Telematik

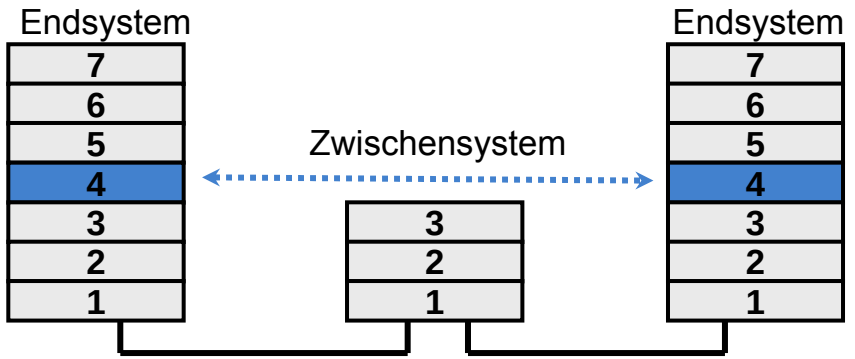
2. Ende-zu-Ende Datentransport

Prof. Dr. Martina Zitterbart
Dipl.-Inform. Thomas Gamer
Dipl.-Inform. Martin Röhricht
[zit | gamer | roehricht]@tm.uka.de

Einordnung Inhaltsverzeichnis

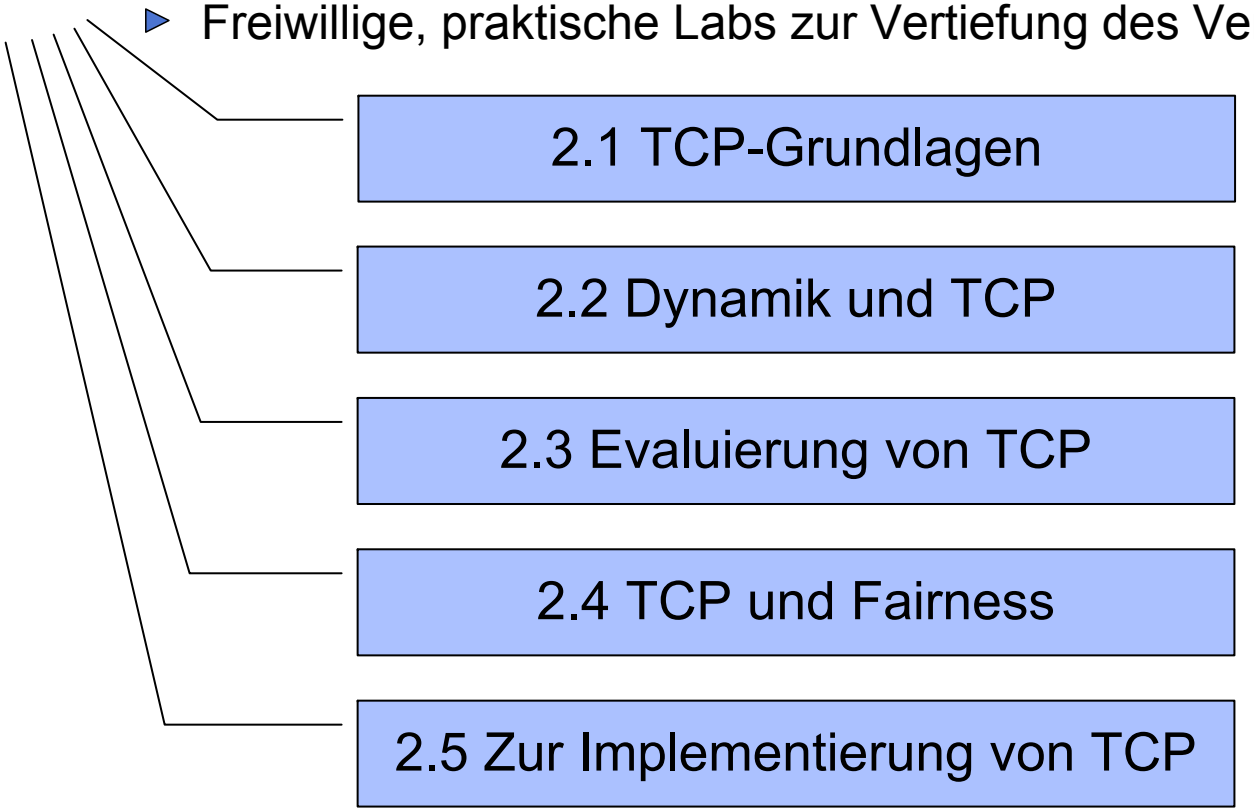
Einordnung in das ISO/OSI-Basisreferenzmodell

- TCP im Schichtenmodell
  - Schicht 4: Transportschicht (Transport Layer)
    - Bietet Anwendungen den Dienst der Übertragung von Daten
      - Ende-zu-Ende-Prinzip
      - Zuverlässiger und unzuverlässiger Dienst möglich
      - Der Anwendung als Dienstnehmer bleiben sämtliche Aspekte der Datenübertragung verborgen
    - Fehlererkennung und -behebung (zuverlässiger Dienst)
    - Adressierung von Transportdienstbenutzern
    - Pufferung
    - Multiplexen von Verbindungen



2. Ende-zu-Ende Datentransport

- Zielsetzung
  - Fundiertes Verständnis von Transportprotokollen und der darin verwendeten Algorithmen am Beispiel von TCP
    - Konzepte, Evaluierung und Implementierung werden berücksichtigt
    - Freiwillige, praktische Labs zur Vertiefung des Verständnisses



2.1 TCP-Grundlagen

- Auf der Basis des Internet-Standards RFC 793 [Post81]
- Grundlegende Aufgabe: Ende-zu-Ende Datentransfer im Internet
- Standardprotokolle im Internet
  - TCP (Transmission Control Protocol)
    - Bietet zuverlässigen, verbindungsorientierten Transportdienst
    - Lokalisiert über unzuverlässigem Dienst von IP (Internet Protocol)
    - Logische Sicht

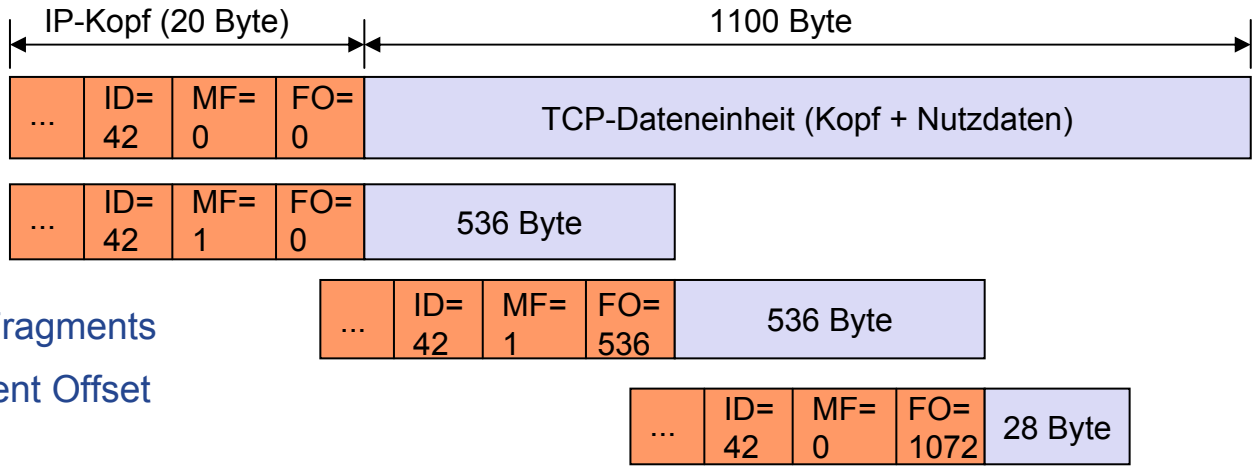
  - UDP (User Datagram Protocol)
    - Bietet unzuverlässigen, verbindungslosen Transportdienst
    - Lokalisiert über unzuverlässigem Dienst von IP (Internet Protocol)

Senden einer TCP-Dateneinheit

- Problemstellung
  - Wann wird aus dem von der Anwendung empfangenen Bytestrom eine TCP-Dateneinheit an IP weitergegeben?
- Spezifikation in RFC 793  [Post81]
  - TCP should „send that data in segments at its own convenience“
- Alternativen zur Realisierung
  - Maximale Größe der zu sendenden Anwendungsdaten (MSS: Maximum Segment Size)
    - Orientierung an der Größe der maximalen Dateneinheit des direkt angeschlossenen Netzes (z.B. lokales Ethernet) oder an der eventuell ermittelten Maximum Transfer Unit (MTU) der Ende-zu-Ende Verbindung
    - Gibt Länge der Anwendungsdaten an. Länge der TCP-Dateneinheit umfasst noch TCP-Kopf
    - Typische Größen (vermeiden das Segmentieren durch IP)
      - 1460 Byte, 536 Byte, 456 Byte
  - Push durch Anwendung (PSH-Flag im Kopf der TCP-Dateneinheit)
    - Sender verlangt hiermit das sofortige Versenden der übergebenen Daten (Vorrangdaten)
    - Bei „kurzen“ Daten bzw. interaktiven Anwendungen sinnvoll (z.B. bei Telnet)
  - Ablauf eines Zeitgebers
    - Nach einem gewissen Zeitintervall der Inaktivität werden vorhandene Daten gesendet
  - In der Regel wird eine Kombination dieser Verfahren angewendet

Segmentieren von TCP-Daten in IP

- Problemstellung
  - Die von TCP generierte Dateneinheit (z.B. MSS + TCP-Kopf) ist zu lang für eine IP-Dateneinheit
- Lösung
  - Segmentieren durch IP
- Beispiel

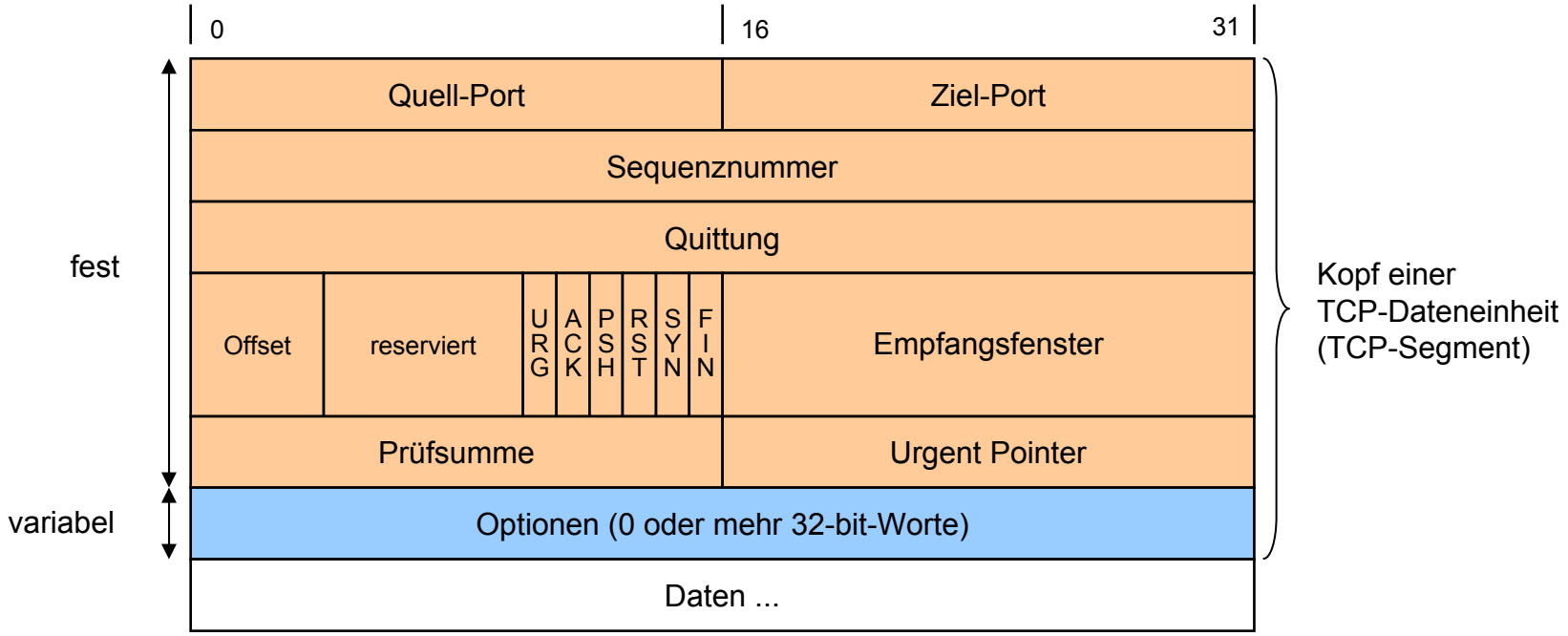


MF = More Fragments

FO = Fragment Offset

  - Statt 1120 Byte werden 1160 Byte gesendet (keine Optionen berücksichtigt)
    - ~3,6 % Overhead
- Probleme
  - Verlust eines IP-Fragments erfordert wiederholtes Senden der kompletten TCP-Dateneinheit
  - Fehlender TCP-Kopf bei IP-Fragmenten kann Probleme bei NATs oder Firewalls erzeugen, die Informationen diesbezüglich benötigen

2.1.1 Format einer TCP-Dateneinheit



URG

Wird auf 1 gesetzt, falls der Urgent Pointer verwendet wird. (Wird i.d.R. nicht benutzt.)

SYN

Wird beim Verbindungsaufbau verwendet, um *Connection Request* (TConReq) bzw. *Connection Confirmation* (TConCnf) anzuzeigen (Synchronization)

ACK

Unterscheidet bei gesetztem SYN-Bit eine TConReq-PDU von einer TConCnf-PDU.
Signalisiert die Gültigkeit des Quittungs-Feldes.

FIN

Gibt an, dass der Sender keine Daten mehr senden möchte

RST

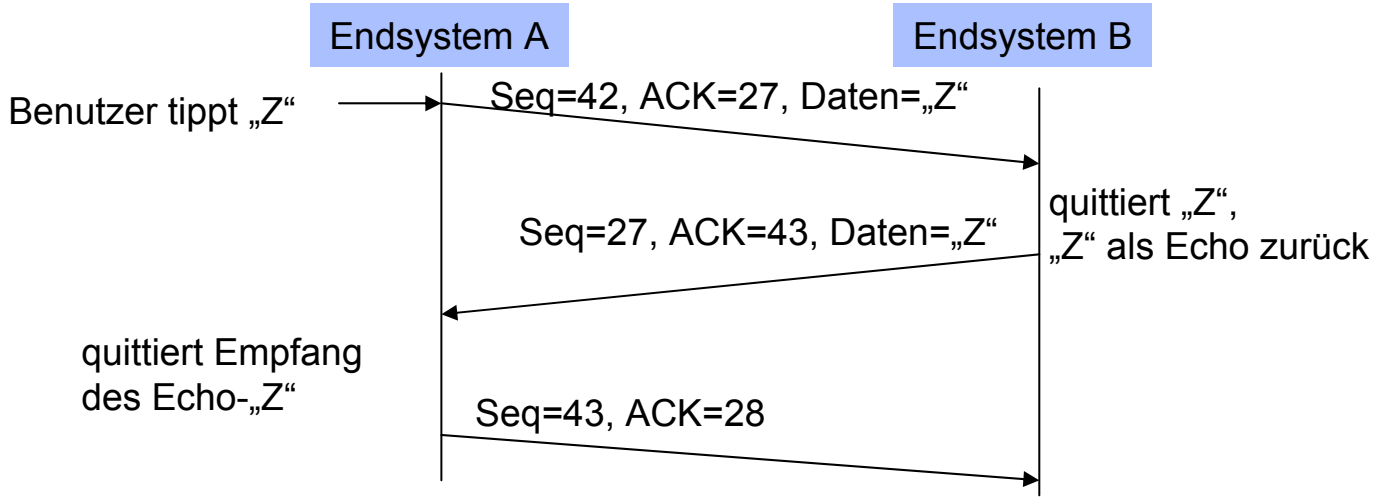
Wird benutzt, um eine Verbindung zurückzusetzen

PSH

Signalisiert, dass die übergebenen Daten sofort weitergeleitet werden sollen (Push).
Gilt sowohl für den Sender als auch für den Empfänger.
(Wird i.d.R. nicht benutzt.)

TCP-Sequenznummern und -Quittungen

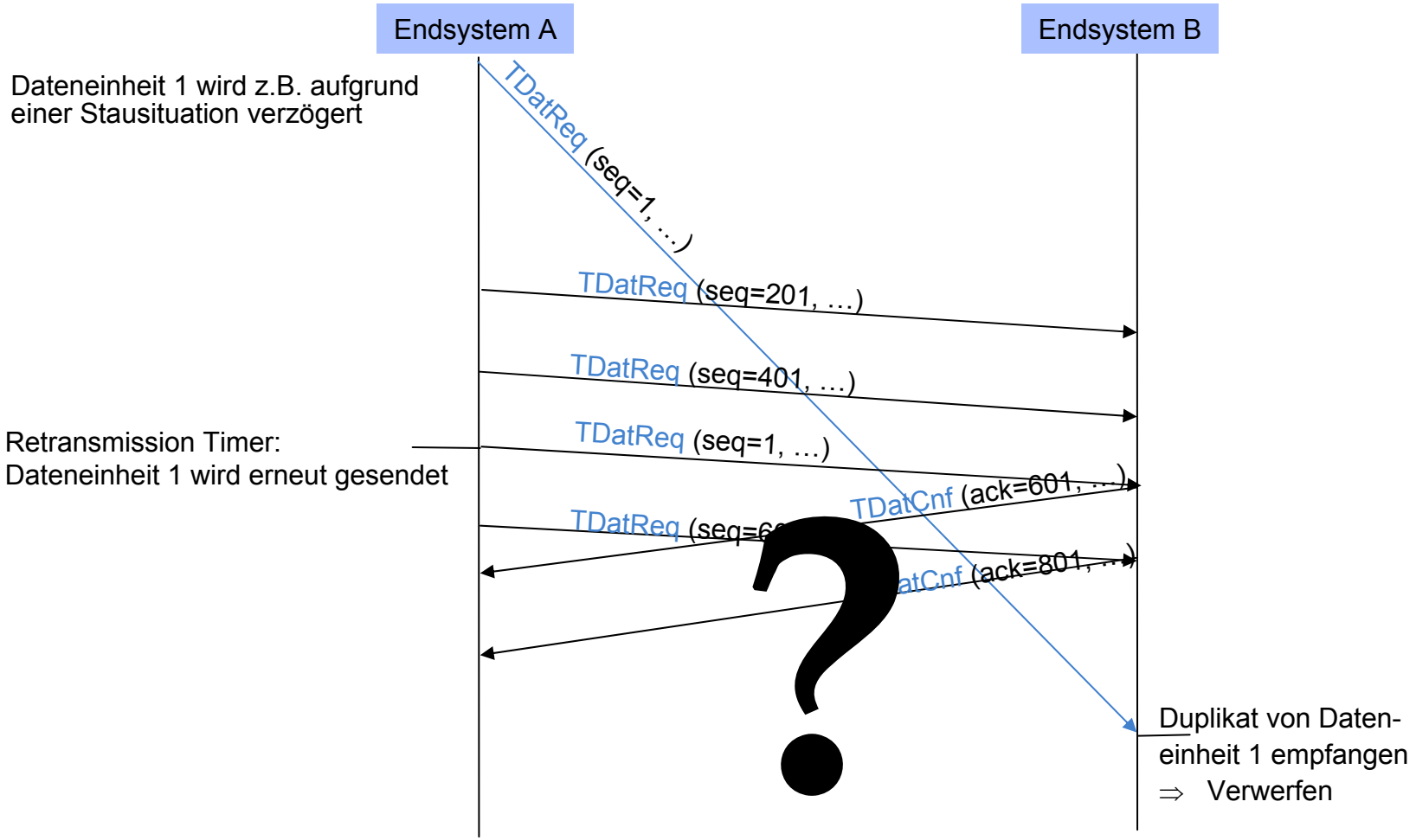
- Sequenznummern
  - Pro Byte, nicht pro Dateneinheit
  - Initiale Sequenznummer wird von beteiligten Endsystemen zufällig gewählt
  - Nummer des ersten Bytes in der Dateneinheit
- Quittungsverfahren
  - Ursprünglicher Entwurf von TCP: Nur positive kumulative Quittungen
  - Sequenznummer des nächsten Bytes, das vom Kommunikationspartner erwartet wird
- Beispiel (Telnet-Szenario)



Duplikaterkennung

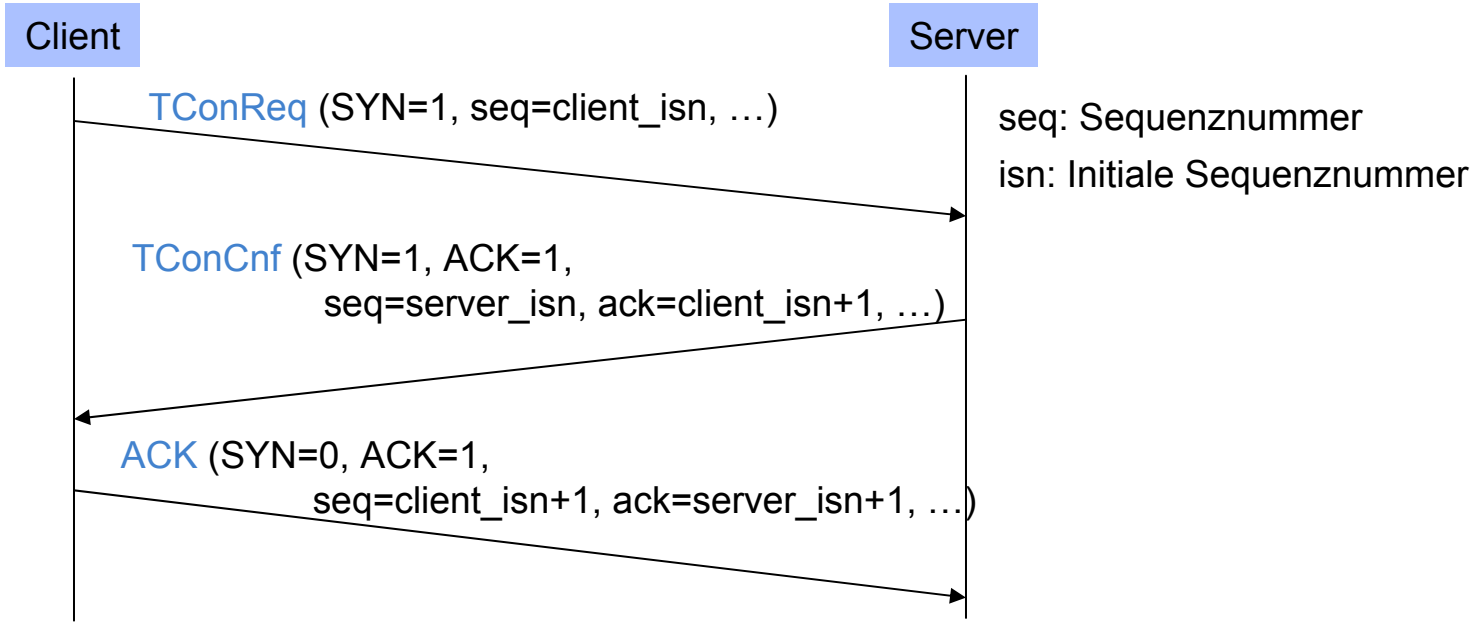
- Beispiel zur korrekten Duplikaterkennung

[Stal06]



2.1.2 TCP-Verbindungsverwaltung

- Verbindungsaufbau
  - Client initiiert Verbindung und Server wartet auf Verbindungswünsche

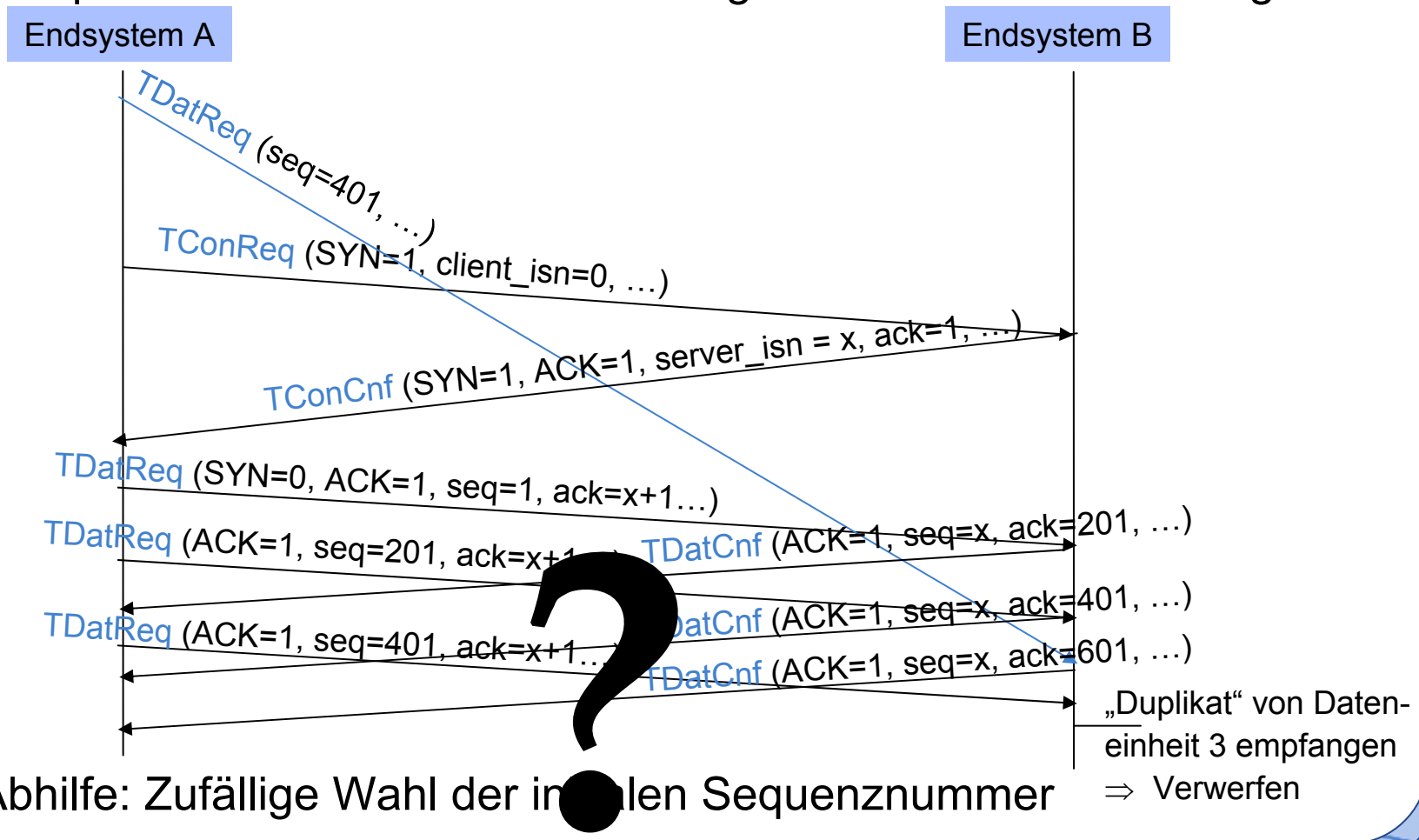


  - Connection request und connection confirm führen keine Nutzdaten mit sich
    - 3-Wege-Handshake. Weshalb genügt 2-Wege-Handshake nicht?
    - Festlegung der initialen Sequenznummern (..._isn). Weshalb erforderlich?
    - Aushandlung der Größe des Flusskontrollfensters
    - Allokation der Puffer …

Fehlerhafte Duplikaterkennung

- Verbindungsaufbau
  - Verspätete Dateneinheit von bereits geschlossener Verbindung.

[Stal06]

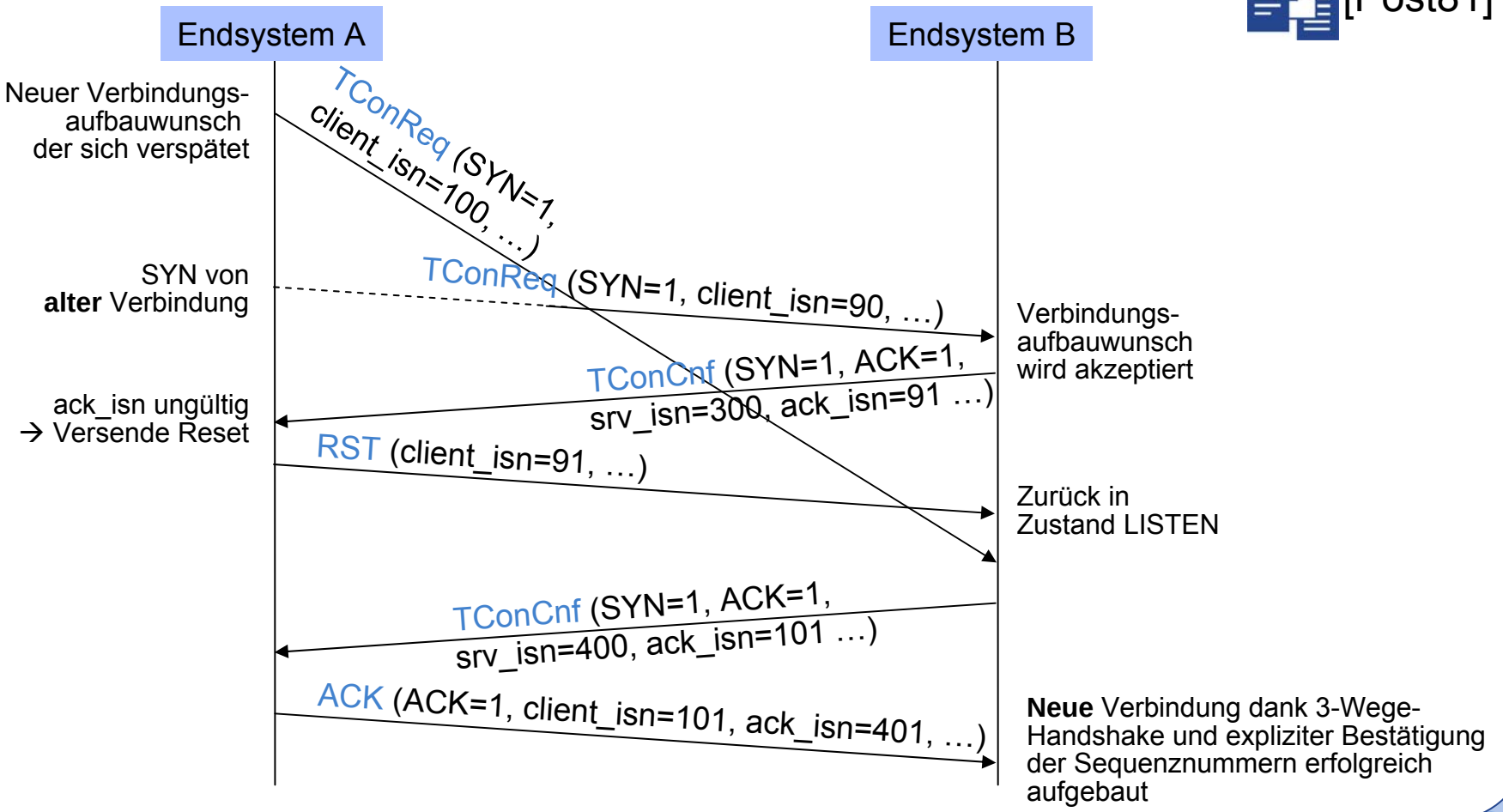


- Abhilfe: Zufällige Wahl der initialen Sequenznummer

Korrekte Duplikaterkennung

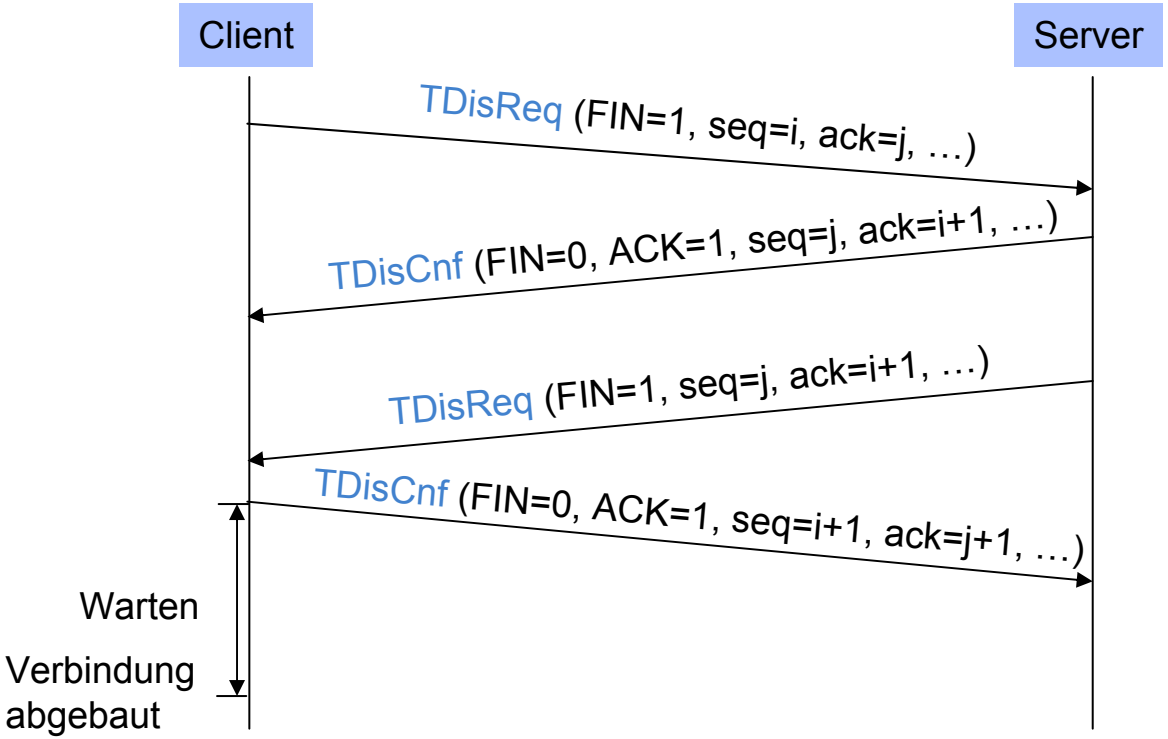
- Verbindungsaufbau
  - Verspätetes SYN-Paket von bereits geschlossener Verbindung

[Post81]



TCP-Verbindungsverwaltung

- Verbindungsabbau
  - Kann sowohl vom Client als auch vom Server initiiert werden



