Vorlesung Kommunikationssysteme
Wintersemester 2024/25

User Datagram Protocol und Transmission Control Protocol

Christoph Lindemann

Comer Buch, Kapitel 24, 25

Zeitplan

Nr.	Datum	Thema
01	18.10.24	Organisation und Internet Trends
02	25.10.24	Programmierung mobiler Anwendungen mit Android
	01.11.24	Keine Vorlesung
03	08.11.24	Protokolldesign und das Internet
04	15.11.24	Anwendungen und Netzwerkprogrammierung
05	22.11.24	LAN und Medienzugriff
06	29.11.24	Ethernet und drahtlose Netze
07	06.12.24	LAN Komponenten und WAN Technologien
08	13.12.24	Internetworking und Adressierung mit IP
09	20.12.24	IP Datagramme
10	10.01.25	Zusätzliche Protokolle und Technologien
11	17.01.25	User Datagram Protocol und Transmission Control Protocol
12	24.01.25	TCP Überlastkontrolle / Internet Routing und Routingprotokolle
13	31.01.25	Ausblick: TCP für Hochgeschwindigkeitsnetze
14	07.02.25	Review der Vorlesung

Überblick

Ziele:

- Einblick in die beiden wichtigen Transportprotokolle des Internets

Themen:

- UDP
  - Paketformat
  - Nachrichtenaustausch
- TCP
  - Paketformat
  - Servicemodell
  - Verbindungsaufbau
  - Flusskontrolle

User Datagram Protocol

Ende-zu-Ende Kommunikation

- IP unterscheidet nicht zwischen verschiedenen Applikationen auf einem Host
  - Beispiel: E-Mail und Webbrowser gleichzeitig geöffnet, mehrere Instanzen von einem Programm
- Felder in Header von IP Datagramm identifizieren nur Host
- IP behandelt Computer als Endpunkt der Kommunikation
- Protokolle der Transportschicht sind Ende-zu-Ende Protokolle
  - Applikation ist Endpunkt der Kommunikation

User Datagram Protocol (1)

- **User Datagram Protocol (UDP)**
  - Eines der zwei Hauptprotokolle der Transportschicht (neben TCP)
  - Weniger komplex und leicht verständlich

- Charakterisierung

  - Ende-zu-Ende: Unterscheidung zwischen verschiedenen Applikationen

  - Verbindungslos

  - Nachrichtenorientiert: Applikation sendet und empfängt individuelle Nachrichten

User Datagram Protocol (2)

- Charakterisierung
  - Best-Effort: Selbe Best-Effort Semantik wie IP
  - Beliebige Interaktion: Senden und Empfangen zu und von beliebig vielen Applikationen
  - Betriebssystemunabhängig: Nicht abhängig von Bezeichnern des lokalen Systems
- Erlaubt Applikationen Senden und Empfangen von IP Datagrammen

Verbindungslosigkeit von UDP

- Kein Verbindungsaufbau und Verbindungsabbau zum Senden
- Daten können zu beliebiger Zeit erstellt und gesendet werden
- Applikation kann beliebig lange zwischen zwei Nachrichten warten
- Sehr geringer Overhead
  - UDP verwaltet keine Zustände
  - UDP sendet nur reine Datennachrichten und keine Kontrollnachrichten

Nachrichtenorientiertes Interface (1)

- Applikation möchte Block von Daten versenden → UDP überträgt diese in einer einzigen Nachricht
- Keine Unterteilung oder Bündelung von mehreren Nachrichten
- Datengrenzen bleiben erhalten: Empfänger bekommt Nachricht wie von Sender verschickt
- Aber: UDP Nachricht muss in IP Datagramm passen

Nachrichtenorientiertes Interface (2)

- Ineffiziente Nutzung des Netzwerks möglich
  - Hoher Overhead bei kleinen Nachrichten
  - Häufige Fragmentierung von großen Datagrammen (eventuell schon vor Versand)
- Programmierer beschränken UDP-Nachrichten oft auf 1400 - 1450 Byte (MTU im Großteil des Internet 1500 Byte)

Best-Effort Semantik

- UDP hat Best-Effort Semantik von IP
  - Verlust, Duplikate, Verzögerung, Falsche Reihenfolge, Datenfehler
- Übertragungsfehler werden nicht erkannt und nicht korrigiert
- Applikation muss immun gegenüber Problemen sein oder selbst Fehler entdecken und korrigieren
- Beispiel:
  - Audio-/Videoübertragung toleriert Fehler
  - Probleme bei Online Shopping (Doppelte Bestellung oder Abbuchung)

