

---

# **Datenkommunikation**

Begleitmaterial für das  
Wintersemester 2018/2019

@Prof. Dr. Peter Mandl

---

---

# Datenkommunikation

## Einführung

Wintersemester 2018/2019

# Inhalt der Vorlesungen, Lehreinheiten

---

- 1 Kommunikationssysteme und verteilte Anwendungen
- 2 Grundlagen der Transportschicht
- 3 Transportprotokolle TCP und UDP
- 4 Grundlagen der Vermittlungsschicht
- 5 Internet und Internet Protocol (IP)
- 6 Routing-Verfahren und -Protokolle
- 7 Internet-Steuerprotokolle und DNS
- 8 Internet Protocol Version 6 (IPv6)
- 9 Netzwerkschnittstelle

# Studienarbeit

---

- Aufgabenstellung (siehe Aufgabenblatt)
  - Programmierung einer verteilten Anwendung und Leistungsanalyse (Chat-Anwendung)
  - Teamarbeit
- Alle Dokumente liegen auf [www.prof-mandl.de](http://www.prof-mandl.de)
  - ... ->Veranstaltungen
  - Datenkommunikation ...
- **Anteil von 0,4 an der Gesamtnote**
- Termine: Planung folgt

- 1 Kommunikationssysteme und verteilte Anwendungen**
- 2 Grundlagen der Transportschicht
- 3 Transportprotokolle TCP und UDP
- 4 Grundlagen der Vermittlungsschicht
- 5 Internet und Internet Protocol (IP)
- 6 Routing-Verfahren und -Protokolle
- 7 Internet-Steuerprotokolle und DNS
- 8 Internet Protocol Version 6 (IPv6)
- 9 Netzwerkschnittstelle

## Definition: Datenkommunikation

---

Datenkommunikation befasst sich mit dem  
Transport von Daten über Übertragungskanäle

# Überblick über das Kapitel

---

- **Terminologie und Referenzmodelle**
- Beispiele für Anwendungsprotokolle
- Chat-Anwendung als begleitende Studie

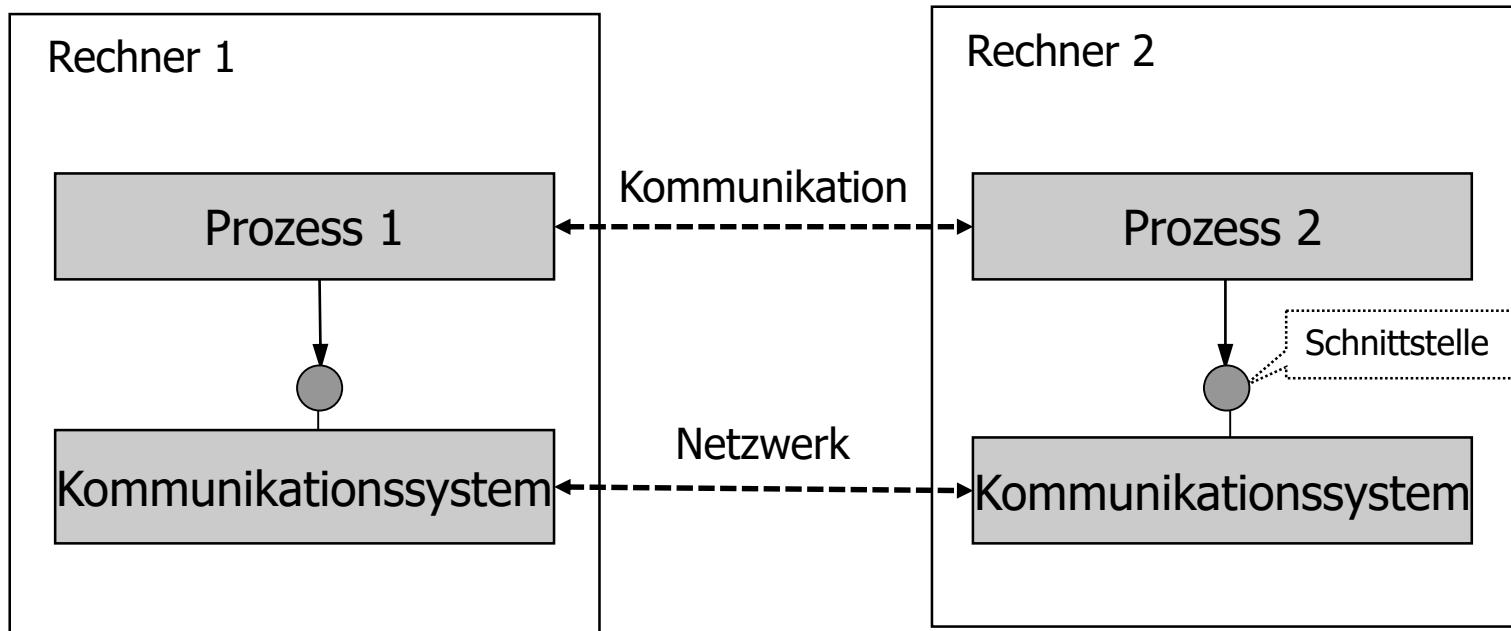
## Informelle Definition: Verteilte Anwendungen

---

- Verteilte Anwendungen verteilen ihre Funktionalität auf mehrere, ggf. auch (sehr) viele Rechnersysteme
- Prozesse der verteilten Anwendungen **kommunizieren miteinander über Nachrichten**
- Prozesse nutzen also Datenkommunikation, um ihre Aufgaben zu erfüllen

# Kommunikationssysteme

- Auf jedem Rechnersystem ist üblicherweise ein Kommunikationssystem verfügbar, das die rechnerübergreifende Kommunikation ermöglicht