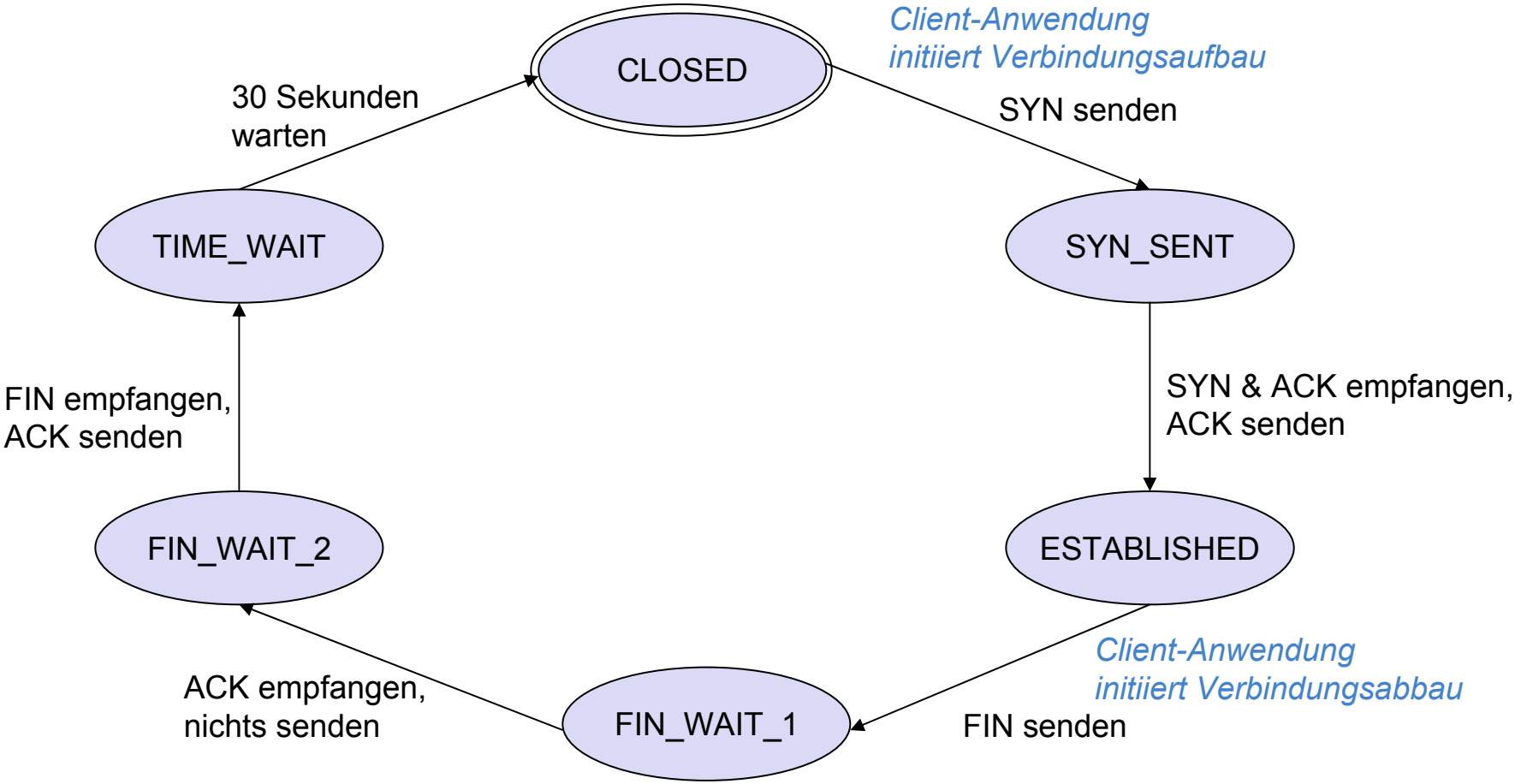
  - Nach letztem ACK wird noch gewartet, bevor lokaler Kontext gelöscht wird
    - z.B. könnte ACK verloren gehen und Server dann sein FIN wiederholen
  - Weitere Varianten für simultanen Verbindungsabbau existieren

Verbindungsabbau

- Unterscheidung zweier grundsätzlicher Varianten
  - Ordnungsgemäßer Abbau (vgl. vorangegangene Folie)
    - Beide Anwendungen müssen unabhängig voneinander ihre "Hälften" der Verbindung schließen (FIN-, ACK- und FIN/ACK-Dateneinheiten)
      - Wurde lediglich eine Richtung geschlossen, so können in der anderen Richtung noch Daten gesendet werden. Damit ist noch ein unidirektionaler Datenfluss möglich
        - Kontrolldaten (z.B. ACK) in der Gegenrichtung sind erlaubt
    - 4-Wege-Handshake für den Abbau
  - Abbruch einer Verbindung
    - Verbindung kann mit Reset abgebrochen werden (RST-Dateneinheit)
    - Verbindung wird unmittelbar geschlossen. Keine Versuche, Daten noch zu wiederholen etc. Kontext wird unmittelbar gelöscht

Zustandsautomat TCP-Client

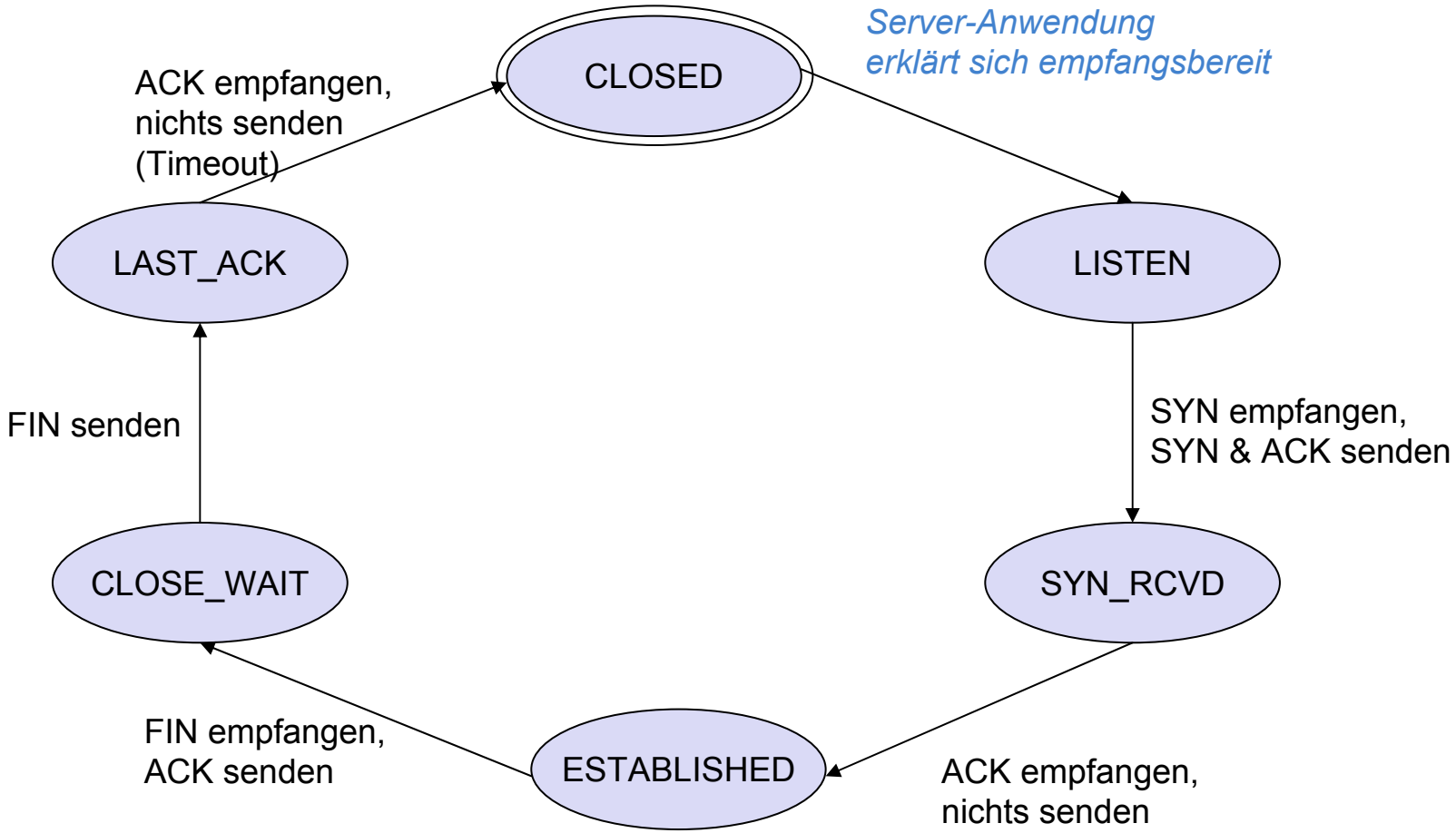
- Typische Sequenz von Zuständen



Dies ist eine vereinfachte Darstellung und nicht (!) der komplette TCP-Zustandsautomat

Zustandsautomat TCP-Server

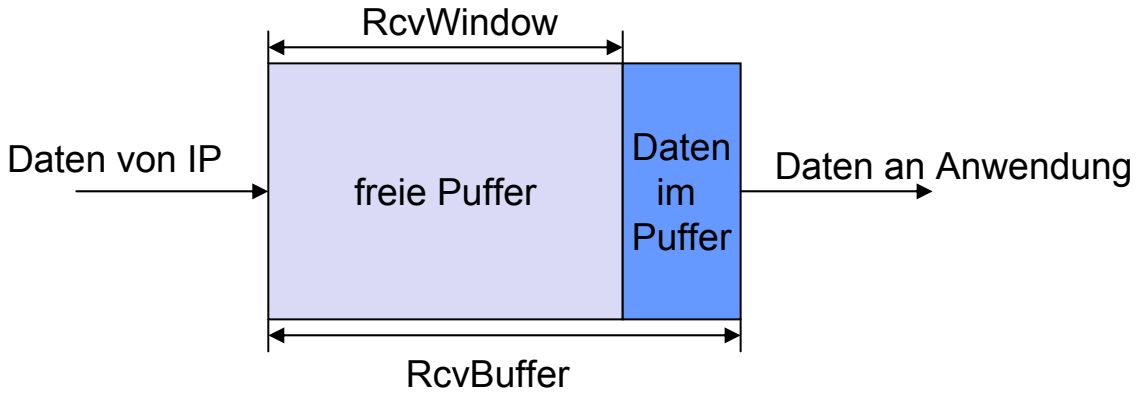
- Typische Sequenz von Zuständen



Dies ist eine vereinfachte Darstellung und nicht (!) der komplette TCP-Zustandsautomat

2.1.4 TCP-Flusskontrolle

- Empfangsfenster
  - Verwaltung des Pufferplatzes, den der Empfänger für diese Verbindung zur Verfügung stellt (explizite Kreditvergabe)
    - Feld Empfangsfenster im Kopf der TCP-Dateneinheit
    - Variablen in der Empfängerinstanz
      - RcvWindow: momentan noch freier Pufferplatz beim Empfänger
      - RcvBuffer: gesamter Pufferplatz für zu empfangende Daten beim Empfänger
  - Dynamische Anpassung möglich
- Schema beim Empfänger



TCP-Flusskontrolle

- Szenario
  - Endsystem A schickt große Datei über TCP an Endsystem B
- Variablen beim Empfänger
  - LastByteRead: Letzte Sequenznummer, die von der Anwendung aus dem Empfangspuffer gelesen wurde
  - LastByteRcvd: Letzte Sequenznummer, die über das Netz empfangen und in den Empfangspuffer geschrieben wurde
- Es muss immer gelten

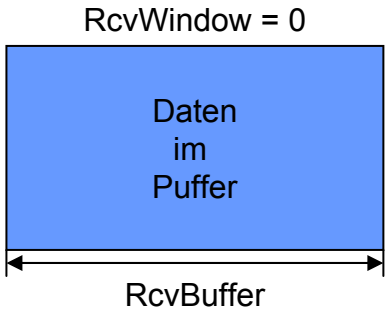
  LastByteRcvd – LastByteRead <= RcvBuffer
- Empfangsfenster

  RcvWindow = RcvBuffer – (LastByteRcvd – LastByteRead)
- Variablen beim Sender
  - LastByteSent: letzte Sequenznummer, die gesendet wurde
  - LastByteAcked: letzte Sequenznummer, die quittiert wurde
- Es muss immer gelten

  LastByteSent – LastByteAcked <= RcvWindow

Zero Window Probing

- Problem: Fenstergröße von Null
  - Endsystem B hat an Endsystem A gemeldet, dass sein Empfangspuffer voll ist
    → Empfangsfenster(B) = 0
  - Konsequenz: Endsystem A darf keine Daten mehr senden
    → Sendefenster(A) = 0
  - Wie soll Endsystem B Endsystem A darüber informieren, dass wieder freier Empfangspuffer existiert?



- Forderung
  - Endsystem A muss auch bei Sendefenster = 0 Dateneinheiten mit 1 Byte senden, die von Endsystem B quittiert werden
    - So kann B aktualisierten Empfangspuffer bekannt geben
  - Spezieller Zeitgeber (Persist-Timer) steuert das Zero Window Probing
    - Um Deadlock zu vermeiden wird Empfänger periodisch abgefragt

2.1.3 Einschub: „TCP-Tools“

- Ziel
  - Beobachtung und Messung des Verhaltens bzw. der Leistungsfähigkeit von TCP
- Dazu erforderlich
  - Datensammlung
    - Erfassung und Sammlung von Daten
  - Datenanalyse
    - Weiterverarbeitung von Daten, beispielsweise um Eigenschaften, wie Minima, Maxima oder Durchschnittswerte zu berechnen
  - Visualisierung
    - Erzeugung von Hilfsmitteln wie Diagrammen oder Tabellen
  - Interpretation
    - Sinnvolle Schlussfolgerungen als Ziel der Beobachtung und Messung
- Kurze Vorstellung zweier nützlicher Tools im Kontext von TCP  [HaJa03]
  - Teilweise für verschiedene Betriebssysteme frei verfügbar

tcpdump

- Paketsniffer
  - Liest an einem Netzwerk-Interface, z.B. an einer Netzwerkkarte, alle empfangbaren Dateneinheiten aus
  - Beispiel

```
$ tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
11:37:41.729131 IP HostA.42980 > Server.www: S 62713440:62713440(0) win 5840
<mss 1460,sackOK,timestamp 8835394 0,nop,wscale 2>
11:37:41.732888 IP Server.www > HostA.42980: S 1853613617:1853613617(0) ack 62713441 win 5792
<mss 1460,sackOK,timestamp 1199883883 8835394,nop,wscale 0>
11:37:41.732907 IP HostA.42980 > Server.www: . ack 1 win 1460 <nop,nop,timestamp 8835395 1199883883>
11:37:41.729297 IP HostA.42980 > Server.www: P 1:112(111) ack 1 win 1460
<nop,nop,timestamp 8835395 1199883883>
11:37:41.729724 IP Server.www > HostA.42980: . ack 112 win 5792 <nop,nop,timestamp 1199883883 8835395>
```

  - Offenbar TCP-Verbindungsaufbau zu Port 80 eines Empfängers
    - 3-Wege Handshake (SYN, SYN/ACK, ACK)
  - Aller Wahrscheinlichkeit nach ein HTTP-Request eines Clients an einen WWW-Server
    - Tcpdump zählt die Sequenznummer nach erfolgreichem Verbindungsaufbau von 1 beginnend (relativ)

tcpdump

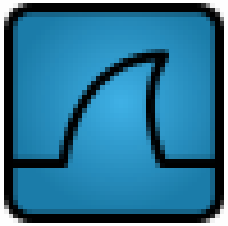
- Fortsetzung

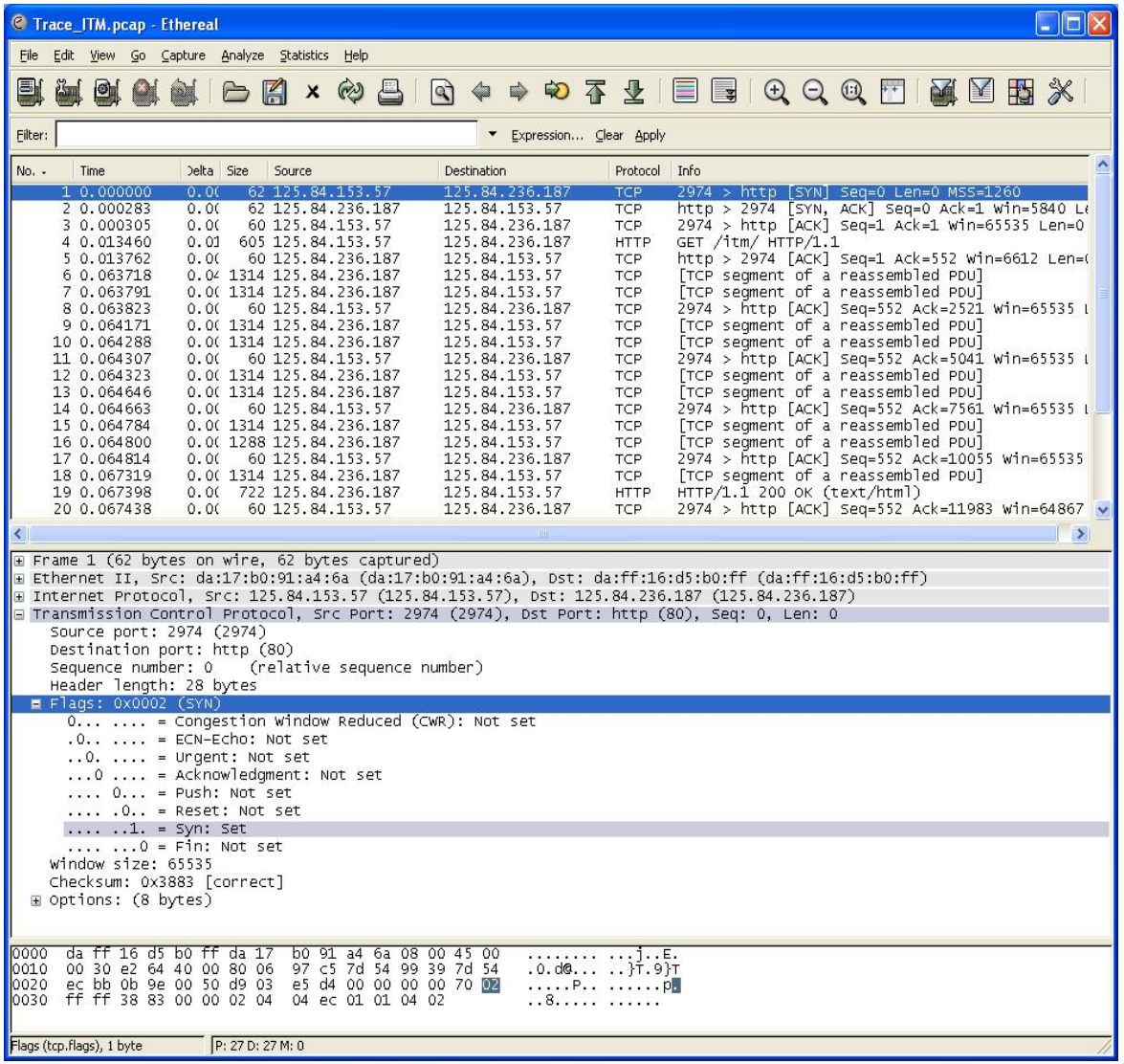
```
11:37:41.730116 IP HostA.42980 > Server.www: . ack 365 win 1728 <nop,nop,timestamp 8835396 1199883884>
11:37:41.730103 IP Server.www > HostA.42980: P 365:1131(766) ack 112 win 5792
<nop,nop,timestamp 1199883884 8835395>
11:37:41.730128 IP HostA.42980 > Server.www: . ack 1131 win 2111 <nop,nop,timestamp 8835396 1199883884>
11:37:41.730577 IP HostA.42980 > Server.www: F 112:112(0) ack 1131 win 2111
<nop,nop,timestamp 8835396 1199883884>
11:37:41.731096 IP Server.www > HostA.42980: F 1131:1131(0) ack 113 win 5792
<nop,nop,timestamp 1199883885 8835396>
11:37:41.731157 IP HostA.42980 > Server.www: . ack 1132 win 2111 <nop,nop,timestamp 8835397 1199883885>
```

- TCP-Verbindungsabbau
  - FIN, FIN/ACK, ACK

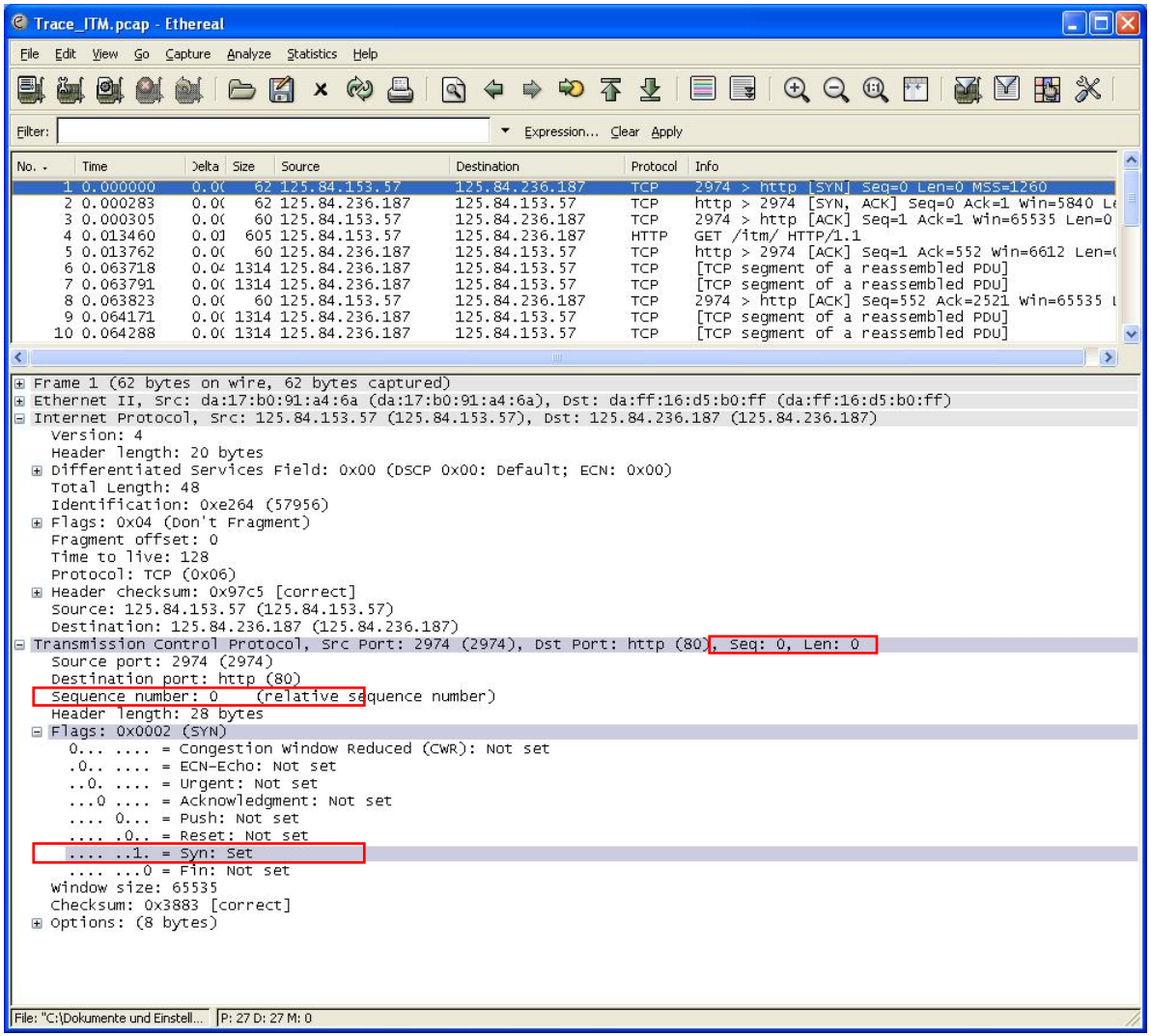
Wireshark

- Grafischer Paketsniffer
  - Dekodierung zahlreicher Protokolle
  - Diagramm-erstellung

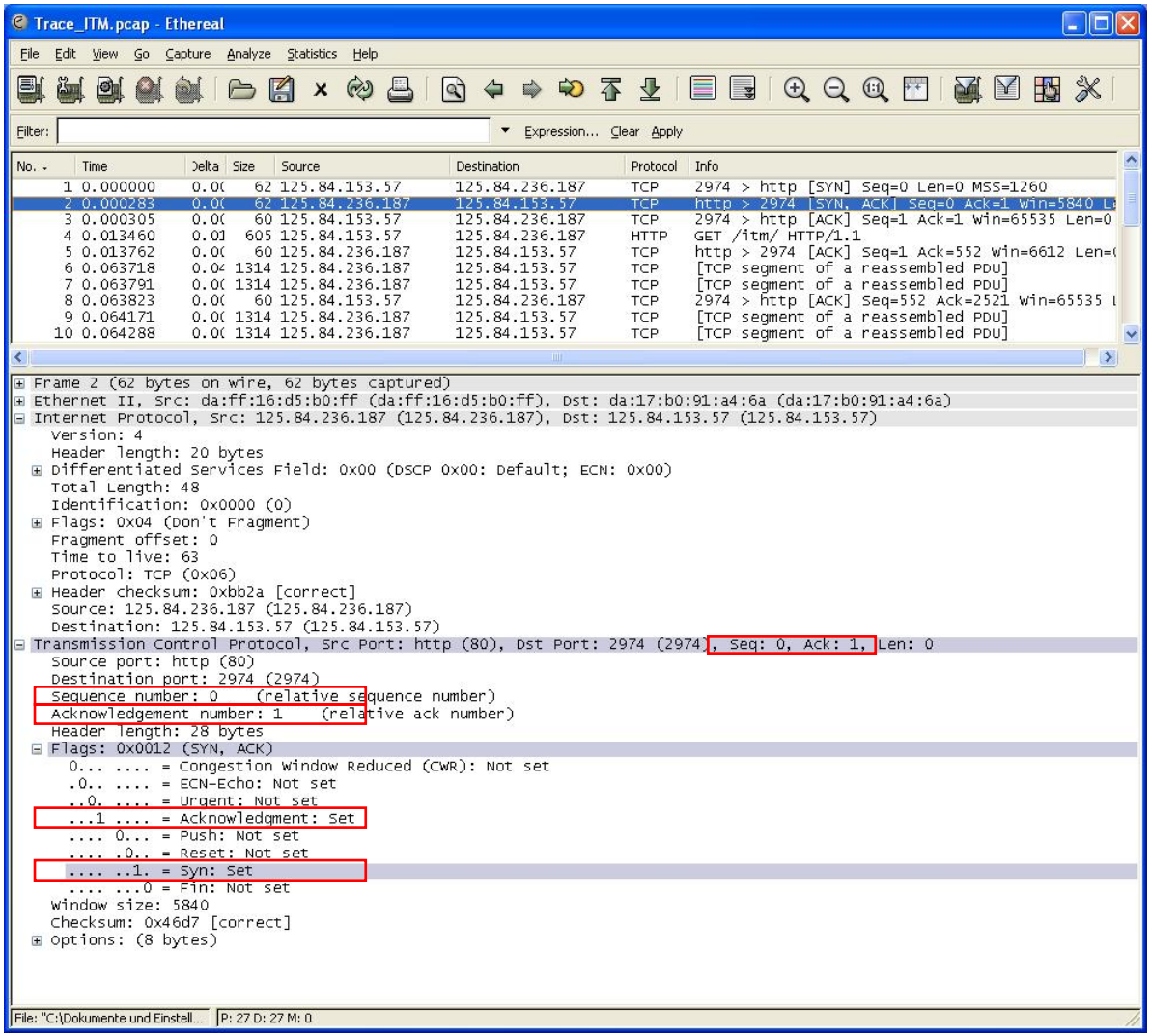




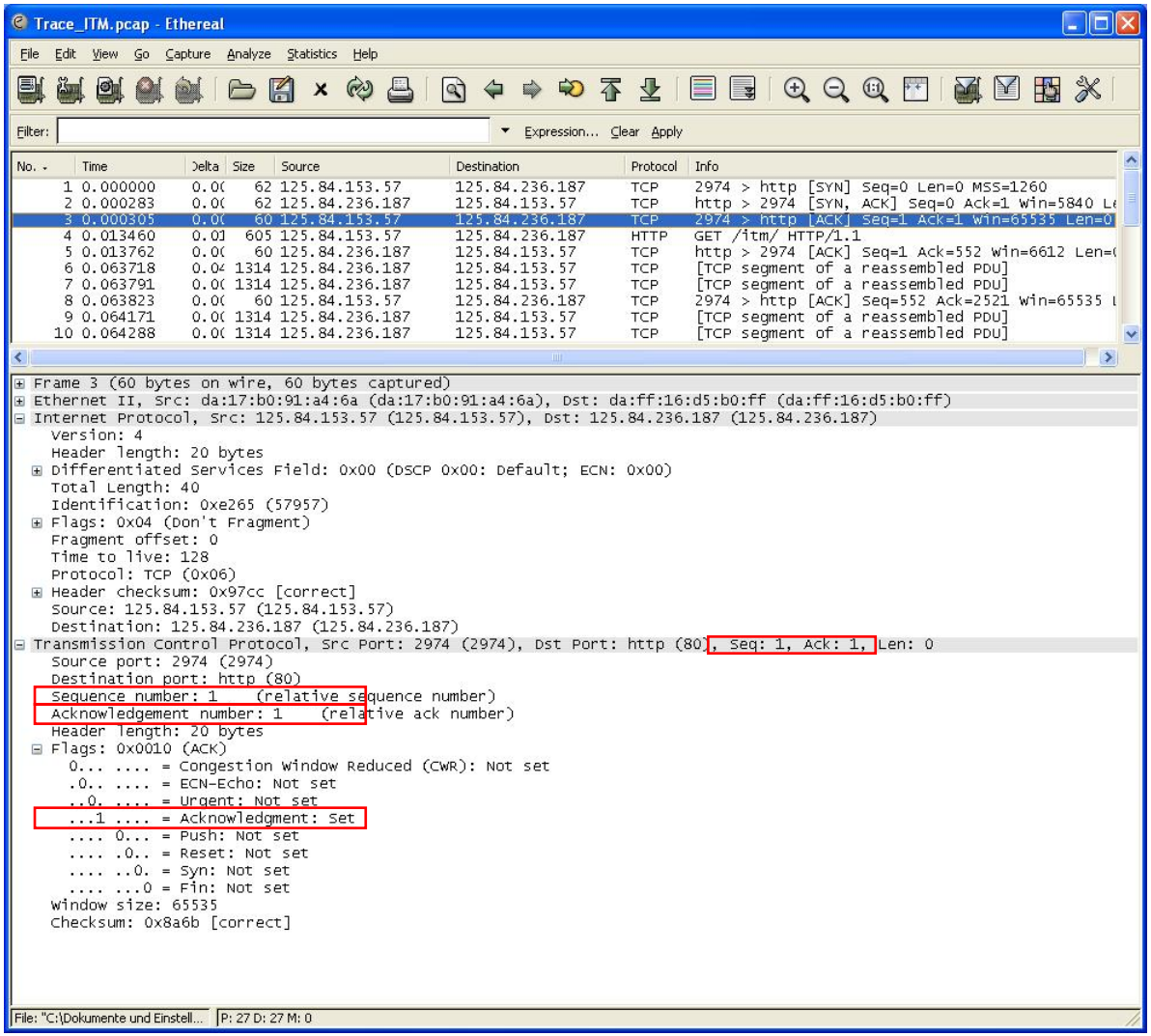
Beispiel TCP Verbindungsaufbau (Connection Request)



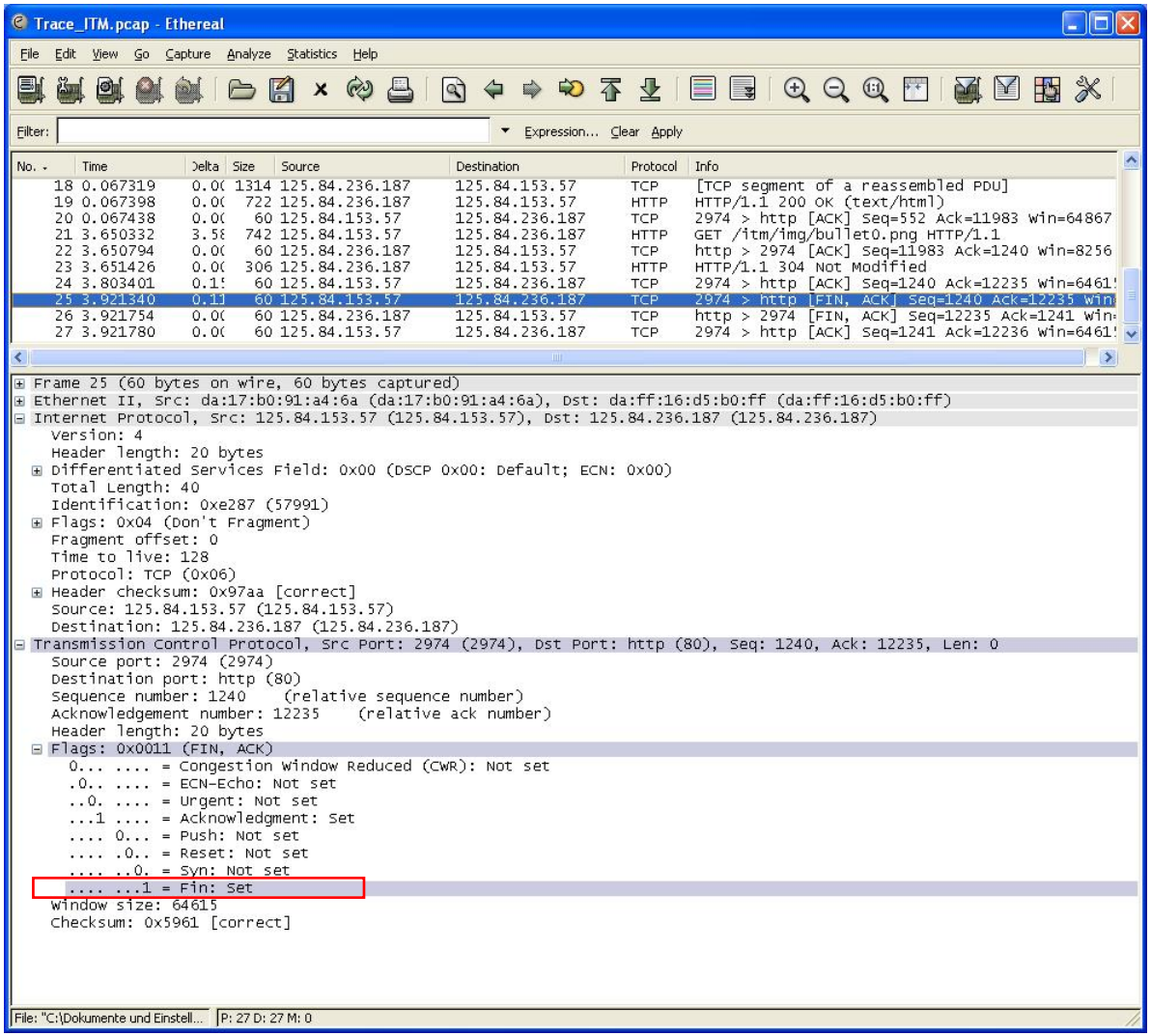
Beispiel TCP Verbindungsaufbau (Connection Confirm)



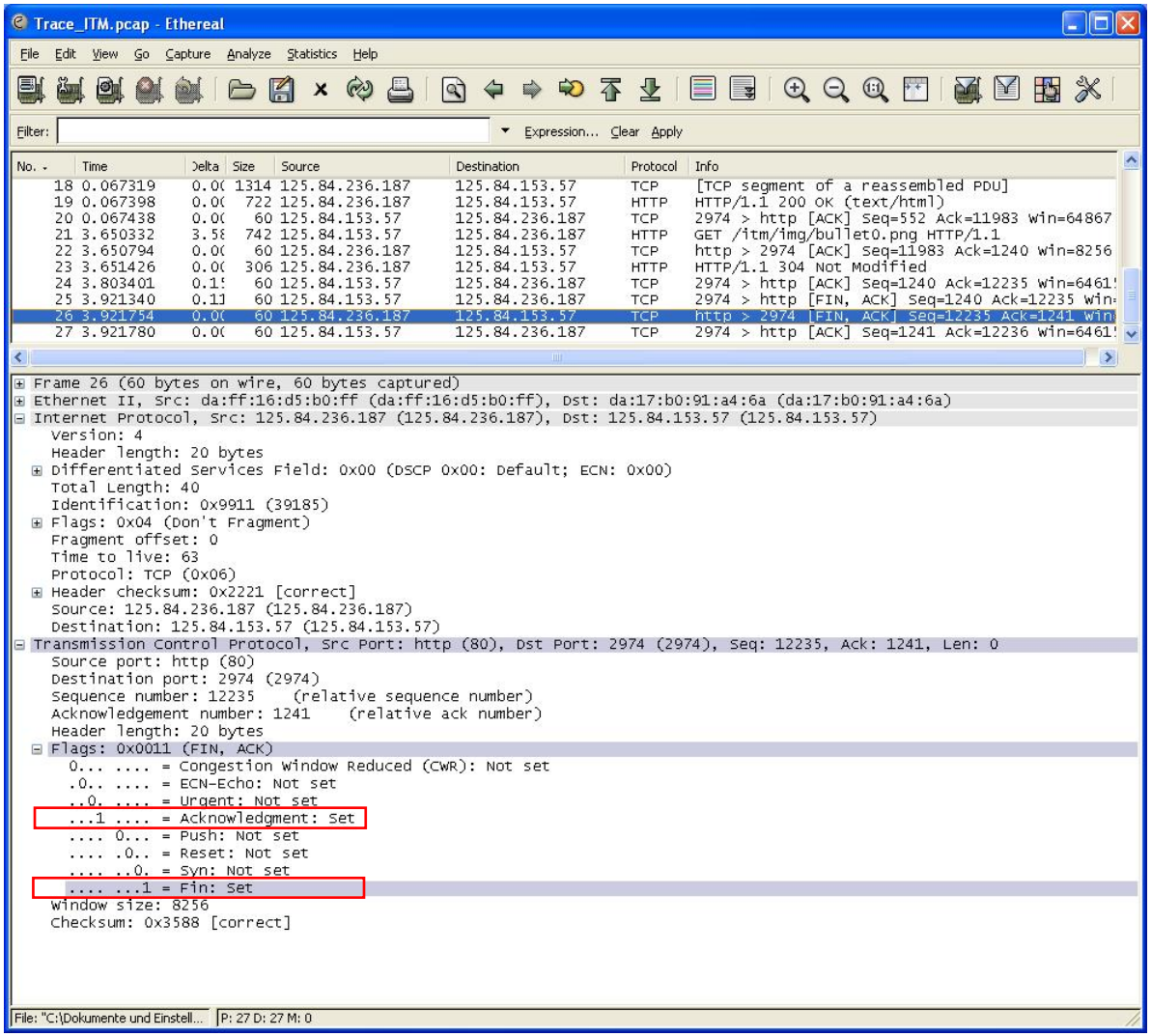
Beispiel TCP Verbindungsaufbau (ACK)