Interaktion und Multicast

- Mögliche Kommunikation:
  - 1-zu-1
  - 1-zu-N
  - N-zu-1
  - N-zu-M

- Verwendung von IP Multicast (oder IPv4 Broadcast) für 1-to-many möglich

- Vermeidet wiederholtes Versenden von Kopie der Nachricht an alle Empfänger

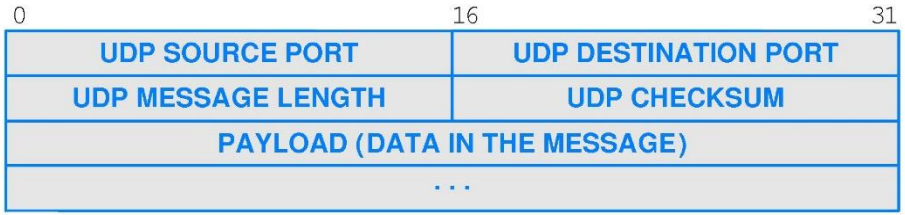
- Vor allem in Ethernet nützlich, da von Hardware effizient unterstützt

UDP Portnummer

- Wie wird Applikation als Endpunkt identifiziert?
- Betriebssysteme nutzen Prozess IDs, Task IDs, Job Namen
  - Nicht ausreichend (Heterogene Computer kommunizieren)
- Protokoll Portnummer (Ports) werden definiert
  - Unabhängig von Betriebssystem
  - Betriebssystem mit UDP muss Mapping zwischen Ports und Prozessen bereit stellen
- Kommunikationsmodus von Applikation bei Erstellung des Socket festgelegt
  - Nur Nachrichten von bestimmter IP und Port (1-zu-1) oder von allen anderen Endpunkten (N-zu-1)

Datagramm Format

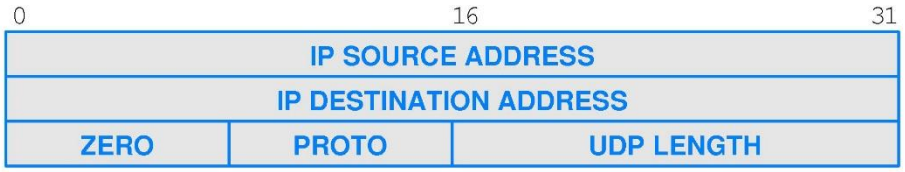
- UDP Nachricht wird **User Datagramm** genannt
  - Kurzer Header mit Sender und Empfänger Programmen
  - Payload mit den Daten
- Source und Destination Port mit 16 Bit (0 - 65535)



UDP Datagramm mit 8 Byte Header

UDP Checksumme

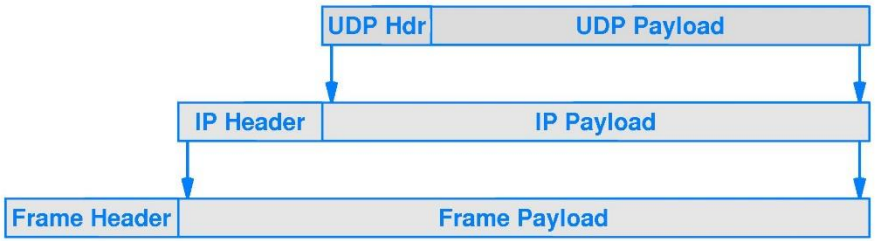
- UDP Checksumme optional
  - Alle Bits auf 0 → wird nicht verwendet
- IP Adressen nicht Teil des UDP Header
  - UDP kann nicht feststellen, ob Datagramm richtiges Ziel erreicht hat
- Checksummen Berechnung erweitert Header um Pseudoheader
  - Enthält IP Source, IP Destination, Type von IP Datagramm sowie UDP Length



Pseudoheader für UDP Checksumme

UDP Kapselung

- UDP Datagramm wird zur Übertragung in IP Datagramm gekapselt
  - Wird als Payload von IP übertragen
- IP wird wiederum in Frame des individuellen Netzwerk übertragen