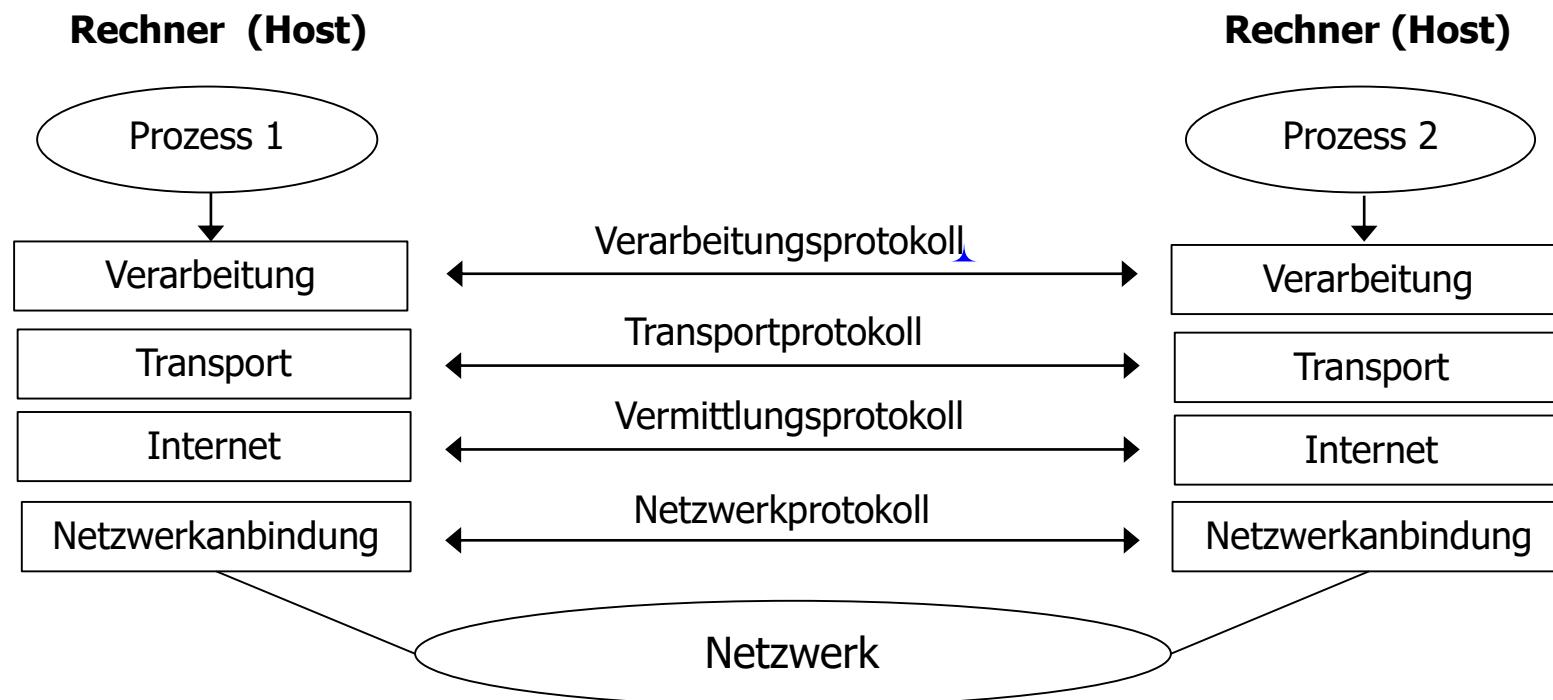
# Kommunikationssysteme

---

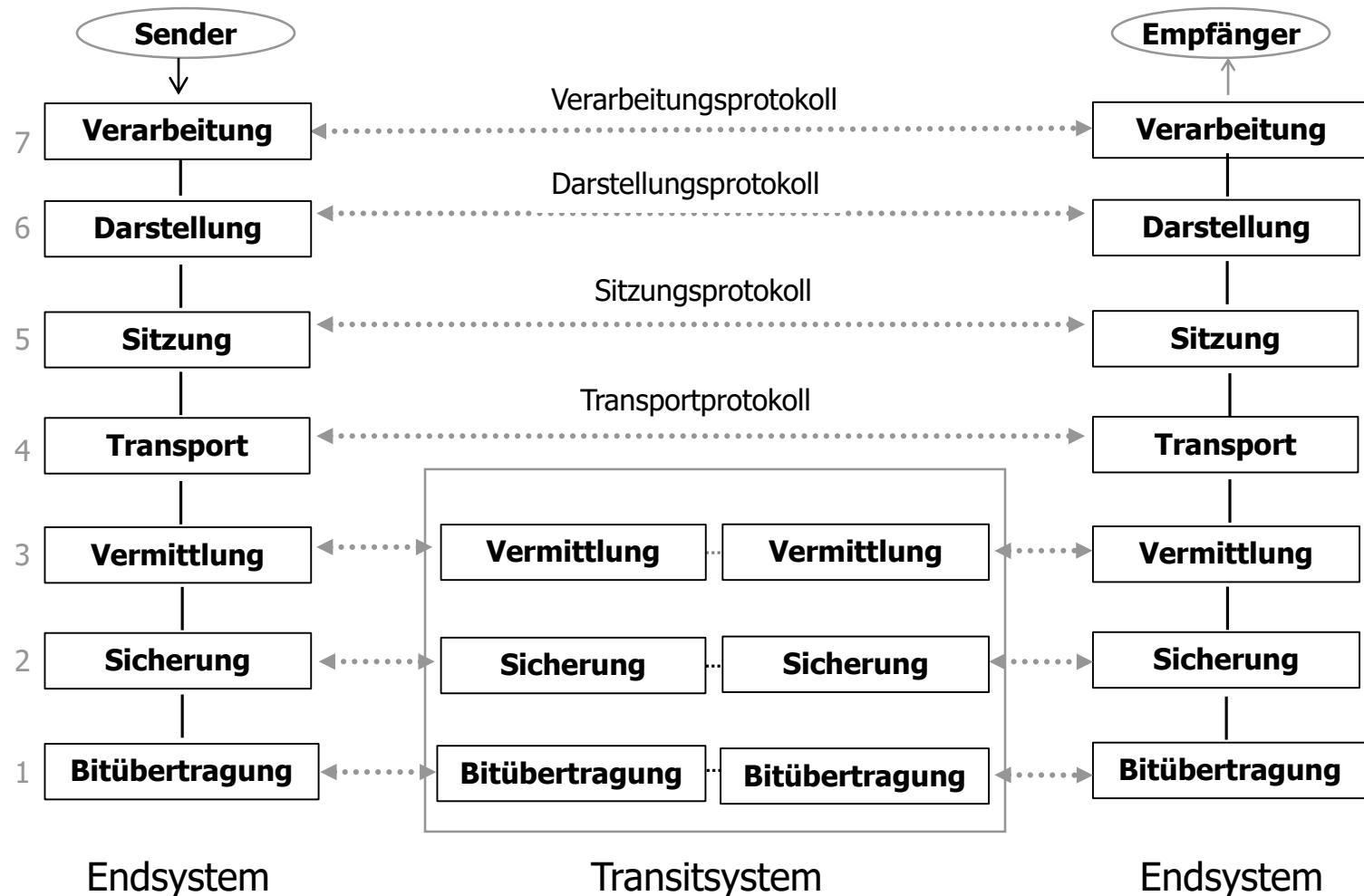
- Um ein Kommunikationssystem überschaubar zu machen, zerlegt man es in **Schichten** (layer)
  - Beim ISO/OSI-Modell sind es 7 Schichten
  - Beim TCP-Referenzmodell sind es 4 Schichten
- Jede Schicht bietet lokal auf einem Rechnersystem einen Dienst für die darüber liegende Schicht
- Jede Schicht kapselt seine Implementierungsdetails
- Man unterscheidet:
  - Endsysteme, auf denen Anwendungen ablaufen
  - Transitsysteme (Router)

# TCP/IP-Referenzmodell

- Vier Kommunikationsschichten
- Unterschiedliche Protokolltypen in den einzelnen Schichten



# ISO/OSI-Referenzmodell: Schichtung



# ISO/OSI-Referenzmodell: Aufgaben der Schichtung

---

- 7 **Verarbeitung**  
Höhere Protokolle und Dienste (File Transfer, Instant Messaging, Web-Kommunikation, Remote Procedure Call, Remote Login, Verteilte Filesysteme, E-Mail, ...)
- 6 **Darstellung**  
Einheitliche Transportsyntax, unterschiedliche lokale Syntaxen
- 5 **Sitzung**  
Synchronisation und Dialogablauf zwischen zwei Kommunikationsteilnehmern, Sessionverwaltung
- 4 **Transport**  
Stellt einen Transportservice bereit, **Ende-zu-Ende-Verbindung zwischen Anwendungsprozessen** 
- 3 **Vermittlung**  
Vermittlung, Paketleitweg der Nachricht **vom Quell- zum Zielrechner** ermitteln (Routing)
- 2 **Sicherung**  
Bitstrom in logische Einheiten umwandeln, Fehlerbearbeitung, **Ende-zu-Ende-Verbindung** zwischen Rechnern
- 1 **Bitübertragung**  
Festlegung der elektrischen, mechanischen und funktionalen Parameter einer physikalischen Verbindung

# ISO/OSI-Referenzmodell: Protokolle und Standards der einzelnen Schichten

---

- |   |                       |   |
|---|-----------------------|---|
| 7 | <b>Verarbeitung</b>   | FTP, HTTP, SNMP, SMTP, IMAP, SOAP, OSI-TP, RPC, XMPP, ...   |
| 6 | <b>Darstellung</b>    | ASN.1, XDR, XML, HTML, UniCode, ...   |
| 5 | <b>Sitzung</b>        | AppleTalk Session Protocol (nicht mehr relevant), OSI Session-Layer Protocol (kaum relevant), ...   |
| 4 | <b>Transport</b>      | OSI-TP4 (kaum relevant), TCP, UDP, ...  |
| 3 | <b>Vermittlung</b>    | IPv4, IPv6, ICMP, OSI-Protokolle, Datex-P X.25 (nicht mehr relevant), Novell IPX (nicht mehr relevant), Connectionless Network Protocol (CLNP) (nicht mehr so relevant), IPSec, ... |
| 2 | <b>Sicherung</b>      | HDLC, SDLC, BSC, Ethernet, Token Ring (kaum relevant), Token Bus (nicht mehr relevant), FDDI, ATM, ...  |
| 1 | <b>Bitübertragung</b> | Ethernet, CSMA/CD, V24, ISDN (kaum mehr relevant) , Token Ring (kaum relevant) FDDI, SONET, ...   |

# ISO/OSI-Referenzmodell: Protokolle und Dienste allgemein

---

- Protokolle sind **Verhaltensrichtlinien**, auf deren Grundlage sich Computersysteme untereinander „unterhalten“ und gegenseitig verstehen:
  - Legt die Spielregeln fest, an die sich Sender und Empfänger halten müssen
  - Notwendig, damit die Übertragungswünsche der Netzeilnehmer nicht im Chaos enden
- Protokolle sind **Vorschriften und Konventionen** zur Regelung der Nachrichtenübermittlung und bei Bedarf für den Verbindungsaufl- und -abbau

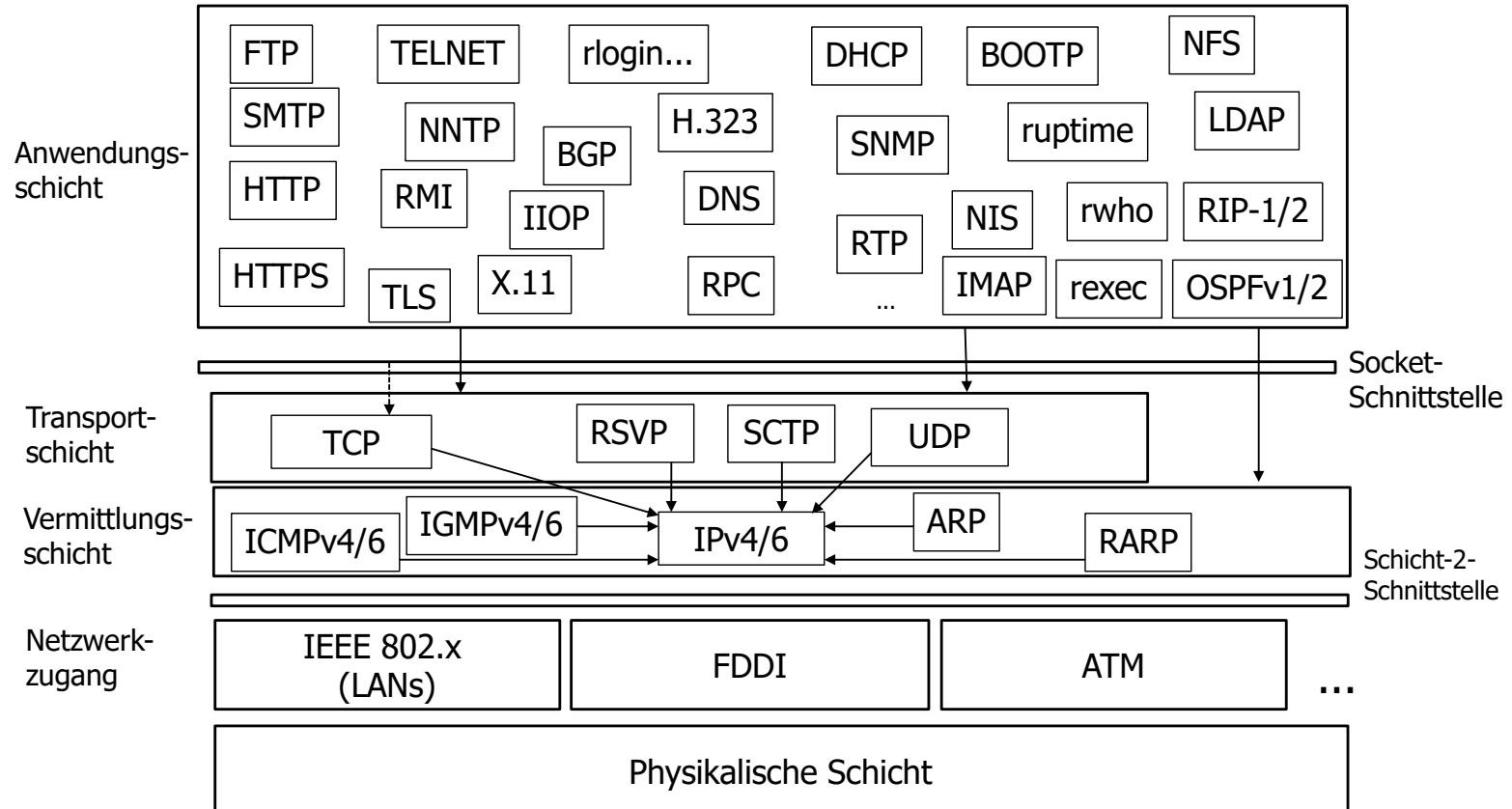
# ISO/OSI-Referenzmodell: Protokolle und Dienste allgemein

---

- Jede Schicht bietet der ihr jeweils übergeordneten Schicht Funktionen, sog. **Dienste** an
- Jede Schicht (bis auf die unterste) kann von der direkt darunter liegenden Schicht Dienste in Anspruch nehmen, **ohne ihre Implementierung zu kennen**
- Protokolle übernehmen verschiedene Aufgaben (je nach Schicht)
  - Verbindung aufbauen und Verbindung abbauen
  - Datenübertragung
  - Fehlererkennung und Fehlerbehebung
  - Staukontrolle (Congestion Control)
  - Flusskontrolle
  - ...