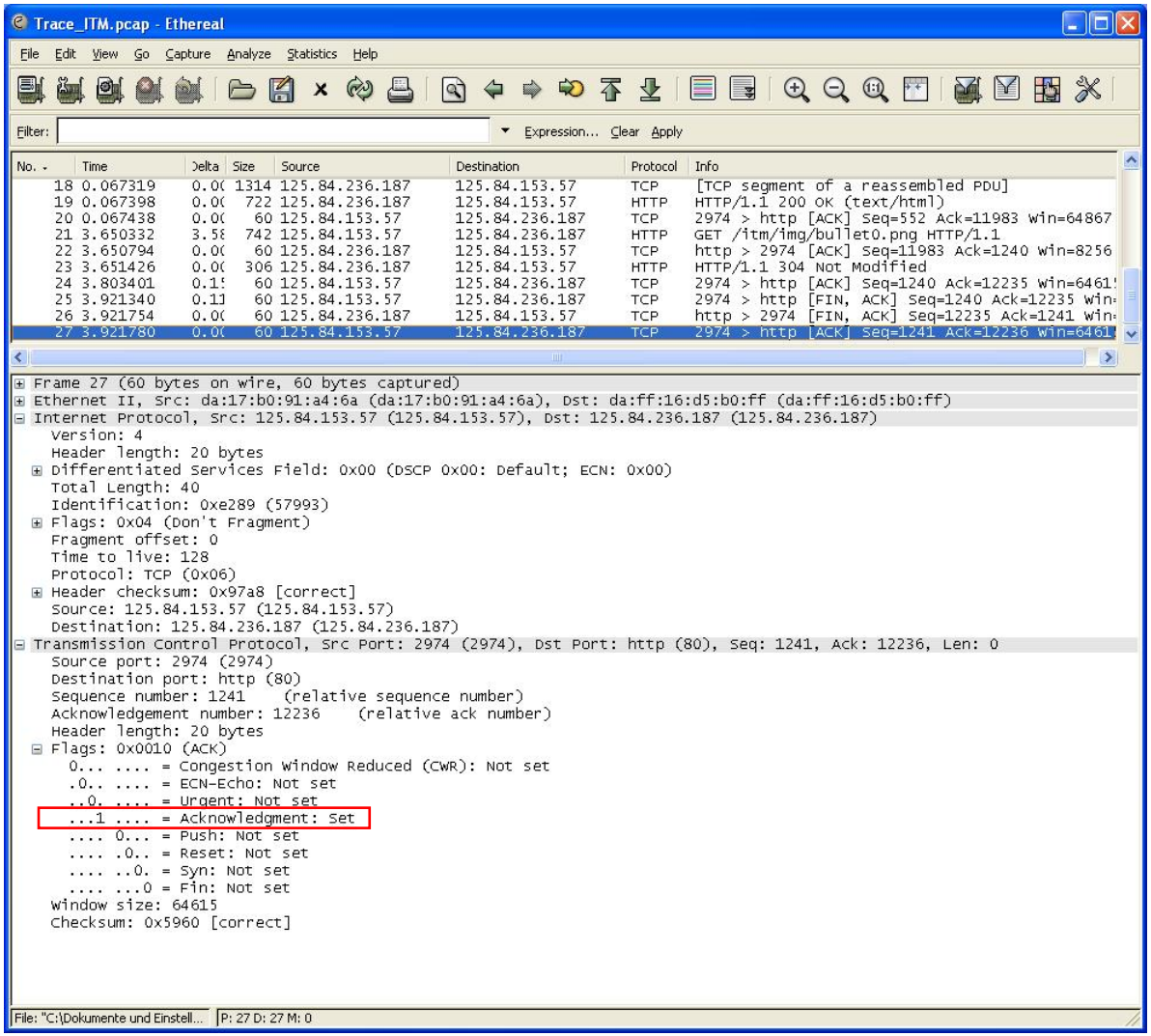
Verbindungsabbau (Client FIN)



Verbindungsabbau (Server ACK + FIN)

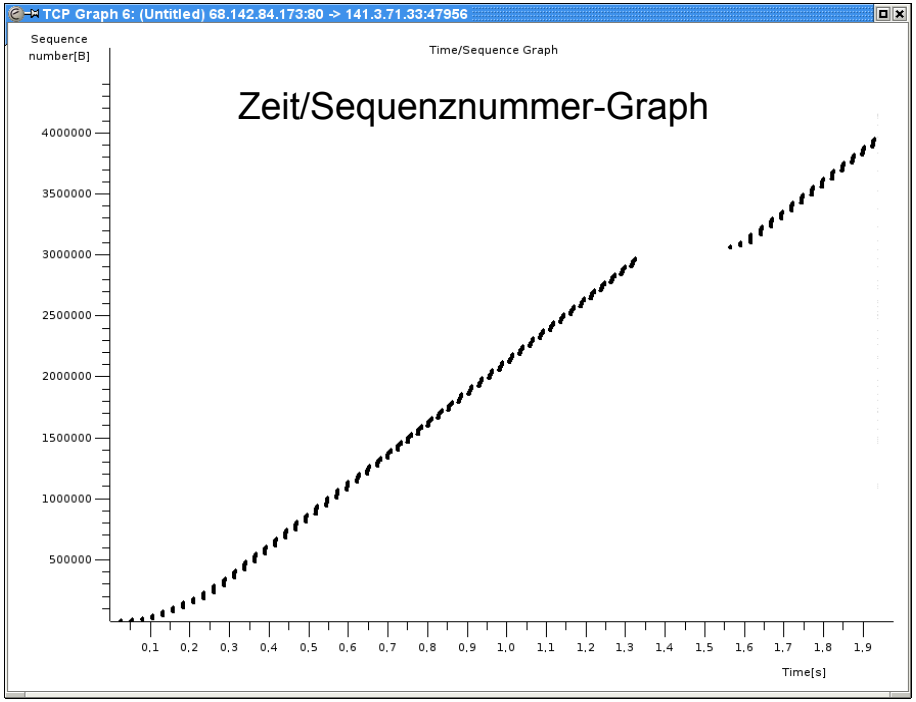


Verbindungsabbau (Client ACK)



Wireshark … Diagramme

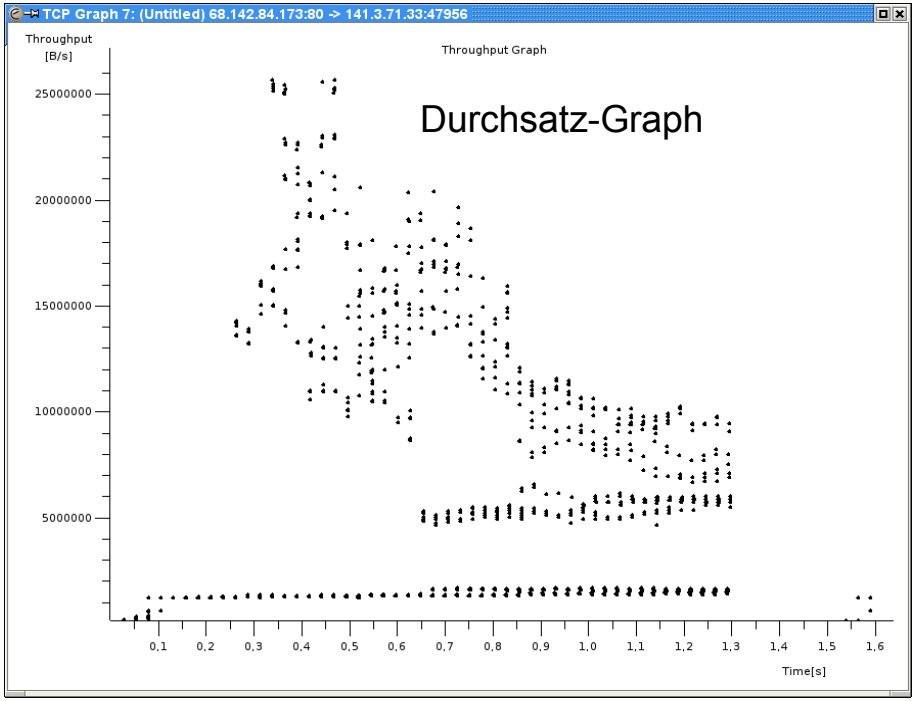
- Wireshark-Diagrammbeispiele einer TCP-Verbindung
  - Download einer größeren Datei



  - Diagramm zeigt, welche Sequenznummer zu welchem Zeitpunkt gesendet wurde
    - Sequenznummer wird pro Byte vergeben
    - Kurze Sendepause (vorheriger Verlust einer Dateneinheit)

Wireshark … Diagramme

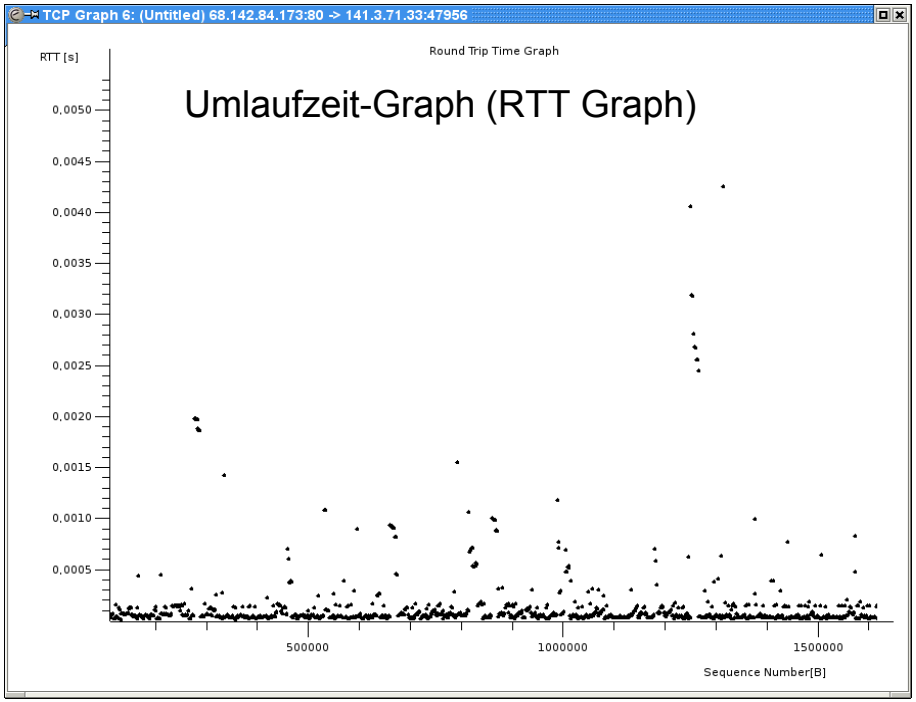
- Wireshark-Diagrammbeispiele einer TCP-Verbindung
  - Download einer größeren Datei



  - Diagramm zeigt den Durchsatz einer TCP-Verbindung
    - Einteilung in äquidistante Zeitintervalle

Wireshark … Diagramme

- Wireshark-Diagrammbeispiele einer TCP-Verbindung
  - Download einer größeren Datei



  - Diagramm zeigt die Umlaufzeiten der gesendeten Dateneinheiten
    - Die Umlaufzeit berechnet sich aus dem Empfangszeitpunkt einer Quittung und dem Sendezeitpunkt der entsprechenden Dateneinheit
    - Dies ist nur möglich, wenn jede Dateneinheit einzeln quittiert wird

2.1.5 Zusammenfassung TCP à la RFC 793

- Verbindungsaufbau
  - 3-Wege-Handshake; Vollduplex-Verbindung
- Verbindungsabbau
  - Separat für beide Übertragungsrichtungen; Halbduplex-Verbindungen möglich
  - 4-Wege-Handshake
- Datentransfer
  - Byte-orientierte Sequenznummern
  - Go-Back-N
    - Positive kumulative Quittungen
  - Flusskontrolle (Sliding Window)
- That´s it … für den Anfang

Homework

- Erstellen Sie zu den gemessenen Abläufen beim Verbindungsauf- und –abbau jeweils die entsprechenden Weg-Zeit-Diagramme
  - Benennen sie die jeweiligen Dateneinheiten korrekt
  - Tragen Sie Sequenznummern etc. ein
- Berechnen Sie den Overhead beim Versenden von 10 KByte Nutzdaten bei verschiedenen Größen der MSS (z.B. 1460 Byte, 536 Byte)
- Spielen Sie den Verlauf der TCP-Flusskontrolle an verschiedenen Szenarien durch
  - Beispiel: unidirektionaler Datenfluss von Endsystem A zu Endsystem B. Je TCP-Dateneinheit werden 200 Byte gesendet. Der anfängliche Sendekredit betrage 1400 Byte. Beginnen Sie bei Sequenznummer 1001. Verwenden Sie verschiedene „Strategien“ zum Versenden der Quittungen. Variieren Sie den von Endsystem B bereitgestellten Kredit.
    - Betrachten Sie auf beiden Endsystemen die Entwicklung des Flusskontroll-fensters und die aktuellen Werte für die Variablen „LastByteRead“ etc.
- Lesen Sie die Veröffentlichung von Van Jacobson („Congestion Avoidance and Control“) als Vorbereitung für die nächste Vorlesung  [JaKa88]

2.2 Dynamik und TCP

- TCP à la RFC 793 läuft in großen, dynamischen Netzen nicht ohne Probleme

 [JaKa88, Nagl84]

  - Drastischer Einbruch des Durchsatzes von TCP im Oktober 1986
    - Es war der erste einer Serie so genannter „Staukollapse“
    - Durchsatz war während einer gewissen Zeit um Größenordnungen niedriger als normal
    - Beispiel: Verbindung zwischen Lawrence Berkeley Laboratory und UC Berkeley:
      - Standorte sind 370 Meter und zwei Router voneinander entfernt
      - Durchsatz sank um mehrere Größenordnungen von 32 kbit/s auf 40 bit/s!
- Als Reaktion darauf Reihe wichtiger Änderungen in TCP eingeführt, z.B.
  - Slow-Start
  - Schätzung der Abweichungen der Umlaufzeit
  - Dynamische Fensterveränderung bei Stau
  - Exponentieller Backoff des Zeitgebers für Übertragungswiederholungen (Retransmission Timer)
  - Agressivere Quittungspolitik beim Empfänger

Überblick

- Grundlegende Beobachtung
  - Verbindung muss in einen stabilen Zustand gebracht werden. Das Gesamtsystem sollte im Gleichgewicht arbeiten.
- Ziel
  - „Conservation of Packets“
    - „Volles“ Fenster im Transit
    - Keine neue Dateneinheit „einfügen“, bevor nicht eine alte Dateneinheit das Gesamtsystem „verlassen“ hat
    - Verbindung befindet sich im Gleichgewicht

[JaKa88]

  - Gesamtsysteme, die dieses Gesetz befolgen sind robust gegen Stausituationen
    - Theorie Dynamischer Systeme und deren Stabilität
    - … mehr Speicher in den Routern hilft auch nicht

2.2.1 „Conservation of Packets“

- Grundlegende Fragestellung
  - In welchen Situationen kann das Erreichen des „Conservation of Packets“-Gleichgewichts scheitern?
- Beobachtungen
  - „Conservation of Packets“ kann aus folgenden Gründen scheitern:

2.2.1.1 Verbindung kommt nicht in das Gleichgewicht

2.2.1.2 Sender sendet neue Dateneinheit zu früh

2.2.1.3 Ressourcenbeschränkungen verhindern Gleichgewicht

2.2.1.1 Verbindung kommt nicht ins Gleichgewicht

- Mögliche Auslöser
  - Neue Verbindung
  - Neustart einer existierenden Verbindung
- „Conservation of Packets“
  - Sender benutzt Quittungen als Auslöser für das Senden neuer Dateneinheiten.
  - Empfänger kann Quittungen nicht schneller erzeugen als die Dateneinheiten durch das Netz weitergeleitet werden.

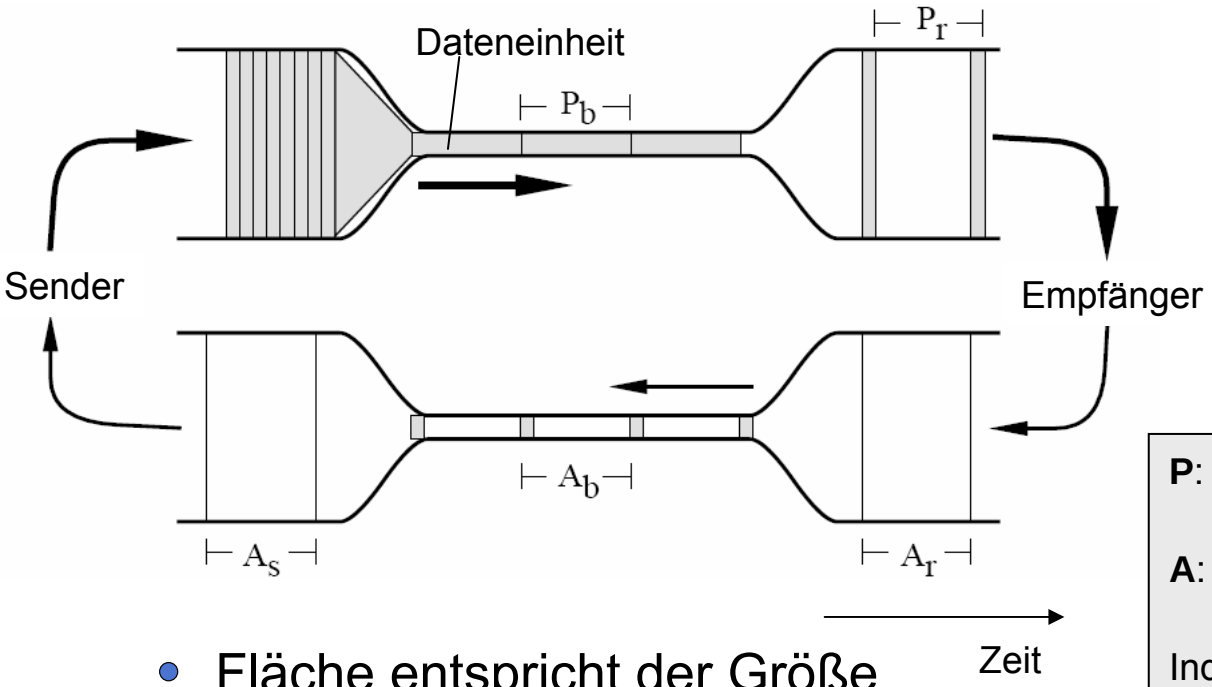
  → Self-Clocking
  - Automatische Anpassung an Änderungen der Datenrate und Verzögerung
    - Für sehr breiten Einsatz in höchst unterschiedlichen Netzen (hohe/niedrige Datenraten)
    - Also benutzbar in sehr verschiedenen dynamischen Systemen
  - Aber …
    - Schwierig beim Neustart
      - Um Daten in das System zu bekommen sind Quittungen erforderlich, aber um Quittungen zu bekommen, müssen Daten gesendet werden

Self-Clocking bei TCP

- Ausgangssituation
  - Sender und Empfänger verfügen über schnellen Netzanschluss
  - Verbindung dieser Netze durch ein langsames Weitverkehrsnetz (Engpass)
  - Sender sendet Dateneinheiten des ersten Fensters direkt aufeinanderfolgend so schnell der lokale Netzanschluss dies erlaubt
    - Sämtliche Dateneinheiten („volles“ Fenster) befinden sich im Transit bedingt durch das langsamere Weitverkehrsnetz.
      - Wieso ist das so?
    - Dateneinheiten treffen beim Empfänger mit der Zwischenankunftszeit ein, die sie auf dem langsamsten Übertragungsabschnitt erzielen.
      - Vergleichen Sie die Zwischenankunftszeiten bei 100 Mbit/s und 64 kbit/s.
    - Empfänger generiert Quittungen mit der Frequenz der eingehenden Dateneinheiten.
    - Sender muss auf Eingang von Quittungen warten bevor er nächste Dateneinheit senden darf.
- Beobachtung
  - Zwischenankunftszeit des langsamsten Übertragungsabschnitts bestimmt Sendeintervall des Senders für die weiteren Dateneinheiten

Self-Clocking illustriert

- Schematische Darstellung



  - Fläche entspricht der Größe der Dateneinheit
- Aber …
  - wie soll die „Uhr“ gestartet werden?

P: Zwischenankunftszeit zweier Dateneinheiten
A: Zwischenankunftszeit zweier Quittungen
Index **b**: langsamer Übertragungs-abschnitt (Weitverkehrsnetz)
Index **r**: schneller lokaler Übertragungs-abschnitt (Empfänger)
Index **s**: schneller lokaler Übertragungs-abschnitt (Sender)

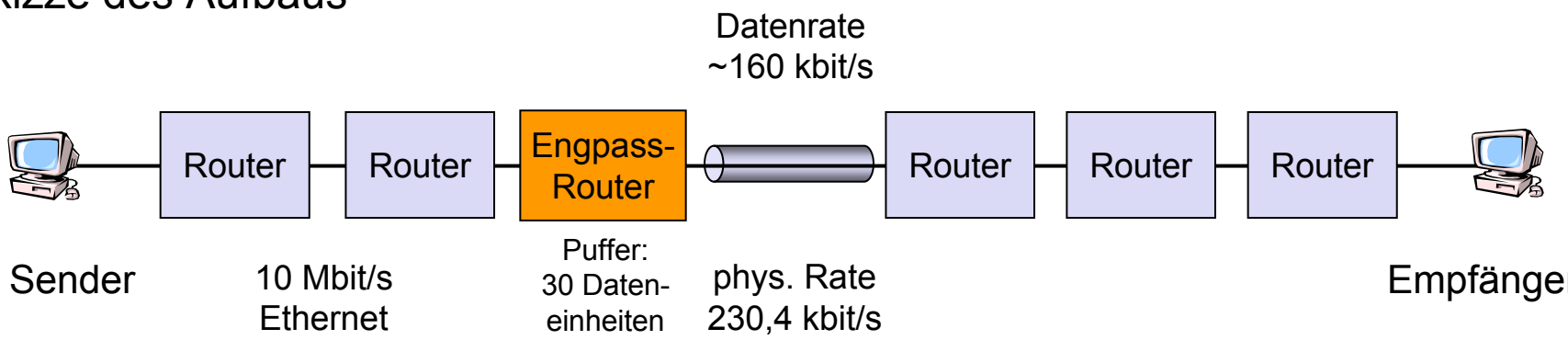
Slow-Start

- Grundlegende Idee
  - Die Anzahl der Dateneinheiten, die pro Verbindung gesendet werden dürfen, wird zunächst „langsam“ erhöht, um das Netz nicht zu überfordern und um damit möglichst ohne zu Übersteuern in das gewünschte Gleichgewicht zu kommen
- Konzept
  - Einführung eines Staukontrollfensters (Congestion Window, CWnd) pro Verbindung
    - Kontrolliert zusammen mit dem Flusskontrollfenster, wieviel Dateneinheiten bzw. Daten vor dem Empfang einer Quittung gesendet werden dürfen
  - Beim Verbindungsstart oder nach Verlust einer Dateneinheit Staukontrollfenster zurücksetzen
    - CWnd = 1
  - Bei jeder eingehenden Quittung für neu gesendete Daten (keine Sendewiederholungen) wird Staukontrollfenster um Eins erhöht
    - CWnd = CWnd + 1
    - Also exponentielles Anwachsen des Staukontrollfensters
  - Gesendet werden darf das Minimum aus Stau- und Flusskontrollfenster
    - Staukontrollfenster: CWnd [# MSS]
    - Empfangsfenster: RcvWindow [Bytes]
      - Wird vom Empfänger zur Flusskontrolle vorgegeben

Startverhalten mit/ohne Slow-Start

[JaKa88]

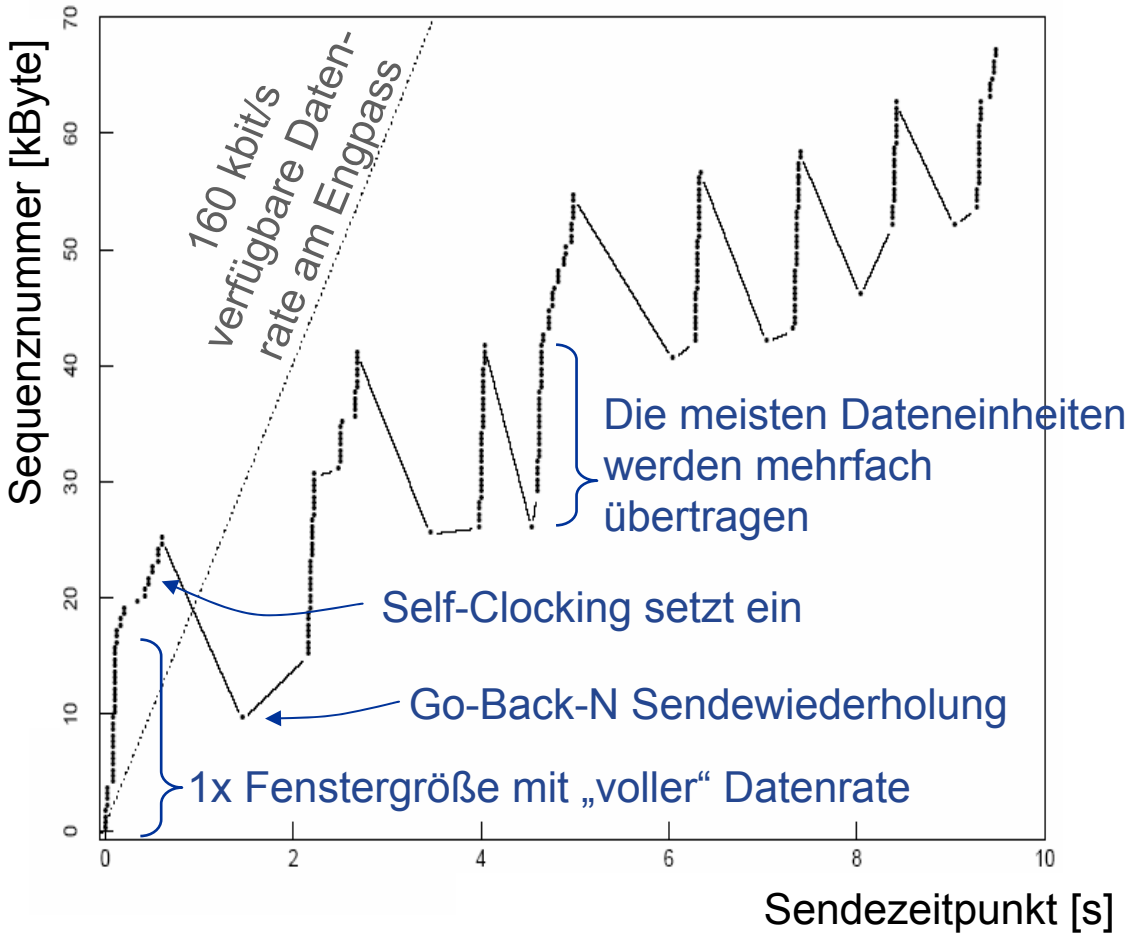
- Ziel
  - Beobachtung des Startverhaltens einer TCP-Verbindung mit und ohne Slow-Start
  - Bestimmung der erzielbaren Datenrate
    - Bei vorgegebener Empfangsfenstergröße und fester Größe der Dateneinheiten
- Aufbau des Experiments
  - Sender und Empfänger in verschiedenen lokalen Netzen mit 10 Mbit/s Ethernet-Anbindung
  - Empfangsfenstergröße 16 KByte (entspricht 32 Dateneinheiten à 512 Bytes Nutzdaten)
  - Engpass-Router verfügt über einen Puffer für 30 Dateneinheiten
  - TCP-Verbindung über sechs Hops mit Engpass-Link
    - Komplettes Fenster kann sich in der Übertragung befinden
- Skizze des Aufbaus



Startverhalten ohne Slow-Start (I)

- Darstellung
  - Jeder Punkt entspricht einer Dateneinheit von 512 Bytes
  - X-Achse: Zeitpunkt, zu dem die Dateneinheit gesendet wird
  - Y-Achse: Sequenznummer im Kopf der Dateneinheit
- Beobachtungen
  - Sendewiederholung
    - Zwei Punkte mit selbem Wert auf Y-Achse und verschiedenen Werten auf X-Achse
  - Go-back-N
    - Sendewiederholung ab erster unbestätigter Dateneinheit

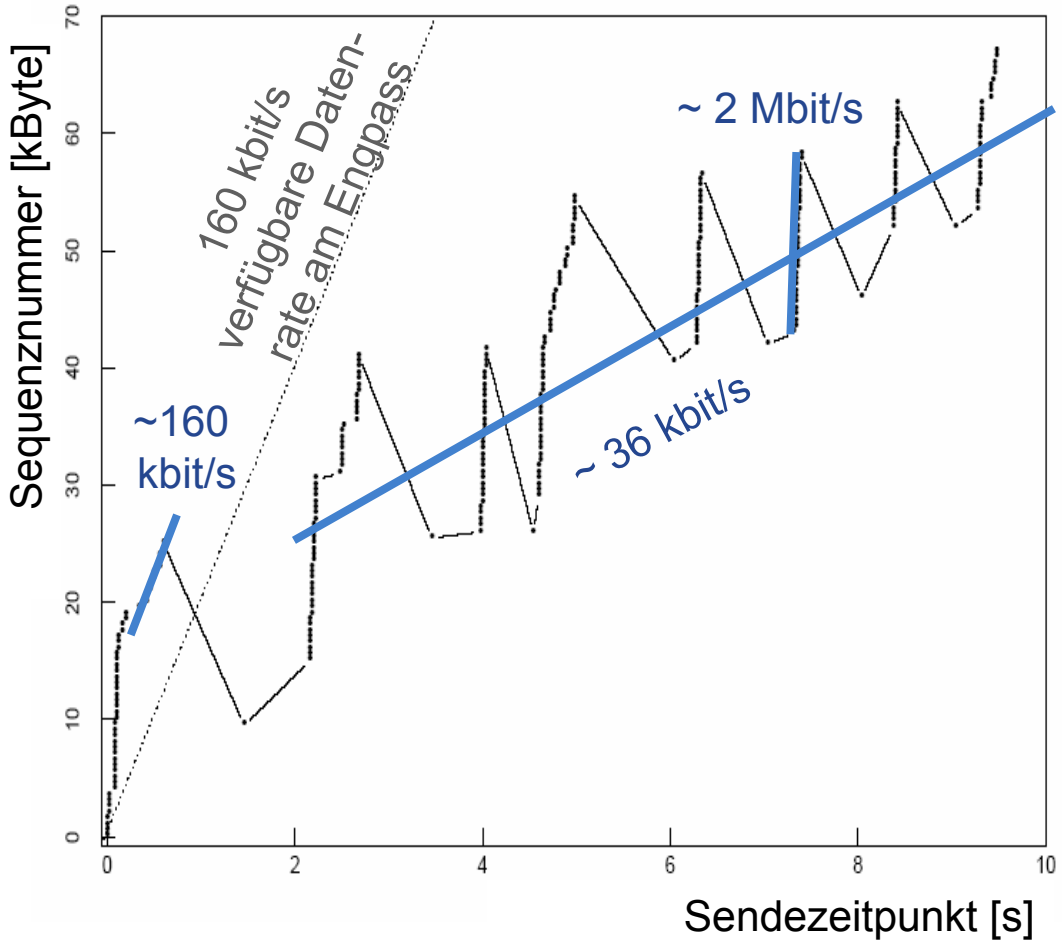
[JaKa88]



Startverhalten ohne Slow-Start (II)

- Gewünschtes Verhalten
  - TCP sollte mit der für diese Verbindung am Engpass verfügbaren Bandbreite senden
  - Relativ gerade Linie von links unten nach rechts oben
- Beobachtungen
  - Nach Verlust der ersten Dateneinheit sendet TCP in Bursts
  - Viele Daten mehrfach wiederholt
  - Effektive Nutzung von nur 35% erzielt

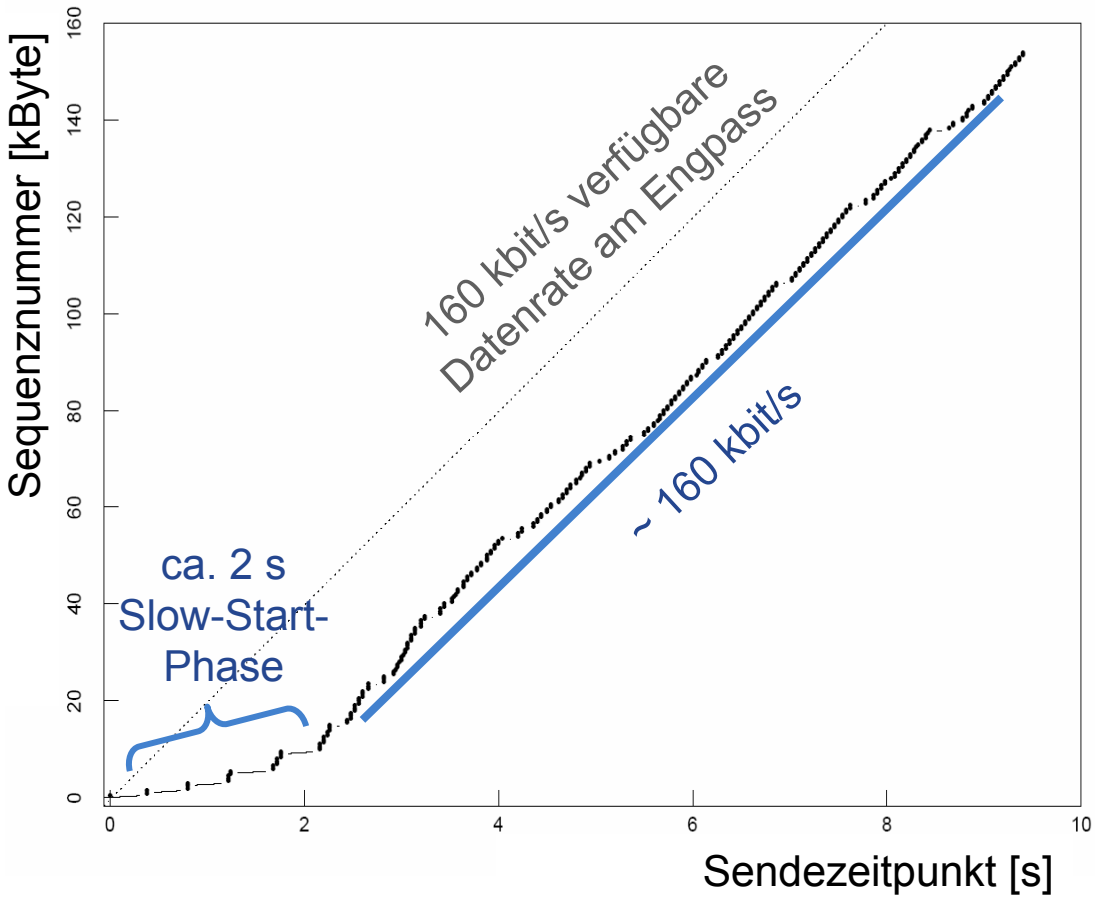
[JaKa88]



Startverhalten mit Slow-Start

- Beobachtungen
  - Slow-Start-Phase von ca. 2 Sekunden
  - Keine Sendewiederholungen
  - Quelle sendet nahezu mit Datenrate am Engpass
  - Effizienz abhängig von Verbindungsdauer
    - Langsamere Slow-Start-Phase fällt bei langen Verbindungen zunehmend weniger ins Gewicht

[JaKa88]



Zusammenfassung: Slow-Start hilft …

- … Burstartiges Senden zu verhindern
  - Verbindung kommt ins Gleichgewicht
    - Senden mit nahezu der maximalen Datenrate am Engpass möglich
    - Kaum Sendewiederholungen notwendig
  - Effizienz weit höher als bei Verbindungen ohne Slow-Start

