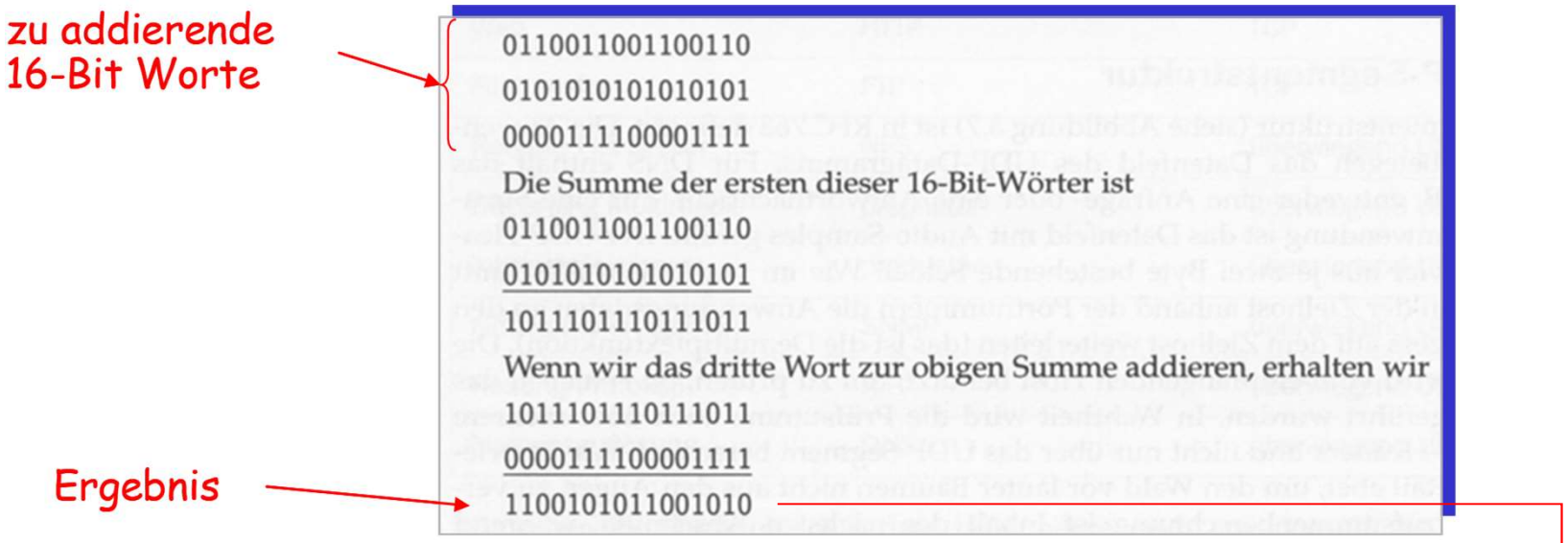
6.2 User Datagram Protocol

- *Die UDP-Prüfsumme*
  - *Berechnung auf Senderseite*
    - *Bildung des Einer-Komplements der Summe aller im Header, im Pseudo-Header und im Datenfeld enthaltenen 16-Bit Worte*
    - *Das Ergebnis wird als „Checksum“ in das UDP-Segment eingefügt*
  - *Auswertung auf Empfängerseite*
    - *Alle 16-Bit Worte und mitgelieferte Checksum werden addiert.*
    - *Falls fehlerfreie Übertragung vorliegt: Ergebnis muss in allen 16 Bit-Stellen „1“ ergeben*
  - *Anm.: Berechnungsschema wird allg. „Internet-Prüfsumme“ genannt*
    - *Für IP, TCP und UDP wird diese Prüfsumme eingesetzt*
    - *Fehler-Erkennungsgrad ist relativ schlecht*

      → Insbesondere bei systematischen Fehlern für gleiche Bitposition

6.2 User Datagram Protocol

- *Ablauf der Prüfsummenberechnung*
  - *1. Schritt Senderseite: Addition (z.B. hier drei 16-Bit-Wörter)*