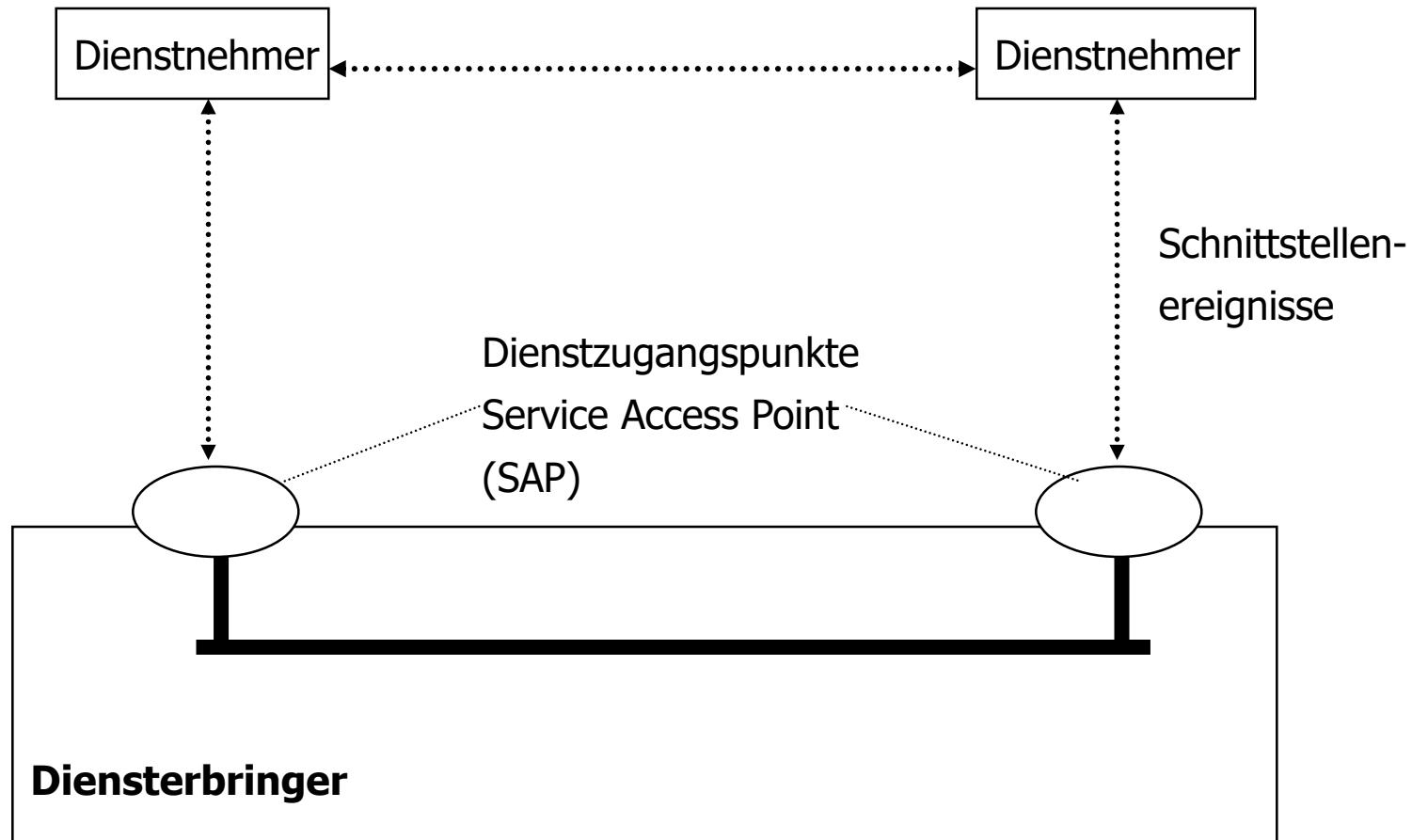
# Übersicht TCP/IP-Protokollfamilie

## Viele Protokolle in der Anwendungsschicht

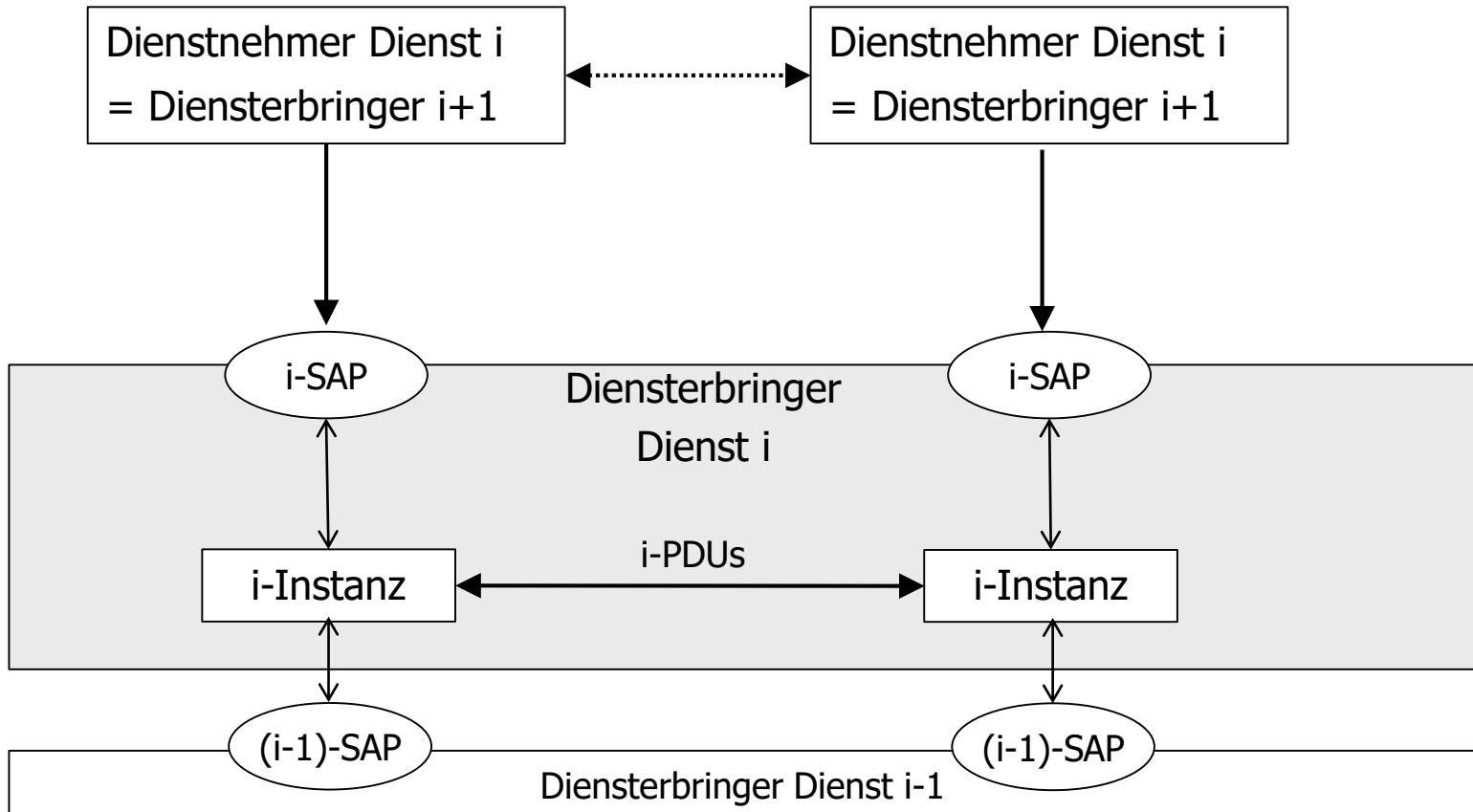


# ISO/OSI-Dienstmodell allgemein

---



# ISO/OSI-Referenzmodell: Hierarchische Dienststruktur (1)



- SAP = Service Access Point, PDU = Protocol Data Unit

## ISO/OSI-Referenzmodell: Hierarchische Dienststruktur (2)

---

- Der **Dienstnehmer** in einer Schicht  $i$  nutzt zur Kommunikation einen **Diensterbringer**  $i$  (Dienstprovider), der wiederum die darunter liegende Schicht  $(i-1)$  nutzt
- Die Dienste werden über **Dienstzugangspunkte** (Service Access Points, SAP) bereitgestellt
- SAPs sind logische Schnittstellen, deren konkrete Realisierung z. B. in Form einer Funktionsbibliothek oder als eigener Systemprozess gegeben sein kann

## ISO/OSI-Referenzmodell: Instanzen

---

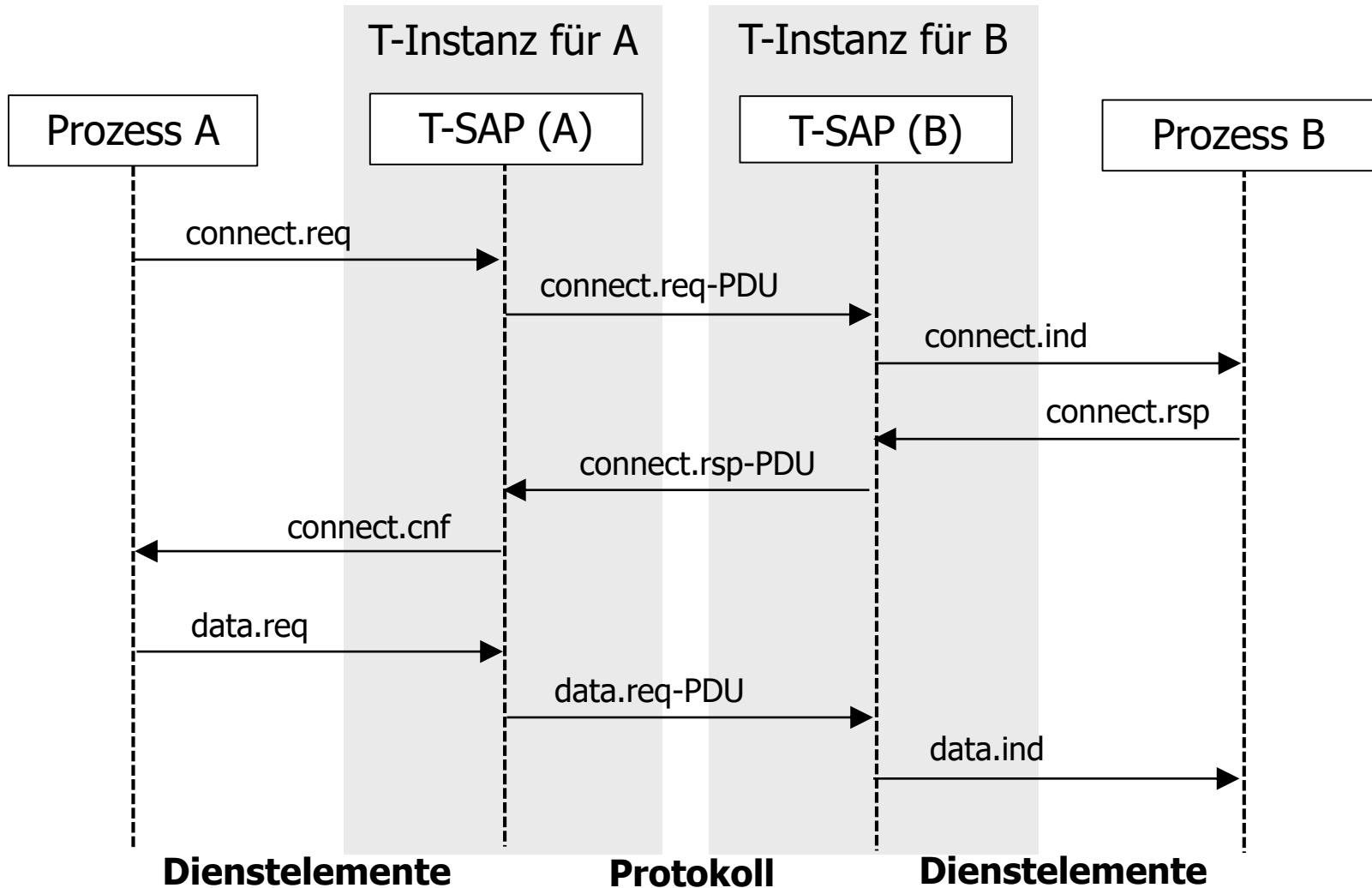
- Die Funktionen innerhalb einer Schicht werden von einer **Instanz** ausgeführt
- Eine Instanz erbringt die Dienstleistung, die ein Dienstnehmer von einem Dienstprovider erwartet
- Instanzen sind auf den kommunizierenden Systemen verteilt
- Instanzen der gleichen Schicht kommunizieren miteinander über Protokolle
- Zwei kommunizierende Instanzen werden als Partnerinstanzen bezeichnet