Zusammenfassung

- UDP bietet verbindungslosen Ende-zu-Ende Transport von Nachrichten
- Kommunikation erfolgt zwischen Anwendungen auf verschiedenen Computern
- Bietet Best-Effort wie IP
- Keine Beschränkung auf 1-zu-1 Interaktionen
- Nutzt Portnummern um Anwendungen zu unterscheiden
- UDP Checksumme ist optional

Transmission Control Protocol

Einführung

- **Transmission Control Protocol (TCP)**
- Bietet zuverlässige Datenübertragung zwischen Anwendungen auf Transportschicht
- Nutzt unzuverlässigen Datagramm Dienst von IP
- Verlust, Verspätung, Duplikate, falsche Reihenfolge muss kompensiert werden
- Verhindert Überlastung der Netzwerke und Router

Dienste von TCP (1)

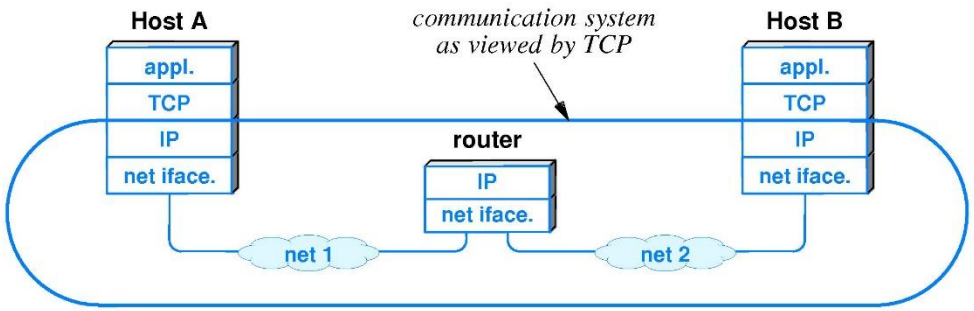
- Verbindungsorientiert: Anwendung muss vor Versand Verbindung zu Ziel anfordern
- Point-to-Point: Jede TCP Verbindung hat zwei Endpunkte
- Zuverlässigkeit: Daten werden exakt so zugestellt wie versendet (vollständig, richtige Reihenfolge)
- Full Duplex: Daten können in beide Richtungen fließen, Anwendungen dürfen zu beliebiger Zeit senden

Dienste von TCP (2)

- Stream Interface:
  - Applikation sendet kontinuierliche Sequenz von Daten
  - Größe der ankommenden Stücke kann ungleich der gesendeten Stücke sein
- Verbindungsaufbau erfolgt zuverlässig
- Verbindungsabbau:
  - Stellt sicher, dass alle Daten vorher übertragen werden
  - Beide Seiten müssen Verbindungsabbau zustimmen

Virtuelle Verbindungen

- Verbindungen virtuell, da nur mit Software erreicht
- Keine Unterstützung durch Hardware und Software im verwendeten Internet
- TCP Nachricht in IP Datagramm gekapselt
- TCP muss nur an beiden Endpunkten der Verbindung vorhanden sein



Techniken von Transportprotokollen

- Hauptprobleme
  - Unzuverlässige Kommunikation
  - Neustart von Endsystem
  - Heterogene Endsysteme
  - Überlastung des Internet
- Transportprotokolle besitzen Techniken um Probleme zu beheben oder zu umgehen

Duplikate, Out-of-Order (1)

- Verwendung von Sequenzierung
  - Sender fügt Sequenznummer zu jedem Paket hinzu
  - Empfänger speichert Sequenznummer S des letzten Pakets in richtiger Reihenfolge
  - Empfänger speichert Liste L von zusätzlichen Paketen in falscher Reihenfolge
- Neues Paket in richtiger Reihenfolge (folgt auf S)
  - Zustellung an nächste Schicht
  - Überprüfung der Liste L, ob weitere Pakete zugestellt werden können

Duplikate, Out-of-Order (2)

- Neues Paket in falscher Reihenfolge
  - Wird zu Liste L hinzugefügt
- Duplikate
  - Prüfung der Sequenznummer des Paket
  - Verwerfen, falls Sequenznummer vor S liegt
  - Verwerfen, falls schon in Liste L enthalten

Verlorene Pakete (1)

- Nutzung von Positive Acknowledgement mit **Retransmission**
- Kleine **Acknowledgment (ACK)** Nachricht wird für empfangenen, intakten Frame gesendet
- Sender hat Verantwortung, dass alle Pakete erfolgreich zugestellt werden
- Bei Versand jedes Pakets wird Timer gestartet