2.2.1.1 Verbindung kommt nicht in das Gleichgewicht 

2.2.1.2 Sender sendet neue Dateneinheit zu früh

2.2.1.3 Ressourcenbeschränkungen verhindern Gleichgewicht

2.2.1.2 Sender sendet neue Dateneinheit zu früh

- Slow-Start
  - bringt TCP-Verbindung in ein Gleichgewicht
- Jetzt
  - … Gleichgewicht muss gehalten werden. Aber …
    - Netzlast kann sich über die Zeit ändern
    - Verkehr kann andere Wege (mit anderer Lastsituation) nehmen
- Problem
  - Netz selbst liefert keine aktive Rückmeldung über Auslastung
  - Überlastsituationen können nur durch Beobachtung vermutet werden
- Zu frühes Senden von Dateneinheiten
  - Probleme mit dem Zeitgeber für Sendewiederholungen (Retransmission Timer)

  → gute Abschätzung der Umlaufzeit essentiell für guten Zeitgeber
    - Zu kleiner Wert: unnötige Sendewiederholungen treten auf
    - Zu großer Wert: langsame Reaktion auf Verlust der Dateneinheit

„Gute“ Zeitgeber

- Wichtiges Merkmal für die Effizienz eines Protokolls: Möglichst optimale Zeitgeber
  - Lokale Zeitmessung bedeutendes Hilfsmittel zur Fehlererkennung
    - v.a. wo andere aktive Rückmeldungen fehlen
  - Zeitgeber entscheiden z.B. wann
    - Verbindung abgebrochen wird
    - Dateneinheit als Verlust deklariert und erneut gesendet werden
  - Gratwanderung zwischen schneller Fehlererkennung und Vermeidung von Fehlalarmen
    - Zeitgeber soll im Fehlerfall so schnell wie möglich auslösen
    - Zeitgeber soll nur im Fehlerfall auslösen
  - Probleme
    - Fehlende Informationen zur Wahl des richtigen Wertes für den Zeitgeber
    - Dynamische Änderungen der Situation

[Zhan86]

Bestimmung der Umlaufzeit

- Messung der Umlaufzeit (Round Trip Time, RTT)
  - Zeitgeber (Granularität variiert, bis zu 500 ms) wird benutzt. Beim Ablauf wird ein Zähler inkrementiert.
    - Typischer Wert bei Linux: 10 ms
  - SampleRTT
    - Gemessene Zeit vom Versenden der Dateneinheit bis zum Empfang der dazugehörigen Quittung
    - Sendewiederholungen werden ignoriert
      - Quittungen können in diesem Fall nicht eindeutig einer gesendeten Dateneinheit zugeordnet werden
      - Gemessene Zeit könnte daher verfälscht sein
- Glätten des gemessenen Wertes
  - Gemessener Wert kann stark schwanken
  - Es wird ein auf mehreren Messungen basierender Wert herangezogen: EstimatedRTT

Umlaufzeit und Zeitgeber

- Nach jedem erfassten Wert für SampleRTT wird der geglättete Wert der Umlaufzeit folgendermaßen bestimmt

$$\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

  - Exponential Weighted Moving Average (EWMA)
  - Neue Werte werden höher gewichtet als alte Werte
  - Einfluss eines gemessenen Wertes sinkt mit der Zeit exponentiell

- Typischer Wert für α: 0.125
  - EstimatedRTT = 0.875 * EstimatedRTT + 0.125 * SampleRTT

- Empfehlung für Zeitgeber für Sendewiederholungen (Retransmission Timer) aus RFC 793

$$\text{Retransmission Timer} = \beta * \text{EstimatedRTT}$$


[Post81]

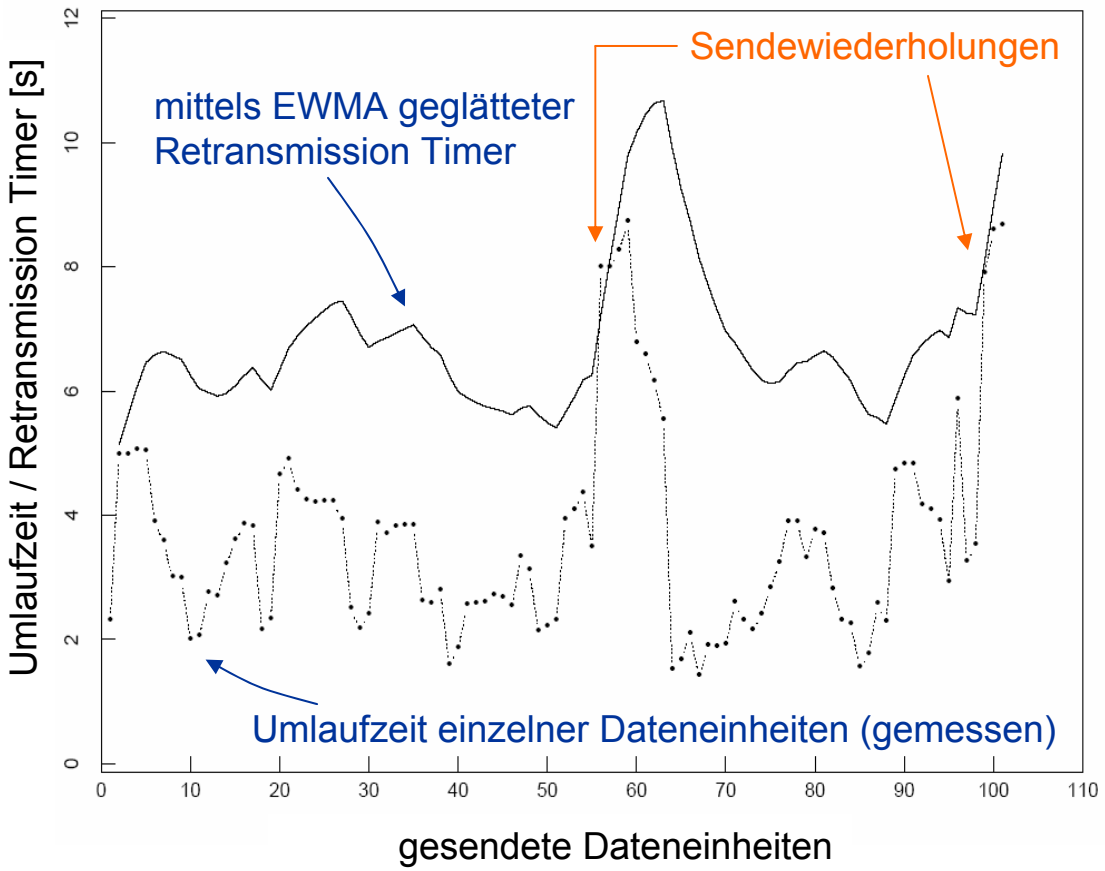
  - In RFC 793 empfohlener Wert für β: 2

Umlaufzeit/Zeitgeber gemäß RFC 793

- Gemessene Daten
  - Standard-TCP-Verbindung über das frühere Arpanet
    - Keine Verluste oder Wiederholungen von Dateneinheiten
  - Zeitgeber gemäß RFC 793
    - α=0,1 und β=2
- Darstellung
  - Punkte repräsentieren Dateneinheit
  - X-Achse: Nummer der Dateneinheit
  - Y-Achse: Zeit zwischen Senden der Dateneinheit und Empfang der zugehörigen Quittung
- Beobachtung
  - Teilweise hohe Abweichungen
  - Gelegentliche Sendewiederholungen

  → bessere Abschätzung der Schwankungen wünschenswert

[JaKa88]



Abschätzung der Schwankungen

- Ziel
  - Vermeidung der beobachteten gelegentlichen Sendewiederholungen
- Beobachtung
  - Schwankungen der Umlaufzeit können in hoch belasteten Netzen stark ansteigen
    - Bei hohen Schwankungen von EstimatedRTT sollte ein höherer „Sicherheitszuschlag“ gegeben werden
    - Fehler, d.h. Unterschied zwischen gemessener und geschätzter Umlaufzeit wird mit einbezogen

$$\text{Deviation} = (1 - \gamma) * \text{Deviation} + \gamma * | \text{SampleRTT} - \text{EstimatedRTT} |$$

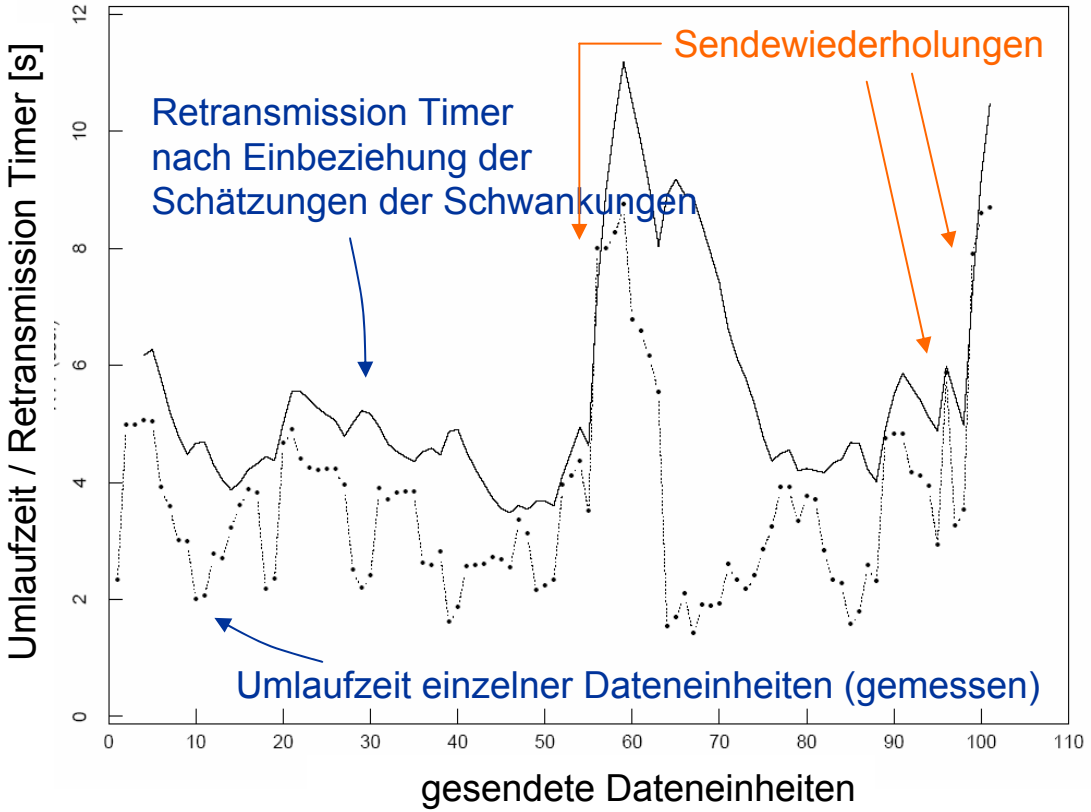
$$\text{Retransmission Timer} = \text{EstimatedRTT} + \beta * \text{Deviation}$$

- Empfohlene Standardwerte:
  - $\alpha = 0{,}125$; $\beta = 4$, und $\gamma = 0{,}25$

Umlaufzeit/Zeitgeber mit Verbesserungen

[JaKa88]

- Gemessene Daten
  - Wie zuvor
  - Verwendete Werte:
    - $\alpha = 0{,}125$; $\gamma = 0{,}25$ und $\beta = 4$
- Beobachtung
  - Abweichungen zwischen Zeitgeber und Umlaufzeit geringer
  - „Sinnvollere“ Sendewiederholungen aufgrund der besseren Schätzung



Mehrfache Sendewiederholungen – Probleme beim Sender

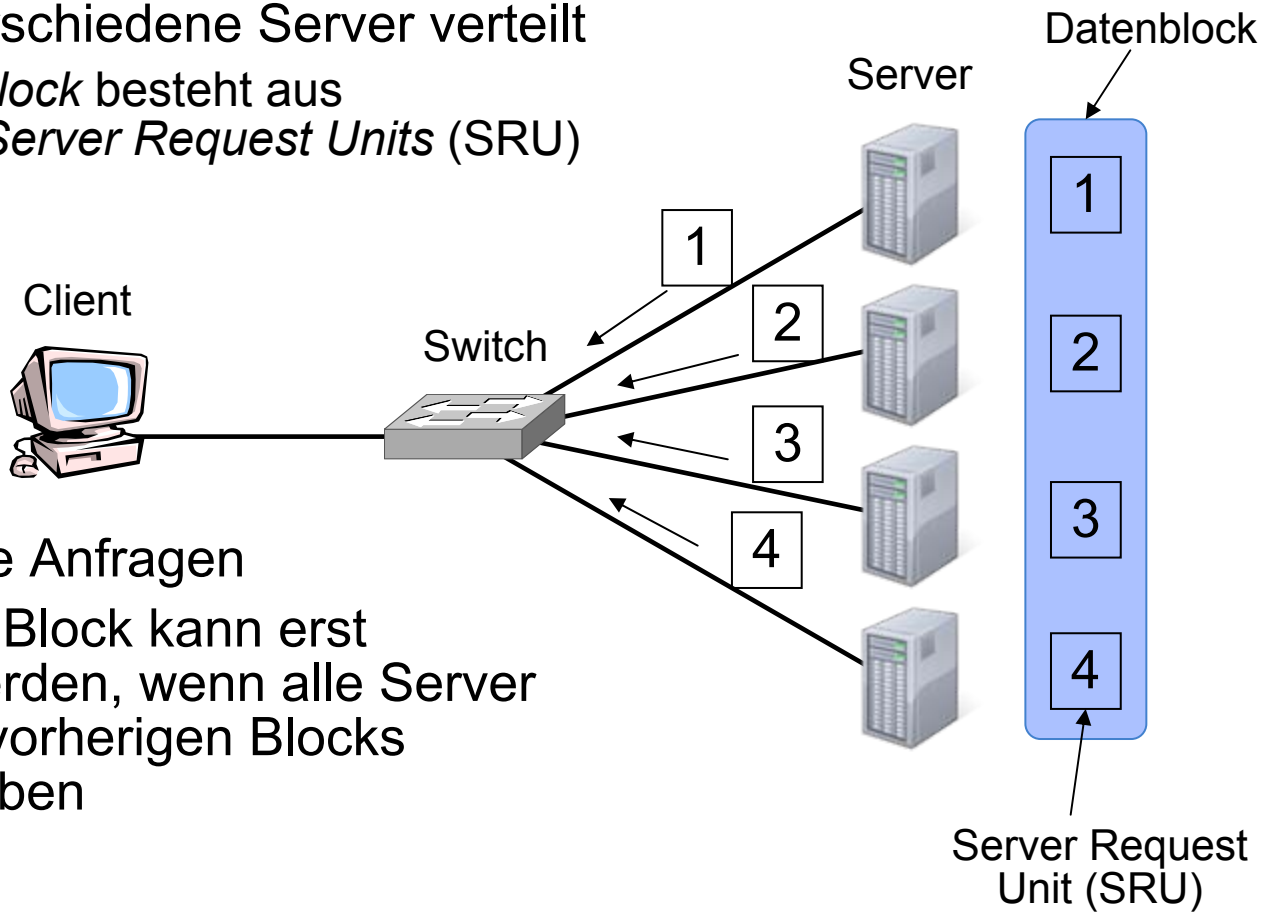
- Problem 1
  - Wie groß ist der Zeitraum zwischen zwei Sendewiederholungen der gleichen Dateneinheit?
- Vorgehensweise
  - Exponentieller Backoff
    - Vgl. Sendewiederholungen bei Ethernet
    - Bei jeder erneuten Wiederholung folgende Berechnung
      - Retransmission Timer = 2 ∗ Retransmission Timer
      - Maximaler Wert kann verwendet werden und sollte ≥ 60 Sekunden sein
- Problem 2
  - Gehört Quittung zu ursprünglicher oder zu wiederholter Dateneinheit?
- Vorgehensweise
  - Karn´s Algorithmus
    - Bei Quittung zu wiederholter Dateneinheit EstimatedRTT und Deviation nicht neu berechnen
    - Backoff wie oben berechnen
    - Zeitgeber auf Basis des Backoff benutzen, bis Quittung zu einer nicht wiederholten Dateneinheit eintrifft
    - Dann ursprünglichen Algorithmus wieder aktivieren

Einschub: Weitere Zeitgeber in TCP

- Neben dem Zeitgeber für Sendewiederholungen (Retransmission Timer) verfügt TCP noch über eine Reihe weiterer Zeitgeber, z.B.
  - Persist Timer
    - Falls Empfangsfenstergröße auf Null ist und Quittungen verloren gehen, kann es zu einem Deadlock kommen
    - Persist-Timer initiiert regelmäßige Nachfragen nach der Empfangsfenstergröße
    - Wert des Zeitgebers wird mit TCP exponential backoff berechnet
  - Keepalive Timer
    - Erkennt, wenn der Kommunikationspartner Probleme hat
      - Über eine TCP-Verbindung im Idle-Zustand fließen keine Daten
      - Server sendet nach Ablauf des Keepalive Timers eine Dateneinheit ohne Nutzdaten
      - Falls kein ACK vom Client empfangen wird, werden weitere n Keepalive-Dateneinheiten gesendet bevor TCP-Verbindung terminiert wird
    - Ist nicht Bestandteil der TCP-Spezifikation, stellt eine Option dar
    - Kann dazu führen, dass bestehende Verbindungen terminieren
      - z.B. bei einem temporären Problem auf der Vermittlungsschicht
    - Kann Servern helfen, Ressourcen nicht unnötig zu belegen, falls Client abgestürzt ist
  - 2MSL-Timer
    - Misst die Zeit, die eine Verbindung im TIME_WAIT-Zustand verbringt

Einschub: TCP in Datacenters

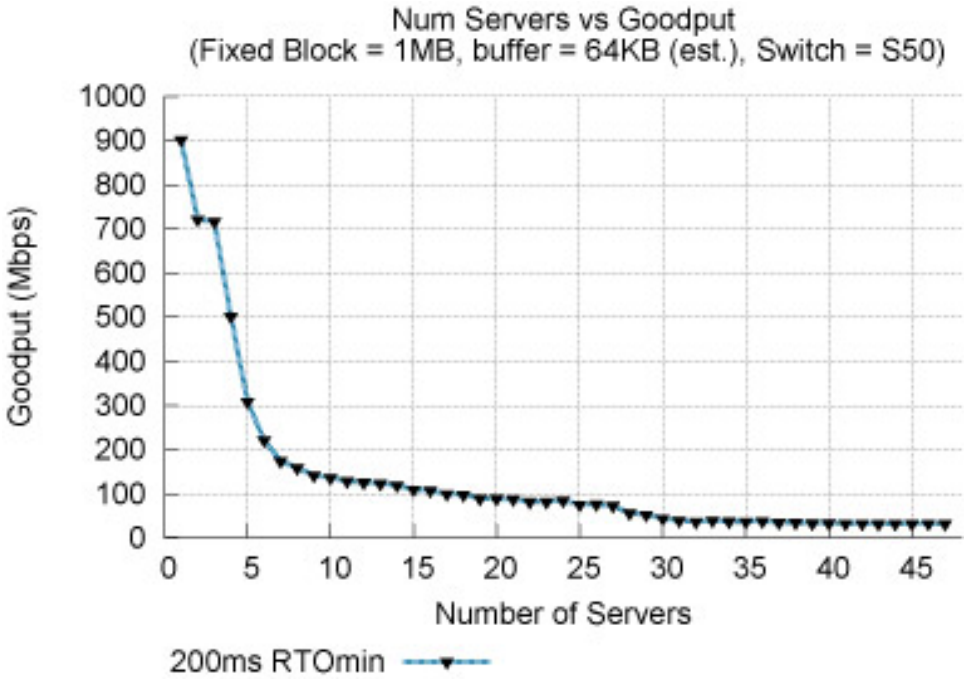
- Cluster-basierte Speichersysteme
  - Sehr geringe Latenz (RTT zw. 10 – 100µs), sehr hoher Durchsatz
  - Daten über verschiedene Server verteilt
    - Ein *Datenblock* besteht aus mehreren *Server Request Units* (SRU)



  - Synchronisierte Anfragen
  - Nächster Data Block kann erst angefordert werden, wenn alle Server ihren Teil des vorherigen Blocks ausgeliefert haben

Einschub: TCP in Datacenters

- Allerdings: TCP Incast Problem
  - Synchronisierte Anfragen können Puffer des Client-Ports am Switch zum Überlaufen bringen
    → Verlust von Dateneinheiten
  - TCP Durchsatz (*Goodput*) kann zusammenbrechen
  - RTO_{min} kritische Variable

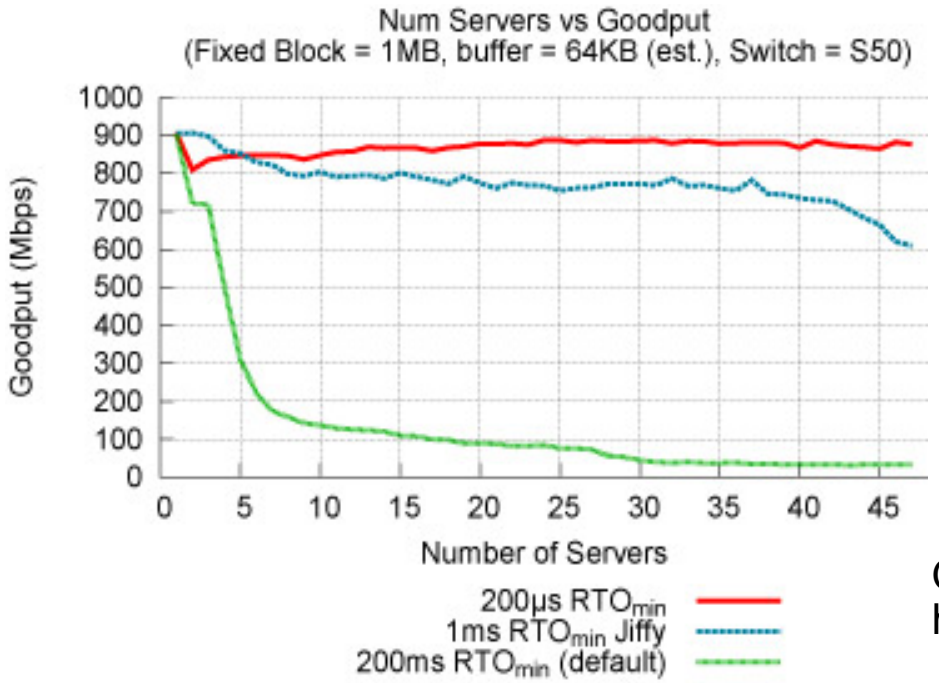


Quelle:
http://www.pdl.cmu.edu/Incast/

Einschub: TCP in Datacenters

- Lösungsansatz: TCPs Minimum Retransmission Timeout (RTO_{min}) reduzieren
  - Granularität von Millisekunden auf Mikrosekunden steigern
    - Mithilfe der *Linux High Resolution Timers*
    - Kein negativer Einfluss auf „wide area transfers“
  - Randomisierungsfaktor zu RTO-Berechnung hinzufügen
    - Retransmissions desynchronisieren

[Vasu09]



Quelle:
http://www.pdl.cmu.edu/Incast/

Zusammenfassung: Zeitgeber sind wichtig…

- Um eine Verbindung im Gleichgewicht zu halten
  - Unnötige Sendewiederholungen vermeiden
  - Optimale Zeitgeber werden benötigt
    - Berechnung abhängig von der geglätteten Umlaufzeit
    - Einbeziehung der tatsächlichen Schwankungen um „sinnvollere“ Sendewiederholungen zu erreichen

2.2.1.1 Verbindung kommt nicht in das Gleichgewicht 

2.2.1.2 Sender sendet neue Dateneinheit zu früh 

2.2.1.3 Ressourcenbeschränkungen verhindern Gleichgewicht

2.2.1.3 Ressourcenbeschränkungen verhindern Gleichgewicht

- Bisher betrachtet
  - Gleichgewicht einer einzelnen Verbindung im Fall eines Engpasses
    - Slow Start ermöglicht Senden mit nahezu maximaler Datenrate am Engpass
    - Zeitgeber werden verwendet um Verbindung im Gleichgewicht zu halten
- Im Folgenden
  - Verhalten mehrerer Verbindungen im Fall eines Engpasses
    - Zeitgeber zeigt Verlust einer Dateneinheit an
      - D.h. bei ausbleibender Quittung wird eine Stausituation vermutet
    - Sender testet durch Erhöhen der Datenmenge, wieviel er ohne Datenverlust senden kann

Ressourcenbeschränkungen verhindern Gleichgewicht

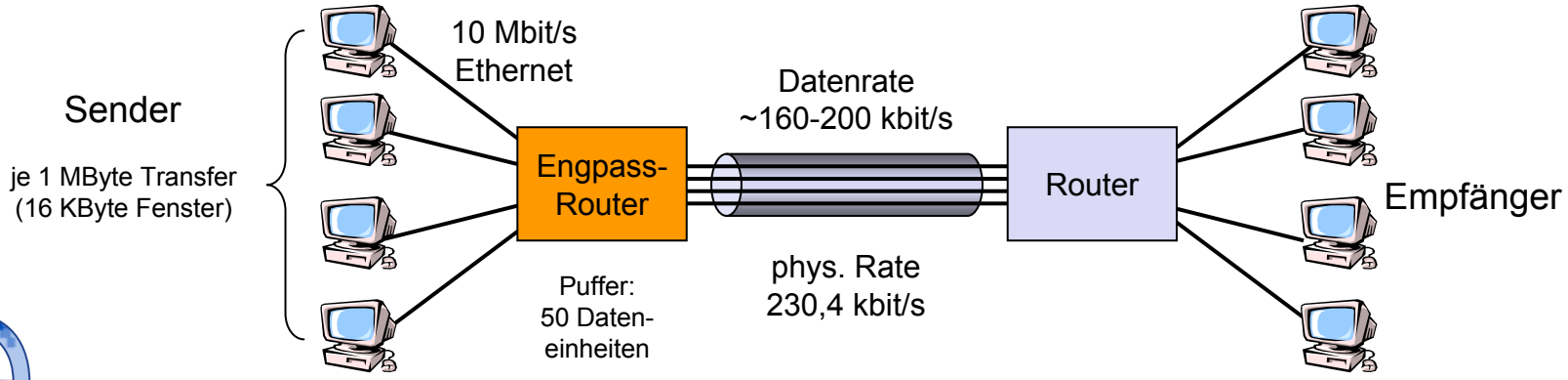
- Aber: Erhöhen der Datenmenge durch Slow Start ist zu aggressiv
  - Van Jacobson: „*it is easy to drive a network into saturation but hard for the net to recover*“
  - Kleinrock: „*long-tail effect during a rush hour period*“
- Die Länge der Warteschlange am Engpass steigt in Stausituationen exponentiell
- Slow Start erhöht das Staufenster nach Vermutung einer Stausituation wieder exponentiell und verschlechtert so die Situation am Engpass zusätzlich
- Das Netz ist dadurch nicht mehr in der Lage sich von der Stausituation zu erholen

- Abhilfe: Congestion Avoidance
  - Lineares Erhöhen des Staukontrollfensters
  - Empfangene Quittung: CWnd + = 1 / CWnd

Stausituation mit/ohne Congestion Avoidance

[JaKa88]

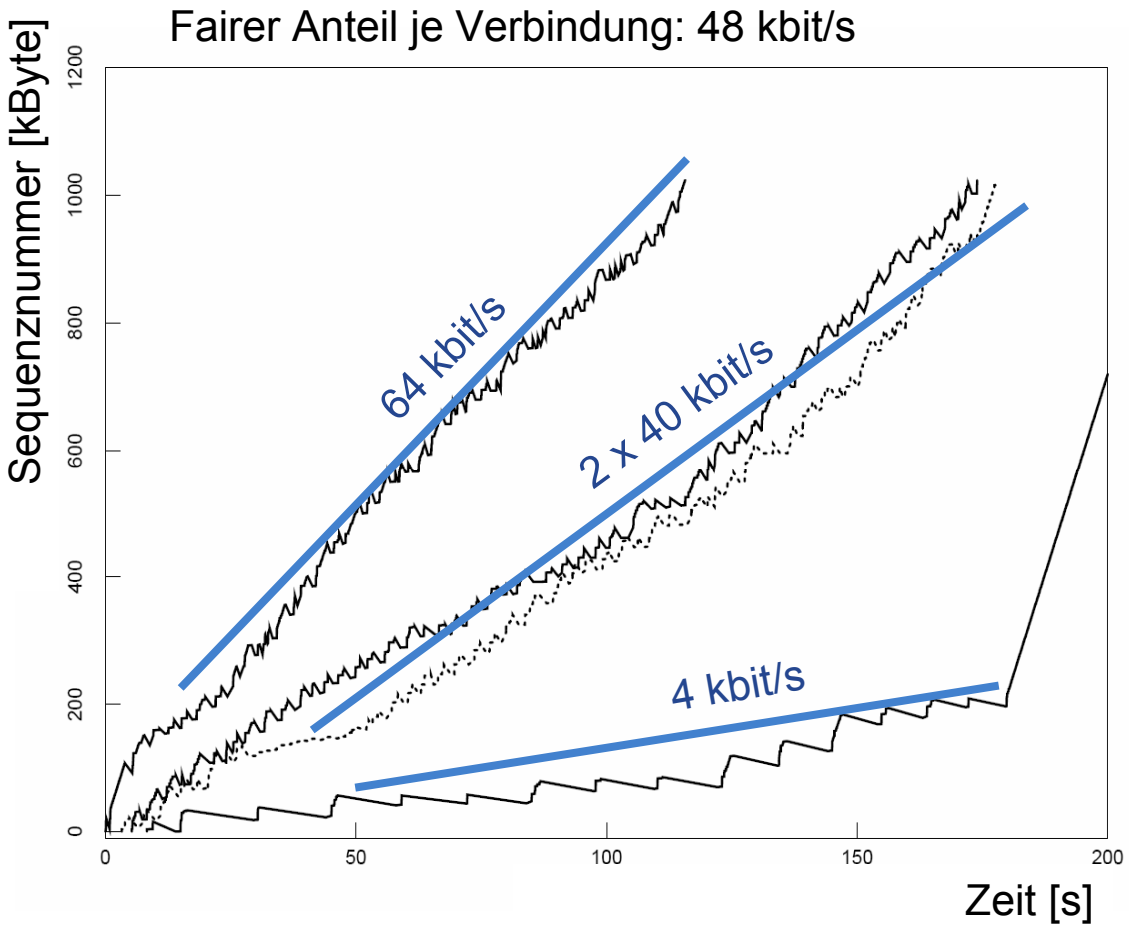
- Ziel
  - Beobachtung des Verhaltens konkurrierender TCP-Verbindungen in Stausituationen mit und ohne Congestion Avoidance
  - Betrachtung der Fairness und der effektiv genutzten Kapazität
- Aufbau des Experiments
  - Mehrere TCP-Verbindungen (max. 4) über einen gemeinsamen Engpass-Link
    - Engpass-Router kann 50 Dateneinheiten puffern
  - Alle 3s Start eines 1 MByte Transfers (2048 Dateneinheiten à 512 Bytes)
    - Empfangsfenster: 16 KByte (32 Dateneinheiten à 512 Bytes)
  - Jeweils zwei Verbindungen können den Puffer des Eingangsrouters überladen, alle vier Verbindungen überlasten die Warteschlangenkapazität um 160%.
- Skizze des Aufbaus



Vier simultane TCP-Verbindungen ohne Congestion Avoidance

[JaKa88]

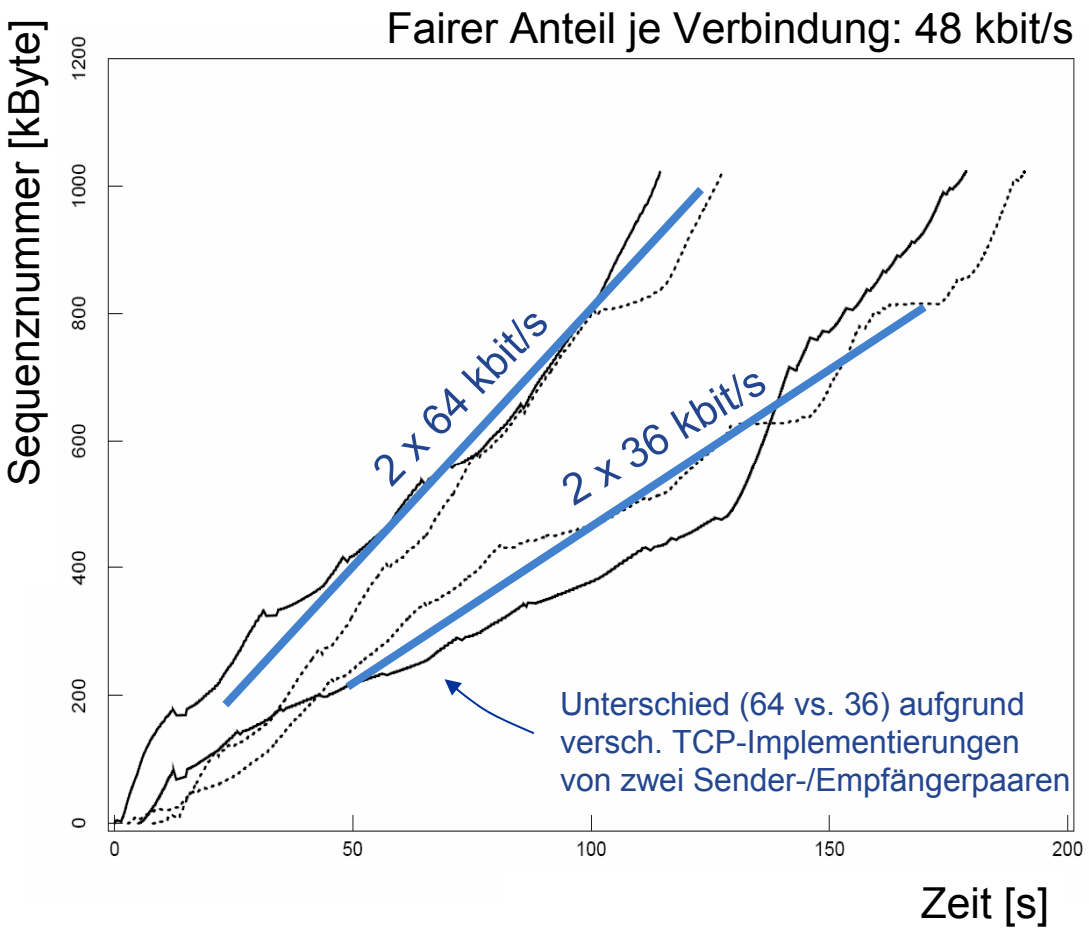
- Beobachtungen
  - 4000 von 11.000 gesendeten Dateneinheiten waren Wiederholungen!
    - Fast 50%
  - Jede Verbindung sollte 48 kbit/s erhalten
    - Aber: sehr unfaire Aufteilung
    - Ca. 48 kbit/s nicht genutzt
      - Für unnötige Wiederholungen verschwendet



Vier simultane TCP-Verbindungen mit Congestion Avoidance

[JaKa88]

- Beobachtungen
  - nur 89 von 8281 gesendeten Dateneinheiten waren Wiederholungen
    - Ca. 1%
  - Keine ungenutzte Kapazität
  - Verschiedene Quittungsstrategien
    - Verzögerte Quittungen bis 35% des Empfangsfenster gefüllt oder 200 ms vergangen
      - 36 kbit/s
      - Bursts mit 5-7 Dateneinheiten je Quittung. Verlustrate von 1,8%
    - Verzögerte Quittung für höchstens eine Dateneinheit
      - 64 kbit/s
      - Max. 2 Dateneinheiten pro Burst. Verlustrate von 0,5%



Zusammenfassung: Congestion Avoidance hilft…

- Faire Aufteilung der verfügbaren Kapazität zu erzielen
  - Langsames „Herantasten“ durch lineares Erhöhen des Staukontrollfensters
  - Keine ungenutzte Kapazität mehr mit Congestion Avoidance
  - Es treten kaum Sendewiederholungen auf

2.2.1.1 Verbindung kommt nicht in das Gleichgewicht 

2.2.1.2 Sender sendet neue Dateneinheit zu früh 

2.2.1.3 Ressourcenbeschränkungen verhindern Gleichgewicht 

Implizite Staukontrolle in TCP

- Kombination von Slow Start und Congestion Avoidance
- Es muss immer gelten

  LastByteSent – LastByteAcked <= min {CWnd, RcvWindow}

- Start:
  - Congestion Window CWnd = 1 MSS bzw.
    - bzw. seit RFC 2581: Initial Window IW <= 2*MSS und CWnd = IW

[AIPB09]