# ISO/OSI-Referenzmodell: Dienstelemente

---

- Dienste (Abstraktion) im ISO/OSI-Modell
  - Beispiel: connect, disconnect, data (Datenübertragung)
- Dienstelemente/Dienstprimitive sind Operationen eines Dienstes
- Typische Dienstprimitive sind
  - Request
  - Indication
  - Confirmation
  - Response
  - Beispiel: connect.req, connect.ind, data.req

# ISO/OSI-Referenzmodell: Dienste und Protokoll im Zusammenspiel



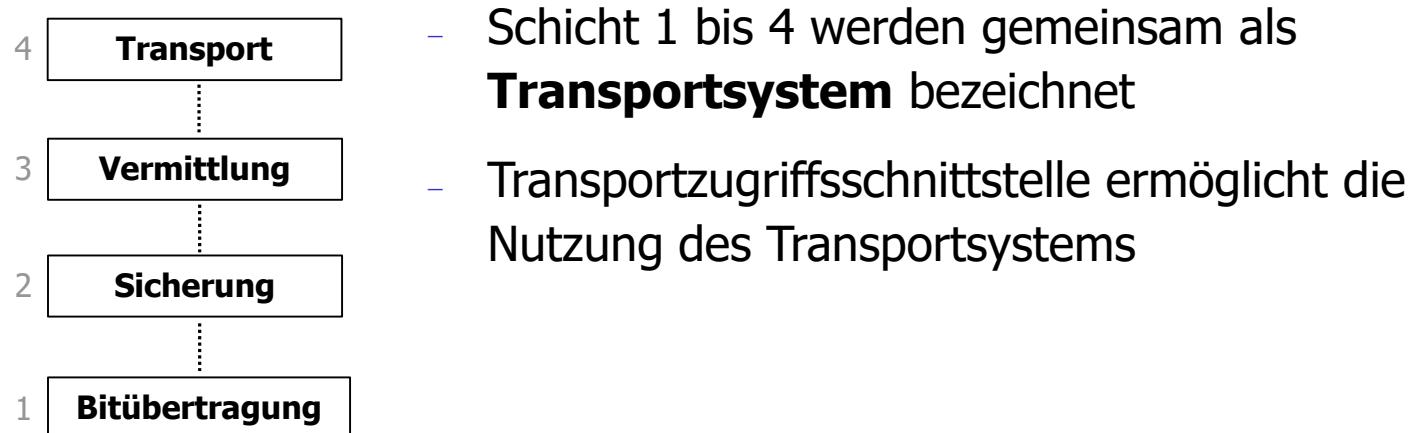
# ISO/OSI-Referenzmodell

## Definition für Transportsystem

---

### ■ **Transportschicht**

- Stellt einen Transportservice bereit
- Ende-zu-Ende-Verbindung zwischen Anwendungsprozessen
- Gesicherte Transportverbindung



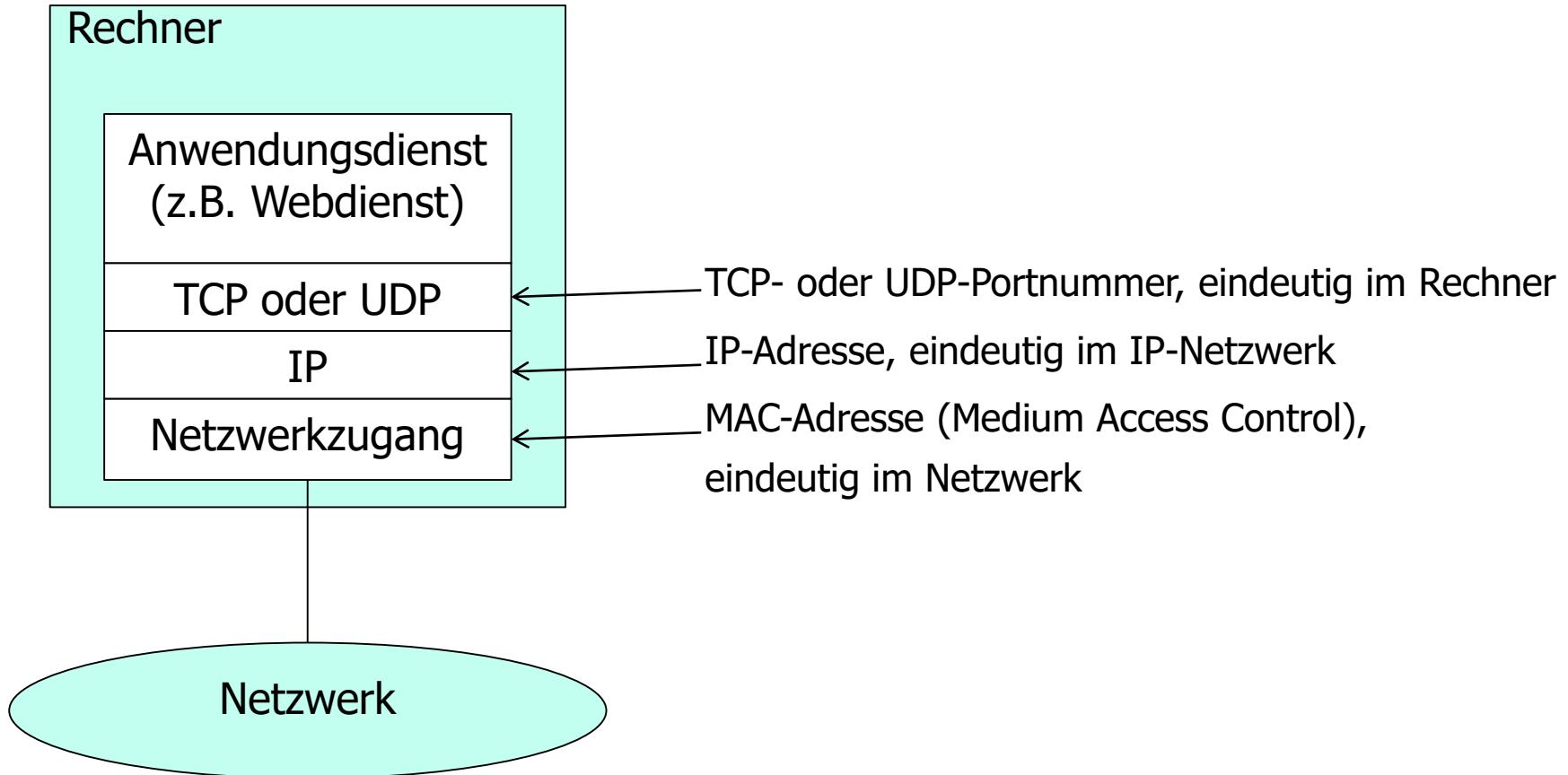
# Adressierung am Beispiel von TCP/IP (1)