Verlorene Pakete (2)

- Timer wird abgebrochen, wenn ACK vor Ablauf ankommt
- Läuft Timer vorher ab, wird neue Kopie versendet und Timer neugestartet → Retransmitting
- Maximalzahl an Neuübertragungen beschränkt
  - Hardwarefehler, Empfänger abgestürzt, Keine Verbindung mit Netzwerk
- Retransmission kann zu Duplikaten bei Verzögerungen führen

Replay

- Lang verzögerte Pakete können zu Replay Fehlern führen
- Beispiel:
  - Paket einer Session wird durch Hardwarefehler verzögert → Wird erst zugestellt, wenn neue Session bereits aktiv ist
  - Kontrollpaket zum Verbindungsabbau wird verzögert → Beendet spätere neue Session bei Zustellung
- Lösung
  - Session bekommt einzigartige ID
  - Wird in jedes Paket der Session eingetragen, Empfänger verwirft andere Pakete
  - ID darf lange Zeit nicht neu genutzt werden

Flusskontrolle (1)

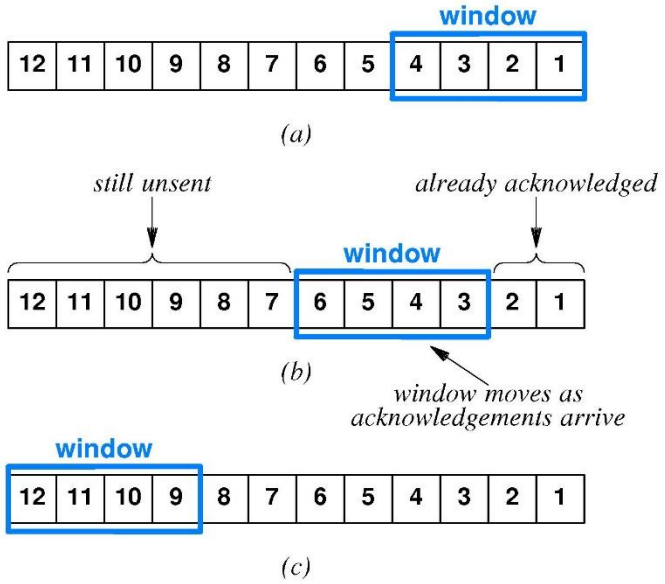
- Schneller Computer kann langsamen Empfänger überfordern
  → **Flusskontrolle** oder **Flow Control** notwendig
- Einfachste Flusskontrolle ist **Stop-and-Go**
  - Sender wartet nach jedem Paket
  - Empfänger sendet nach empfangenem Paket Kontrollnachricht
- Führt zu sehr geringem Durchsatz
  - Paketgröße 1000 Byte, Durchsatz 2 Mbps, 50 ms Verzögerung
  - Nur ein Paket alle 100 ms möglich → 80.000 bps (4% der Kapazität)

Flusskontrolle (2)

- Für höheren Durchsatz wird **Sliding Window** verwendet
- **Window Size**: Maximalmenge an Daten, die ohne Empfang von ACK gesendet werden kann
- Empfänger hält Pufferspeicher, welcher mind. die Größe der Window Size hat
- Paket in richtiger Reihenfolge → wird an höhere Schicht geleitet und ACK versandt
- Wird ACK empfangen sendet Sender nächstes Paket

Flusskontrolle (3)

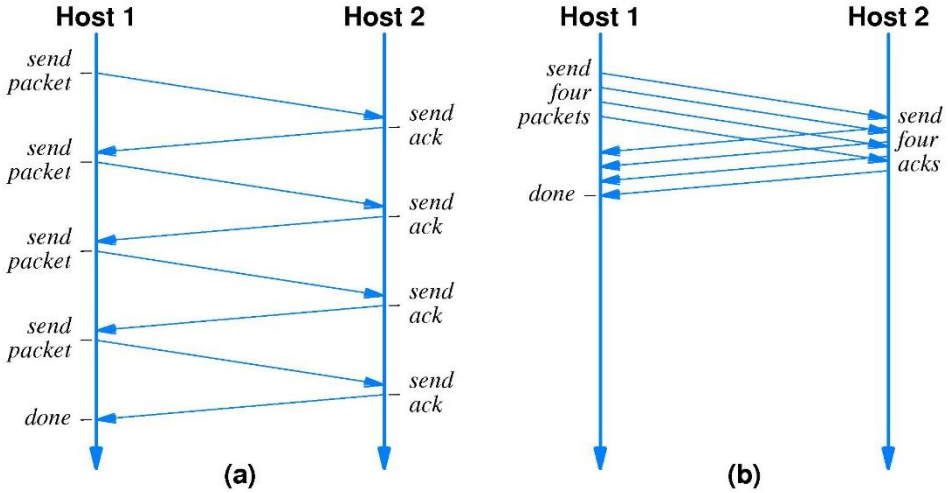
- Vorstellbar als ein Fenster, welches über Daten geschoben wird