  - Schwellenwert SSThresh markiert den Übergang zwischen Slow Start und Congestion Avoidance
    - Initial auf Unendlich gesetzt
  - Weitere Annahme
    - Für jede empfangene Dateneinheit wird eine Quittung gesendet
      - D.h. es werden keine kumulativen Quittungen verwendet
- Ablauf
  - Solange CWnd ≤ SSThresh und Quittungen rechtzeitig empfangen Slow-Start
    - Exponentielles Erhöhen des Staukontrollfensters
    - Verdopplung des Staukontrollfensters nachdem alle gesendeten Dateneinheiten eines „vollen“ Staukontrollfensters mit ACKs bestätigt wurden (und diese ACKs empfangen wurden)
      - Empfangene Quittung: CWnd += 1

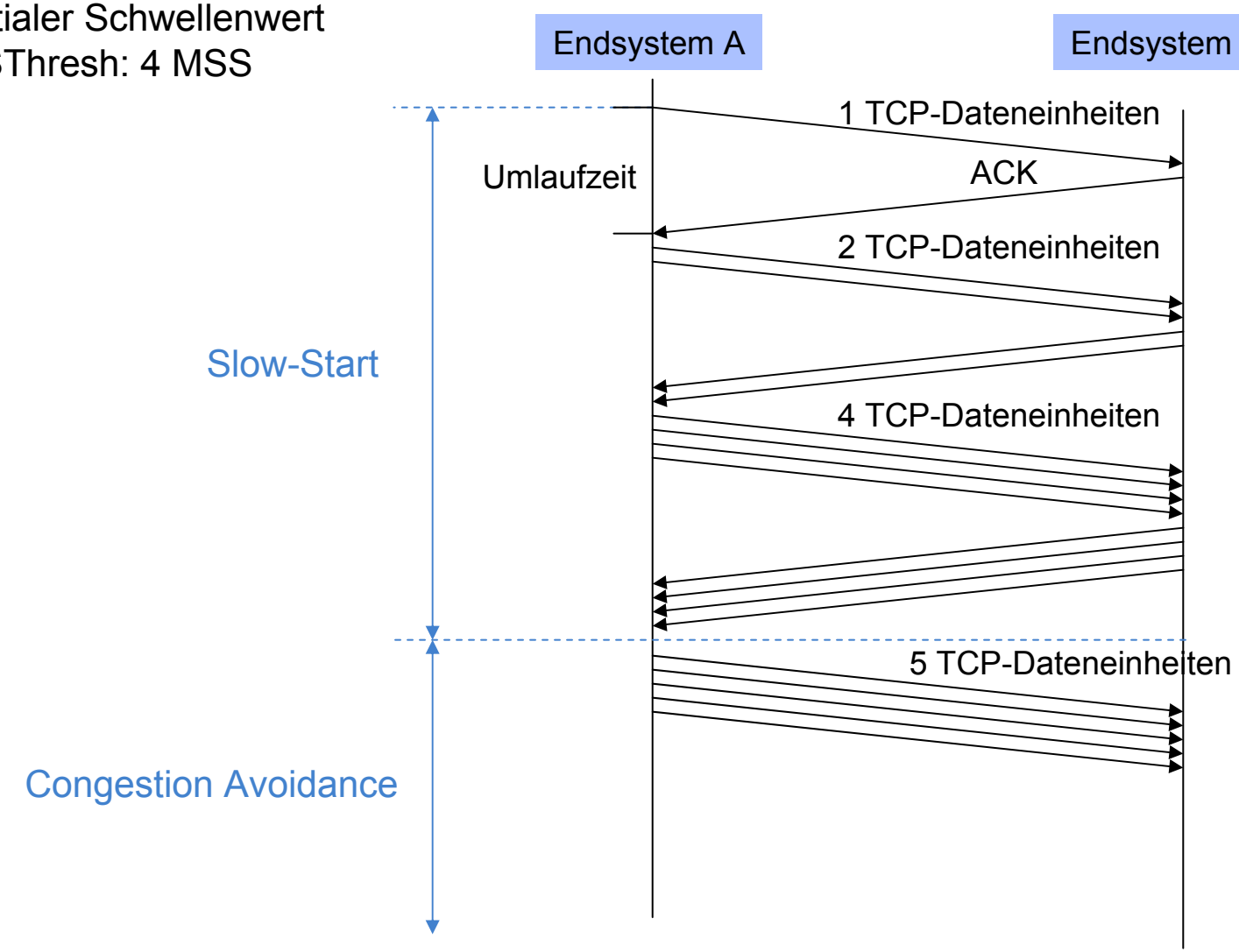
Implizite Staukontrolle in TCP

- CWnd > SSThresh und Quittungen rechtzeitig empfangen
  Congestion Avoidance
  - Lineares Erhöhen des Staukontrollfensters (additive increase)
  - Erhöhen des Staukontrollfensters um 1 pro vollständig gesendetem und bestätigtem Staukontrollfenster
    - Empfangene Quittung: CWnd += 1 / CWnd
- Quittung nicht rechtzeitig empfangen (Zeitgeber abgelaufen)
  - Signal für Datenverlust wegen Stausituation (Stausituation wird vermutet)
  - SSThresh folgendermaßen neu gesetzt (multiplicative decrease)
    - SSThresh = max (FlightSize/2, 2*MSS)
    - FlightSize: Menge an Daten, die gesendet, aber noch nicht quittiert wurden
  - CWnd zurücksetzen (neue Slow-Start-Phase)
    - CWnd = 1 MSS bzw. CWnd = IW

- AIMD (additive increase, multiplicative decrease)
  - Grundlegende Vorgehensweise
    - Additives Erhöhen des Fensters um 1 je Umlaufzeit
    - Multiplikatives Erniedrigen des Fensters um Faktor 2 bei vermutetem Datenverlust (Quittung nicht rechtzeitig empfangen)

Weg-Zeit-Diagramm zur Staukontrolle

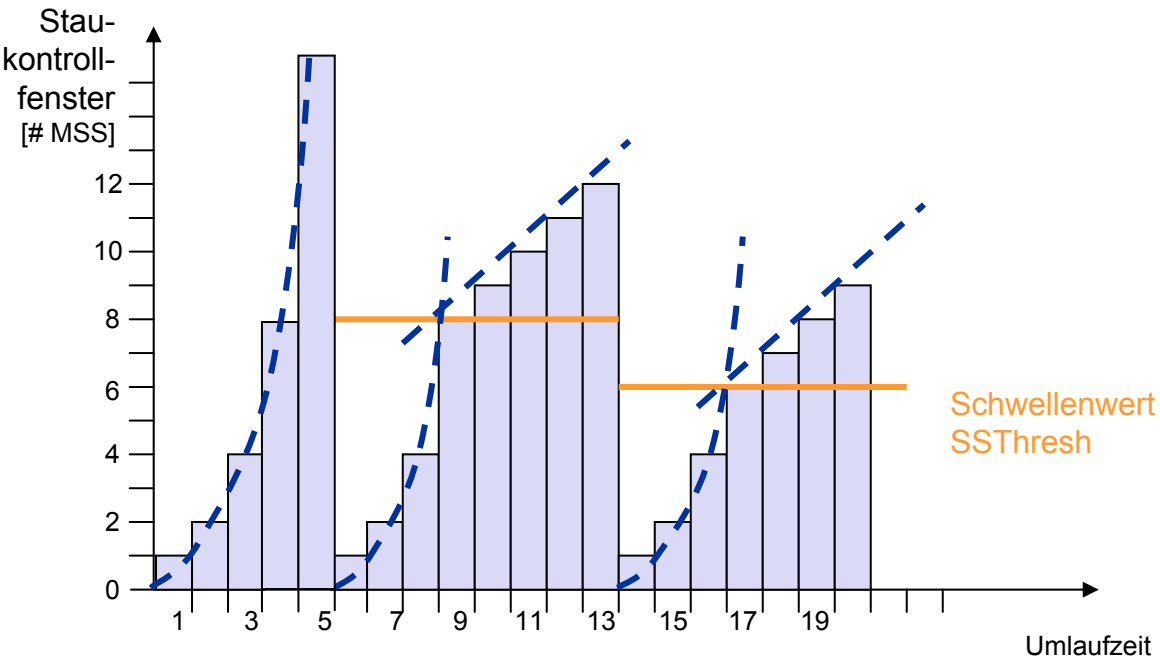
Initialer Schwellenwert
SSThresh: 4 MSS



Staukontrolle: Tahoe, Reno und Vegas

- ... verschiedene Varianten der TCP-Staukontrolle
- Tahoe
  - Der gerade beschriebene Algorithmus
  - Problem
    - Sender muss bei Datenverlust evtl. lang auf den Ablauf des Zeitgebers warten
    - Nicht jeder Datenverlust geht auf eine Stausituation zurück (z.B. aktives Warteschlangen-Management, Kap. 2.2.2)

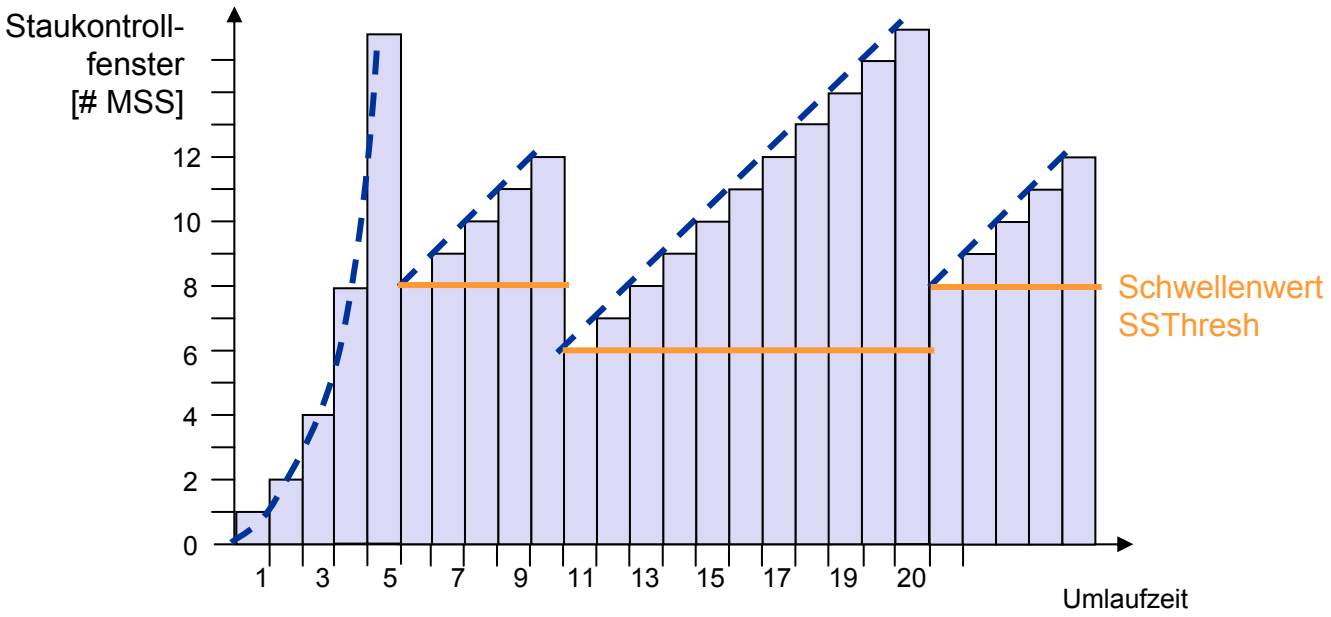
[Brad89]



Staukontrolle: Tahoe, Reno und Vegas

- ... verschiedene Varianten der TCP-Staukontrolle
- Reno
  - Umfasst Fast-Retransmit
    - Empfänger sendet sofortige Quittung, wenn er eine Dateneinheit außerhalb der Reihenfolge erhält
    - Sender startet Sendewiederholung, falls 4 gleiche Quittungen (sog. Duplikate) empfangen wurden
  - Umfasst Fast-Recovery
    - Slow-Start-Phase wird nach Fast-Retransmit nicht verwendet
  - Ist in vielen Betriebssystemen implementiert, z.B. in den Linux-Distributionen Ubuntu, Debian und openSUSE sowie in den Windows-Versionen 98 und 2000

[AIPB09]



Staukontrolle: Tahoe, Reno und Vegas

- ... verschiedene Varianten der TCP-Staukontrolle


[FIHG04]

- NewReno
  - Verbessert Fast-Recovery für Fälle, in denen mehrere Dateneinheiten verloren wurden
    - Kumulative Quittung für die wiederholte Dateneinheit bestätigt nicht alle ausstehenden Daten (partial ACK)
    - Fast Retransmit wird für die nächste fehlende Dateneinheit sofort angestoßen
      - Erkennbar an der kumulativen Quittungen für die wiederholte Dateneinheit
    - Retransmission Timer wird zurückgesetzt
      - Verhindert „Rückfall“ in Slow Start aufgrund eines Timeouts


[BrMP94, HeBG00]

- Vegas
  - Idee: Stausituationen erkennen bevor Datenverluste auftreten, dann lineare Reduzierung (anstelle der multiplikativen Erniedrigung bei Tahoe und Reno)
  - Vermuten einer Stausituation basierend auf steigenden Umlaufzeiten
  - Heute wenig implementiert
  - Schlechtere Fairness
    - Vegas-Verbindungen bekämen ggfs. mehr Bandbreite als simultane (New) Reno-Verbindungen

Zur Generierung von TCP-Quittungen

Ereignis	Reaktion des TCP-Empfängers
• Ankunft einer Dateneinheit in Reihenfolge • Alle Daten davor bereits quittiert • Keine Lücken	• Verzögerte Quittung • Bis zu 500 ms warten auf weitere reihenfolgetreue Dateneinheit • Wird keine weitere empfangen, Quittung senden
• Ankunft einer Dateneinheit in Reihenfolge • Alle Daten davor bereits quittiert • Bereits eine weitere reihenfolgetreue auf Quittung wartende Dateneinheit • Keine Lücken	• Sofortiges Senden einer kumulativen Quittung • Beide Dateneinheiten werden quittiert
• Ankunft einer Dateneinheit mit einer nicht erwarteten höheren Sequenznummer • Erkennen einer Lücke	• Sofortiges Versenden einer duplizierten Quittung mit der Sequenznummer des nächsten erwarteten Bytes
• Ankunft einer Dateneinheit, die teilweise oder komplett eine Lücke bei den empfangenen Daten auffüllt	• Sofortiges Versenden einer Quittung, falls die Dateneinheit an der unteren Schranke der Lücke beginnt

- Stehen sendebereite Daten beim Empfänger zur Verfügung, werden Quittungen per Piggyback gesendet

[AIPB09]

Zusammenfassung „Conservation of Packets“

- An dieser Stelle seien noch einmal die drei wesentlichen Aspekte aufgezählt, um eine TCP-Verbindung im Gleichgewicht zu halten

2.2.1.1 Verbindung kommt nicht in das Gleichgewicht — *Slow-Start*

2.2.1.2 Sender sendet neue Dateneinheit zu früh — *Zeitgeber*

2.2.1.3 Ressourcenbeschränkungen verhindern Gleichgewicht — *Congestion-Avoidance*

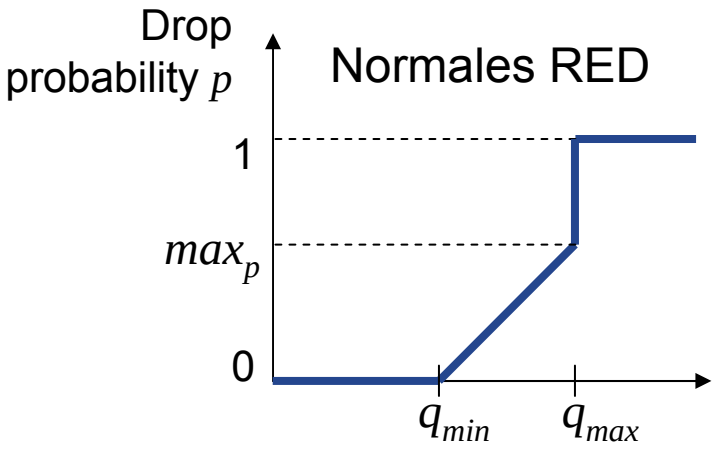
- Durch diese Verbesserungen, die aus Beobachtungen und Analysen entstanden sind, wurde TCP bis heute „fit“ gehalten

2.2.2 Aktives Warteschlangemanagement

- Problem: Netzwerk ist „Black Box“
  - Endsysteme schließen auf Stausituation nur indirekt über Verlust einer Dateneinheit, d.h. Ausbleiben einer Quittung
    - Ruft teilweise Synchronisation von TCP-Strömen hervor
- Ansatz
  - Verhalten des Netzes ändern
  - Aktives Warteschlangenmanagement verhindert, dass Warteschlange in Überlastbereich kommt
- RED (Random Early Detection)
  - Verwerfen von Dateneinheiten im Netz, auch wenn noch Platz in der Warteschlange vorhanden ist
  - Nicht alle TCP-Verbindungen sollen in einer Stausituation gleichzeitig Dateneinheiten verwerfen

RED – Random Early Detection

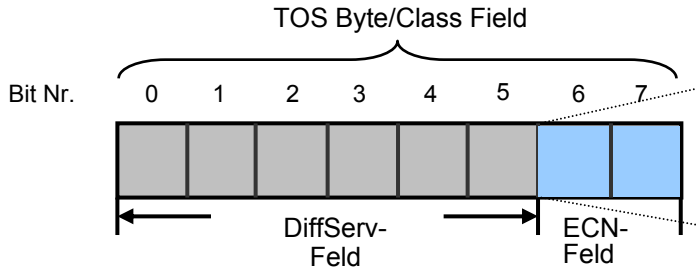
- Idee
  - Frühzeitiges Verwerfen von Dateneinheiten, bevor Warteschlange maximal gefüllt ist
    - Verhindert gleichzeitige globale Synchronisierung aller Sender
    - Zufällige Auswahl sorgt für mehr Fairness
    - Durchschnittliche Warteschlangenlänge wird kürzer
    - Stausituationen können frühzeitig erkannt werden
      - Explizite Staukontrolle
    - Aufwändigerer Warteschlangen-Mechanismus
    - RED-Parameter sind nicht einfach zu setzen



- Ablauf
  - Solange Länge der Warteschlange q_{min} nicht überschreitet, wird keine Dateneinheit verworfen (p = 0)
  - Nach Überschreitung von q_{min} steigt die Wahrscheinlichkeit, dass die Dateneinheit verworfen wird, linear mit der Warteschlangenlänge
  - Sobald die Warteschlange maximal gefüllt ist, werden alle Dateneinheiten verworfen (p = 1)

2.2.3 Explizite Staukontrolle: ECN

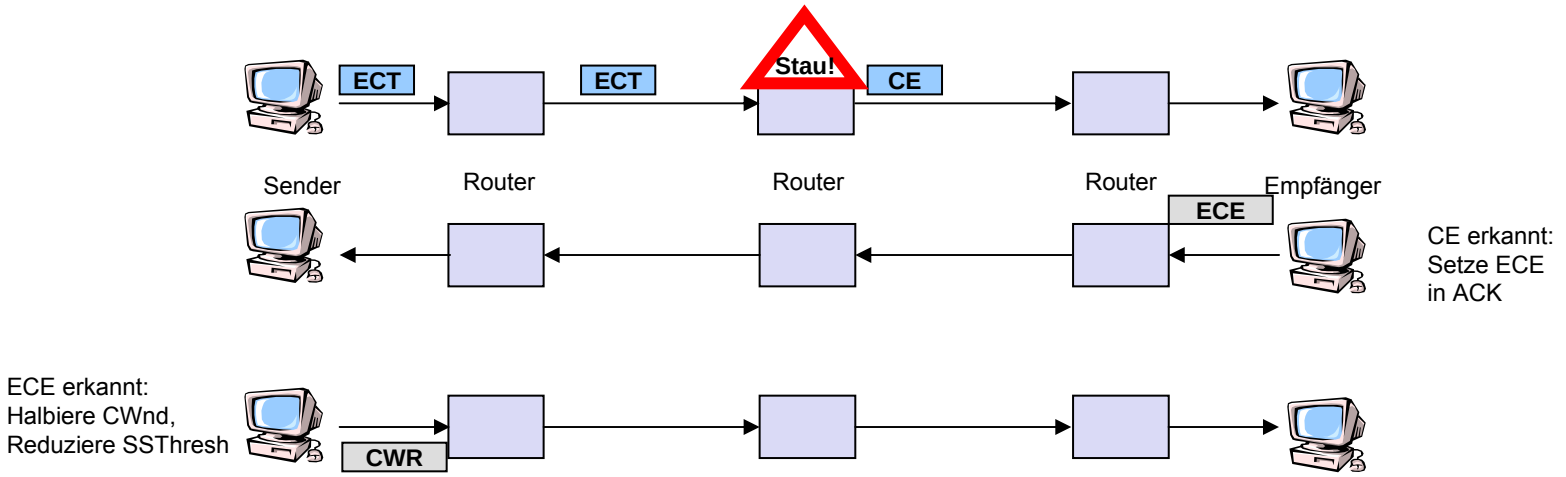
- Problem: Netzwerk ist „Black Box“ → Endsysteme schließen auf Stausituation nur indirekt über Verlust einer Dateneinheit, d.h. Ausbleiben einer Quittung
- Verlust einer Dateneinheit zur Erkennung einer Stausituation für verzögerungssensitive Anwendungen (z. B. Remote Login) ungünstig (Timer + RTT)
- Alternativer Vorschlag: Explizite Staukontrolle (Explicit Congestion Notification – ECN)
  - Vermeiden von Verlusten durch explizite Stauanzeige des Netzes [RaFB01]
    - IP-Erweiterung Explicit Congestion Notification (ECN)
    - Signalisiert durch niederwertigste 2 Bit im ehemaligen Type-of-Service-Feld des IP-Kopfs
  - ECN-Fähigkeit muss signalisiert werden, um Unfairness zu vermeiden
    - ECN-Capable Transport (ECT) Bits
  - Setzt aktives Warteschlangenmanagement (z.B. RED) im Router voraus
    - Anzeige muss erfolgen, bevor Warteschlange wirklich voll ist
    - Markierung der IP-Dateneinheit anstatt sie zu verwerfen (Congestion Experienced – CE)



0	0	Not-ECT
0	1	ECT(1)
1	0	ECT(0)
1	1	CE

Explizite Staukontrolle: ECN

- Signalisierung einer Stausituation durch CE-Bit (Schicht 3) an Empfänger
  - Reaktion auf Stau erfolgt aber beim Sender (Schicht 4)
  - Zusätzliche Flags im TCP-Kopf
    - ECE-Bit: Signalisiert dem Sender die Stausituation
    - CWR-Bit: Bestätigt den Erhalt einer Staumeldung
    - Beim TCP-Verbindungsaufbau werden beide Flags gesetzt, um ECN-Fähigkeit zu signalisieren
  - Sender-Reaktion auf Staumeldung
    - Halbiere Staukontrollfenster CWnd
    - Reduzieren Slow Start Schwellenwert SSThresh



2.2.4 Zusammenfassung „Dynamik und TCP“

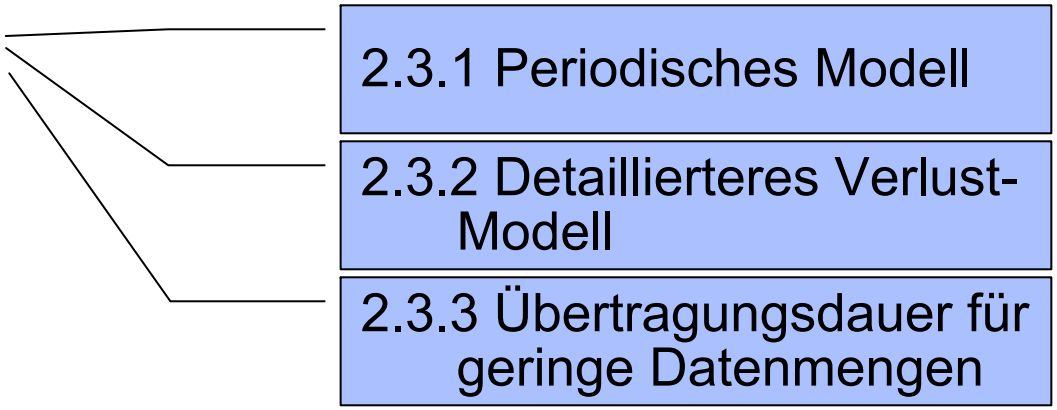
- „Conservation of Packets“
  - TCP-Verbindungen ins Gleichgewicht bringen …
    - Slow Start hilft dabei die verfügbare Kapazität optimal zu nutzen
  - … und im Gleichgewicht halten
    - „Gute“ Zeitgeber benötigt um unnötige Sendewiederholungen zu vermeiden
    - Congestion Avoidance vermeidet Unfairness im Fall von Ressourcenbeschränkungen (additive increase, multiplicate decrease)
- Aktives Warteschlangenmanagement
  - Vermeiden, dass TCP-Ströme gleichzeitig in Stausituation kommen
    - Frühzeitiges Verwerfen von Dateneinheiten im Netz
- Explizite Staukontrolle
  - Erkennen einer Stausituation vor dem Verlust einer Dateneinheit
    - Markieren von Dateneinheiten statt Verwerfen
    - Sender reagiert auf Stausituation mit Reduzierung des Staufensters

Homework

- Nutzen Sie das Tool Wireshark um verschiedene TCP-Verbindungen zu beobachten. Interpretieren Sie insbesondere die RTT sowie den Verlauf der empfangenen Sequenznummern. Wo sind die Effekte von Slow Start und Congestion Avoidance zu beobachten?
- Gegeben sei eine TCP-Verbindung mit einer Umlaufzeit von 120 ms. Der Schwellenwert sei auf 1600 Byte gesetzt. Die MSS betrage 500 Byte. Wie lange dauert die Slow-Start Phase und wieviel Daten konnten während dieser gesendet werden?
- Eine sendende TCP-Instanz sendet mit der „vollen“ Geschwindigkeit des angeschlossenen Links (100 Mbit/s). Auch der Empfänger ist mit 100 Mbit/s an das Netz angeschlossen. Dazwischen befindet sich eine Strecke mit einer Übertragungsrate von 100 kbit/s. Die MSS betrage 500 Byte. Mit welcher Zwischenankunftszeit kommen die Dateneinheiten beim Empfänger an?

2.3 Evaluierung von TCP

- Problemstellung
  - Ist TCP für meine Zwecke geeignet?
  - Welche Leistung (z.B. Datenrate, Verzögerung), welches Verhalten kann ich wann erwarten?
- Alternative Vorgehensweisen
  - Messung
  - Modellierung
    - Simulativ
    - Analytisch



Analytische Modellierung von TCP

- Stets zu beachten
  - Welche Variante von TCP mit welchen Erweiterungen wird betrachtet?
    - Im Folgenden meist Varianten, die ähnlich zu TCP Reno sind
- Grundlegende Frage
  - Welche Schlüsselaspekte sind für das Verhalten von TCP ausschlaggebend?
  - Im Folgenden zwei Schlüsselaspekte modelliert
    - Das dynamische Verhalten des Staukontrollfensters
    - Verluste von Dateneinheiten
      - Lässt u.a. auf Stausituationen im Netz schließen

Verwendete Variablen

- Für den besseren Überblick
  - X: Übertragungsrate in der Einheit Dateneinheiten pro Zeit [Dateneinheiten/s]
  - RTT: Umlaufzeit (Round Trip Time) [Sekunden]
  - p: Verlustwahrscheinlichkeit einer Dateneinheit
  - MSS: Maximale Größe einer Dateneinheit (Maximum Segment Size) [bit]
  - W: Größe des Staukontrollfensters in der Einheit Anzahl Dateneinheiten [MSS] (angegeben als Vielfaches der MSS)
  - E[z]: Erwartungswert des Parameters z
  - O: Größe der zu übertragenden Daten [bit]
  - D: Datenrate in der Einheit bit pro Zeit [bit/s]
  - b: Anzahl der pro Quittung quittierten Dateneinheiten (meist implizit b=1) angenommen

Dynamik des Staukontrollfensters

- Grundlegende Dynamik


[HaJa03]

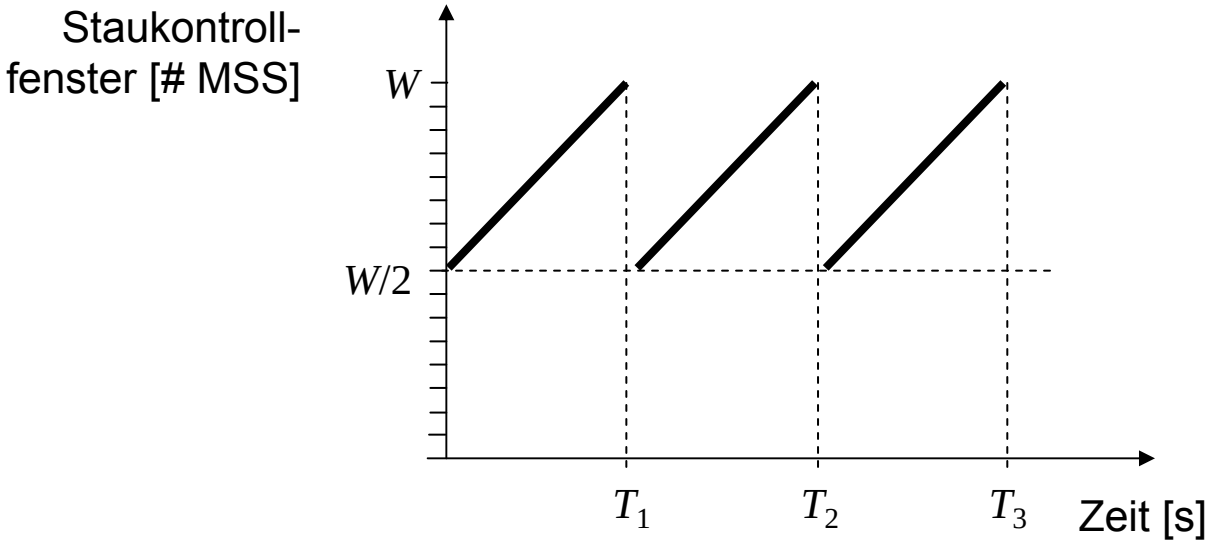
  - Bei Congestion Avoidance steigt Fenstergröße linear und wird multiplikativ erniedrigt (gemäß AIMD)
  - Lineare Erhöhung
    - Normalerweise um eine Dateneinheit je Umlaufzeit (Round Trip Time, RTT)
  - Multiplikatives Erniedrigen
    - Normalerweise auf die Hälfte der erreichten Fenstergröße beim Ausbleiben einer Quittung
  - Übertragungsrate X
    - Annahme: Fenstergröße W steht in Relation zu dieser Rate und zur Umlaufzeit RTT

$$X(t) = \frac{W(t)}{RTT}$$

      - Annahme: Steigende Rate hat vernachlässigbaren Einfluss auf Wartezeiten in den Warteschlangen der Zwischensysteme im Netz. Die Umlaufzeit bleibt dann effektiv konstant.

2.3.1 Periodisches Modell

- Einfaches Modell zur Modellierung von TCP
  - Basiert auf keiner konkreten Version von TCP
  - Geht bezüglich des Staukontrollfensters von linearer Erhöhung und multiplikativer Erniedrigung aus
    - Staukontrollfenster erreicht stets wieder gleichen Maximalwert W, bevor Verlust einer Dateneinheit auftritt
  - Konstante Wahrscheinlichkeit p für Verlust einer Dateneinheit
    - Im Mittel $1/p$ Dateneinheiten übertragen, bevor Verlust einer Dateneinheit auftritt
- Zeitlicher Verlauf des Staukontrollfensters



- Obwohl reales Verhalten stark vereinfacht wird, Herleitung wichtiger Eigenschaften von TCP möglich

Verhalten von Congestion Avoidance

- Makroskopische Analyse

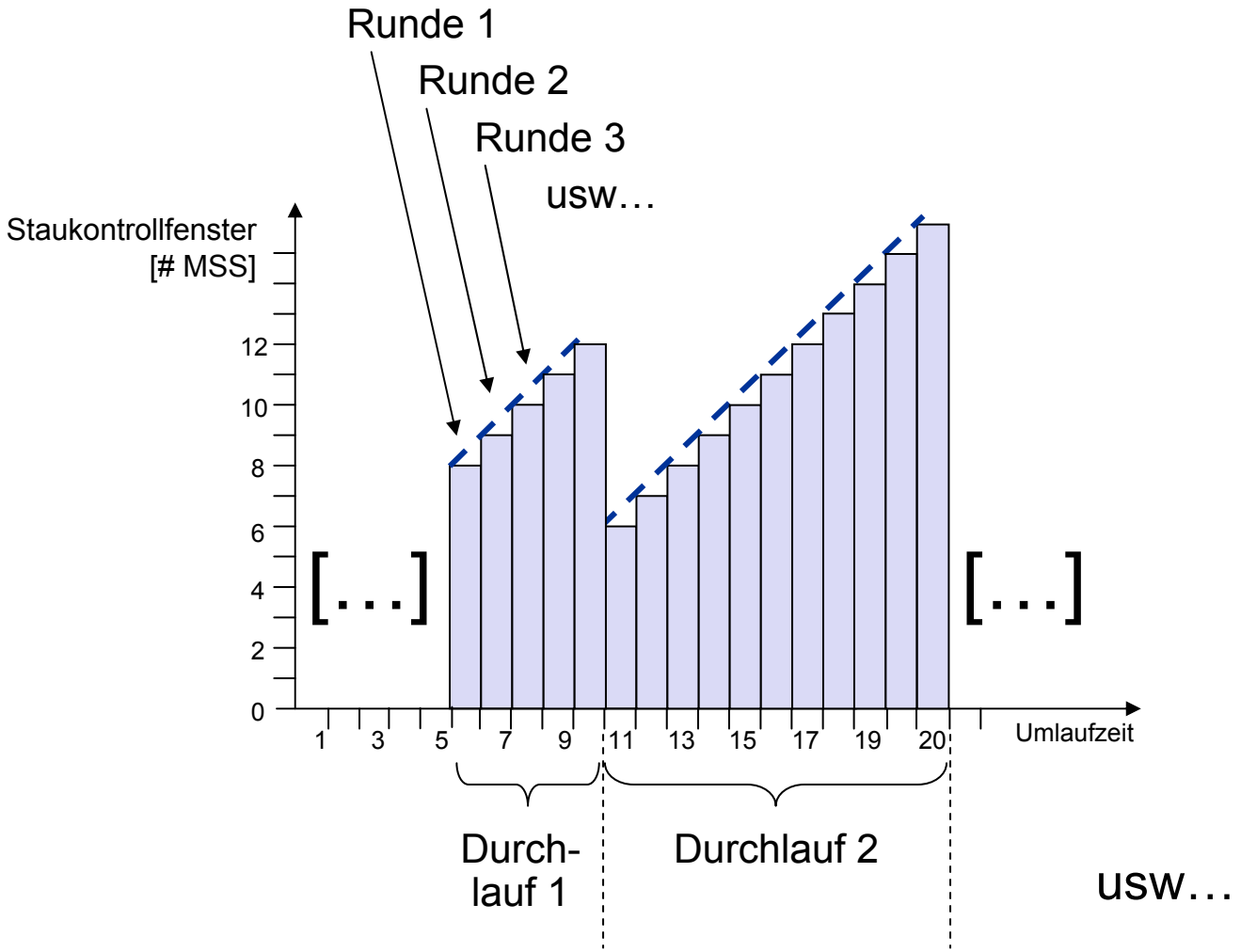

[Math97]

  - Annahmen
    - Sender hat stets Daten zu senden
      - Z.B. Übertragung einer großen Datei
    - Idealer TCP Congestion Avoidance
      - Slow Start wird vernachlässigt
      - Andere Einflüsse auf Leistungsfähigkeit werden vernachlässigt, d.h. Congestion Avoidance bestimmt allein die resultierende Leistungsfähigkeit
    - Unabhängige Stausignale
      - Stausignal entspricht Verlust einer Dateneinheit
    - p: Verlustwahrscheinlichkeit für eine Dateneinheit
      - Korrekte Übertragung von $1/p$ konsekutiven Dateneinheiten bevor Verlust auftritt
    - Stausignale treten periodisch auf
  - Beobachtung
    - Staukontrollfenster durchläuft eine perfekte Sägezahnkurve

Begriffe und Definitionen

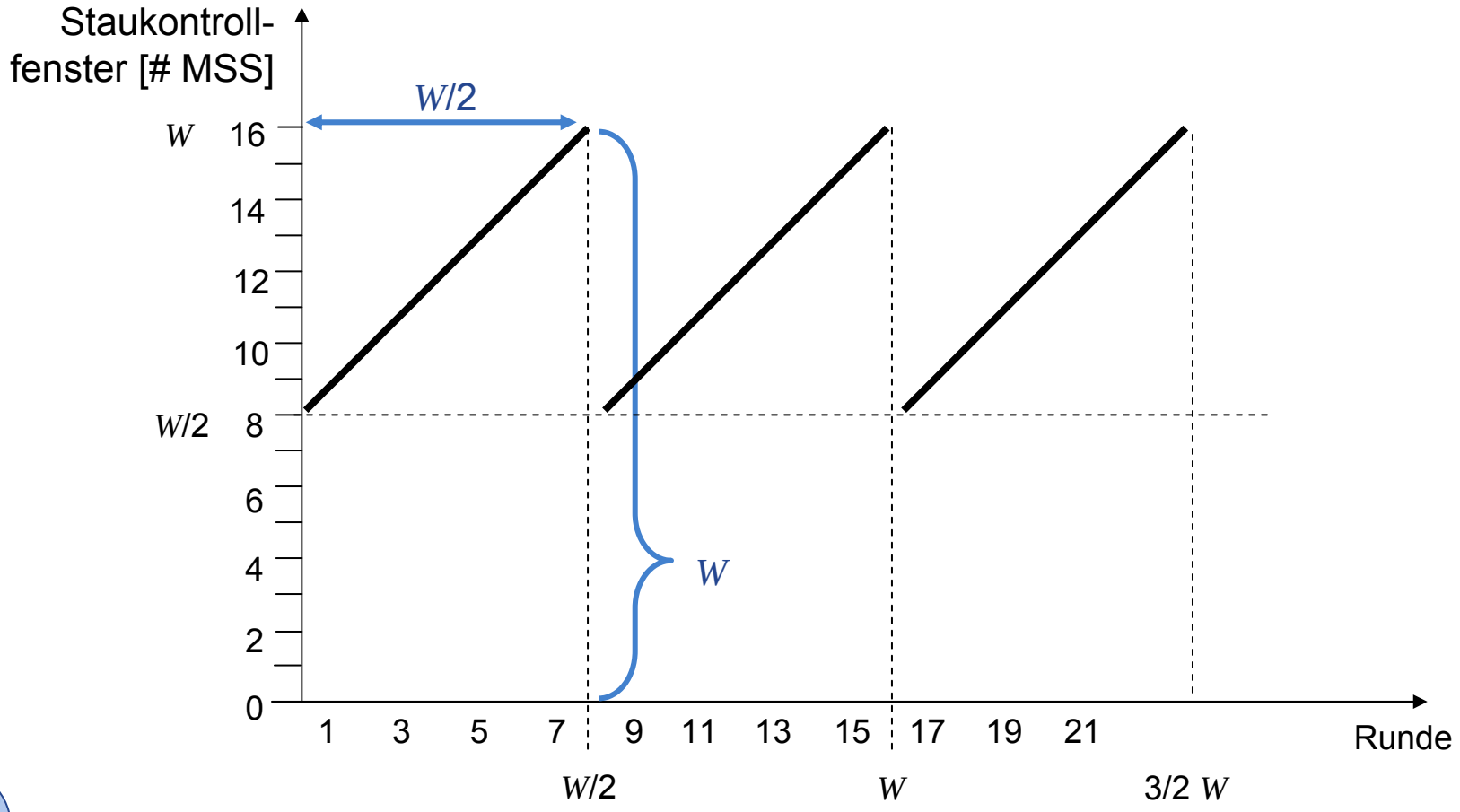
- Das Konzept Runde
  - Sei W die Größe des Staukontrollfensters
    - Annahme hier: Sendefenster nie der limitierende Faktor
  - Runde startet mit der Übertragung von W Dateneinheiten
    - Sind alle Dateneinheiten gesendet, so können vor Empfang einer Quittung keine weiteren gesendet werden
    - Diese Quittung markiert Ende der momentanen Runde und Beginn der nächsten Runde
  - Später in der Modellierung entspricht die Dauer einer Runde der Umlaufzeit RTT
- Durchlauf
  - (Mehrere) Runden bis zum Auftreten des Verlustes einer Dateneinheit.
    - Nach Verlust beginnt nächster Durchlauf

Beispiel für Runde und Durchlauf



Verlauf Staukontrollfenster

- Sei W diejenige Größe des Staukontrollfensters, ab der Datenverluste auftreten.