---

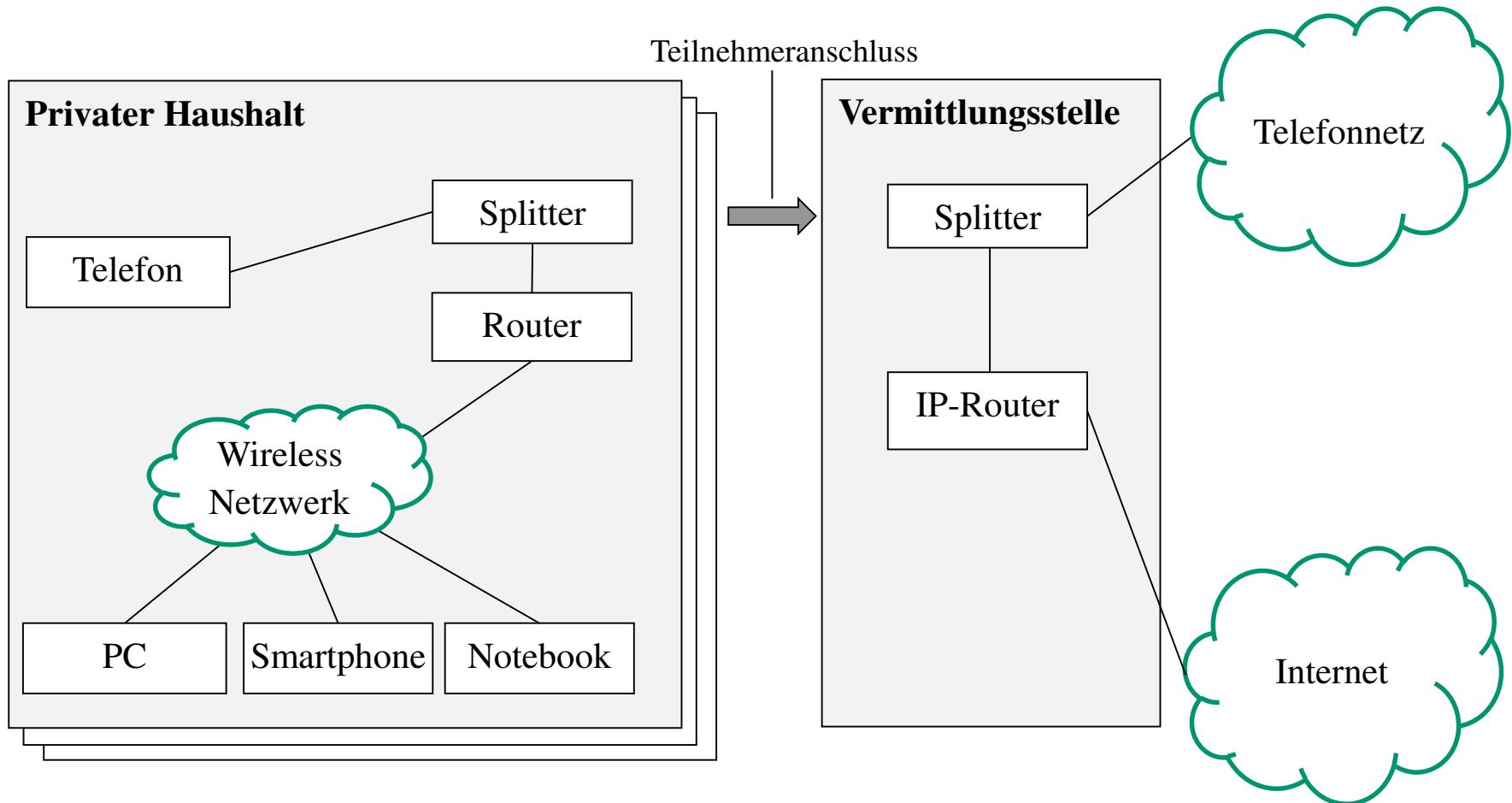
- **Adressierung von Anwendungen (Diensten)**
  - Eindeutige Portnummern (16 Bit lange Integerzahlen)
  - Portnummern sind z.T. fest zugeordnet, z.B. 80 für Web-Kommunikation
- **Rechneradresse = Internetadresse = IP-Adresse**
  - Jeder Netzwerkzugang hat eine IP-Adresse: Bei IPv4 32-Bit lang, z.B. 192.168.10.2
  - Bei IPv6 128 Bit, z.B. 2001:0db8:85a3:08d3:1319:8a2e:0370
- **Adressierung eines Kommunikationsendpunkts durch eine Socket-Adresse**
  - Tupel (IP-Adresse, Portnummer), z.B. (195.168.8.1, 80) = Port 80 auf Rechner mit Netzzugang 195.168.8.1
- **Adressierung des Netzwerkzugangs durch MAC-Adresse**
  - Im LAN zum Beispiel die Ethernet-Adresse, z.B. 00:50:56:c0:00:08

## Adressierung am Beispiel von TCP/IP (2)

---



# Netzwerzugang am Beispiel von Ethernet und DSL



# Überblick

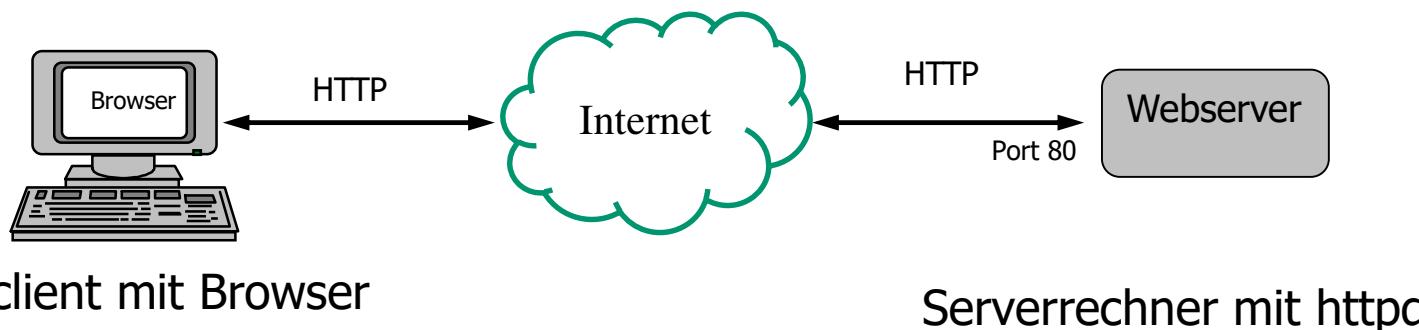
---

- Terminologie und Referenzmodelle
- **Beispiele für Anwendungsprotokolle**
- Chat-Anwendung als begleitende Studie

# Fallbeispiel: Web-Kommunikation: Zusammenspiel

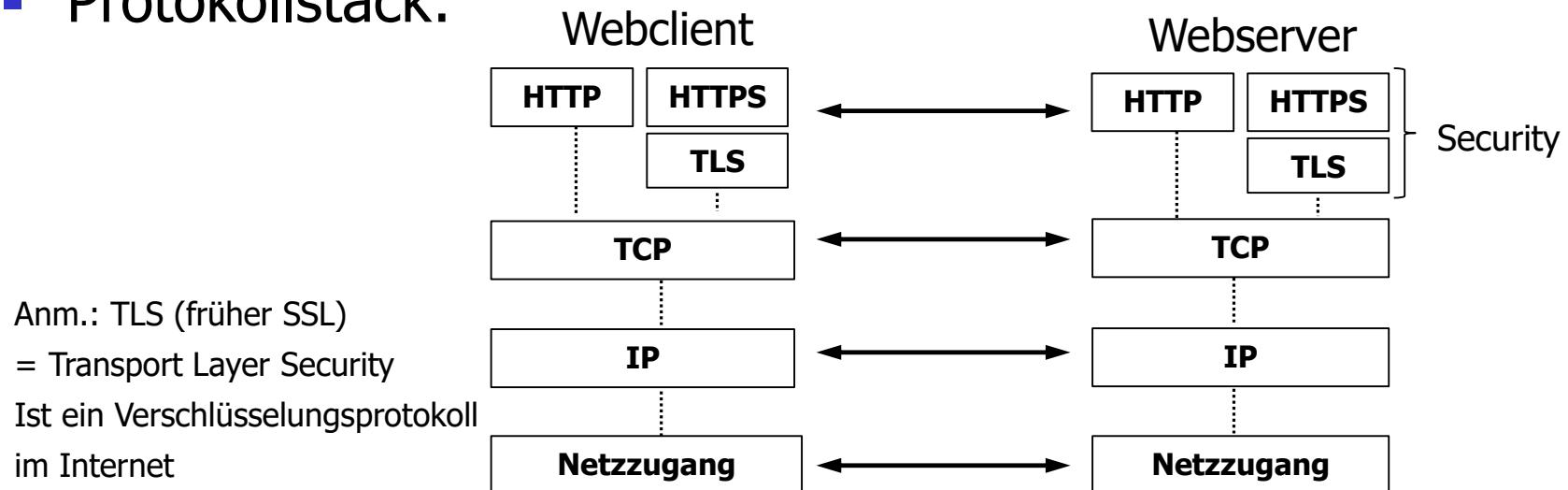
---

- HTML (Hypertext Markup Language) dient als Auszeichnungssprache (Markup Language)
- HTTP (Hypertext Transfer Protocol) auf Basis von TCP/IP als textbasiertes Kommunikationsprotokoll
- Web-Browser auf der Clientseite als grafische Benutzeroberfläche
- Webserver (httpd) auf der Serverseite als Container für Anwendungen



# Fallbeispiel Web-Kommunikation: HTTP-Protokollstack

- HTTP ist ein **verbindungsorientiertes** Protokoll
- Darstellung der zu übertragenden Objekte:
  - HTML oder XML
  - Nutzung von **MIME**-Bezeichnern für content-type (Multipurpose Internet Mail Extensions), z.B. text/html, image/gif
- Protokollstack:



# Fallbeispiel Web-Kommunikation: HTTP-Protokolloperationen

---

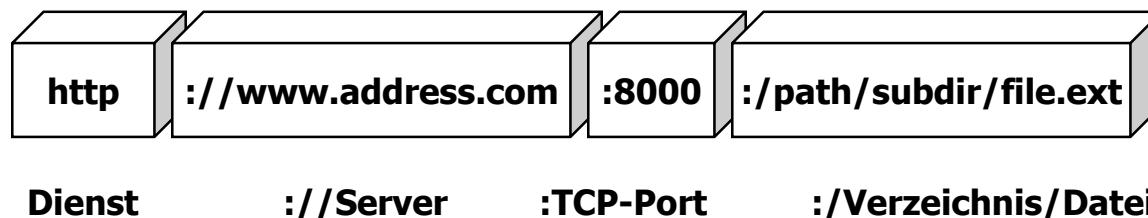
- HTTP ist vom Protokolltyp her ein **zustandsloses Request/Response-Protokoll**
  - Weder Sender noch Empfänger merken sich irgendwelche Stati zur Kommunikation
  - Ein Request ist mit einem Response vollständig abgearbeitet
- **Protokolloperationen** sind z.B. GET, HEAD, PUT, POST, DELETE, ...
  - GET: Lesen einer Ressource mit Antwort
  - POST: Anlegen einer neuen Ressource ohne URI
  - PUT: Erzeugen oder Aktualisieren einer Ressource mit URI
  - HEAD: Lesen einer Ressource ohne Datentransfer, nur Metadaten
  - DELETE: Löschen einer Ressource