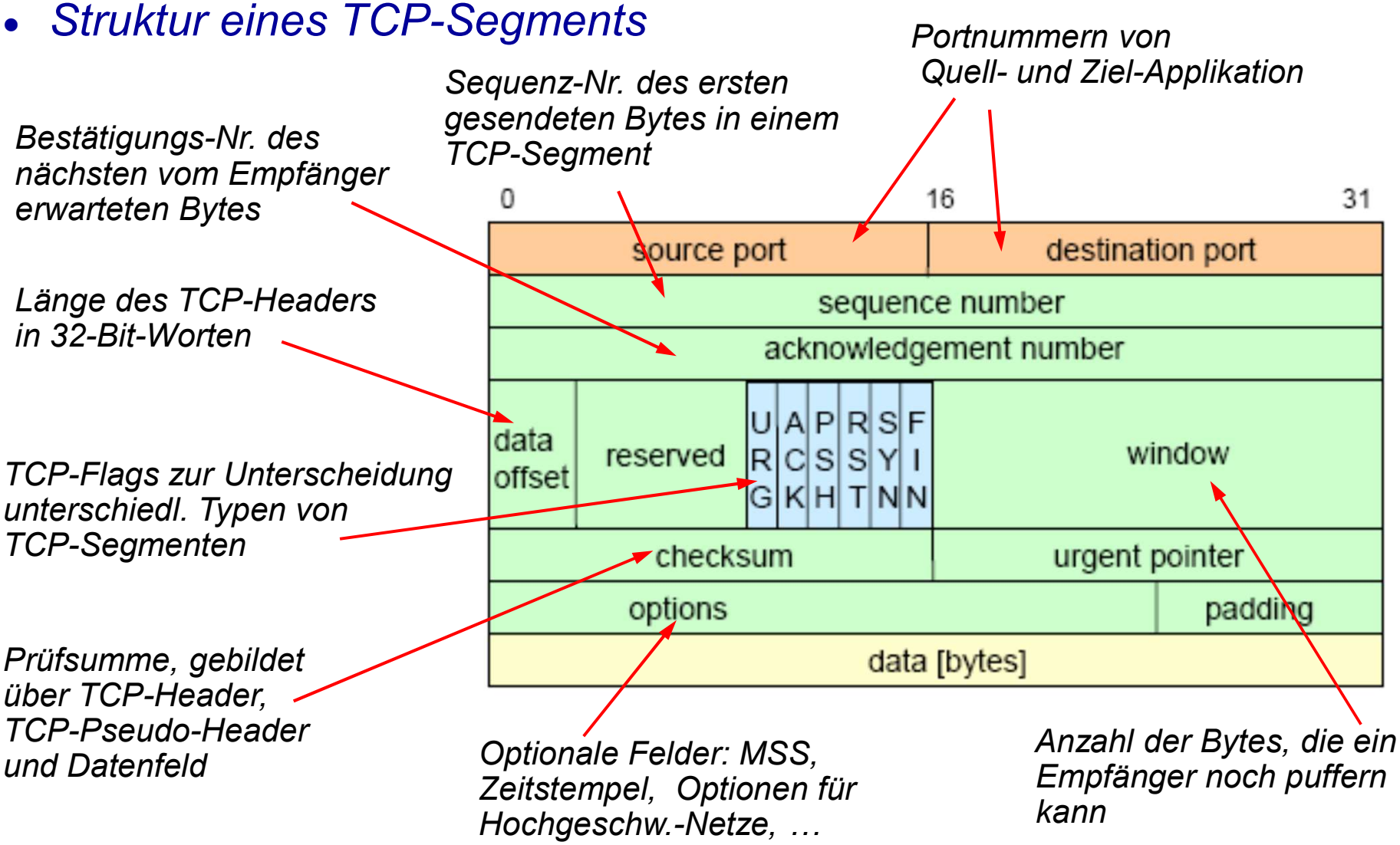
  - *2. Schritt Senderseite: Aus dem Ergebnis wird Einer-Komplement gebildet, also: 0011010100110101 (=Prüfsumme)*
  - *Empfängerseite: Addition aller 3 16-Bit-Wörter und der Prüfsumme; Ergebnis muss 1111111111111111 sein, falls keine Fehler vorliegen*

6.3 Transmission Control Protocol

- *TCP (Transmission Control Protocol)* [Ref 1] Abschnitt 3.5, Seite 272-280
  [Ref 2] Abschnitt 6.5, Seite 628-639
  - *Garantiert eine fehlergesicherte, zuverlässige Transport-Verbindung zwischen Verbindungsendpunkten (Sockets)*
  - *Eigenschaften von TCP*
    - *Gesicherter Verbindungs-Aufbau und -Abbau*
    - *„Full-Duplex“-Kommunikation über virtuelle Verbindungen*
    - *Einhaltung der Segment-Reihenfolge*
    - *Fluss- und Staukontrolle mittels Fenstermechanismen*
    - *Fehlerkontrolle*
      → *Folgenummern, Prüfsummen, Quittierungsnummern, Übertragungswiederholungen*
    - *Unterstützung priorisierter Daten*
    - *„Demultiplex“-Dienst*