Sliding Window: (a) initial, (b) dazwischen, (c) am Ende

Flusskontrolle (4)

- Stop-and-Go: Verzögerung von N in Netzwerk → Gesamtzeit 8N
- Sliding Window: Kurze Verzögerung zwischen einzelnen Paketen → Gesamtzeit 2N + ε



(a) Stop-and-Go, (b) Sliding Window

Flusskontrolle (5)

- Bei großer Anzahl an Paketen kann ε vernachlässigt werden
- Potenzielle Verbesserung durch Sliding Window: $T_w = T_g \times W$
  - T_w Durchsatz mit Sliding Window, T_g Durchsatz mit Stop-and-Go, W Window Size
- Durchsatz kann mit größerem Window nicht beliebig erhöht werden
  - Obere Schranke durch Kapazität des Netzwerks: $T_w = \min(C, T_g \times W)$, C Kapazität des Netzwerks

Vermeidung von Überlast (1)

Betrachte folgendes Szenario:

- Jede Verbindung hat Kapazität 1 Gbps
- Beide Computer an Switch 1 senden an Computer an Switch 2 → Switch 1 empfängt mit 2 Gbps, kann nur mit 1 Gbps an Switch 2 senden
- Switch kann temporär Pakete speichern, führt dennoch zu Verzögerung

Vermeidung von Überlast (2)

- Bei weiterer Überlastung, wird Speicher voll und Pakete verworfen
- Retransmission führt wieder zu neuen Paketen
- Netzwerk kann unbenutzbar werden → **Congestion Collapse**
- Transportprotokolle versuchen Congestion Collapse zu verhindern
- Nutzen **Congestion Control** bzw. **Überlastkontrolle**

Vermeidung von Überlast (3)

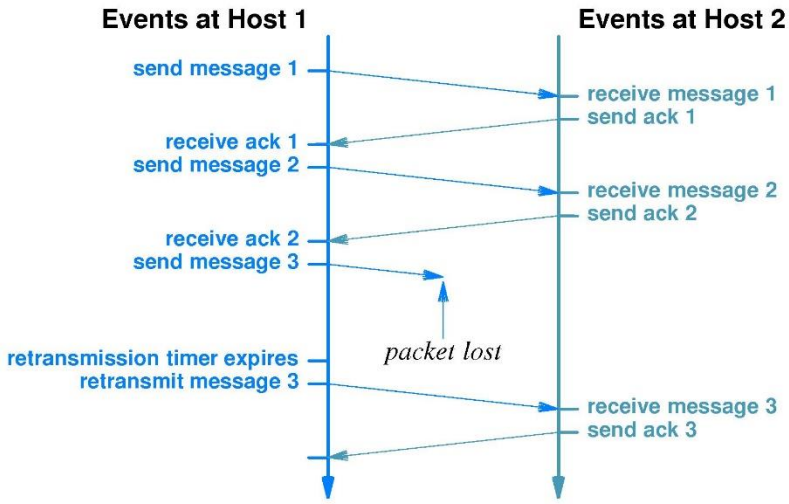
- Zwischenliegende Systemkomponenten (ie Router) könnten Sender über Überlast informieren
  - Spezielle Nachricht an Sender oder gesetztes Bit in verzögerten Paketen
- Schätzen von Überlast anhand erhöhter Verzögerung oder Verlust
  - Empfänger informiert Sender mit Informationen in ACK
- Moderne Netzwerkhardware funktioniert zuverlässig → Verluste und Verzögerung meist durch Überlast
- Bei Überlast wird Rate verringert
  - Sliding Window verkleinert Window temporär