# Fallbeispiel Web-Kommunikation: HTTP-Adressierung im WWW

---

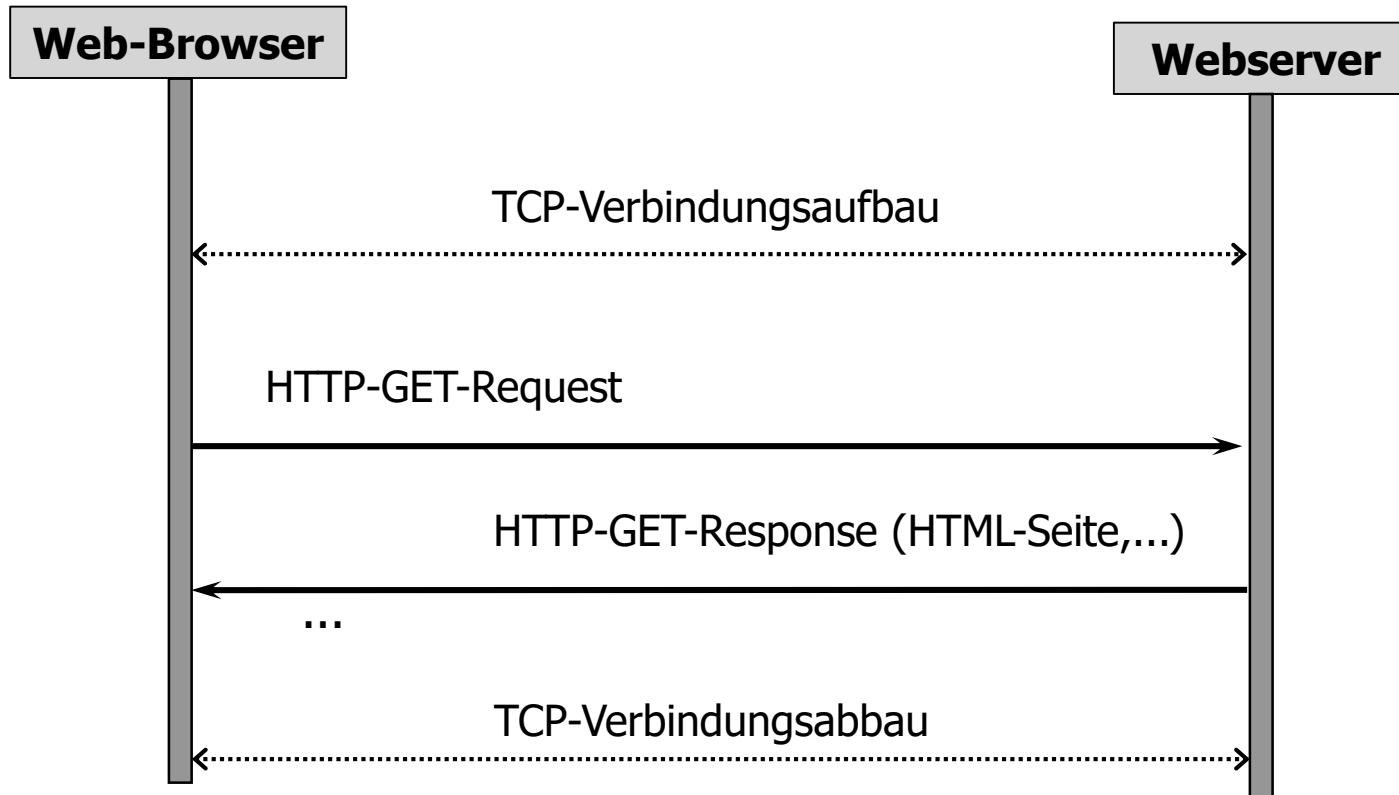
- Die Adressierung über **URIs bzw. URLs**
- Uniform Resource Locator (URL) identifiziert das Dokument
- Uniform Resource Identifier (URI): Allgemeinerer Begriff als Überbegriff für alle Adressierungsmuster, die im WWW unterstützt werden

## URL-Aufbau:



# Fallbeispiel Web-Kommunikation: Ablauf der Kommunikation

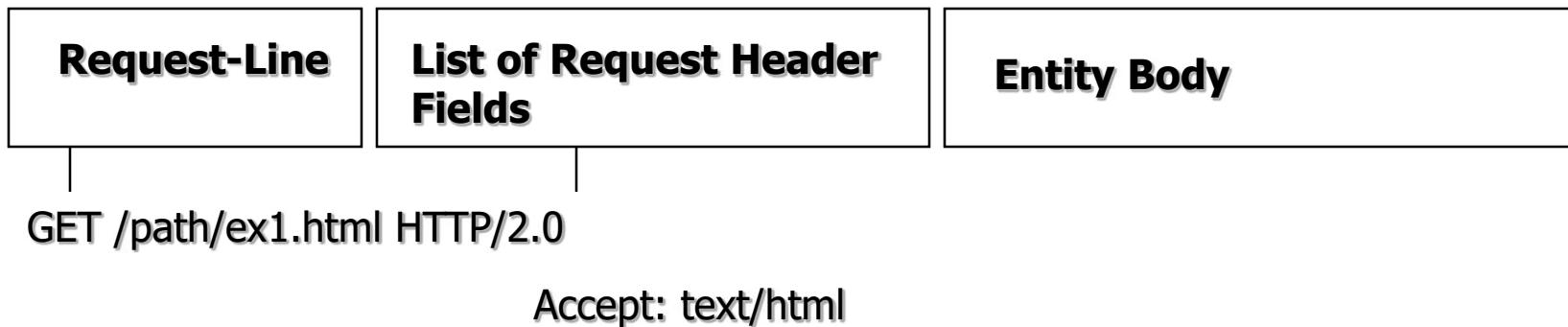
---



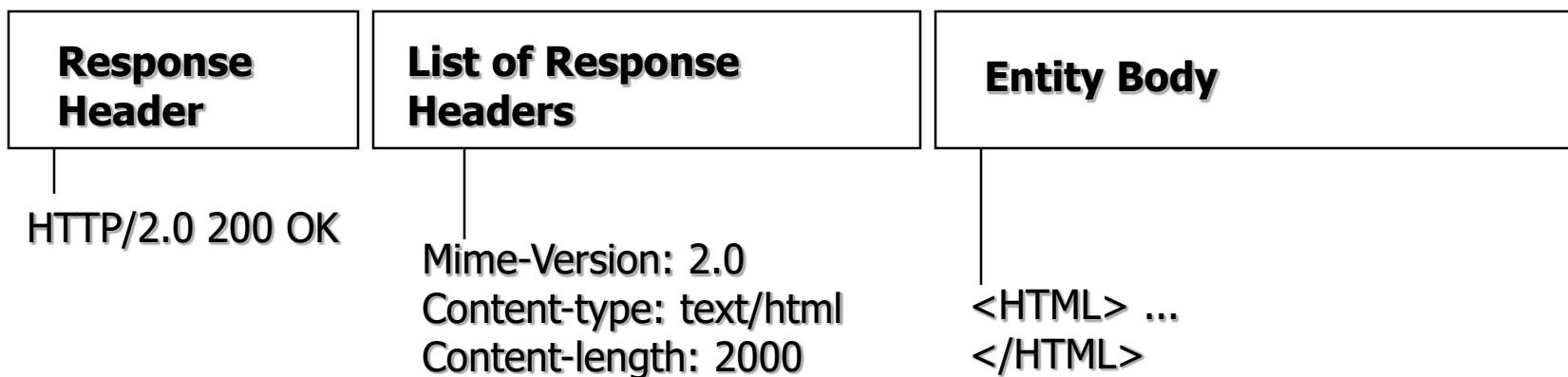
# Fallbeispiel Web-Kommunikation: Aufbau der HTTP-PDUs

---

## HTTP-Request

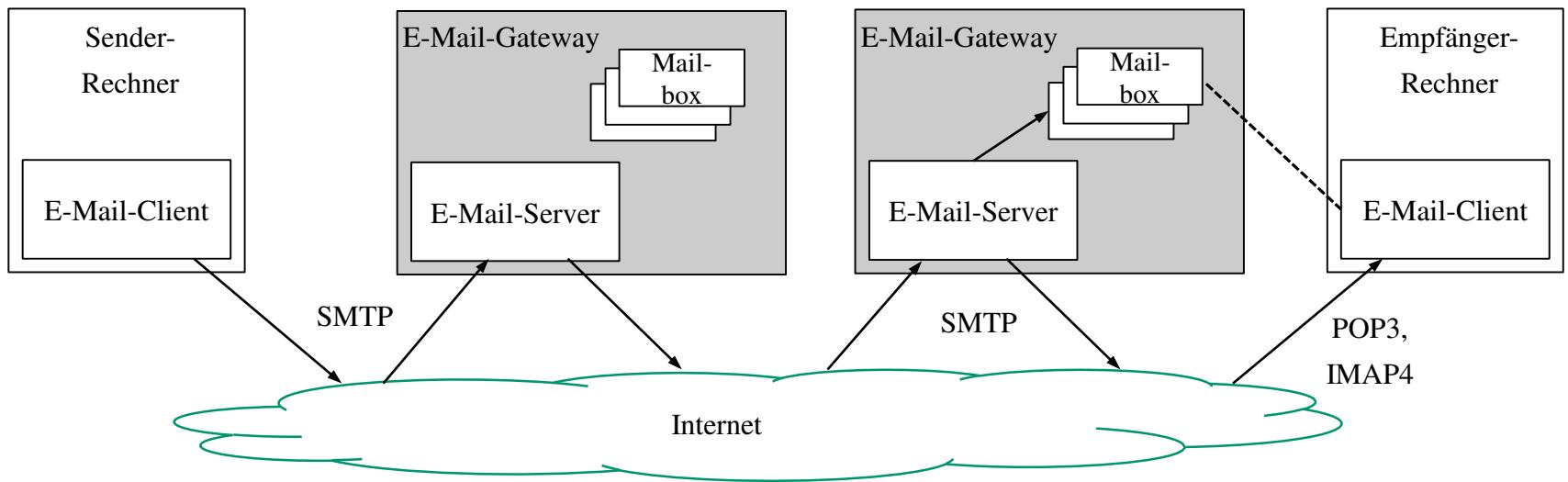


## HTTP-Response



# Fallbeispiel E-Mail-Kommunikation: Architektur

- Austausch von elektronischen Nachrichten
- Jeder E-Mail-Client (*Microsoft Outlook, Mozilla Thunderbird,...*) baut eine TCP-Verbindung zu seinem Mail-Gateway auf (*sendmail, qmail, MS Exchange*), z.B. bei seinem Internet-Provider
- SMTP-Server kommunizieren untereinander