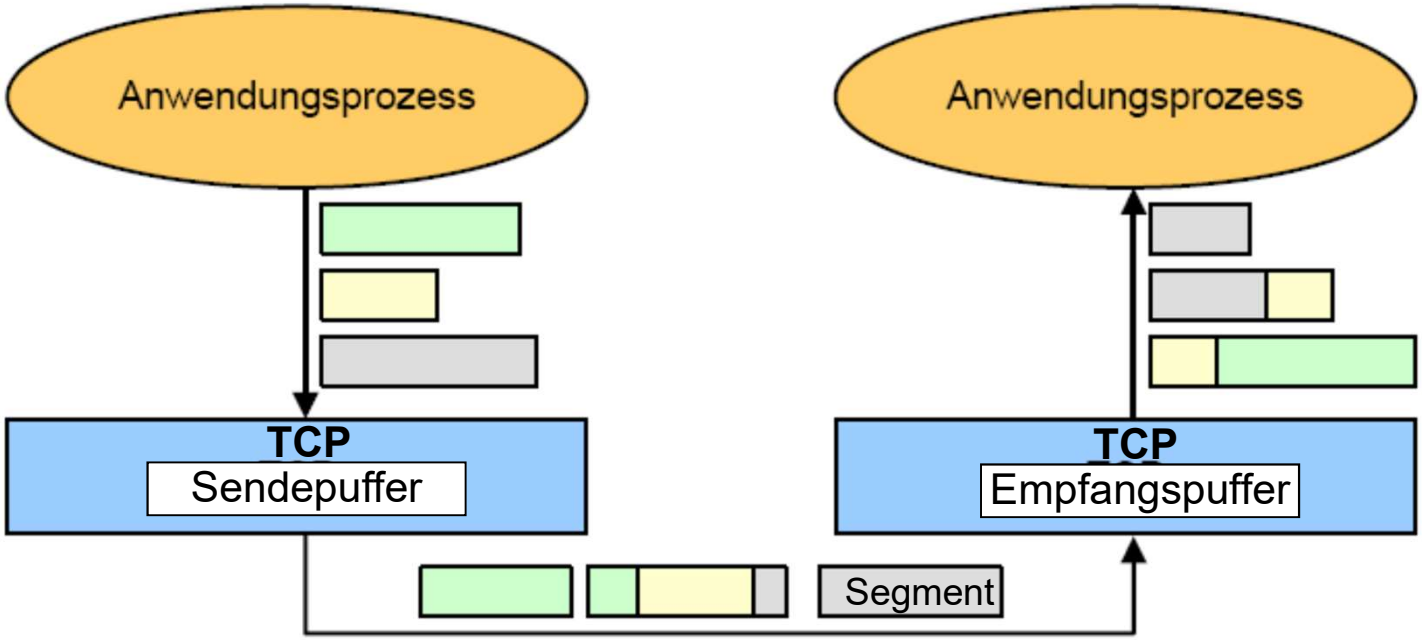
6.3 Transmission Control Protocol

- *Struktur eines TCP-Segments*



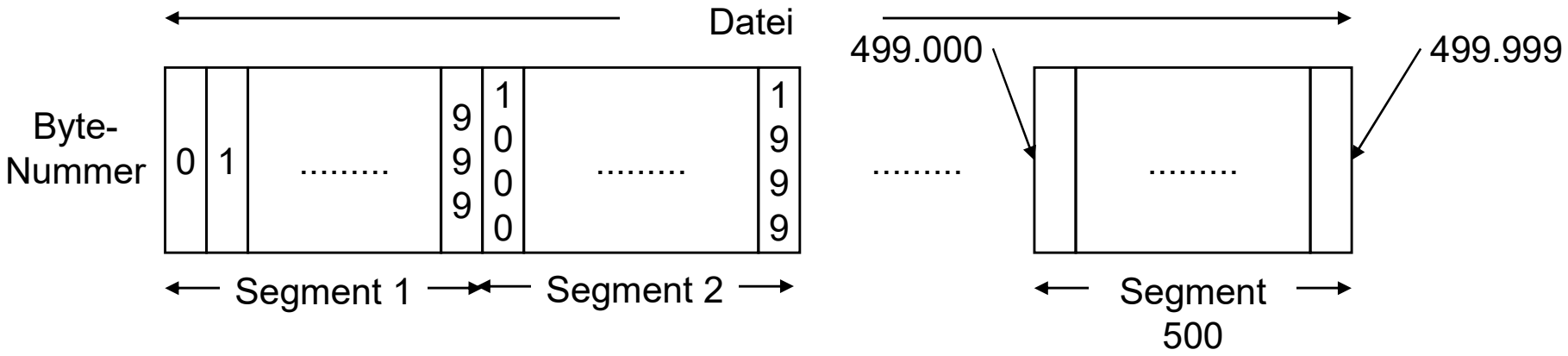
6.3 Transmission Control Protocol

- *TCP ist Byte orientiert*
  - *Anwendungsprozesse senden und empfangen Byteströme*
    - *TCP puffert Byteströme, bildet Segmente und überträgt diese*
    - *Ein Empfänger puffert Segmente und liest diese zum gewünschten Zeitpunkt aus*



6.3 Transmission Control Protocol

- *Ablauf der TCP-Kommunikation*
  - *Daten werden Bytestrom orientiert übertragen*
    - *Sequenz- und Bestätigungsnummern:*
      - Spiegeln den Bytestrom, aber nicht die Serie der gesendeten Segmente wieder
  - *MSS (Maximum Segment Size)*
    - *Maximale Datenmenge in Bytes, die ein Segment transportieren kann*
    - *Orientiert sich an MTU des eingesetzten LAN-Protokolls, um Fragmentierung zu vermeiden*