Leistungsfähigkeit von TCP

- Beobachtungen
  - Im eingeschwungenen Zustand Größe minimales Staukontrollfenster: $\frac{W}{2}$
    - Staukontrollfenster wächst um Eins je Runde
    - Durchläufe der Länge $\frac{W}{2}$ Umlaufzeiten, bzw. $\frac{W}{2} * RTT$ Sekunden entstehen
  - Anzahl der ausgelieferten Dateneinheiten ergibt sich aus Fläche unterhalb der Sägezahnkurve
    - Anzahl Dateneinheiten = $\frac{1}{2} * \frac{T}{RTT}\left(\frac{W}{2} + W\right)$
      - T entspricht einem Durchlauf bis Datenverlust auftritt:
    - Es gilt: $T = RTT * W/2$
    - Damit: Anzahl der Dateneinheiten in einem Durchlauf = $\frac{W}{4}\left(\frac{W}{2} + W\right) = \frac{1}{p}$
      - Im Beispieldiagramm der vorangegangenen Folie: 64 + 32 = 96 Dateneinheiten
      - Verlustwahrscheinlichkeit p = 1/96
  - Damit ergibt sich für W: $W = \sqrt{\frac{8}{3p}}$ bzw. für p: $p = \frac{1}{\frac{3}{8}W^2}$

Durchschnittliche Übertragungsrate einer TCP-Quelle

- Durchschnittliche Übertragungsrate
  - Dateneinheiten, die während der Dauer eines Durchlaufs gesendet werden

$$\overline{X}(p) = \frac{\#\ \text{Dateneinheiten}}{\text{Durchlauf}}$$

  - „Inverse Square-Root p Law“

$$\overline{X}(p) = \frac{1/\mathrm{p}}{\mathrm{RTT} * \mathrm{W}/2} = \frac{1}{RTT}\sqrt{\frac{3}{2p}}$$

  - Fundamentaler Zusammenhang zwischen Verlustwahrscheinlichkeit von Dateneinheiten p, der Umlaufzeit RTT und der möglichen Übertragungsrate von TCP
    - Übertragungsrate ist invers proportional zur Umlaufzeit und zur durchschnittlichen Verlustwahrscheinlichkeit

Beispiel zur mittleren Übertragungsrate

- Beispielkonfiguration
  - Umlaufzeit RTT = 200 ms
  - Verlustwahrscheinlichkeit einer Dateneinheit p = 0,05
  - Mittlere Übertragungsrate gemäß „Inverse Square Root p Law“

$$\overline{X} = \sqrt{\frac{3}{2p}} * \frac{1\,\text{Dateneinheit}}{RTT}$$

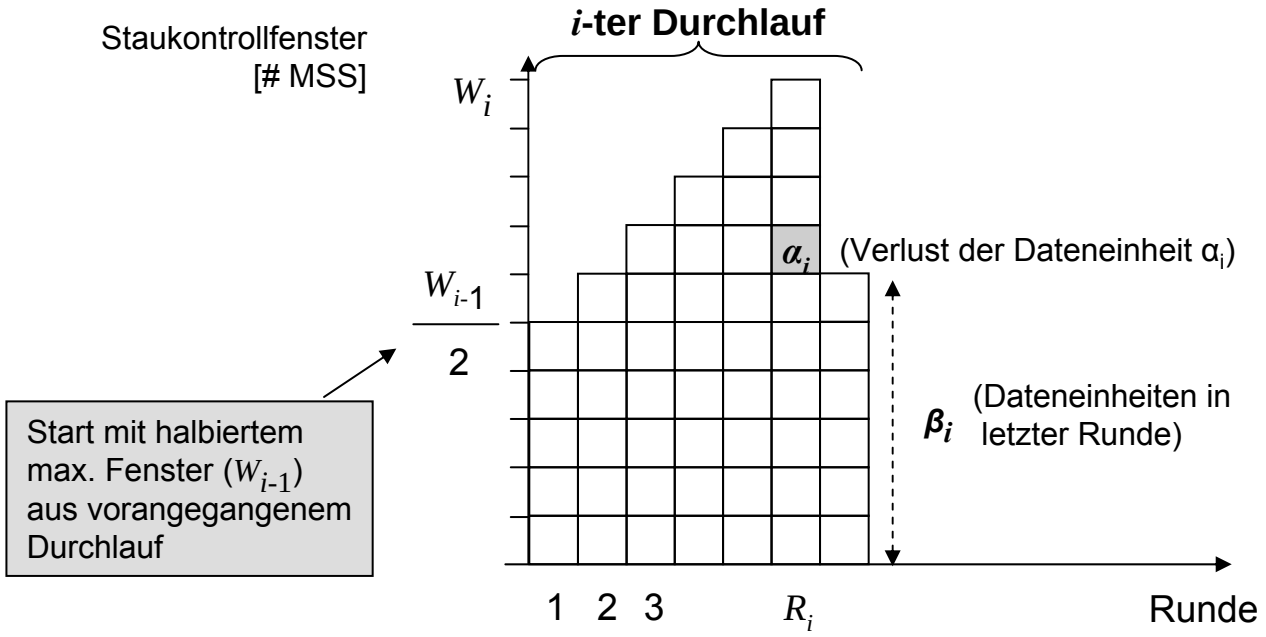
Achtung: Einheit im Beispiel soll Dateneinheiten/s sein, daher keine Multiplikation mit MSS!

$$\overline{X} = \sqrt{\frac{3}{2*0{,}05}} * \frac{1\,\text{Dateneinheit}}{0{,}2\,\text{s}} = 27{,}4\,\frac{\text{Dateneinheiten}}{\text{s}}$$

2.3.2 Detaillierteres Paketverlust-Modell

- Ziel
  - Bestimmung der durchschnittlichen Übertragungsrate im eingeschwungenen Zustand
- Annahme jetzt
  - Auslöser für das Erkennen des Verlustes einer Dateneinheit: Dreifacher Empfang einer duplizierten Quittung
    - In dieser Modellierung daher kein Slow-Start durch den Sender, lediglich Congestion Avoidance
- Vorgehensweise
  - Vergleichbar mit derjenigen beim periodischen Modell
  - Berechnen der Anzahl übertragener Dateneinheiten Y_i zwischen zwei aufeinander folgenden Verlusten von Dateneinheiten, d.h. in einem Durchlauf
    - A_i : Dauer des i-ten Durchlaufs
  - Bestimmt werden soll also $\overline{X} = \frac{E[Y]}{E[A]}$

Verlauf Staukontrollfenster

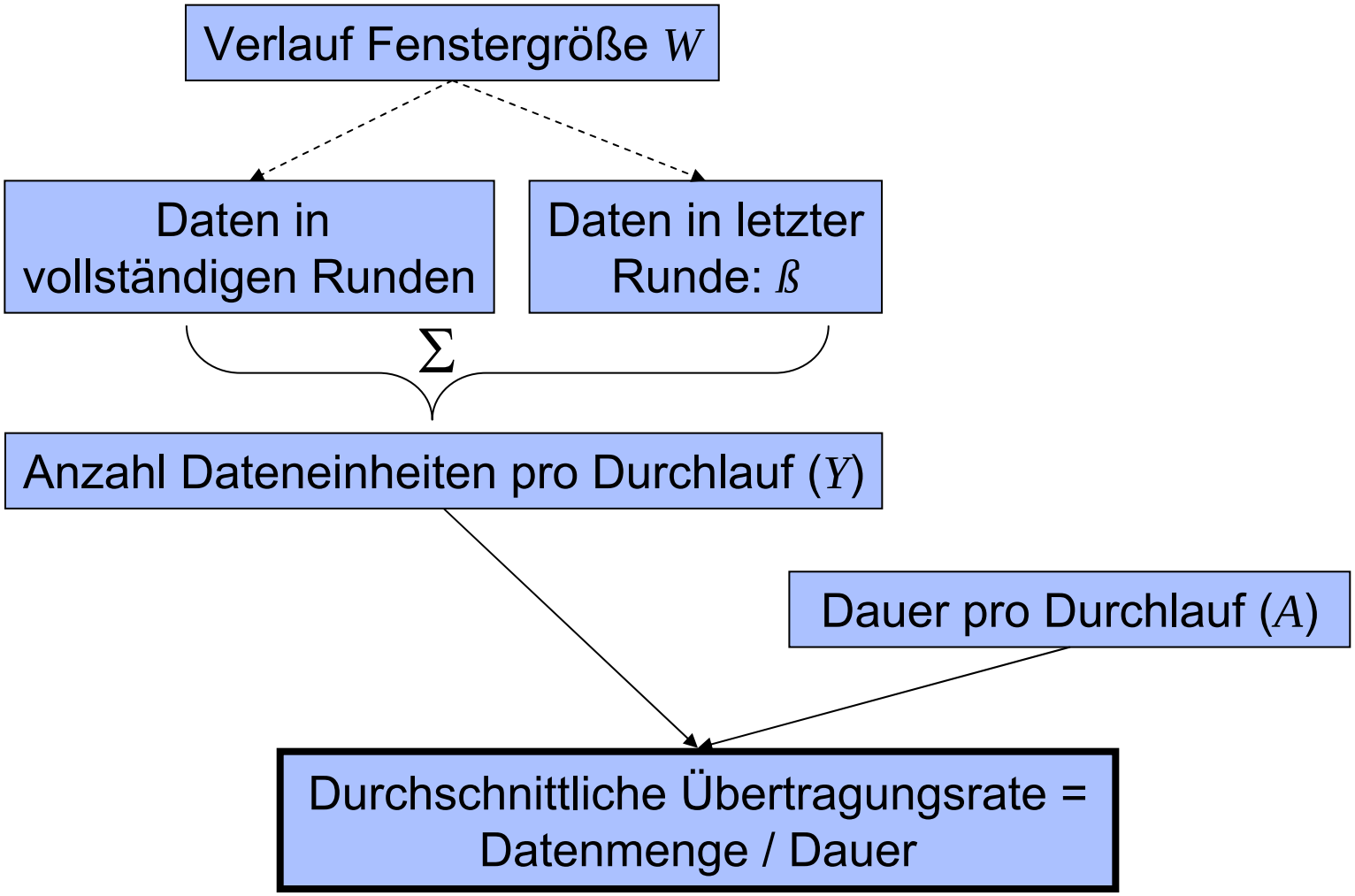


- Sei α_i Nummer der verlorenen Dateneinheit in Runde R_i des i-ten Durchlaufs
  - Nach der Dateneinheit α_i werden $W_i - 1$ weitere Dateneinheiten gesendet bevor Verlust erkannt wird und der i-te Durchlauf endet
  - In i+1 Runden werden somit $Y_i = \alpha_i + W_i - 1$ Dateneinheiten gesendet
  - Damit ergibt sich $E[Y] = E[\alpha] + E[W] - 1$

Anzahl gesendeter Dateneinheiten

- Annahme
  - Dateneinheiten α_i der einzelnen Durchläufe gehen unabhängig voneinander mit Wahrscheinlichkeit p verloren
  - Geht eine Dateneinheit α_i verloren, dann auch alle verbleibenden Dateneinheiten bis zum Ende der Runde
  - Dateneinheiten werden unabhängig von ihrem Schicksal (empfangen oder nicht) in die Berechung der Übertragungsrate einbezogen
- Wahrscheinlichkeit p, dass k – 1 Dateneinheiten gesendet wurden, bevor ein Verlust auftritt:
$$P[\alpha = k] = (1-p)^{k-1} p$$
- Erwartete Position der ersten verlorenen Dateneinheit
$$E[\alpha] = \sum_{k=1}^{\infty} k(1-p)^{k-1} p = \frac{1}{p}$$
- Damit folgt für $E[Y]$
$$E[Y] = E[\alpha] + E[W] - 1 = \frac{1-p}{p} + E[W]$$
- … noch erforderlich: Ausdrücke für $E[W]$ und $E[A]$ um $\overline{X} = \frac{E[Y]}{E[A]}$ zu bestimmen

Vorgehen detailliertes Modell



Anmerkung Verlauf Staukontrollfenster

- Im Modell: Dauer einer Runde entspricht der Umlaufzeit
  - Dauer einer Runde ist unabhängig von der Fenstergröße
  - Implizite Annahme: Alle Dateneinheiten eines Fensters werden in einer Zeitspanne kürzer der Umlaufzeit (RTT) gesendet
- b sei die Anzahl der Dateneinheiten, die mit einer Quittung quittiert werden
  - In der ersten Runde werden W/b Quittungen empfangen
  - Jede Quittung erhöht Fenstergröße um $1/W$
  - Fenstergröße zu Beginn der zweiten Runde: $W' = W + 1/b$
  - Fenstergröße steigt linear mit einer Steigung von $1/b$ per Umlauf
  - … falls b im folgenden nicht explizit erwähnt ist, wird b=1 impliziert

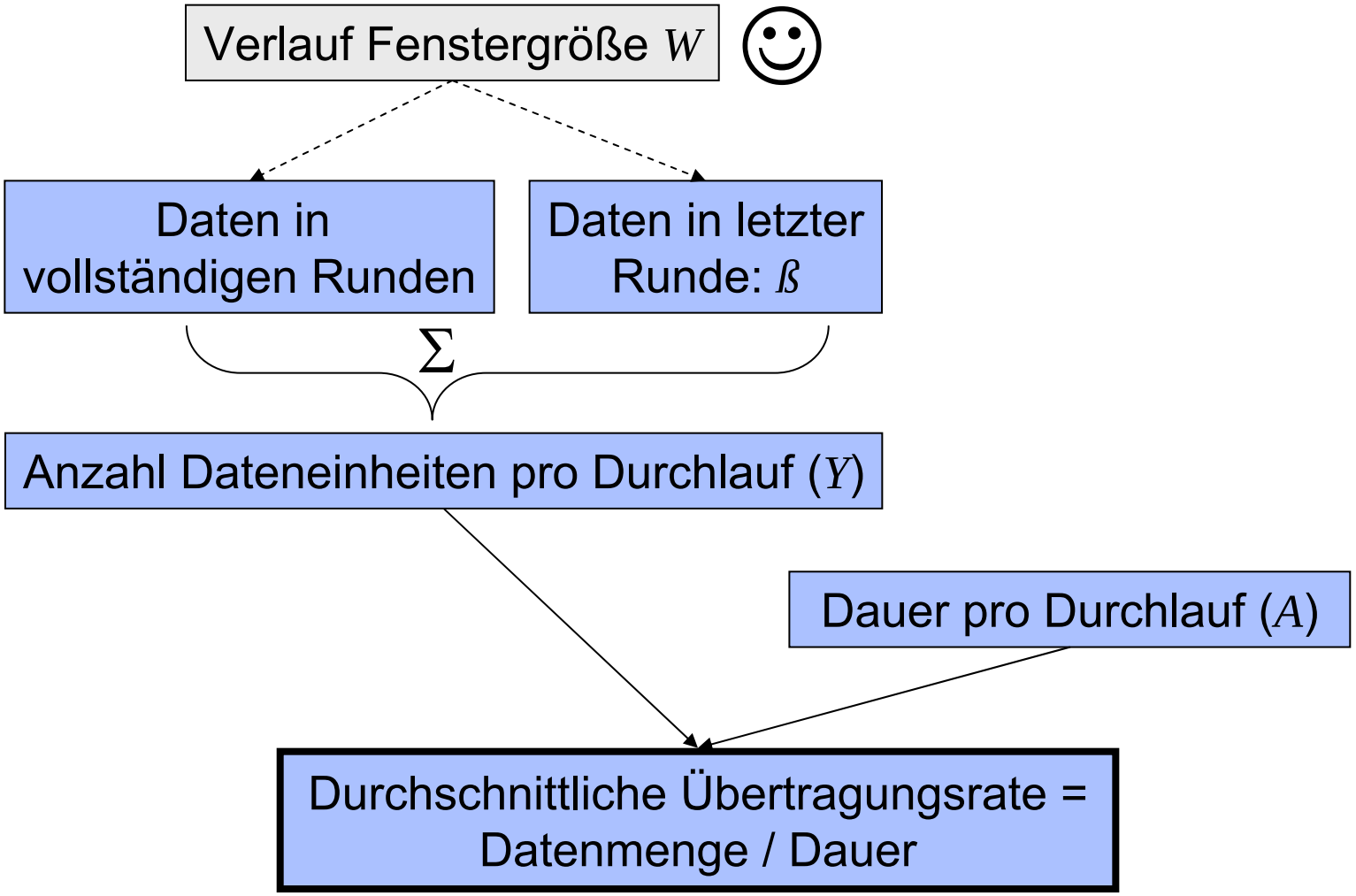
Größe des Fensters

- Gesucht: $E[W]$
- Beobachtung
  - Übertragungsfenster W wird in jeder Runde vergrößert, daher gilt für die letzte Fenstergröße W_i: $W_i = \frac{W_{i-1}}{2} + \frac{R_i}{b}$

b: Anzahl Dateneinheiten, die mit einer Quittung quittiert werden.

- Annahme
  - R_i und W_i seien voneinander unabhängige Folgen von Zufallsvariablen
  - Es folgt $E[W] = \frac{2}{b} E[R]$

Vorgehen detailliertes Modell



Größe des Fensters

- Neuer Ausdruck für die Gesamtzahl gesendeter Dateneinheiten Y_i eines Durchlaufs:

$$Y_i = \sum_{k=0}^{R_i/b-1} \left(\frac{W_{i-1}}{2} + k \right) b + \beta_i$$

$$= \frac{R_i W_{i-1}}{2} + \frac{R_i}{2} \left(\frac{R_i}{b} - 1 \right) + \beta_i$$

$$= \frac{R_i}{2} \left(W_{i-1} + \frac{R_i}{b} - 1 \right) + \beta_i$$

$$= \frac{R_i}{2} \left(\frac{W_{i-1}}{2} + W_i - 1 \right) + \beta_i$$

  - Mit $ß_i$ = Anzahl der Dateneinheiten in der letzten Runde R_{i+1}

Anzahl Dateneinheiten

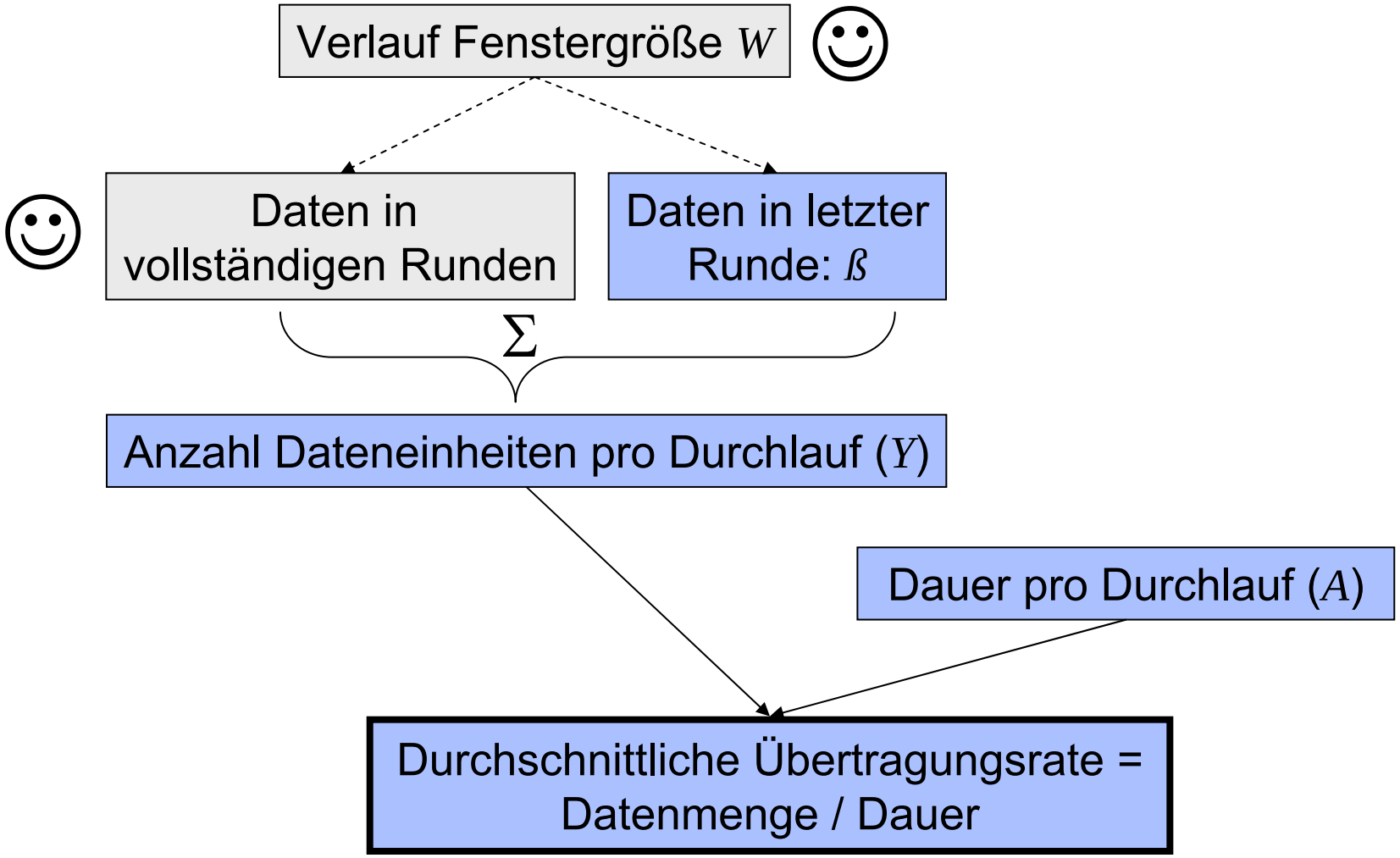
- Annahme
  - Anzahl von Runden je Durchlauf unabhängig von Fenstergröße
- Damit

$$E[Y] = \frac{E[R]}{2}\left(\frac{E[W]}{2} + E[W] - 1\right) + E[\beta]$$

  - Mit $E[W] = \frac{2}{b}E[R]$ und b = 1 gilt

$$E[Y] = \frac{E[W]}{4}\left(\frac{E[W]}{2} + E[W] - 1\right) + E[\beta]$$

Vorgehen detailliertes Modell



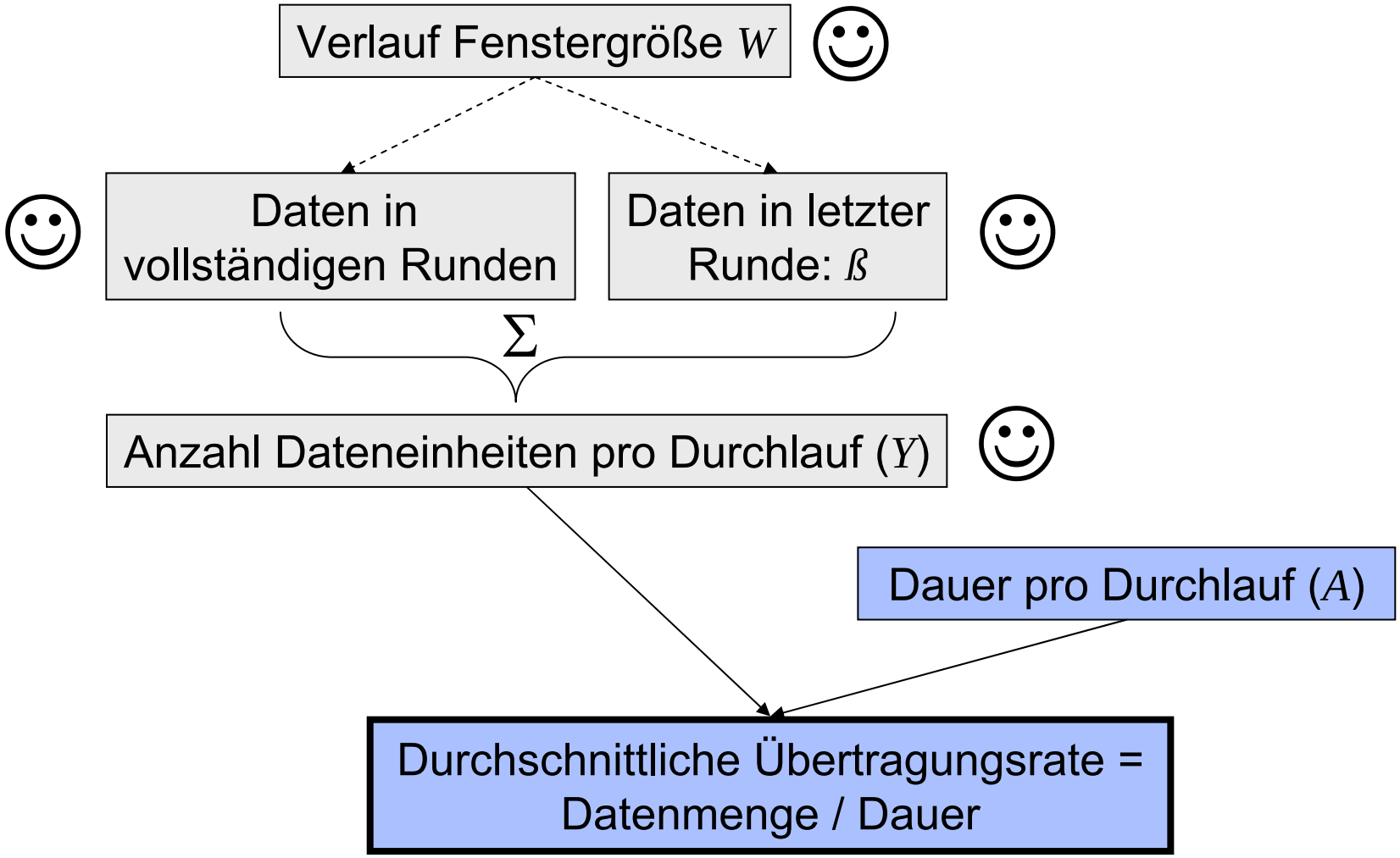
Anzahl Dateneinheiten

- Annahme
  - Anzahl von Runden je Durchlauf unabhängig von Fenstergröße
- Damit

$$E[Y] = \frac{E[W]}{4}\left(\frac{E[W]}{2} + E[W] - 1\right) + E[\beta]$$

- Weiterhin
  - Anzahl der gesendeten Dateneinheiten in der letzten Runde gleichverteilt zwischen 1 und $W_i - 1$
  - Also $E[\beta] = \frac{E[W]}{2}$
  - Damit $E[Y] = \frac{E[W]}{4}\left(\frac{E[W]}{2} + E[W] - 1\right) + \frac{E[W]}{2}$

Vorgehen detailliertes Modell



Anzahl Dateneinheiten

- E[Y] auf zwei Arten ausgedrückt:

$$E[Y] = \frac{E[W]}{4}\left(\frac{E[W]}{2} + E[W] - 1\right) + \frac{E[W]}{2}$$

$$E[Y] = \frac{1-p}{p} + E[W]$$

Anzahl Runden

- Gleichsetzen der beiden Ausdrücke für die erwartete Anzahl gesendeter Dateneinheiten

$$\frac{1-p}{p} + E[W] = \frac{E[W]}{4}\left(\frac{E[W]}{2} + E[W] - 1\right) + \frac{E[W]}{2}$$

- Ergebnis: quadratische Gleichung $3\left(E[W]\right)^2 - 6E[W] + \frac{8(p-1)}{p} = 0$

- Da $E[W]$ >= 0: Positive Lösung $E[W] = 1 + \sqrt{\frac{8}{3p} - \frac{5}{3}}$

- Falls Verlustwahrscheinlichkeit p nahe Null, Abschätzung der erwarteten Fenstergröße: $E[W] \approx \sqrt{\frac{8}{3p}}$

- Erwartete Anzahl an Runden pro Durchlauf $E[R] = \frac{1}{2} + \sqrt{\frac{2}{3p} - \frac{5}{12}}$

Dauer pro Durchlauf

- Noch benötigt: Erwartete Dauer pro Durchlauf $E[A]$
  - Zeitspanne zwischen Start der j-ten Runde (Versand 1. Dateneinheit in Runde j in Durchlauf i) und dafür erhaltener Quittung ist eine Umlaufzeit rtt_{ij}, somit:

$$A_i = \sum_{j=1}^{R_i+1} rtt_{ij}$$

- Annahme: rtt unabhängig von Fenstergröße, dann:

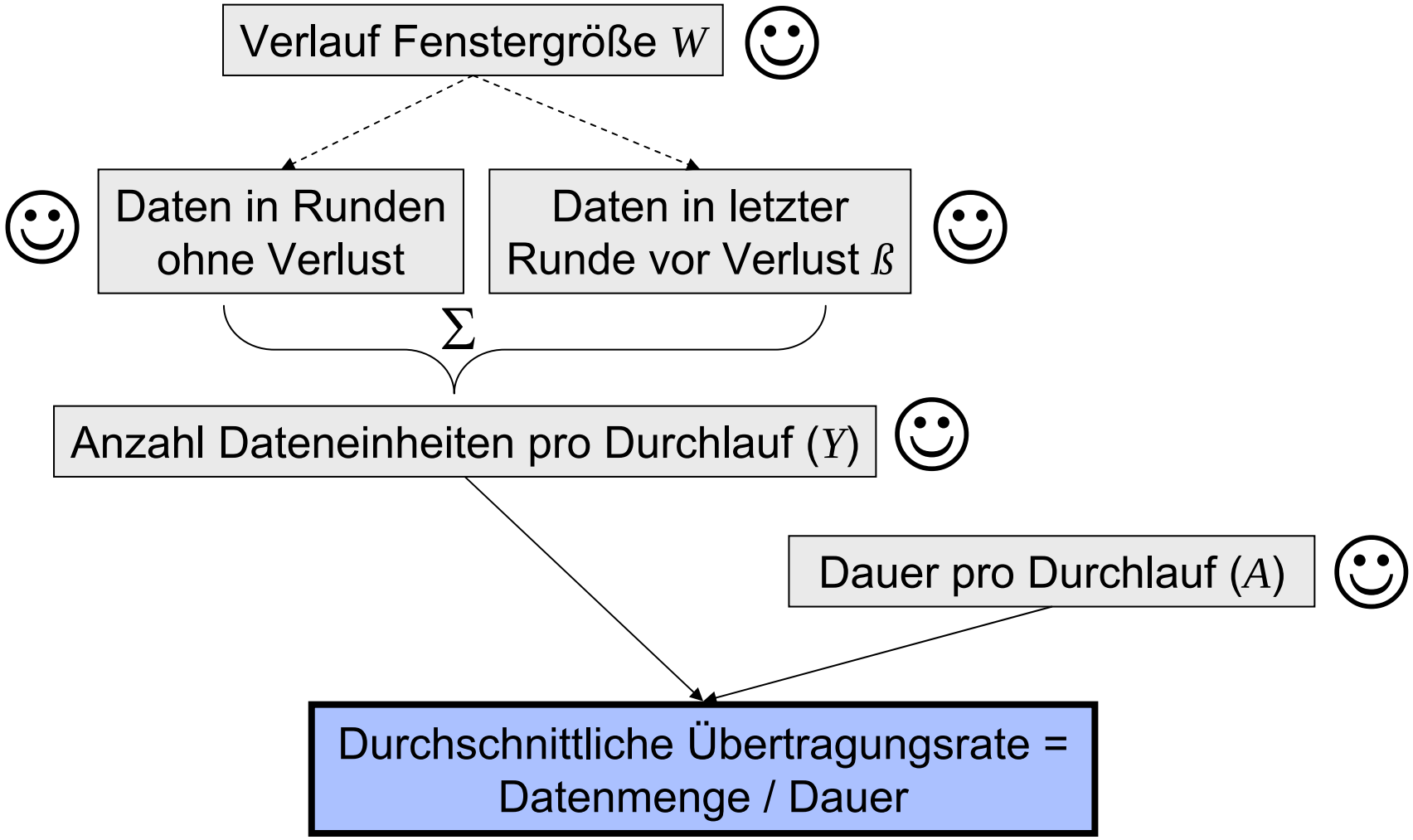
$$E[A] = (E[R]+1)E[rtt]$$

$$= \left(\frac{1}{2} + \sqrt{\frac{2}{3p} - \frac{5}{12}} + 1\right) RTT$$

$$= RTT\left(\frac{3}{2} + \sqrt{\frac{2}{3p} - \frac{5}{12}}\right)$$

  - RTT ist durchschnittliche Umlaufzeit

Vorgehen detailliertes Modell



Durchschnittliche Übertragungsrate

- Damit: durchschnittliche Übertragungsrate im eingeschwungenen Zustand

$$X(p) = \frac{E[Y]}{E[A]}$$

„Einheit“ von $X(p)$: Dateneinheiten/s

$$= \frac{\frac{1-p}{p} + 1 + \sqrt{\frac{8}{3p} - \frac{5}{3}}}{RTT\left(\frac{3}{2} + \sqrt{\frac{2}{3p} - \frac{5}{12}}\right)}$$

$$= \frac{\frac{1}{p} + \sqrt{\frac{8}{3p} - \frac{5}{3}}}{RTT\left(\frac{3}{2} + \sqrt{\frac{2}{3p} - \frac{5}{12}}\right)}$$

- Falls Verlustwahrscheinlichkeit p für Dateneinheiten sehr klein, Abschätzung möglich:

$$X(p) \approx \frac{1}{RTT}\sqrt{\frac{3}{2p}}$$

  - Identisch zu Abschätzung durch einfaches periodisches Modell

Beispiel zur durchschnittlichen Übertragungsrate

- Selbe Ausgangssituation wie beim Beispiel im periodischen Modell
  - RTT = 0,2 s = 200 ms
  - Verlustwahrscheinlichkeit von Dateneinheiten p = 0,05
  - Ergibt eine durchschnittliche Übertragungsrate X wie folgt

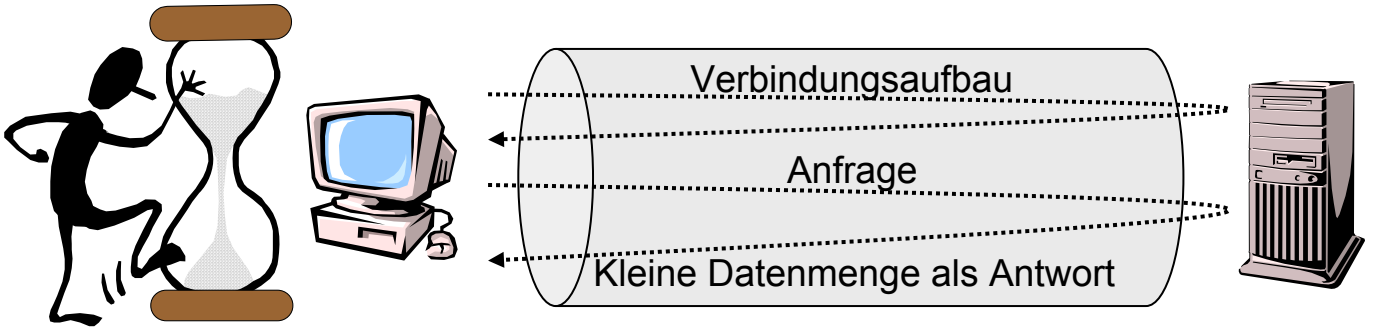
$$X = \frac{\frac{1}{0{,}05} + \sqrt{\frac{8}{3*0{,}05} - \frac{5}{3}}}{0{,}2\left(\frac{3}{2} + \sqrt{\frac{2}{3*0{,}05} - \frac{5}{12}}\right)} \frac{\text{Dateneinheiten}}{\text{s}}$$

$$= 26{,}69 \frac{\text{Dateneinheiten}}{\text{s}}$$

  - Zum Vergleich: Resultat durch einfaches periodisches Modell war 27,4 Dateneinheiten/s

2.3.3 Wartezeit auf die vollständige Übertragung geringer Datenmengen

- Hier
  - Übertragung einer geringen Menge von Daten, etwa einzelne Objekte einer Webseite
  - In 2.3.1 und 2.3.2 wurde jeweils Übertragung großer Datenmengen betrachtet
- Beobachtung
  - Verbindungsaufbauzeit und Slow-Start-Phase können die Wartezeiten erheblich beeinflussen
- Für die Interaktivität ist die Wartezeit von immenser Bedeutung
  - Gefühlte Reaktionsgeschwindigkeit von webbasierten Diensten
  - Benutzer sind äußerst ungeduldig



Wartezeit auf die vollständige Übertragung geringer Datenmengen

- Wartezeit, die hier interessiert
  - Zeit zwischen der Initiierung einer TCP-Verbindung und dem Erhalt der angeforderten Daten
- Annahmen
  - Netz ist unbelastet
  - Direkte Verbindung zwischen Client und Server
- Weitere vereinfachende Annahmen
  - Staukontrollfenster limitiert Datenmenge, die Sender senden kann (d.h. großer Empfangspuffer ist vorhanden)
  - Es treten keine Sendewiederholungen auf
  - Overhead, der durch Kontrollinformation erzeugt wird (z.B. TCP-Kopf) wird vernachlässigt
  - Die zu übertragenden Daten sind ganzzahlige Vielfache der MSS
  - Sendezeit von Quittungen etc. wird vernachlässigt.