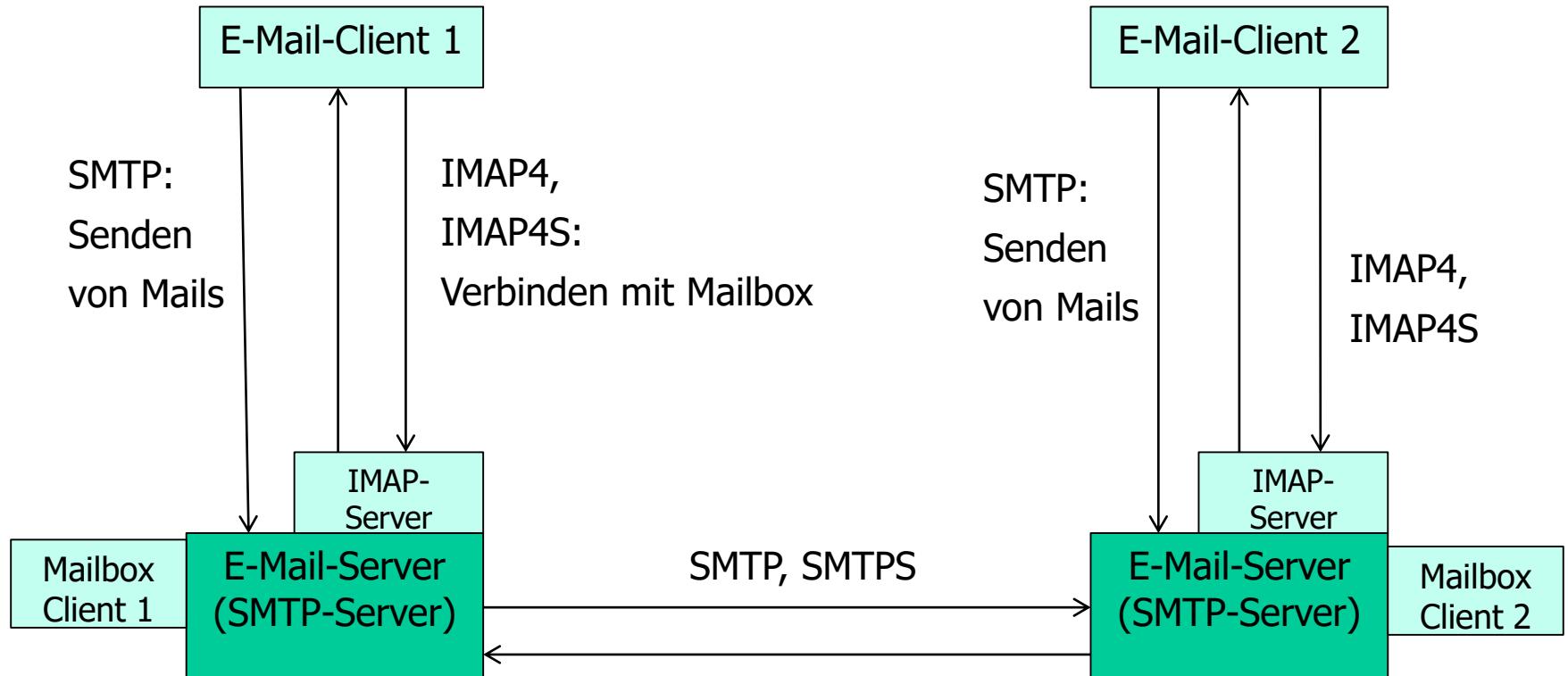
# Fallbeispiel E-Mail-Kommunikation: Grundlegendes

---

- Elektronische Mailbox mit einer eindeutigen E-Mail-Adresse (z.B. mandl@hm.edu) in einem **E-Mail-Gateway**
- Die E-Mail-Gateways kommunizieren untereinander über das Protokoll **SMTP** bzw. **SMPTS** (Simple Mail Transfer Protocol, well-known TCP-Port = 25)
  - Transportfunktionalität wird logisch als Message Transfer Agent (**MTA**) bezeichnet
- E-Mail-Client (Mail User Agent = **MUA**) senden Mails über SMTP/SMTPS zum zuständigen E-Mail-Server (Mail Submission Agent = **MSA**)
- **Mailzugangsprotokolle** (Zugang zur Mailbox)
  - **POP3** bzw. **POP3S** (Post Office Protocol, Version 3, well-known TCP-Port = 110) → für Privatleute
  - **IMAP4** bzw. **IMAP4S** (Internet Message Access Protocol, well-known TCP-Port = 143) → für Unternehmen

# Fallbeispiel E-Mail-Kommunikation: Zusammenspiel Client - Server

---

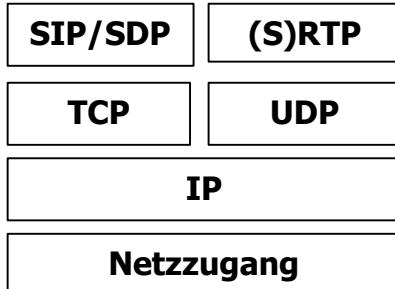


# Fallbeispiel: Internet-Telephonie

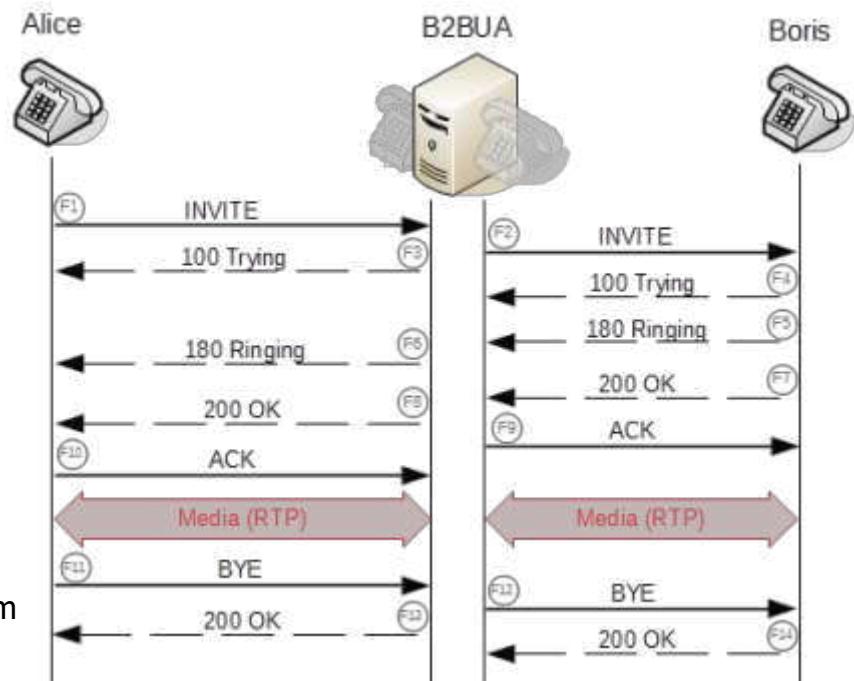
## SIP und RTP

- **SIP = Session Initiation Protocol**, zum Aufbau, zur Steuerung und zum Abbau einer Kommunikationssitzung zwischen zwei und mehr Teilnehmern, RFC 3261

- SIP handelt die Sitzung aus (HTTP-ähnlich)
- SIP nutzt SDP als Session Description Protocol zum Aushandeln der Medien-Details, SIP nutzt auch TCP
- SIP nutzt RTP (Realtime Transport Protocol) für den Medienstrom
- RTP nutzt wiederum UDP
- SRTP mit Verschlüsselung



<https://upload.wikimedia.org/wikipedia/commons/e/e2/SIP-B2BUA-call-flow.png>



# Fallbeispiel: Instant (Multimedia) Messaging: Überblick

---

- Bei **Instant Messaging (IM)** tauschen Personen Nachrichten über einen gemeinsamen Kanal aus
- Bei **Instant Multimedia Messaging (IMM)** werden auch multimediale Objekte (Bilder, Videos, Audio) ausgetauscht
- Varianten:
  - Zwei Personen kommunizieren direkt miteinander (siehe WhatsApp (gehört seit 2014 Facebook))
  - Eine Gruppe kommuniziert zu einem bestimmten Topic miteinander (siehe WhatsApp-Gruppen)

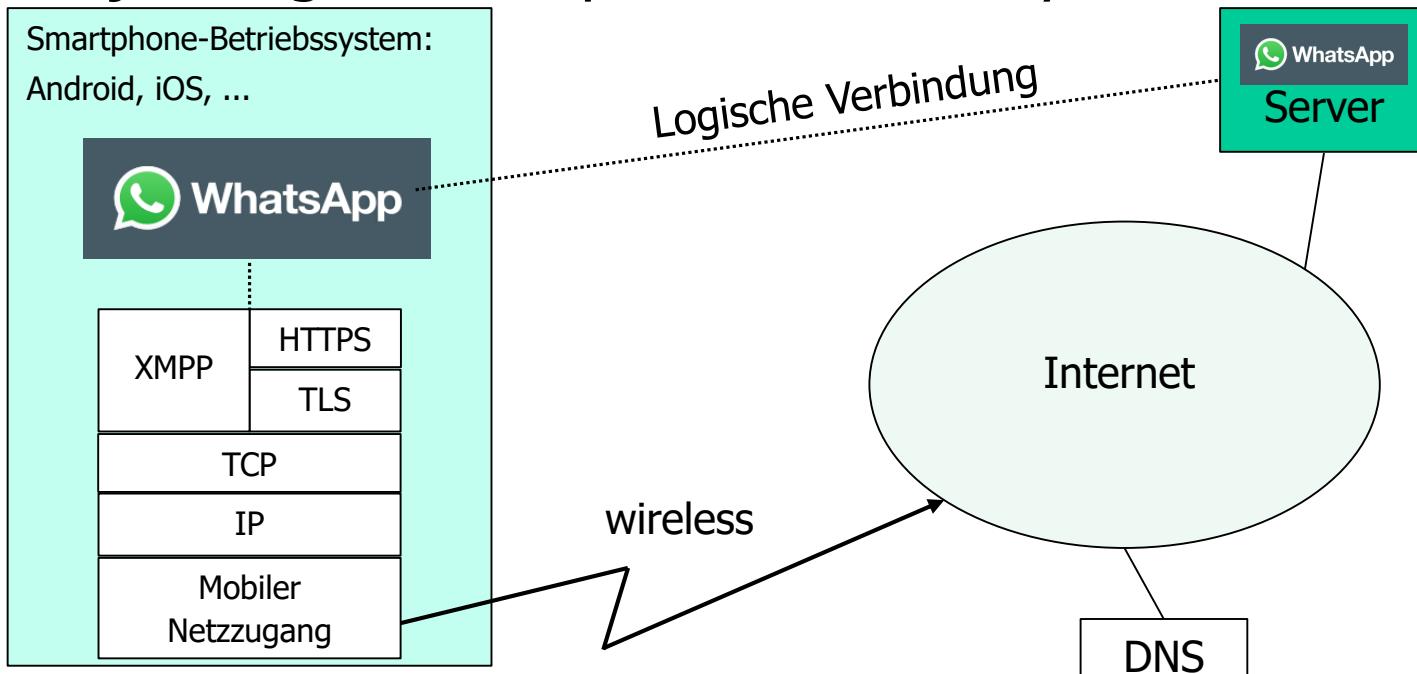
# Fallbeispiel Instant Multimedia Messaging: WhatsApp

---

- Kommunikation über Smartphone
  - Adressierung über Handynummer bzw. WhatsApp-Identifikation
  - Textnachrichten, Sprachnachrichten, Fotos, Videos, Kontakte, Standortinformation an einzelne Empfänger oder an Gruppen
  - Vorsicht:
    - Adressbuch wird an den Server von WhatsApp übertragen, sonst geht es nicht wirklich
    - Zugriff auf Dateien, Fotos, SMS, Kamera, Mikrofon, Standort, ...
    - Patriot Act in USA: US-amerikanisches Bundesgesetz aus 2001, das das Abhören jeglicher Kommunikationsmittel von Terror-Verdächtigen erlaubt
    - Verstößt gegen das europäische / deutsche Datenschutzgesetz