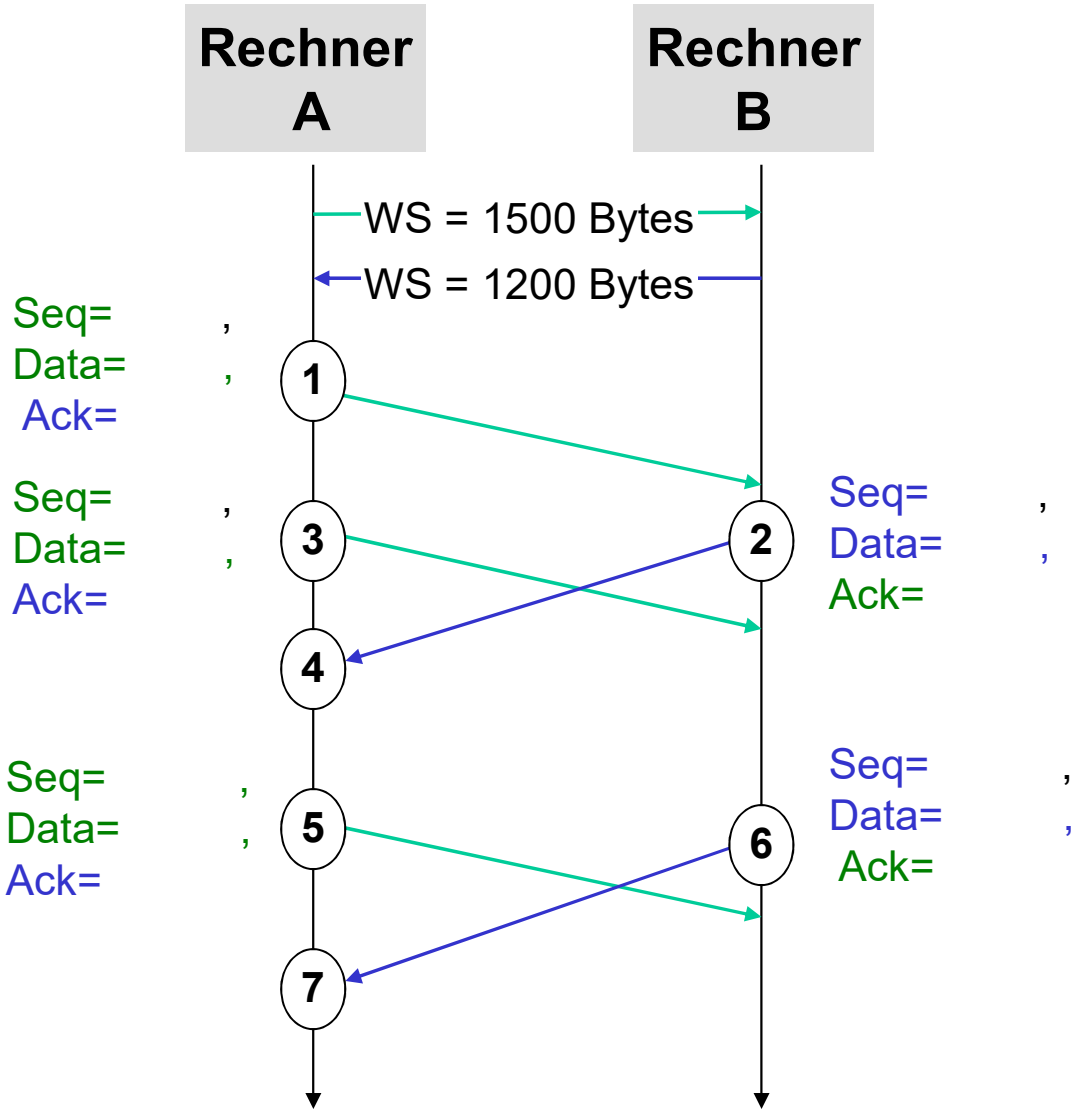


6.3 Transmission Control Protocol

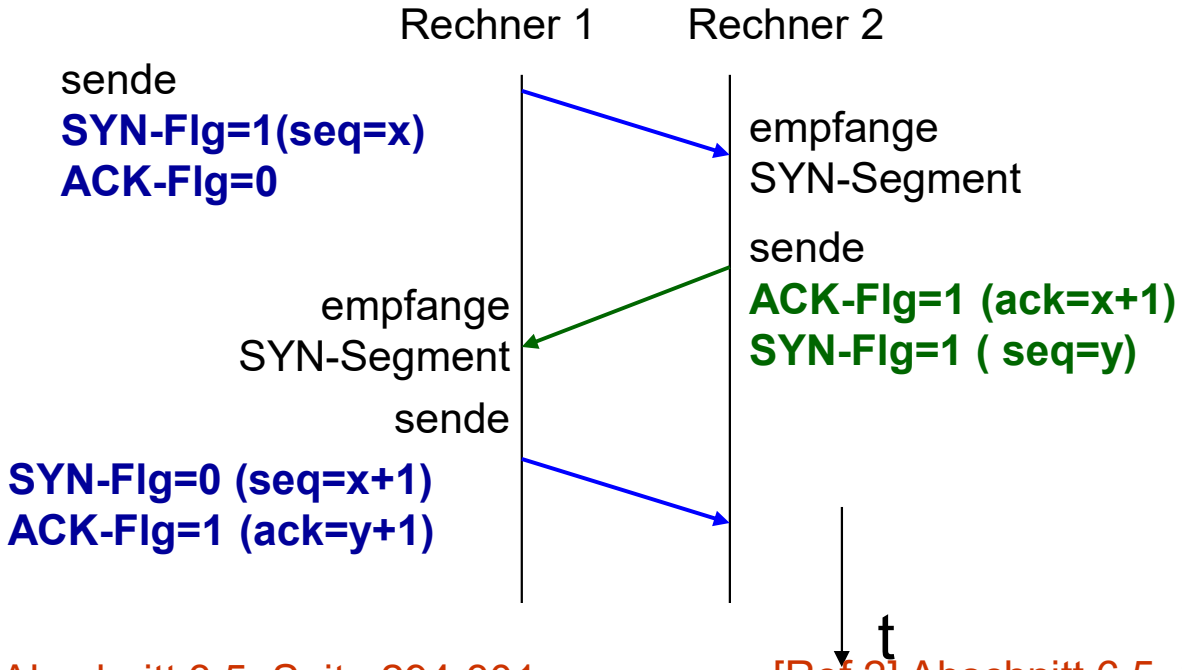
- *TCP: Definition der Sequenznummer*
  - *Ist die Bytenummer des ersten Bytes eines versandten Segments*
    - *Jedes Byte im Datenstrom wird von TCP nummeriert*
    - *Für Beispiel: 1. Segment: Seq=0*
      2. Segment: Seq=1000 usw.
- *TCP: Definition der Bestätigungsnummer*
  - *Ist die Bytenummer des nächsten Bytes, das der Empfänger vom Sender erwartet*