Protokolldesign

- Design nicht trivial: Kleine Designfehler führen zu falschem Ablauf, unnötigen Paketen, Verzögerungen
- Trade-off bei Größe der Sequenznummern: Häufige Wiederbenutzung oder Platzverschwendung in Header
- Flow Control mit Sliding Window nutzt mehr Kapazität um Durchsatz zu verbessern vs. Congestion Control verringert Zahl an Paketen, um Überlast zu vermeiden
- Neustart von Empfänger während Verbindung → Empfang von Daten aus der Mitte des Streams

TCP Paketverlust (1)

- Verschiedene Techniken werden kombiniert
- Empfängt TCP Daten, wird Acknowledgement an Sender gesendet
- TCP nutzt Timer für versendete Daten und sendet neu bei Ablauf



TCP Paketverlust (2)

- Wie lange muss bis Retransmitting gewartet werden?
  - ACKs in LAN werden in wenigen Millisekunden erwartet
  - Bei Verbindung über Satellit mehrere hundert Millisekunden
- Zu lange warten: Verringert Durchsatz, Netzwerk untätig
- Zu kurz warten: Unnötiger Traffic verbraucht Bandbreite und verringert Durchsatz
- Verzögerung kann sich durch Überlast rasch ändern

Adaptive Retransmission (1)

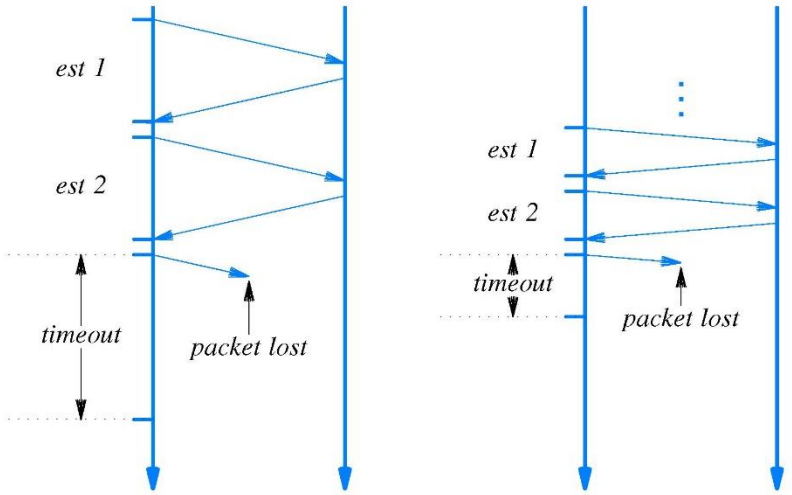
- Vor TCP wurde nur feste Verzögerung genutzt
- Retransmission in TCP wurde adaptiv gestaltet
- TCP überwacht Verzögerung jeder Verbindung und passt Retransmission Timer an
- TCP schätzt Round-Trip Verzögerung anhand Zeit um Antwort zu bekommen
  - Zeit des Versand muss gespeichert werden

Adaptive Retransmission (2)

- Erhält Sequenz von Schätzungen für Round-Trip Time, bildet gewichteten Durchschnitt
  - Linearkombination aus geschätzten Mittelwert und geschätzter Varianz der Round-Trip Time
- Varianz: Schnelle Reaktion, falls Verzögerung nach Burst steigt
- Gewichteter Durchschnitt: Falls Delay sinkt, wird Timer schnell zurück gesetzt
- Konstante Verzögerung: Timer auf Wert gesetzt, der nur wenig größer als Round-Trip Verzögerung ist

Vergleich der Retransmission

- TCP setzt Retransmission Timeout wenig größer als mittlere Round-Trip Verzögerung
- Größerer Timeout, falls vorher große Verzögerung