Verwendete Variablen

- Für den besseren Überblick
  - X: Übertragungsrate in der Einheit Dateneinheiten pro Zeit [Dateneinheiten/s]
  - RTT: Umlaufzeit (Round Trip Time) [Sekunden]
  - p: Verlustwahrscheinlichkeit einer Dateneinheit
  - MSS: Maximale Größe einer Dateneinheit (Maximum Segment Size) [bit]
  - W: Größe des Staukontrollfensters in der Einheit Anzahl Dateneinheiten [MSS] (angegeben als Vielfaches der MSS)
  - $E[z]$: Erwartungswert des Parameters z
  - O: Größe der zu übertragenden Daten [bit]
  - D: Durchschnittliche Datenrate in der Einheit bit pro Zeit [bit/s]

Wartezeit bei statischem Fenster

- Analyse
  - 1. Fall: $\frac{W * MSS}{D} > RTT + \frac{MSS}{D}$

[KuRo04]

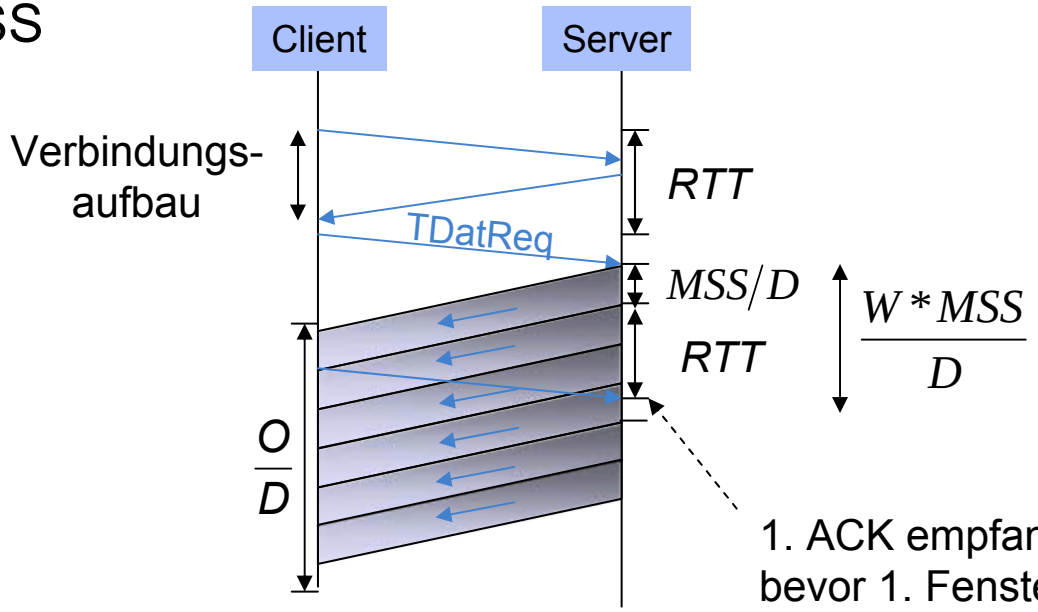
    - Server empfängt Quittung bevor er die Übertragung des ersten Fensters beendet
    - Server kann ohne Pause senden

  - 2. Fall: $\frac{W * MSS}{D} < RTT + \frac{MSS}{D}$

    - Server sendet komplettes Fenster bevor er eine Quittung erhält
    - Dann muss er auf Quittung warten, bevor er weiter senden darf

Wartezeit bei statischem Fenster

- 1. Fall
  - Sei W = 4 MSS



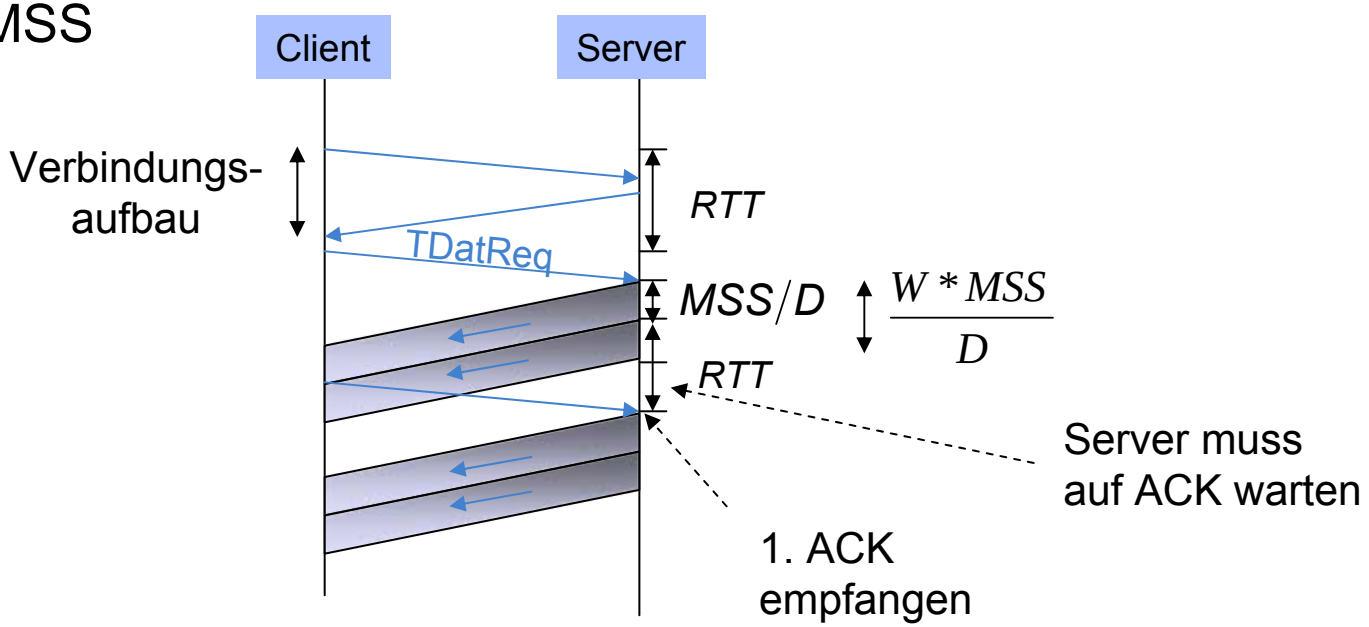
1. ACK empfangen bevor 1. Fenster komplett übertragen wurde

⇨ Server kann ohne Pause weitersenden

$$\text{Wartezeit} = 2 * RTT + \frac{O}{D}$$

Wartezeit bei statischem Fenster

- 2. Fall
  - Sei W = 2 MSS

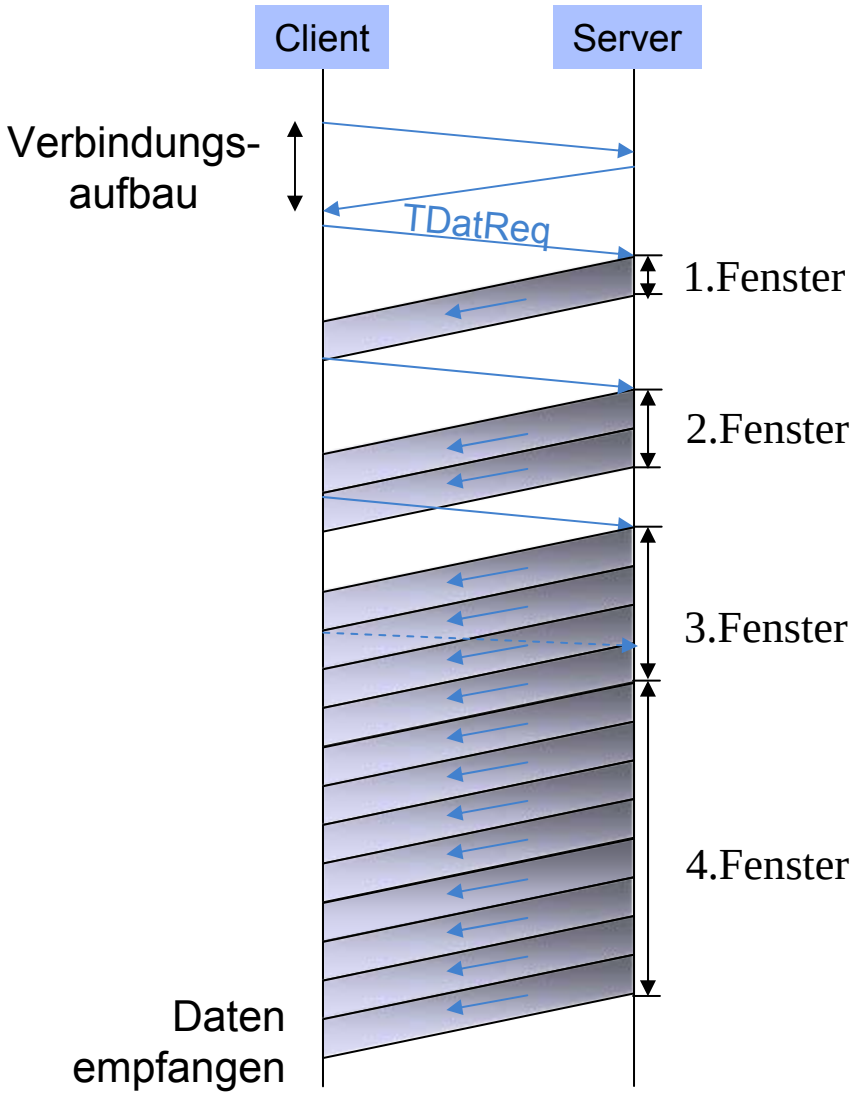


$$\text{Wartezeit} = 2 * RTT + \frac{O}{D} + (K-1)\left[\frac{MSS}{D} + RTT - \frac{W * MSS}{D}\right]$$

mit $K = \left\lceil \frac{O}{W * MSS} \right\rceil$ (= Anzahl Fenster für die Übertragung der Datenmenge O)

Wartezeit bei dynamischem Fenster

- Beobachtung
  - Nachdem der Server ein Fenster an Daten übertragen hat, muss er ggfs. auf eine Quittung warten
  - Die Wartezeit besteht aus drei Komponenten
    - Verbindungsaufbau
    - Sendedauer
    - Wartezeit



Weitere wichtige Modellierungsaspekte

- Komplexere Modelle beziehen weitere Aspekte von TCP mit ein
  - Berücksichtigung von Timeouts
    - Bisher Erkennung von Verlusten nur durch Duplicate ACKs modelliert
      - D.h. kein Rückfall in Slow-Start-Phase
    - Timeouts haben jedoch große Auswirkungen auf Datenrate von TCP
      - Wartezeiten bis Timeout
      - Slow-Start

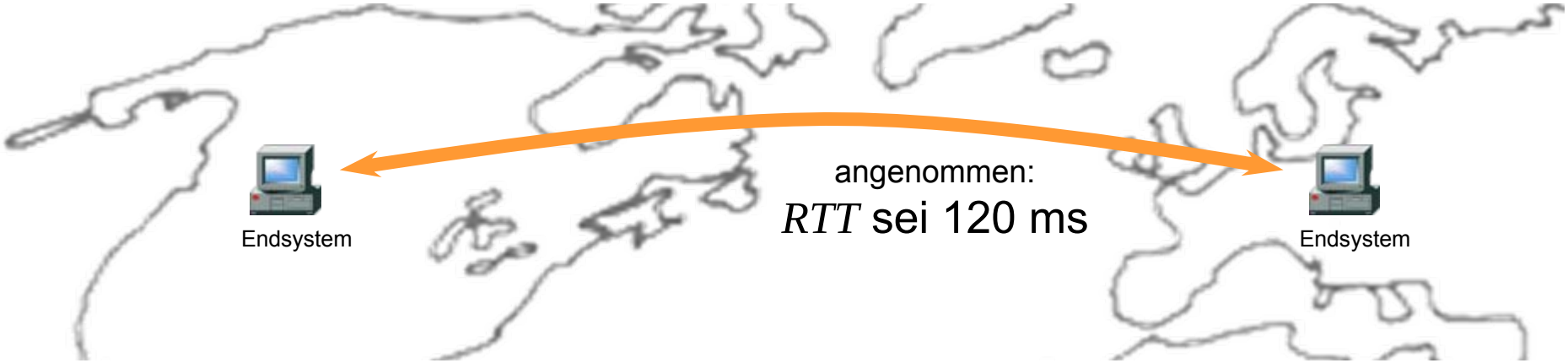
    ⇨ Deutliche Reduktion der Datenrate
  - Berücksichtigung des Flusskontrollfensters
    - Beschränkungen des Empfängers können erzielbare Datenrate reduzieren
      - Bisher stets Annahme, dass unlimitiert schnell empfangen werden kann
  - Bursthaftes Auftreten von Verlusten von Dateneinheiten
  - Verlustwahrscheinlichkeit abhängig von Sendefenstergröße
  - Einfluss paralleler TCP-Verbindungen
  - usw.
- Ziel: Verfeinerung der Aussagen zum Verhalten von TCP und der erreichbaren TCP-Datenrate
  - … weiterführende Literatur hierzu

[HaJa03, Padh00]

Einschub: Praktische Auswirkung der Flusskontrolle

- Nach so viel Staukontrollfenster, jetzt mal etwas zum Flusskontrollfenster
  - Kopffeld „Empfangsfenster“ 16 Bit breit (Standard-TCP)
    - Maximal 65536 Bytes unbestätigt unterwegs
      - Selbst wenn Empfänger mehr empfangen könnte
    - Quittungen treffen erst nach einer Umlaufzeit (RTT) ein
  - Problematisch bei Verbindungen hoher Bandbreite und großer Ausbreitungsverzögerung (*Long Fat Pipes*)
    - Hohes Bandbreiten-Verzögerungsprodukt
    - Beispielsweise Transatlantik- oder Satelliten-Links
    - Trotz aller verfügbarer Übertragungskapazität Engpässe durch Flusskontrolle möglich!

Beispiel: Transatlantik-Verbindung



- Maximal erreichbare Datenrate mit Standard-TCP limitiert:
  - Maximale Datenrate = Fenstergröße / Umlaufzeit [Post81]
- Beispiel bei 120 ms Umlaufzeit:
  - Maximale Datenrate = 65536 Bytes / 120 ms = 546125 Bytes/s
  - Unabhängig von verfügbarer Übertragungskapazität maximal 4,2 Mbit/s möglich

Erweiterungen von TCP für *Long Fat Pipes*

- Einige Abhilfen für die Beschränkungen durch Standard-TCP:

[JBB92, AGS99]

  - Window Scaling
    - Hebt 16 Bit-Beschränkung der Flusskontrollfenstergröße auf
    - Signalisierung größerer Empfangsfenster per TCP-Option
  - Protection Against Wrapped Sequence Space (PAWS)
    - Vermeidet Probleme mit der Duplikaterkennung bei Überlauf des Sequenznummernfeldes
  - Round-Trip Time Measurements (RTTM)
    - Zeitstempel als TCP-Option verbessern Messung der Umlaufzeit
- Mehr in der Vorlesung Hochleistungskommunikation

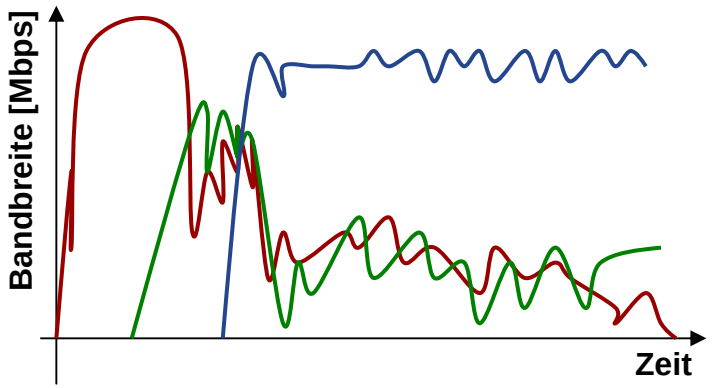
Simulative Evaluierung von TCP

- Problem: Bewertung von komplexeren Szenarien
  - Mehrere hundert bis tausend Netzwerkknoten
  - Einfluss anderer Protokolle auf TCP
- Analytische Betrachtung sehr komplex
- Experimenteller Ansatz teuer bzw. schwer handhabbar
- Alternative: Simulation des Netzwerks
  - Voraussetzungen
    - Sorgfältige Modellierung des Systems
    - Simulationsdurchführung
    - Auswertung der Ergebnisse

- Vorlesung: „**Modellierung und Simulation von Netzen und verteilten Systemen**“
  - Ehemals: Simulation von Rechnernetzen
  - Findet immer im Sommersemester statt
  - Prof. Dr. H. Hartenstein
  - http://dsn.tm.uni-karlsruhe.de

Praktikum „Modellierung und Simulation von Netzen und verteilten Systemen“

- Kennenlernen und Ausprobieren der wichtigsten Simulationstools (für Netzwerksimulation):
  - Ns-3
  - Omnet++
- „Hands-on experience“ in Simulationsplanung, -durchführung und Ergebnisanalyse
  - Kenntnisse sind grundlegend für die Forschung im Telematikbereich
- Das Praktikum findet vorlesungsbegleitend statt (→ rechtzeitig anmelden!)
- Beispiel
  - TCP und Fairness: Simulatives Vorgehen
    - 2 TCP Verbindungen (rot, grün)
    - 1 UDP mit CBR Verkehr (blau)

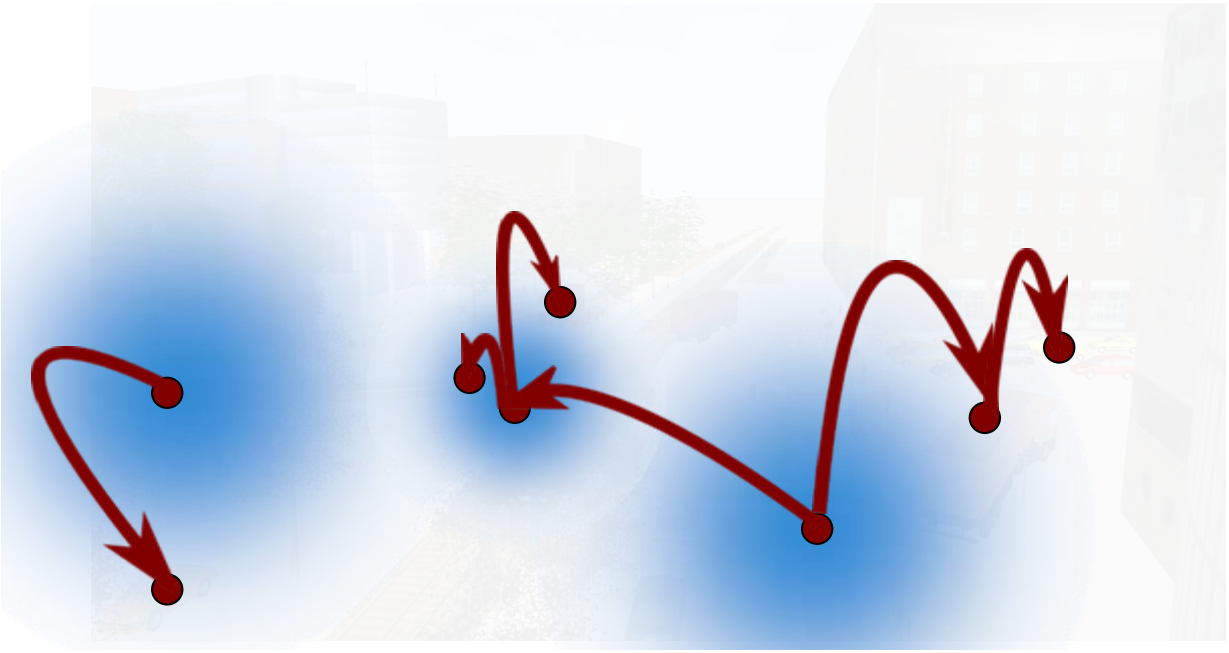


(Straßen-) Verkehr

- Mikroskopische Simulation
- Verkehrssimulator VISSIM der PTV AG

Kommunikation

- Ereignis-diskrete Netzwerksimulation
- Parallele Simulation auf Höchstleistungsrechner im Steinbuch Centre for Computing (SCC)



Leistungsbewertung durch gekoppelte und verteilte Simulationen

- Bewertung des Kommunikationssystems im Straßenverkehr
- Bewertung von Fahrzeugnetzen hinsichtlich **Verkehrssicherheit** und **Verkehrseffizienz**

Agentenbasierte Simulationen

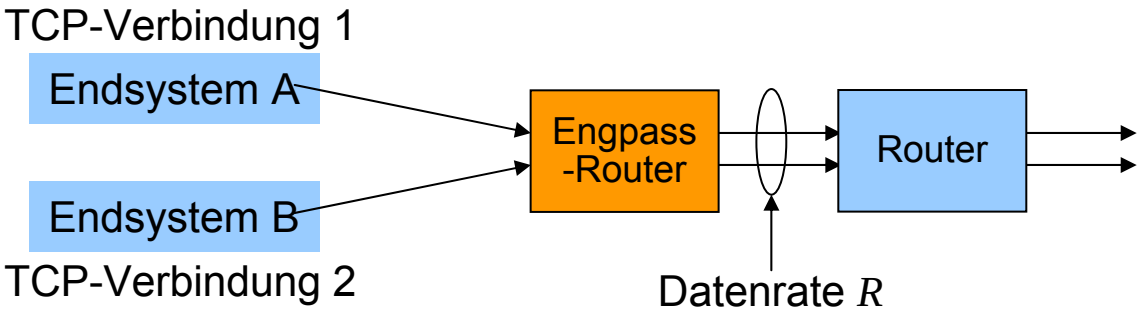
- Soziale Netzwerke
  - Informationsausbreitung
  - Privacy Studien
- Verhalten von Stromnetzen
- Ausbreitung von Krankheiten
- Logistik
- Konsumverhalten

[Argonne National Laboratory, Power Supply System Illinois]

delicious facebook flickr twitter

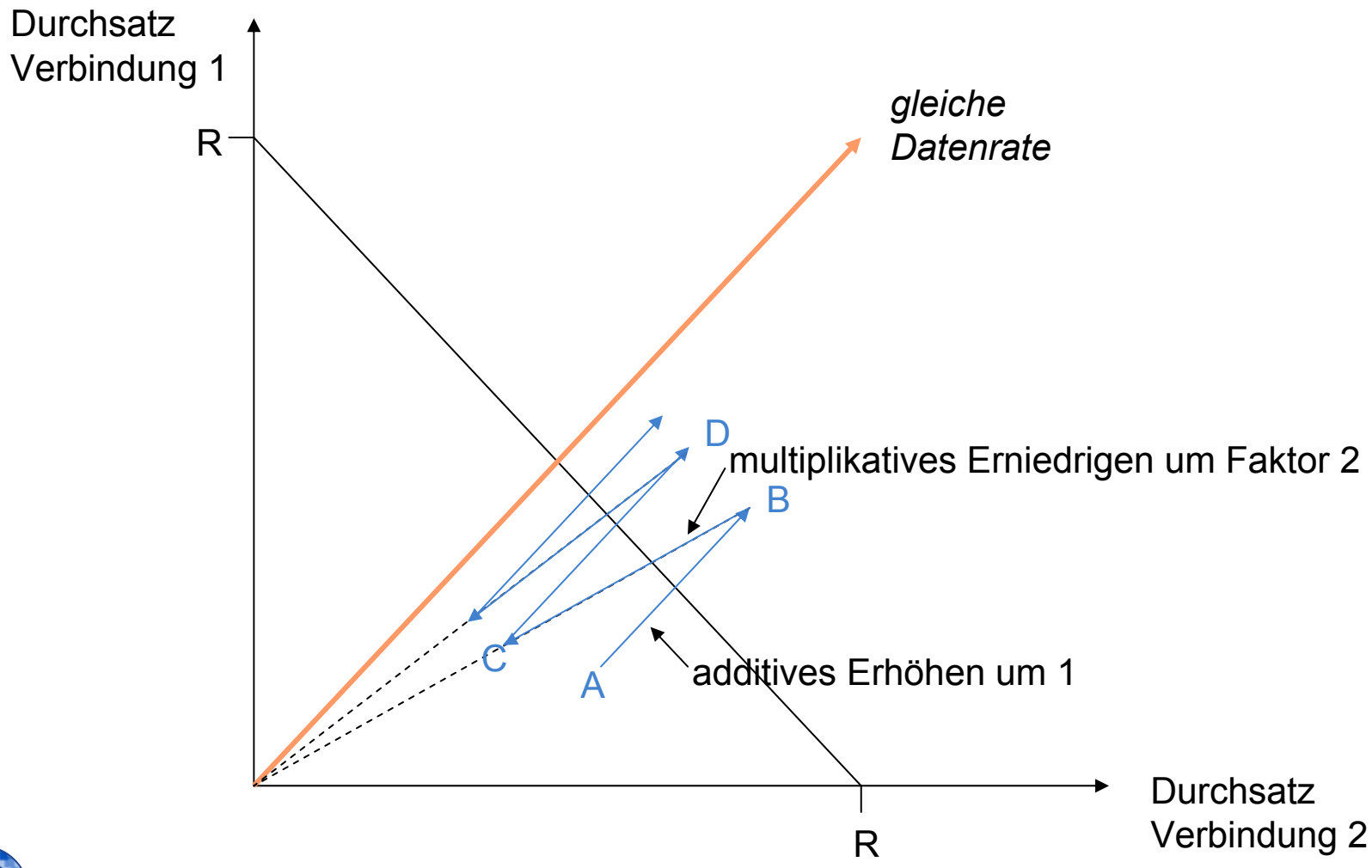
2.4 TCP und Fairness

- Ziel
  - Falls sich N TCP-Verbindungen einen Engpass teilen, dann sollte jede dieser Verbindungen den gleichen Anteil der Kapazität erhalten ($1/N$)
- Beispielszenario



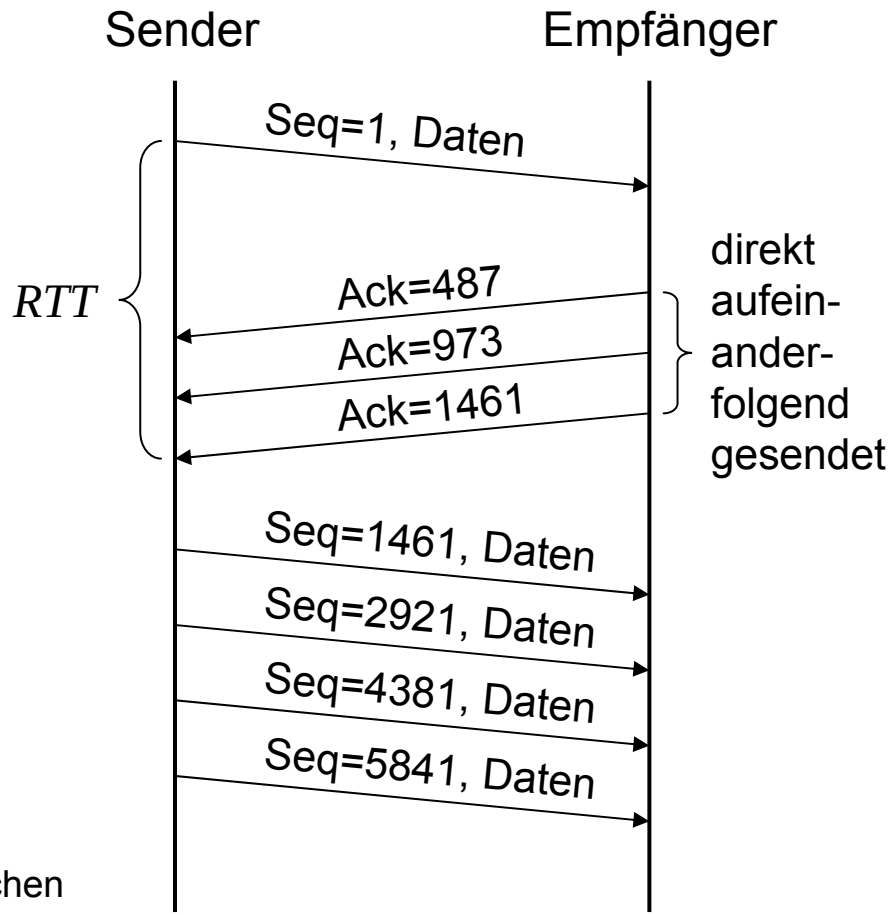
  - Zwei Verbindungen teilen sich eine Leitung mit der Datenrate R
    - Beide Verbindungen konkurrieren somit um Übertragungskapazität
  - Annahmen
    - Beide Verbindungen haben die gleiche MSS und die gleiche Umlaufzeit
  - Das folgende Diagramm zeigt die erzielten Datenraten der beiden Verbindungen

TCP und Fairness



Unfairer TCP-Empfänger

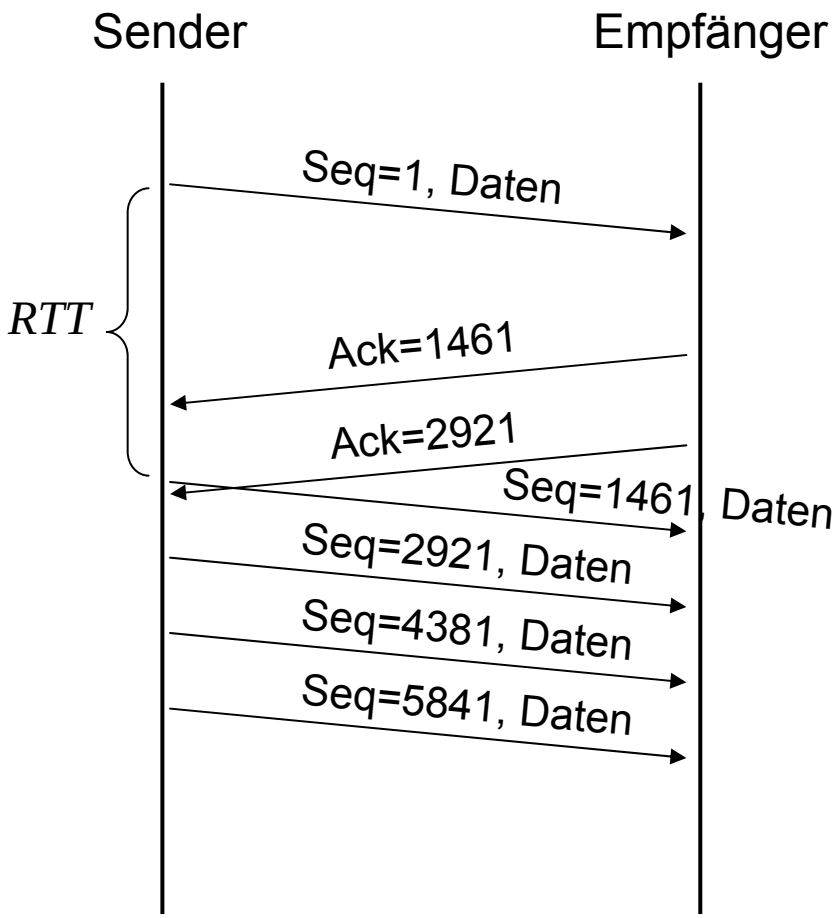
- TCP ist anfällig gegen unfairen Verbrauch von Übertragungskapazität
  - Einzelner Empfänger kann sich Vorteil verschaffen
    - Z.B. kann ein entsprechender Web-Browser schneller WWW-Seiten laden, ohne dass eine Modifikation des WWW-Servers nötig wäre
- Der Ansatzpunkt für diese „Unfairness“ sind Inkonsistenzen in der TCP-Spezifikation
  - TCP-Sequenznummern zählen Bytes
  - Staukontrollfenster zählt in Dateneinheiten
  - Angriff wäre nicht möglich, wenn Staukontrollfenster ebenfalls in Bytes zählen würde
- Angriff
  - Bestätige jede Dateneinheit in einzelnen Häppchen
  - Dadurch öffnet sich das Staukontrollfenster schneller
    - weil schneller mehr erfolgreiche Quittungen beim Sender eintreffen



[Sava99]