6.3 Transmission Control Protocol

- *Ablauf der Datenüber-tragung bei TCP*
  - *Die Window-Größe wird in Bytes angegeben und entspricht anfänglich der Empfangspuffergröße*
  - *Später entspricht die Window-Größe der max. Anzahl Bytes, die vom Empfänger noch zwischen gepuffert werden können*
  - *Eine TCP-Instanz kann in einem TCP-Segment gleichzeitig Daten verschicken und empfangene Daten quittieren*



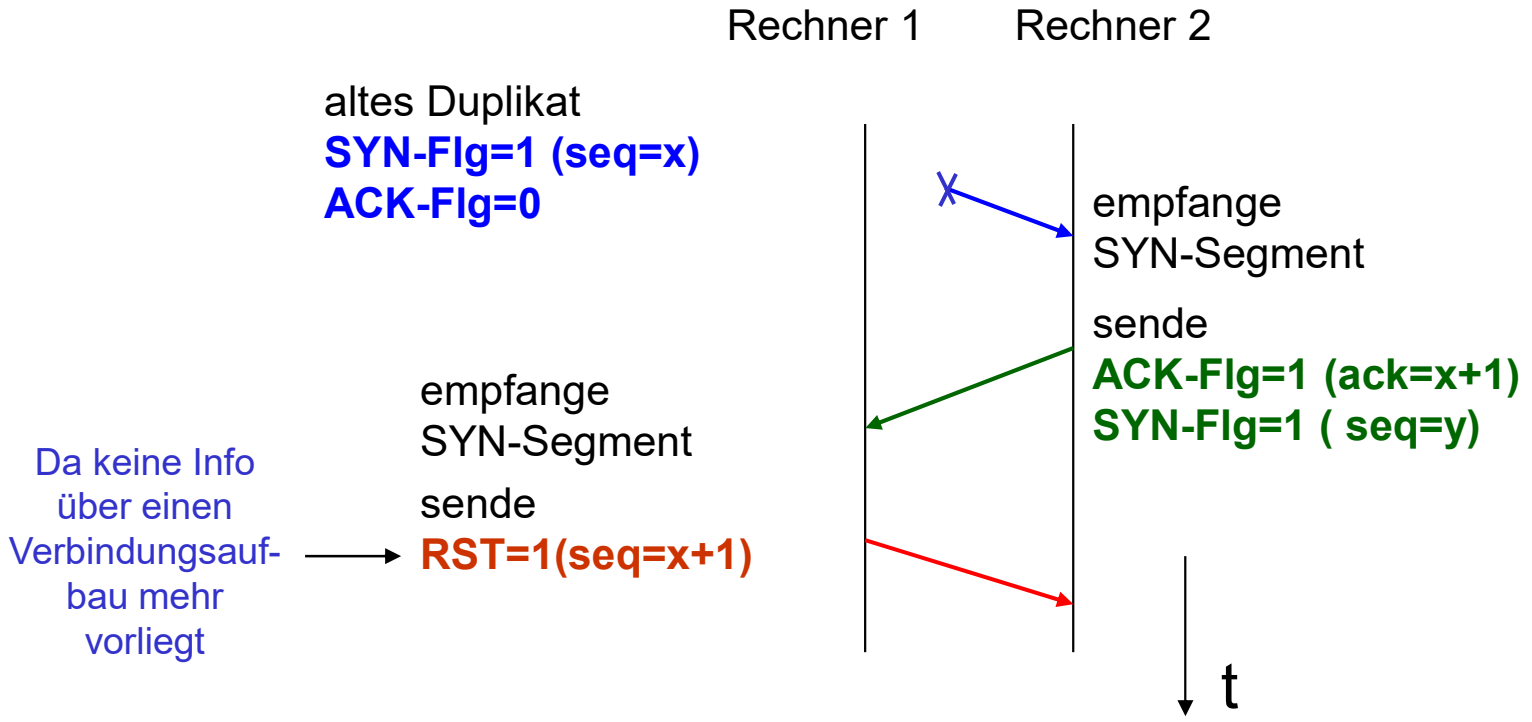
- *TCP-Verbindungsaufbau: „Three-Way-Handshake“*
  - *Problem: Das Netz kann Daten speichern, duplizieren oder verlieren*
  - *Es muss auch mit „verzögerten Duplikaten“ gerechnet werden*
  - *Deshalb wurde die „Three-Way-Handshake“-Methode eingeführt:*