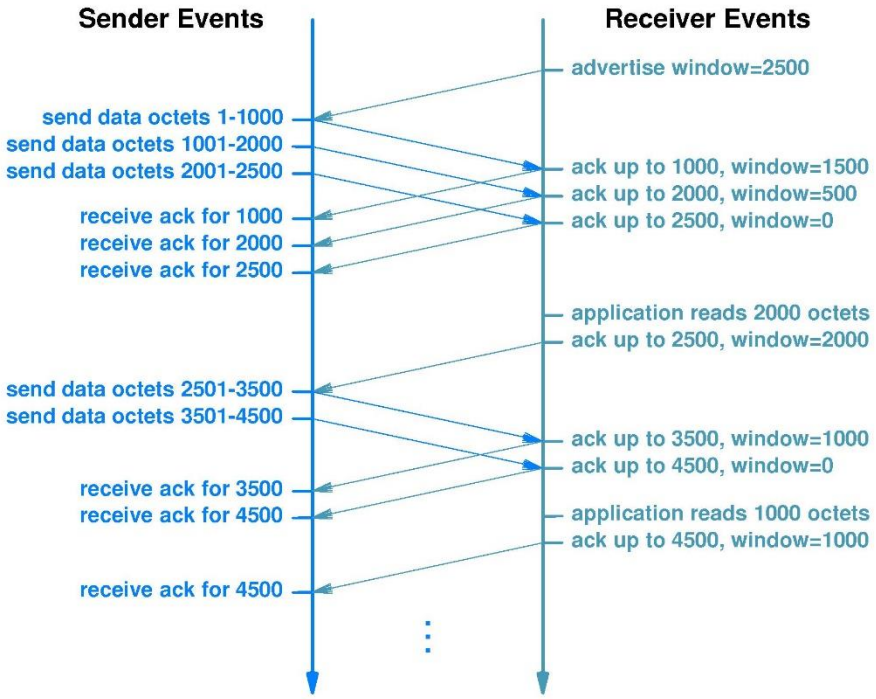
Paketverlust in zwei Verbindungen mit verschiedenen RTT

Buffer, Flusskontrolle, Window (1)

- Flusskontrolle in TCP nutzt Window Mechanismus
  - Gemessen in Bytes und nicht in Anzahl Pakete
- Bei Verbindungsaufbau allokiert Empfänger Buffer und teilt Größe an Sender mit
- TCP am Empfänger sendet ACK mit verbleibender Größe des Buffer (**Window Advertisement**)
- Falls Daten schneller gesendet als Empfangen werden (z.B. langsamere CPU), füllt sich Buffer → Zero Window
  - Empfängt Sender Zero Window Advertisement sendet er nicht bis Window wieder positiv ist

Buffer, Flusskontrolle, Window (2)

- Sender sendet schneller als Empfänger Daten liest
- Zusätzliches Window Advertisement nachdem Daten gelesen



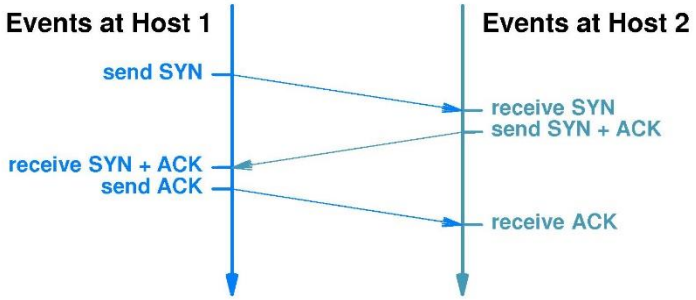
TCP Three-Way Handshake (1)

- Verbindungen werden zuverlässig an- und aufgebaut
- **Three-Way Handshake**: Drei Nachrichten werden ausgetauscht

- Jede Seite sendet Sequenznummer und initiale Buffer Größe

- Robust gegenüber Verlust, Duplikate, Verzögerung, Replay

- Verbindung nicht geöffnet bis beide Endpunkte zugestimmt haben



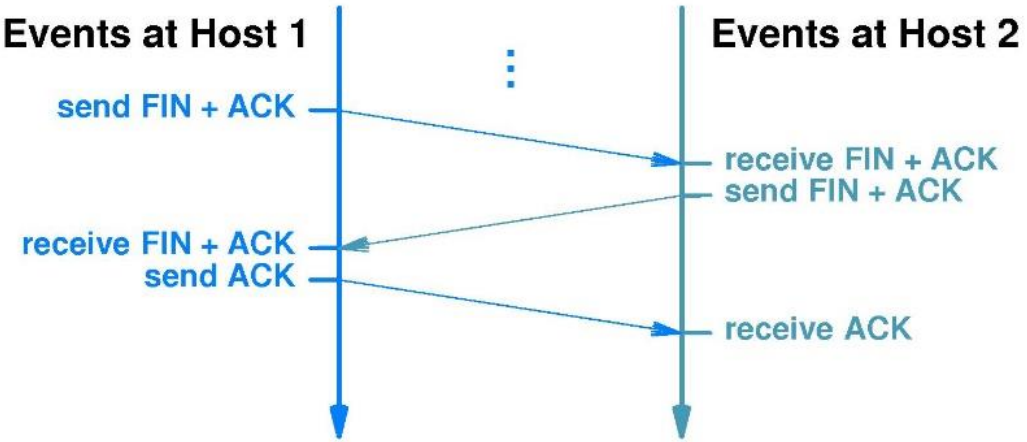
TCP Three-Way Handshake (2)