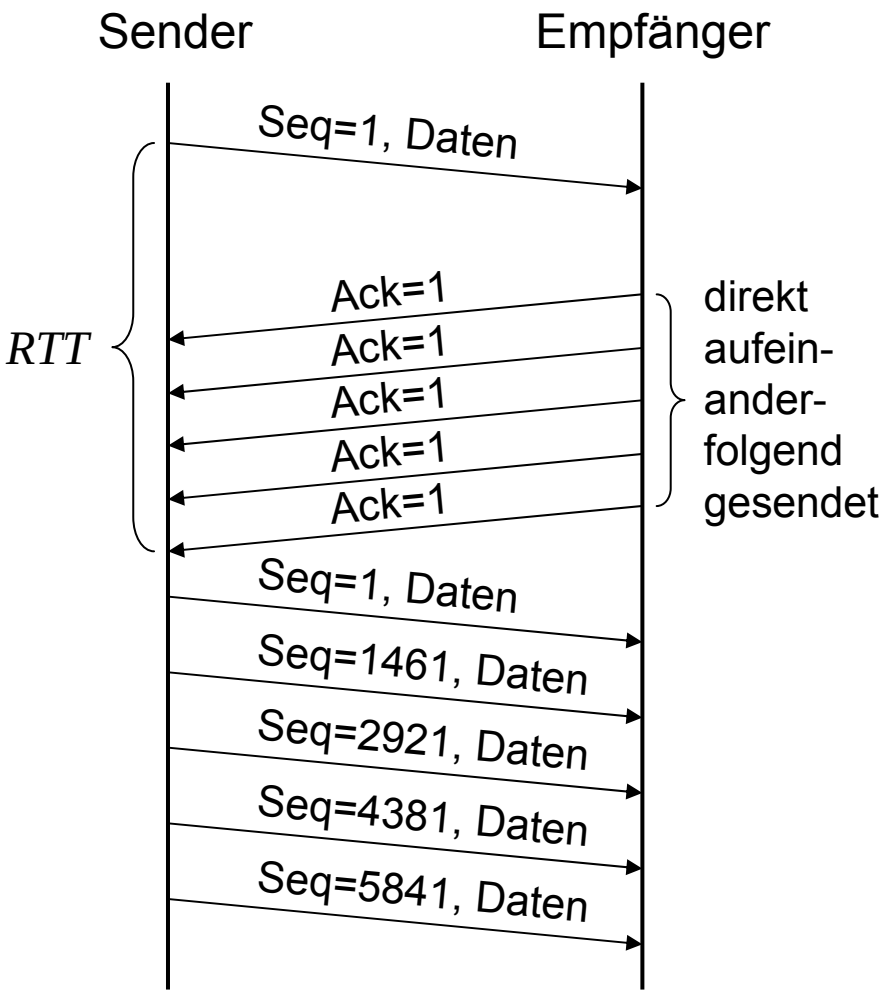
Unfairer TCP-Empfänger

- Verluste von Dateneinheiten sind selten
  - Empfänger kann Empfang antizipieren
- Quittungen sind kein Beweis für Empfang
  - Empfänger kann Quittungen fälschen
  - Würden Quittungen tatsächlich den korrekten Empfang beweisen, wäre der Angriff unmöglich
- Angriff
  - Bestätige Dateneinheiten schneller als sie eintreffen
- Abhilfe
  - Protokolländerung erforderlich
  - Alternativ: Erhöhung der Verlustwahrscheinlichkeit von Dateneinheiten macht Angriff unrentabel
    - z.B. durch gelegentliches Auslassen einer Dateneinheit beim Sender



Unfairer TCP-Empfänger

- Wiederholung von identischen Quittungen (Duplicate ACKs) hat doppelte Bedeutung
  - Anzeige fehlender Dateneinheiten
    - Fehlende Dateneinheit soll wiederholt werden
  - Positive Quittung
    - Nachfolgende Dateneinheiten wurden empfangen
      - Welche kann jedoch nicht übermittelt werden.
- Gemäß TCP Reno läuft das Self-Clocking weiter und auch das Staukontrollfenster wird weiter geöffnet
  - D.h. es werden immer weiter neue Dateneinheiten übertragen bis das Flusskontrollfenster erschöpft ist
  - Abhilfe: Retransmission Timeout
    - Staukontrollfenster zurücksetzen



2.5 Implementierung

- Getreu der Schichtenarchitektur lassen sich Implementierungsfragen in zwei Kategorien einteilen
  - Nutzung der TCP-Protokollimplementierung
    - Nutzung der Dienstschnittstelle, die das Betriebssystem für TCP bereitstellt
    - Sockets
  - Programmierung innerhalb der TCP-Protokollimplementierung
    - Fragestellungen und Probleme bezüglich der möglichst effizienten Programmierung von TCP
    - Implementierungsprobleme von TCP an sich

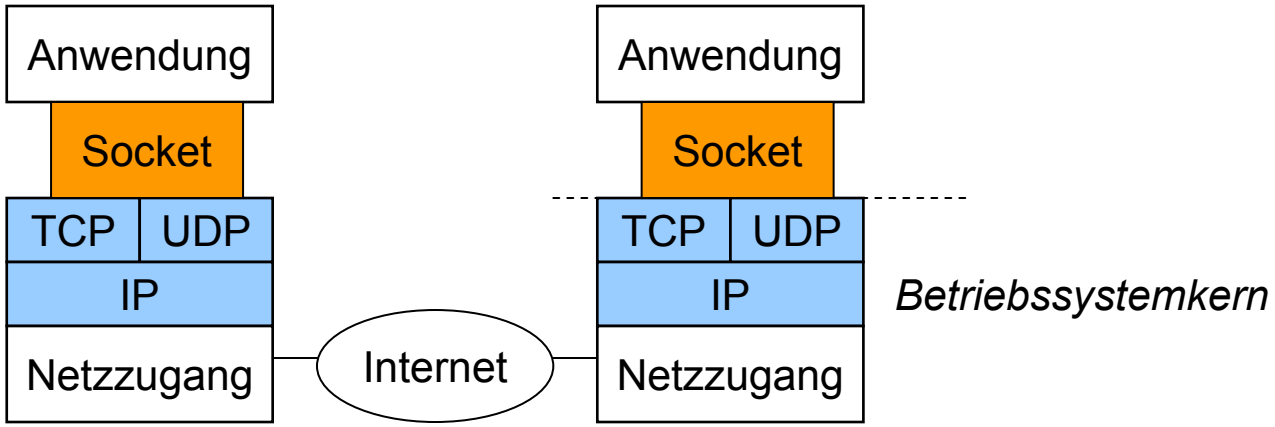

[Paxo99]

- Im Folgenden wird zunächst TCP aus Sicht eines Programmierers behandelt, der die TCP-Protokollimplementierung des Betriebssystems für eine Anwendung nutzen möchte
- Danach schließen sich Bemerkungen zur protokollinternen Programmierung an, also der TCP-Protokollimplementierung selbst

2.5.1 Sockets: Programmierschnittstelle

[Stev03]

- Sockets
  - Schnittstelle zwischen Anwendung und Betriebssystem
  - Eingeführt in Berkeley UNIX, 1981
  - Entwickelt für die Programmiersprache C
  - In der Praxis weit verbreitet
  - Flexible Schnittstelle für die Nutzung verschiedener Kommunikationsprotokolle
    - Interprozesskommunikation, TCP, UDP, IP, X.25 ...
  - Socket-Objekt
    - Kommunikationsendpunkt für bidirektionalen Datenaustausch

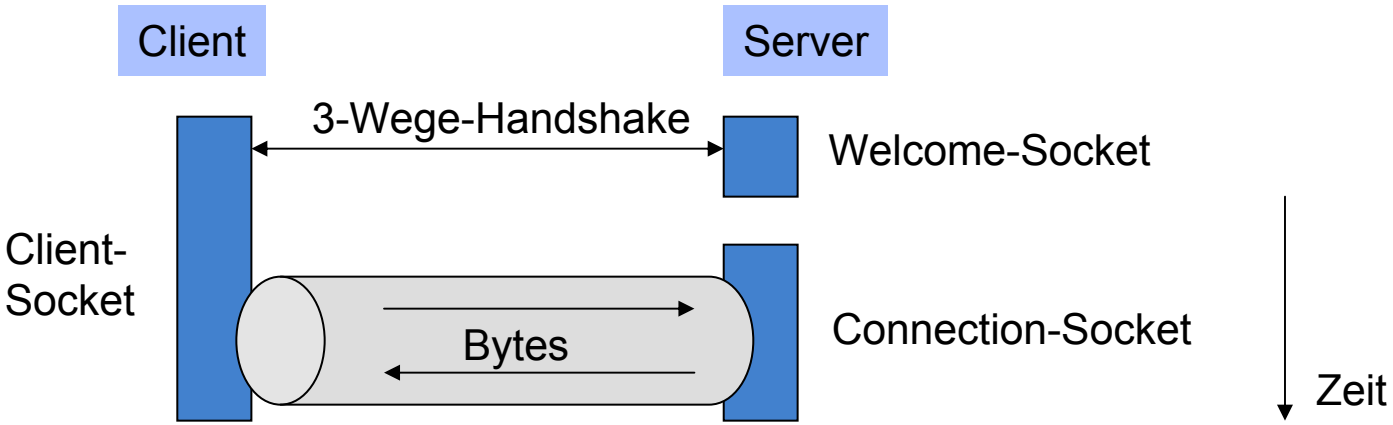


Sockets

- Eigenschaften
  - Sockets werden von Anwendungen erzeugt, genutzt und wieder freigegeben
  - Adressierung auf Anwendungsebene über Ports
  - Sockets ermöglichen das Lesen und Schreiben von Daten
    - Interne Adressierung erfolgt über Deskriptoren
      - Ablauf analog zu einfachem Dateizugriff (Filedeskriptor)
      - Transfer der Daten lokal oder über Netzwerk
  - Sockets können empfangene oder zu sendende Daten kurzzeitig speichern
    - Sockets belegen Ressourcen
  - Sockets folgen dem Client/Server-Paradigma

Socketprogrammierung mit TCP

- Server benutzt zwei Sockets
  - Welcome-Socket, über den er angesprochen wird
    - Muss ständig aktiv sein
  - Connection-Socket, der dann für den eigentlichen Datentransfer etabliert wird
- Clientprozess kann dann eine Verbindung zum Serverprozess initiieren
  - Kreiert ein Socket-Objekt, den Client-Socket
  - Parameter: Ziel-IP-Adresse, Ziel-Portnummer
  - 3-Wege-Handshake zum Aufbau der TCP-Verbindung transparent für Clientprozess



Portnummern

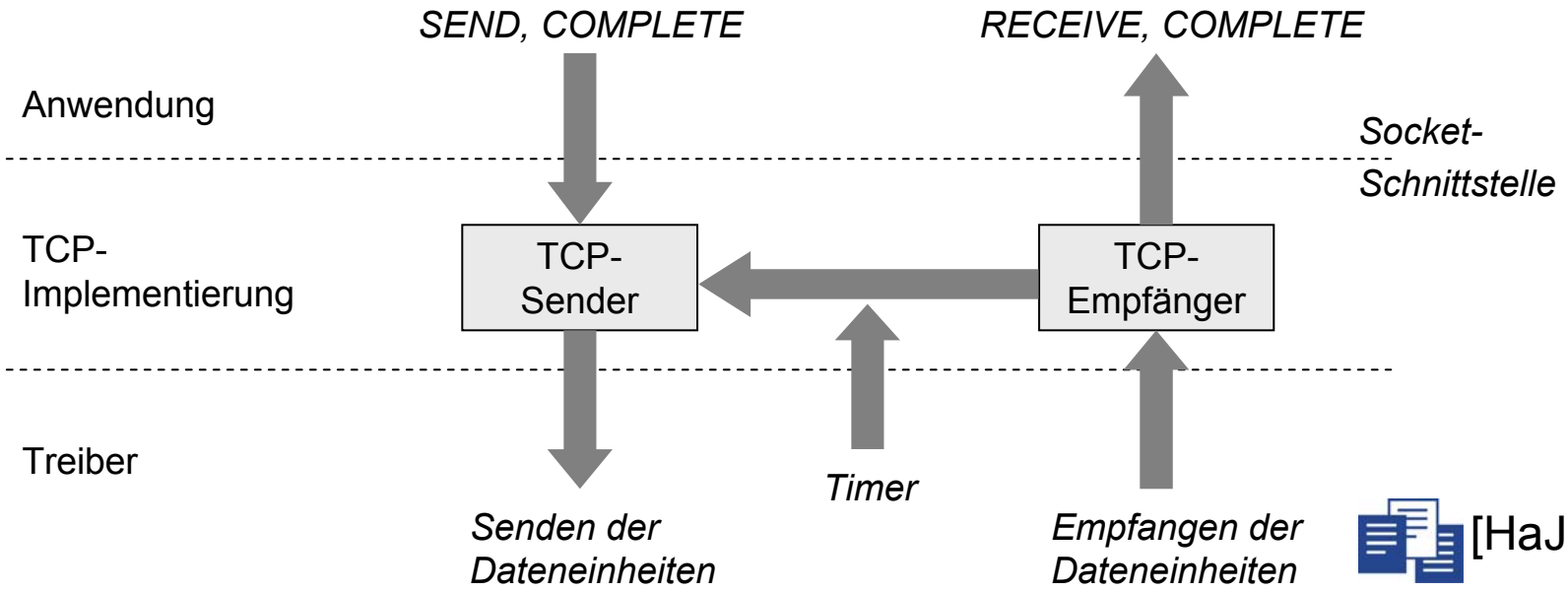
- Eine Verbindung wird eindeutig identifiziert durch
  - Quelle
    - IP-Adresse und Portnummer
  - Ziel
    - IP-Adresse und Portnummer
  - Transportprotokoll
- Vergabe der Portnummern [IANA09]
  - Well known port numbers (0 – 1023)
    - Vergabe durch IANA (Internet Assigned Numbers Authority)
    - Nutzung nur durch System- oder privilegierte Prozessen
  - Registered port numbers (1024 – 49151)
    - Registrierung durch IANA
    - Nutzung durch Nutzer- oder gewöhnliche Prozesse
  - Dynamic and/or private ports (49152 – 65535)
    - Keine Registrierung vorgesehen

2.5.2 Protokollimplementierung

- Vorangegangene Folien
  - Inanspruchnahme von TCP-Diensten durch eine Anwendung
    - Nutzung der Socket-Schnittstelle
- Im Folgenden
  - Wissenswertes zur Implementierung **innerhalb** des TCP-Protokollimplementierung
  - Durch welche Architektur und Mechanismen stellt das Betriebssystem den Anwendungen die Socket-Schnittstelle zur Verfügung?
  - Welche Besonderheiten sind durch die Implementierung von TCP/IP innerhalb des Betriebssystems zu beachten?
    - Möglichst hohe Effizienz (und damit Performance) eines der Haupt-Designziele

Protokollimplementierung

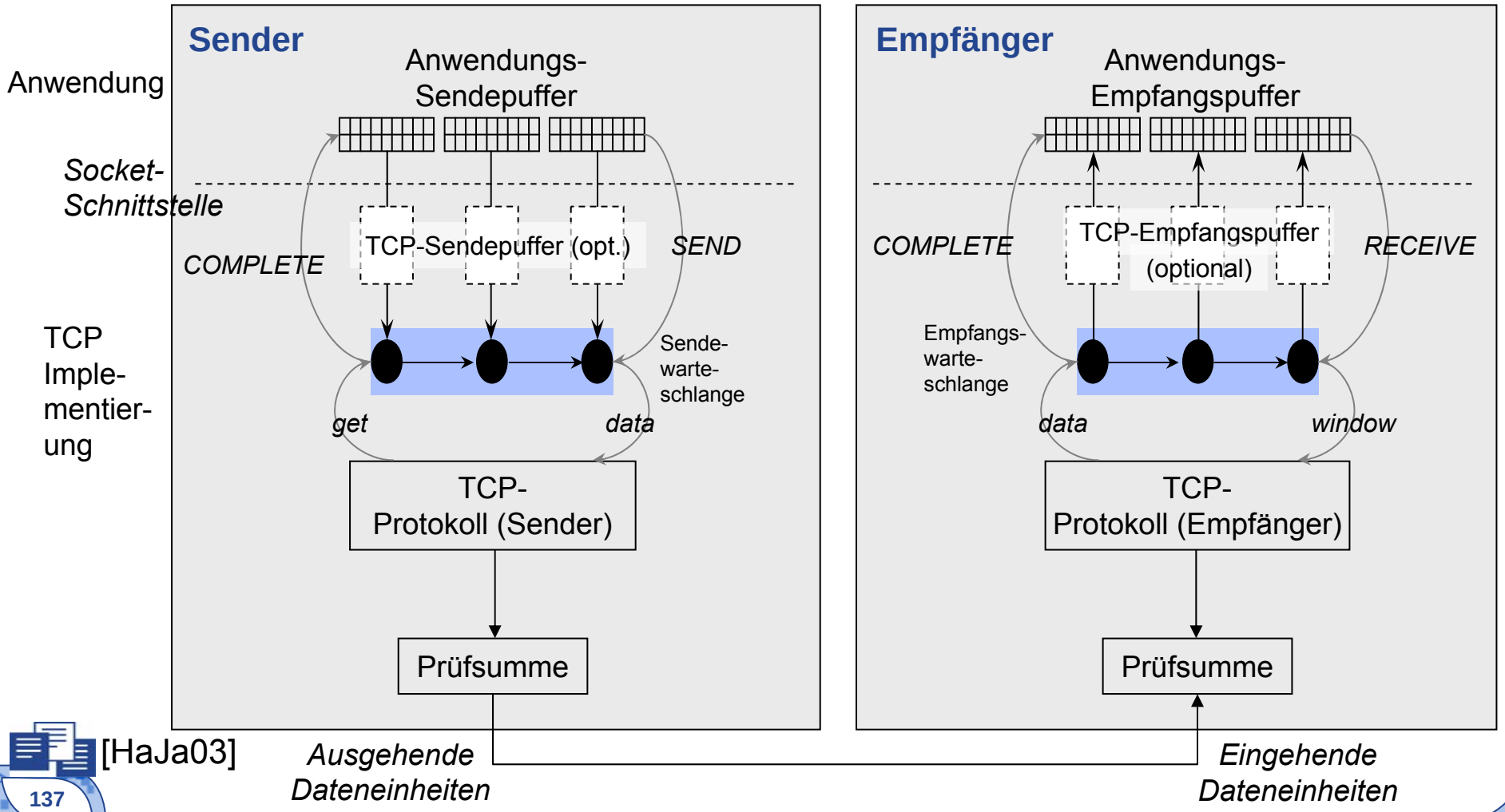
- RFC 793 definiert Architektur für TCP-Protokollimplementierung [Post81]
  - Aufteilung in Sender und Empfänger
    - Weil TCP bidirektional arbeitet: Stets beide vorhanden und aktiv
  - Grundlegendes Basis-Interface für Anwendungen zur Nutzung der TCP-Protokollimplementierung
    - Anwendung entweder User-Programm oder darüber liegendes Protokoll (z.B. HTTP)
    - *SEND*-, *RECEIVE*- und *COMPLETE*-Dienstaufrufe an Schnittstelle



[HaJa03]

Protokollimplementierung

- Datenstrukturen und Abläufe innerhalb einer TCP-Implementierung



[HaJa03]

Protokollimplementierung

- Anwendung überträgt Daten durch *SEND*-Aufrufe in TCP-Sendewarteschlange
  - TCP-Sendewarteschlange enthält
    - zu verschickende Daten, die noch nicht abgesandt wurden
    - verschickte Daten, für die noch keine Quittung (ACK) empfangen wurde
- Anwendung empfängt Daten durch *RECEIVE*-Aufrufe aus TCP-Empfangswarteschlange
  - TCP-Empfangswarteschlange enthält eingegangene Daten, die von der Anwendung noch nicht abgerufen wurden
- TCP-Sender/Empfänger…
  - …arbeiten als Zustandsautomat
  - … arbeiten üblicherweise asynchron, sobald nötig (z.B. beim Eingang einer Dateneinheit)
  - TCP-Sender entnimmt (*get*) zu verschickende Daten (*data*) aus der TCP-Sendewarteschlange
  - TCP-Empfänger reiht eingegangene Daten (*data*) in die TCP-Empfangswarteschlange ein
    - Erst nach Abholung durch die Anwendung (*RECEIVE*) kann das Empfangsfenster wieder weiter geöffnet werden (*window*)

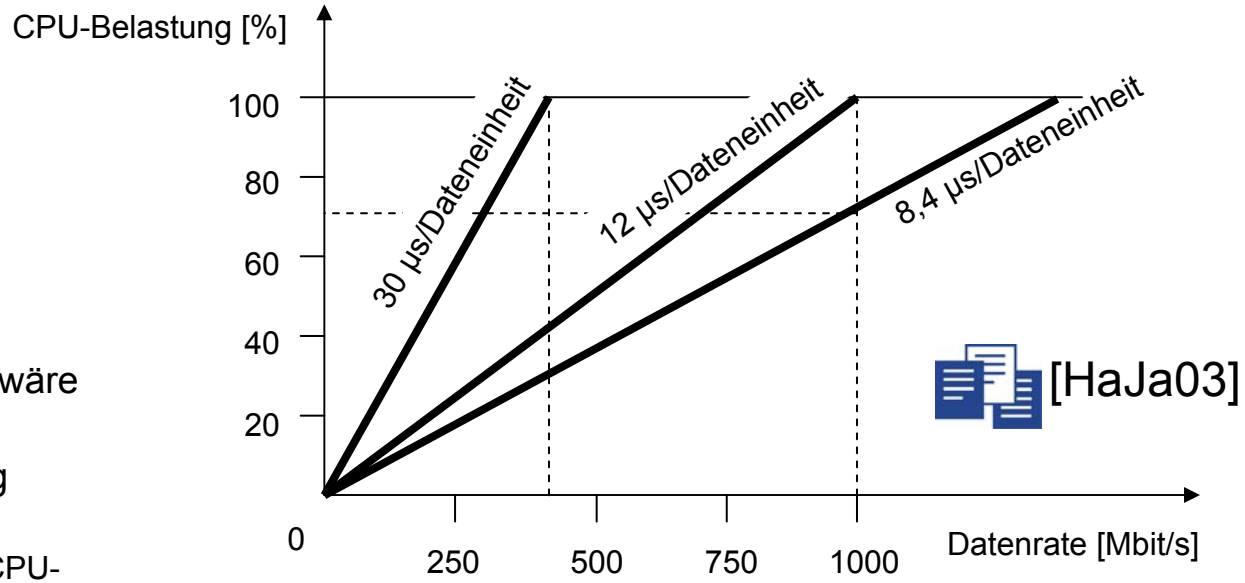
Puffer und Speicherkopien

- RFC 793 fordert Datenintegrität nach Aufruf der Standard-*SEND/RECEIVE*-Operationen
  - Problem
    - Anwendung könnte eigenen Puffer auch nach Aufruf von *SEND* noch verändern
    - TCP soll jedoch die Daten, die zum Zeitpunkt des *SEND*-Aufrufes gültig waren, verschicken
  - Übliche Vorgehensweise:
    Daten werden bei Aufruf von *SEND* aus dem Speicherkontext der Anwendung in den Speicherkontext der TCP-Implementierung kopiert
    - (und umgekehrt bei *RECEIVE*-Aufrufen)
  - Nachteile
    - Aufwand durch Kopieraktionen
    - Erhöhte Systembelastung (Speicherverbrauch, Prozessorzeit)
    - Kann Limitierung darstellen

Overhead

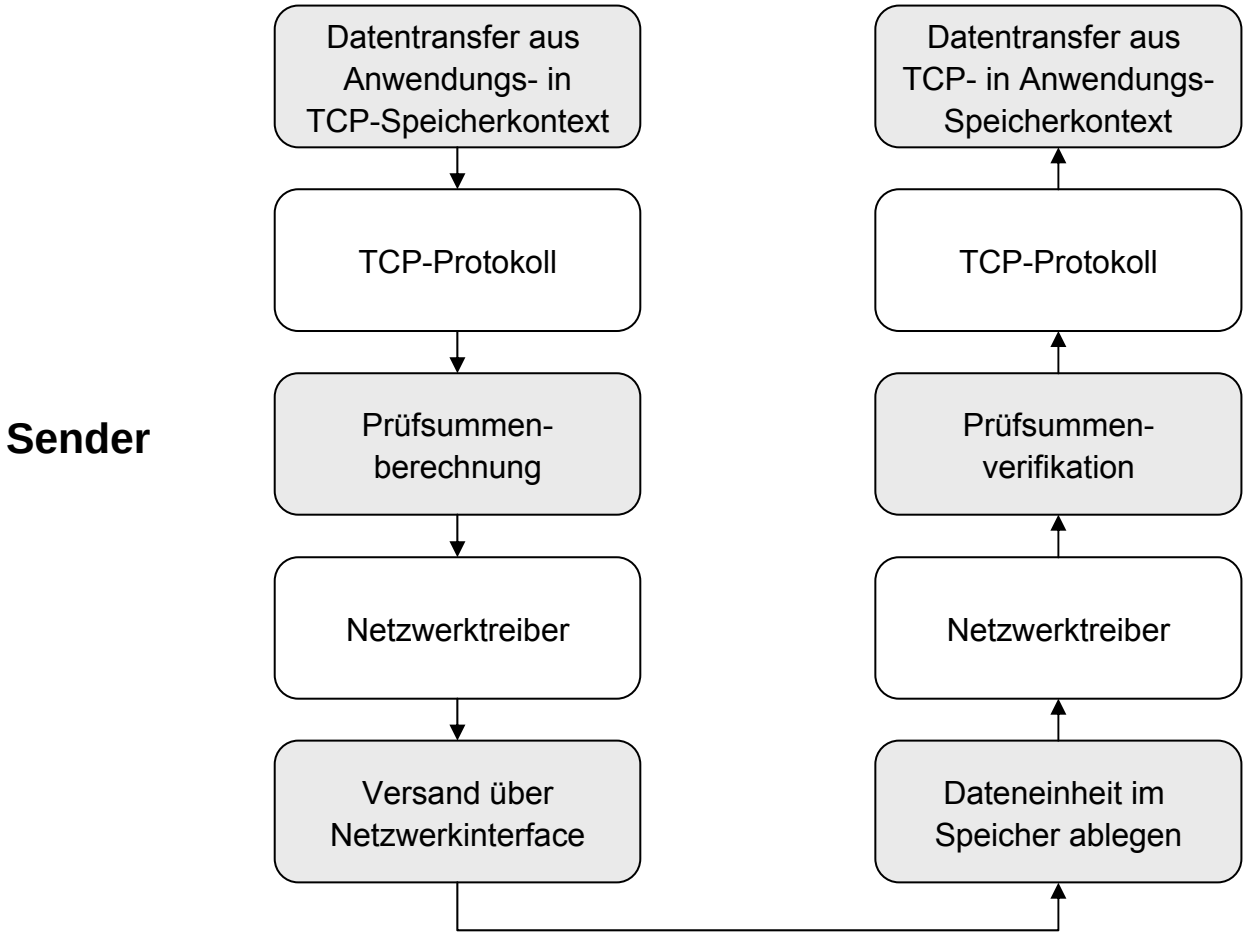
- Nutzung der TCP-Implementierung benötigt Ressourcen
  - Zeit für die Abarbeitung addiert sich zu Übertragungsverzögerungen
  - Wirtssystem wird belastet
    - Worst-Case: Wirtssystem (Prozessorleistung, Speicherbandbreite) kann begrenzender Faktor der erzielbaren Datenrate werden (Host-Limitierung)
    - Einfaches Beispiel für linearen Zusammenhang zwischen Datenrate und CPU-Belastung unter verschiedenen Bearbeitungszeiten pro Dateneinheit:

- Beispiel:
  - Annahme: Genutzte TCP-Implementierung benötigt 20 CPU-Zyklen pro Byte Verarbeitungszeit
  - Mit 1 GHz-CPU: 30 µs/Dateneinheit
  - Bei 100 Mbit/s-Anbindung wäre System netzwerklimitiert
  - Bei 1000 Mbit/s-Anbindung wäre System Host-limitiert
    - Erreicht aufgrund der CPU-Überlastung nur 400 Mbit/s



Overhead

- Ursachen von Endsystem-Overhead bei TCP



[HaJa03]

Overhead

- Overhead-Ursachen lassen sich klassifizieren
  - Pro-Übertragung-Overhead
    - Aufwand für *SEND/RECEIVE*-Aufrufe
      - Z.B. dafür notwendige Betriebssystemaufrufe, Pufferallokationen, etc.
  - Overhead-pro-Dateneinheit
    - Aufwand für Verarbeitung einer Dateneinheit
      - Z.B. Ausführung des TCP/IP-Codes, Interrupts, Pufferallokation, etc.
  - Pro-Byte-Overhead
    - Aufwand für Speicherkopien und Prüfsummenberechnung
      - (Fallen an grau gekennzeichneten Aktionen auf der vorangegangenen Folie an)

Overhead

- Reduzierung des Endsystem-Overheads durch
  - Einsparung von Interrupts
    - z.B. durch verzögertes Auslösen von Interrupts bei zahlreichen eingehenden oder versandten Dateneinheiten
  - Integration der Prüfsummenberechnung in andere Operationen, welche die Daten durch die CPU schleusen
    - z.B. während Kopieraktionen, Ver- oder Entschlüsselung
  - Übernahme der Prüfsummenberechnung durch Hardware
    - Prozessoren vieler Netzwerkkarten können Prüfsumme bei Datenübergabe gleich mit berechnen, bzw. prüfen
  - …

Overhead durch Interrupts

- Interrupts sind bedeutende Quelle für Overhead-pro-Dateneinheit
  - Beispiel
    - Eine Netzwerkkarte empfängt 1500 Byte große Dateneinheiten über eine 1 Gbit/s-Anbindung:
      - 80.000 Interrupts pro Sekunde für eingehende Dateneinheiten
      - 40.000 Interrupts pro Sekunde zur Signalisierung des Versands von Quittungen

    ⇨ Skaliert auf diese Weise nicht
  - Einsparung von Interrupts notwendig
    - Interrupt coalescing / Interrupt suppression
      - Interrupts werden bei zahlreichen eingehenden oder versandten Dateneinheiten nur verzögert ausgelöst
      - Behandlungsroutine kann hierdurch mehrere Ereignisse während eines Aufrufes abarbeiten
    - Erste Schritte der TCP-Protokollbearbeitung gleich während der Interrupt-Behandlung anstatt nochmal extra Interrupt zum Aufruf der Protokollimplementierung auszulösen
      - Spart die sonst zusätzlich belastenden Software-Interrupts ein

Homework

- Schreiben Sie ein Programm mit dem Sie in der Lage sind, eine Datei über UDP zu übertragen. Setzen Sie direkt auf der Sockets-Schnittstelle auf.
- Schreiben Sie ein Programm mit dem Sie in der Lage sind, eine Datei über TCP zu übertragen. Setzen Sie direkt auf der Sockets-Schnittstelle auf.

2.6 Literatur

Bücher

[HaJa03] M. Hassan, R. Jain; High Performance TCP/IP Networking – Concepts, Issues and Solutions; Prentice Hall, 2003

[KrYo02] H. Kredel, A. Yoshida; Thread- und Netzwerk-Programmierung mit Java; dpunkt Verlag, 2002

[KuRo04] J. Kurose, K. Ross; Computer Networking; Addison-Wesley, 3rd Edition, 2004

[Stal06] W. Stallings; Data and Computer Communications; Prentice Hall, Inc., New Jersey, 2006, 8th Edition, ISBN 0-132-4331-09

[Stev03] W. R. Stevens, B. Fenner, A. M. Rudoff; Unix Network Programming; Addison-Wesley, 2003, 3rd Edition, ISBN 0-13-141155-1

[Wehr01] K. Wehrle, F. Müller; H. Ritter, D. Müller, M. Bechler; Linux-Netzwerkarchitektur – Design und Implementierung von Netzwerkprotokollen im Linux-Kern; Addison-Wesley, 2001

Links

[IANA09] Internet Assigned Numbers Authority (IANA); Port Numbers; Sep. 2009, http://www.iana.org/assignments/port-numbers

Literatur

RFCs

[AGS99] M. Allman, D. Glover, L. Sanchez; Enhancing TCP Over Satellite Channels using Standard Mechanisms; RFC 2488, Januar 1999

[AlPB09] M. Allman, V. Paxson, E. Blanton; TCP Congestion Control; RFC 5681, September 2009

[Brad89] R. Braden; Requirements for Internet Hosts -- Communication Layer; RFC 1122, Oktober 1989

[FlHG04] S. Floyd, T. Henderson, A. Gurtov; The NewReno Modification to TCP's Fast Recovery Algorithm; RFC 3782, April 2004

[JBB92] V. Jacobson, R. Braden, D. Borman; TCP Extensions for High Performance; RFC 1323, May 1992

[Nagl84] J. Nagle; Congestion Control in IP/TCP Internetworks; RFC 896, Januar 1984

[Paxo99] V. Paxon (Ed.); Known TCP Implementation Problems; RFC 2525, März 1999

[Post81] J. Postel (Ed.); Transmission Control Protocol; RFC 793, September 1981

[RaFB01] K. Ramakrishnan, S. Floyd, D. Black; The Addition of Explicit Congestion Notification (ECN) to IP; RFC 3168, September 2001

Literatur

Vertiefende Literatur

[BrMP94] L. S. Brakmo, S. W. O'Malley, L. L. Peterson; TCP Vegas: New Techniques for Congestion Detection and Avoidance; ACM SIGCOMM Computer Communication Review, Vol. 24, Issue 4, Oktober 1994, pp. 24-35

[HeBG00] U. Hengartner, J. Bolliger, T. Gross; TCP Vegas Revisited; Proceedings of INFOCOM 2000, pp. 1546-1555

[JaKa88] V. Jacobson, M. J. Karels; Congestion Avoidance and Control; Proceedings of SIGCOMM'88, Stanford, CA, August 1988, pp. 314-329

[Math97] M. Mathis, J. Semke, J. Mahdavi, T. Ott; The Macroscopic Behaviour of the TCP Congestion Avoidance Algorithm; ACM Computer Communication Review, Vol. 27, Issue 3, Juli 1997, pp. 67-82

[Padh00] J. Padhye, V. Firoiu, D. Towsley, J. Kurose; Modeling TCP Reno Performance: A Simple Model and its Empirical Validation; IEEE/ACM Transactions on Networking, Vol. 8, Issue 2, April 2000, pp. 133-145

[Sava99] S. Savage, N. Cardwell, D. Wetherall, T. Anderson; TCP Congestion Control with a Misbehaving Receiver; ACM Computer Communications Review, Vol. 29, Issue 5, October 1999, pp. 71-78

[Vasu09] V. Vasudevan et al.; Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communications; Barcelona, Aug. 2009, ACM SIGCOMM 09

[Zhan86] L. Zhang; Why TCP Timers Don't Work Well; ACM SIGCOMM Computer Communication Review, Vol. 16, Issue 3, August 1986, pp. 397-405

2.7 Aufgaben

1) Angenommen, TCP operiert über eine 1 Gbit/s-Verbindung. Wie lange dauert es, bis alle Sequenznummern einmal durchlaufen wurden?
2) Ein 3-Wege-Handshake baut eine TCP-Verbindung ordnungsgemäß auf. Welche Schritte benötigt ein ordnungsgemäßer Abbau?
3) Diskutieren Sie die Wirkungsweise von Slow-Start und die Motivation für dessen Einführung.
4) Welches ist der wichtigste Timer innerhalb des TCP-Protokolls und was ist dessen Aufgabe? Erläutern Sie die Auswirkungen ungünstiger Timer-Werte auf die TCP-Performance.
5) Welche Aufgaben haben das Flusskontrollfenster und das Staukontrollfenster? Weshalb benötigt man zwei Fenster?
6) Beschreiben Sie die Mechanismen von TCP zur Vermeidung von Stausituationen (Congestion Avoidance). Erläutern Sie weshalb TCP sich fair verhält bei Stausituationen.

Aufgaben

8) Diskutieren Sie, wann genau ein Empfänger Quittungen zum Sender zurückschickt, wenn er reihenfolgetreue Daten empfängt, alle Daten davor quittiert sind und es keine Lücken gibt?

9) Was versteht man unter Piggyback von Quittungen? Wann kann diese Technik bei TCP verwendet werden?

10) Welche zwei Parameter haben entscheidenden Einfluss auf die erreichbare Datenrate einer TCP-Verbindung?

11) Diskutieren Sie Vor- und Nachteile der Explicit Congestion Notification.

12) Was versteht man unter einem Socket? Was ist der Unterschied zwischen einem Welcome-Socket und einem Connection-Socket?

13) Nennen Sie verschiedene Ursachen für Overhead innerhalb der TCP-Verarbeitung.