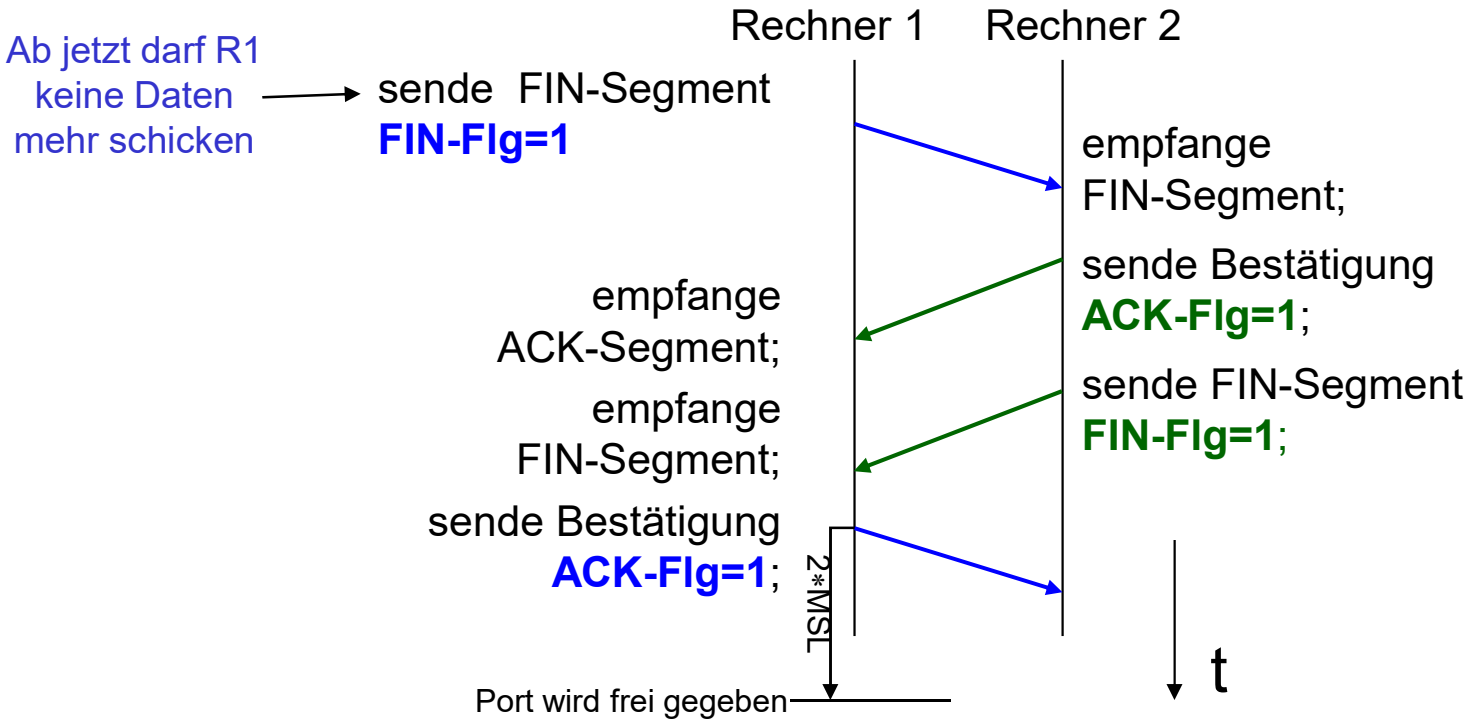
[Ref 1] Abschnitt 3.5, Seite 294-301

[Ref 2] Abschnitt 6.5, Seite 637-638

- *Beispiel: Verzögertes Duplikat bei einem TCP-Verbindungsaufbau*
  - *Altes Duplikat kommt bei Rechner 2 ohne Wissen von Rechner 1 an*
  - *Rechner 2 sendet Rechner 1 eine Verbindungsaufbau-Bestätigung*
  - *Rechner 1 erkennt, dass von ihm keine gültige Verbindungsaufbau-Anforderung ausging und weist Anforderung von Rechner 2 ab*