- 2-Way Handshake reicht nicht
  - A sendet Request an B, B bestätigt → B weiß nicht, ob Bestätigung angekommen
- **Synchronization Segment (SYN):** Kontrollnachrichten für Verbindungsaufbau in TCP
- Jeder Endpunkt generiert zufällig initiale Sequenznummer
  - Verhindert Verarbeitung von Daten alter Verbindungen

TCP Three-Way Handshake (3)

- **Finish Segment (FIN):** Kontrollnachrichten für Verbindungsabbau in TCP

- ACK in jede Richtung um zu garantieren, dass vor Verbindungsabbau alle Daten angekommen sind



Zusammenfassung

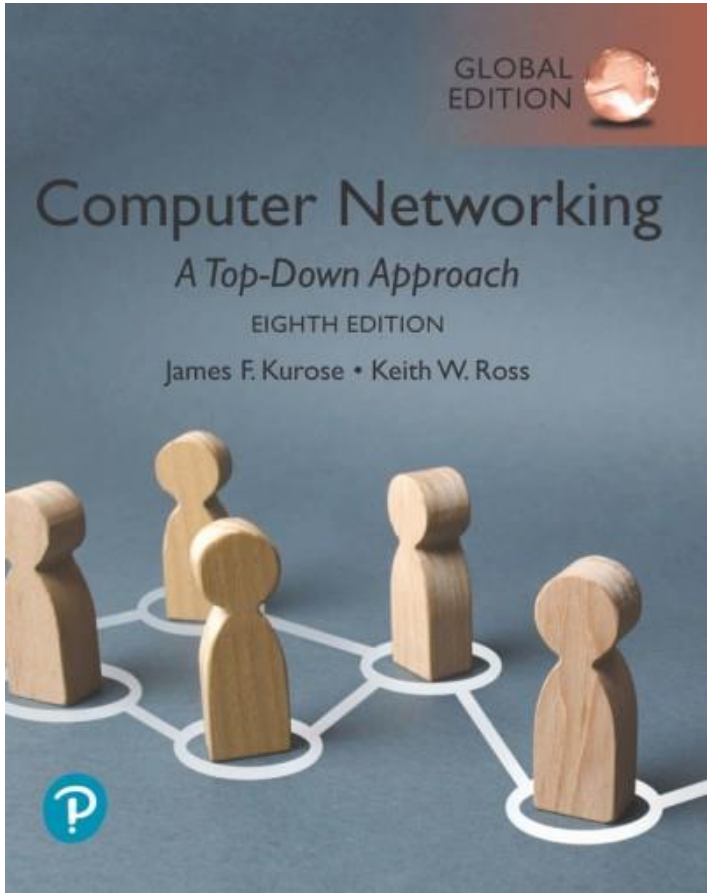
- TCP bietet zuverlässige Datenübertragung zwischen Anwendungen auf Transportschicht
- Nutzt unzuverlässigen Datagramm Dienst von IP
- Verbindungsorientiert
- Verlust, Verspätung, Duplikate, falsche Reihenfolge wird kompensiert
- Verhindert Überlastung der Netzwerke und Router

Weiterführendes Lehrbuch zur Vorlesung

James Kurose, Keith Ross, Computer Networking: A Top-Down Approach, Global Edition, 8. Auflage, Pearson, 2021.

An Uni als E-Book

https://katalog.ub.uni-leipzig.de/Record/0-1771738375



Selbststudium

Zum Vertiefen der Inhalte dieser Vorlesung

Leseaufgabe zum Selbststudium bis 7.2.2025:

James Kurose, Keith Ross, Computer Networking: A Top-Down Approach, Pearson, 2021. S. 211 - 331

Chapter 3: The Transport Layer.

Klausur

Termin:

27.02.2025, 10:30 Uhr bis 11:30 Uhr

Ort:

Audimax, Augusteum

Viel Erfolg!!