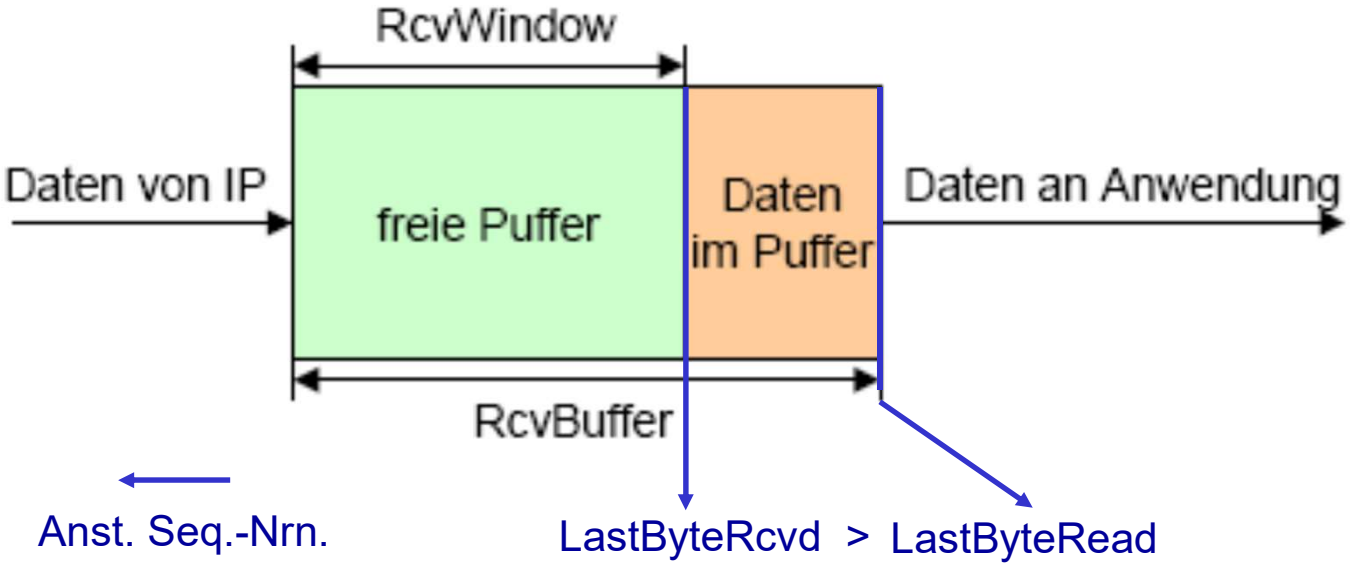
- *TCP-Verbindungsabbau*
  - *Der Abbau durch nur eine Seite kann zu Datenverlust führen*
    - *Deshalb ist beiderseitige Bestätigung des Verbindungsabbaus nötig*
    - *Dies wird als hinreichend sicher gewertet, da kein 100% sicheres Protokoll existiert; zusätzlich wird ein Timout-Mechanismus eingesetzt*



- *TCP-Flusskontrolle im Detail* [Ref 1] Abschnitt 3.5, Seite 292-294

  - *Auf Empfängerseite*
    - *Berechnung des aktuellen Empfangsfensters „RcvWindow"*
      - *→ Gibt an, über wie viel freien Pufferplatz der Empfänger aktuell verfügt*
    - *Bestimmung des aktuellen Empfangsfensters:*
      - *→ LastByteRead: Nummer des letzten Bytes, das Empfänger-Prozess aus Empfangspuffer gelesen hat*
      - *→ LastByteRcvd: Nummer des letzten Bytes, das im Empfangspuffer angekommen ist*
    - *Für die Empfangspuffergröße RcvBuffer muss immer gelten*
      LastByteRcvd - LastByteRead <= RcvBuffer
    - *Für Empfangsfenster RcvWindow gilt:*
      RcvWindow = RcvBuffer - (LastByteRcvd - LastByteRead)

- *Berechnung des aktuellen Empfangsfensters: RcvWindow*



- *TCP-Flusskontrolle…*
  - *Auf Senderseite*
    - *Empfänger teilt in jedem an Sender gerichteten Segment aktuellen Wert von RcvWindow mit*
      - *→ Zu Beginn gilt: RcvWindow = RcvBuffer des Empfängers*
      - *→ Dieser Wert wird dem Sender beim Verbindungsaufbau mitgeteilt*
    - *Sender verwaltet zwei Variablen:*
      - *→ LastByteSend (letztes gesendetes Byte)*
      - *→ LastByteAcked (letztes vom Empfänger bestätigtes Byte)*
    - *Es gilt:*
      - *→ LastByteSend - LastByteAcked = Menge der unbestätigten Daten*
    - *Sender verhält sich so, dass zu jedem Zeitpunkt gilt:*
      LastByteSend - LastByteAcked <= RcvWindow

6.3 Transmission Control Protocol

- *TCP-Überlast-Kontrolle*

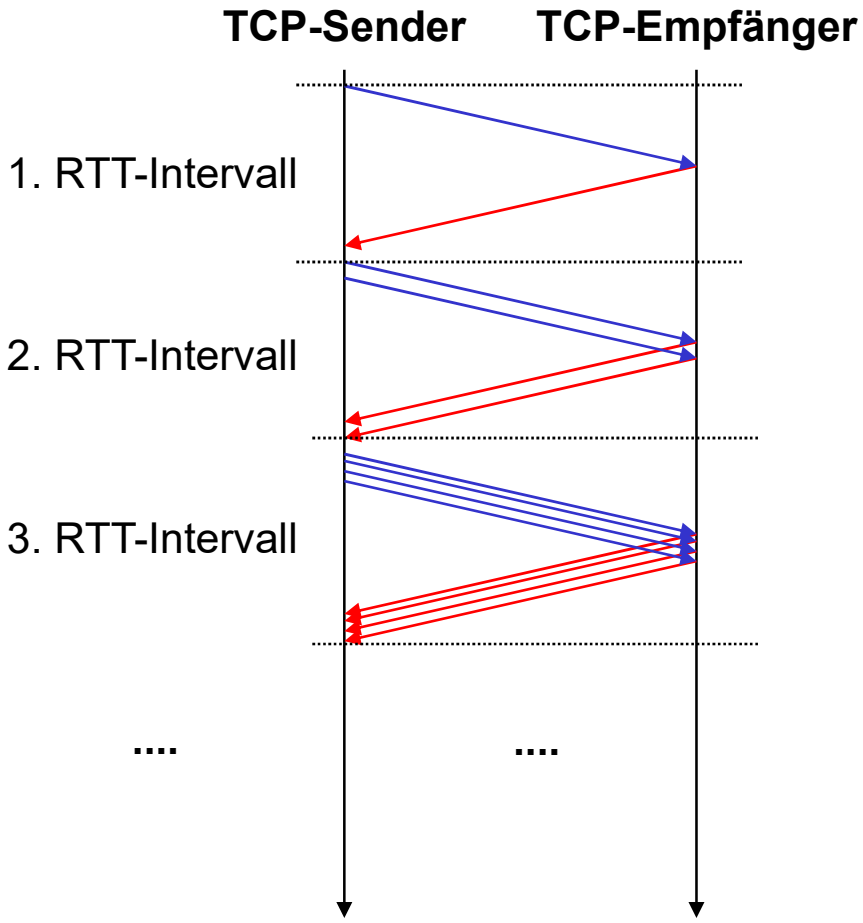
[Ref 1] Abschnitt 3.7, Seite 311-318
[Ref 2] Abschnitt 6.5, Seite 649-658

  - *Ansatz*
    - *Paket-Verluste/-Verzögerungen werden meist durch Überlastzustände in Routern verursacht*
    - *Sender muss ermitteln können, wie viel Übertragungskapazität zur Verfügung steht*
  - *Erweiterung des TCP-Übertragungsverfahrens*
    - *TCP-Sender verwaltet zwei zusätzliche Variablen: CongestionWindow und Threshold*
    - *Maximale Anzahl nicht bestätigter Bytes begrenzt durch MaxWindow= MIN(CongestionWindow, RcvWindow)*
  - *Annahme für weitere Betrachtungen*
    - *RcvWindow sei immer genügend groß, so dass immer gilt RcvWindow $\geq$ CongestionWindow*

- *TCP: Die „Slow-Start“-Phase*
  - *1. Fall: Direkt nach Verbindungsaufbau ist Threshold noch unbekannt*
    - *CongestionWindow=1•MSS Bytes (Max. Segment Size) setzen*
      - → *Also zu Beginn des 1. RTT(Round Trip Time)-Intervalls 1 Segment senden*
    - *Falls ACK rechtzeitig erfolgt: CongestionWindow verdoppeln**
      - → *Zu Beginn des nächsten RTT-Intervalls doppelte Anzahl Segmente senden*
    - *Falls ACKs für alle Segmente rechtzeitig eintreffen, weiter mit *)*
      - → *Generell: Verfahren wird fortgesetzt, bis ein Timeout erfolgt (ACK zu spät)*
    - *Bei Timeout:*
      - → *Threshold wird auf Hälfte des aktuellen CongestionWindows festgelegt*
      - → *CongestionWindow wird auf 1•MSS Bytes zurückgesetzt*
  - *2. Fall: Während einer laufenden Verbindung ist Threshold bekannt*
    - *Falls keine Timeouts:*
      - → *Ist MaxWindow < Threshold: Pro RTT-Intervall CongestionWindow verdoppeln*
      - → *Ist MaxWindow ≥ Threshold: weiter mit Congestion-Avoidance-Phase*
    - *Bei Timeouts:*
      - → *Weiter mit Multiple Decrease-Phase*

6.3 Transmission Control Protocol

- *TCP: Ablauf der TCP-Slow-Start-Phase*



- *TCP: „Congestion-Avoidance“-Phase*
  - *Falls MaxWindow Wert von Threshold erreicht:*
    - *Pro RTT-Intervall wird CongestionWindow nur linear um MSS Bytes vergrößert, falls kein Timeout erfolgt*
    - *Bei Timeouts: Multiple Decrease-Phase*
- *TCP: „Multiple Decrease“-Phase*
  - *Immer wenn in laufender Verbindung Timeouts auftreten*
    - *Threshold auf die Hälfte des akt. CongestionWindows setzen*
    - *Danach CongestionWindow auf 1•MSS Bytes setzen*
    - *Weiter mit TCP-Slow-Start-Phase*

- *Beispiel: TCP-Überlastkontrolle*
  - *Bis zum Zeitpunkt T=6*RTT keine Fehler, dann Timeout*
  - *T=7*RTT: Threshold wird auf 8*MSS Bytes festgelegt*
  - *Slow-Start-Phase von T=7*RTT bis T=10*RTT*
    - *CongestionWindow wächst exponentiell bis 8*MSS Bytes*
  - *Ab T=10*RTT Congestion Avoidance-Phase bis T=14*RTT*
    - *CongestionWindow wächst linear bis 12*MSS Bytes*
  - *T=14*RTT: Timeout*
    - *Mutiple Decrease Phase: Threshold auf 6*MSS Bytes reduziert; CongestionWindow auf 1*MSS Bytes gesetzt*
    - *Weiter mit Slow-Start-Phase*

6.3 Transmission Control Protocol

- *Beispiel: TCP-Überlastkontrolle…*