# Fallbeispiel Instant Multimedia Messaging: WhatsApp, Smartphone-Einbettung

- Smartphone-Kommunikationssystem wird genutzt
- Nachrichten-Notifikation geht über den Push-Dienst des jeweiligen Smartphone-Betriebssystem

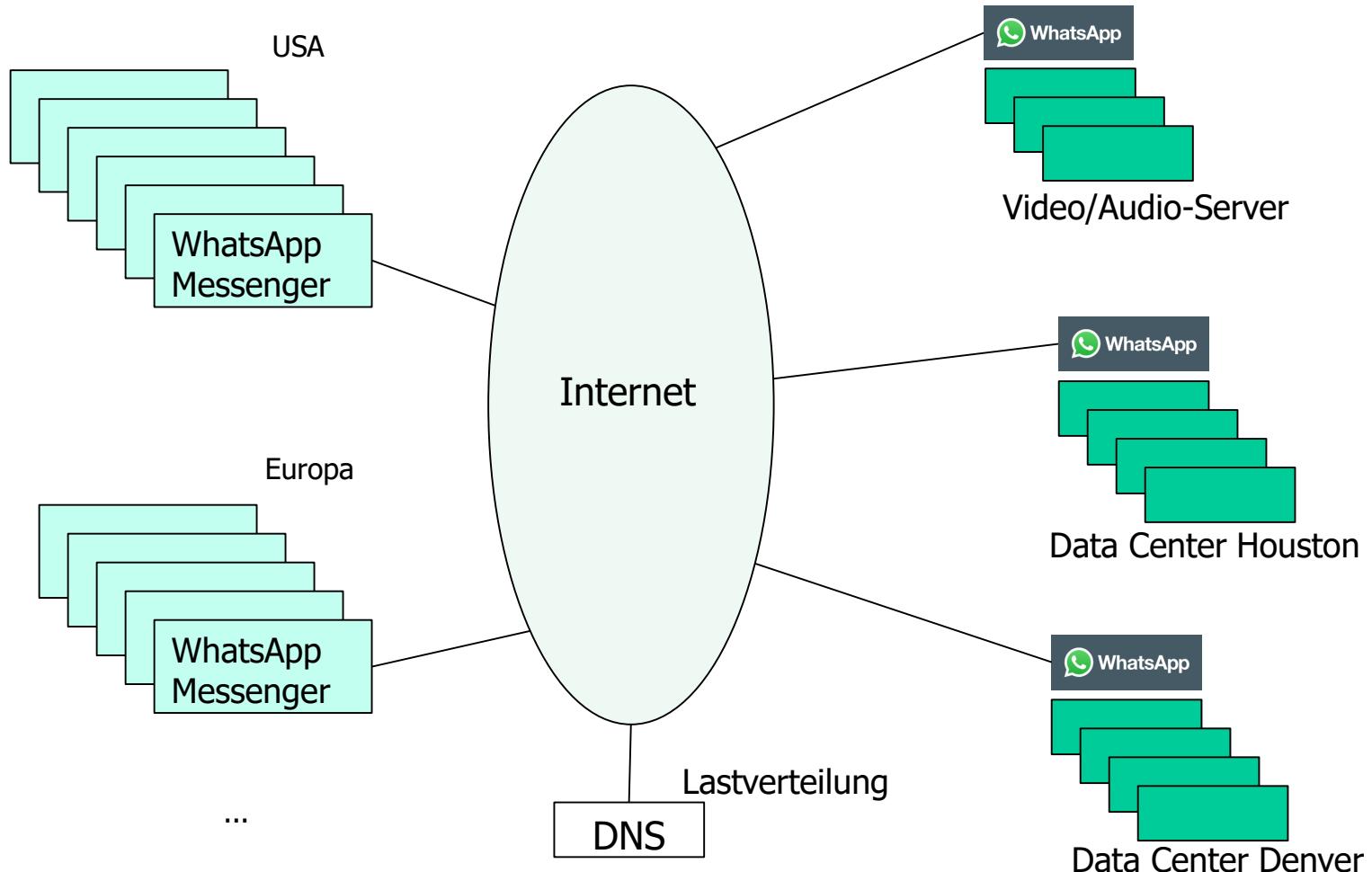


HTTPS = HyperText Transfer Protocol Secure

TLS = Transport Security Layer, Verschlüsselungsprotokoll zur Internetkommunikation

XMPP = Extensible Messaging and Presence Protocol

# Fallbeispiel Instant Multimedia Messaging: WhatsApp, Verteilte Architektur (grob)



DNS: Domain Name System, ist für sich  
wieder ein weltweit verteiltes System, später mehr

## Exkurs (1):

### WhatsApp-Daten (Stand: 2014)

---

- WhatsApp-Serverfarm ist aktuell auf wenige US-Standorte (Houston, Denver) verteilt
- Ein Internet Service Provider wird verwendet:  
**SoftLayer** (vor der Facebook-Übernahme), Autonomes System, Nr. 36351 (später mehr)
- Spezielle HTTP-Server werden für Video und Audio verwendet
- 500 Millionen Benutzer weltweit
- Nachrichtenaufkommen:
  - 64 Milliarden Nachrichten pro Tag, davon
    - 700 Millionen Fotos
    - 100 Millionen Videos

Quelle: Fiadino, F. et al.: Vivisecting WhatsApp in Cellular Networks: Servers, Flows, and Quality of Experience. IFIP 2015. DOI: 10.1007/978-3-319-17172-2\_4.

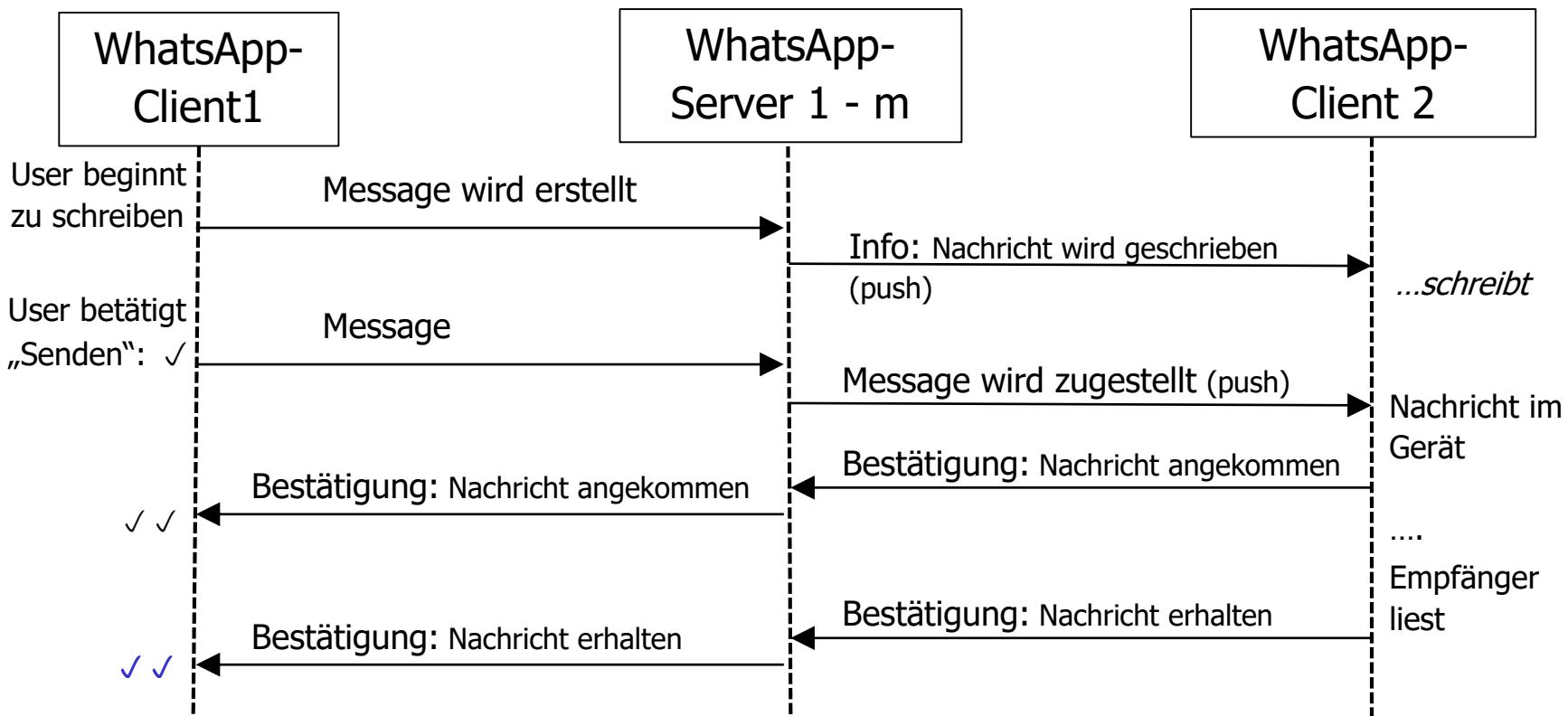
# Fallbeispiel Instant Multimedia Messaging: WhatsApp, Kommunikation und Bestätigungsverfahren

---

- Kommunikation über TCP
  - Die TCP-Verbindung zwischen WhatsApp-Messenger auf dem Smartphone und WhatsApp-Server bleibt bestehen bzw. wird bei Abbruch wieder aufgebaut
- Bestätigungen bei WhatsApp-Nachrichten
  - Sender schreibt seine Nachricht: „*schreibt*“
  - Nachricht erfolgreich versendet → Häckchen: ✓
  - Nachricht auf dem Gerät des Empfängers angekommen → 2 Häckchen: ✓ ✓
  - Der Empfänger hat die Nachricht erhalten: 2 blaue Häckchen: ✓ ✓

# Fallbeispiel Instant Multimedia Messaging: WhatsApp, Typischer Nachrichtenfluss

- Grober Nachrichtenfluss, genauer Nachrichtenaufbau nicht veröffentlicht, Protokoll XMPP angepasst.



## Weitere verteilte Anwendungen und dazugehörige Protokolle

---

- Filetransfer: FTP
- Netzwerkdateisysteme: NFS
- Voice over IP: Skype
- Netzwerkmanagement: SNMP
- Remote Login: telnet, ssh
- Stark verteilte Internet-Dienste:
  - YouTube
  - Facebook
  - DropBox
  - Google+
  - Amazon Web Services, ...

# Überblick

---

- Terminologie und Referenzmodelle
- Beispiele für Anwendungsprotokolle
- **Chat-Anwendung als begleitende Studie**

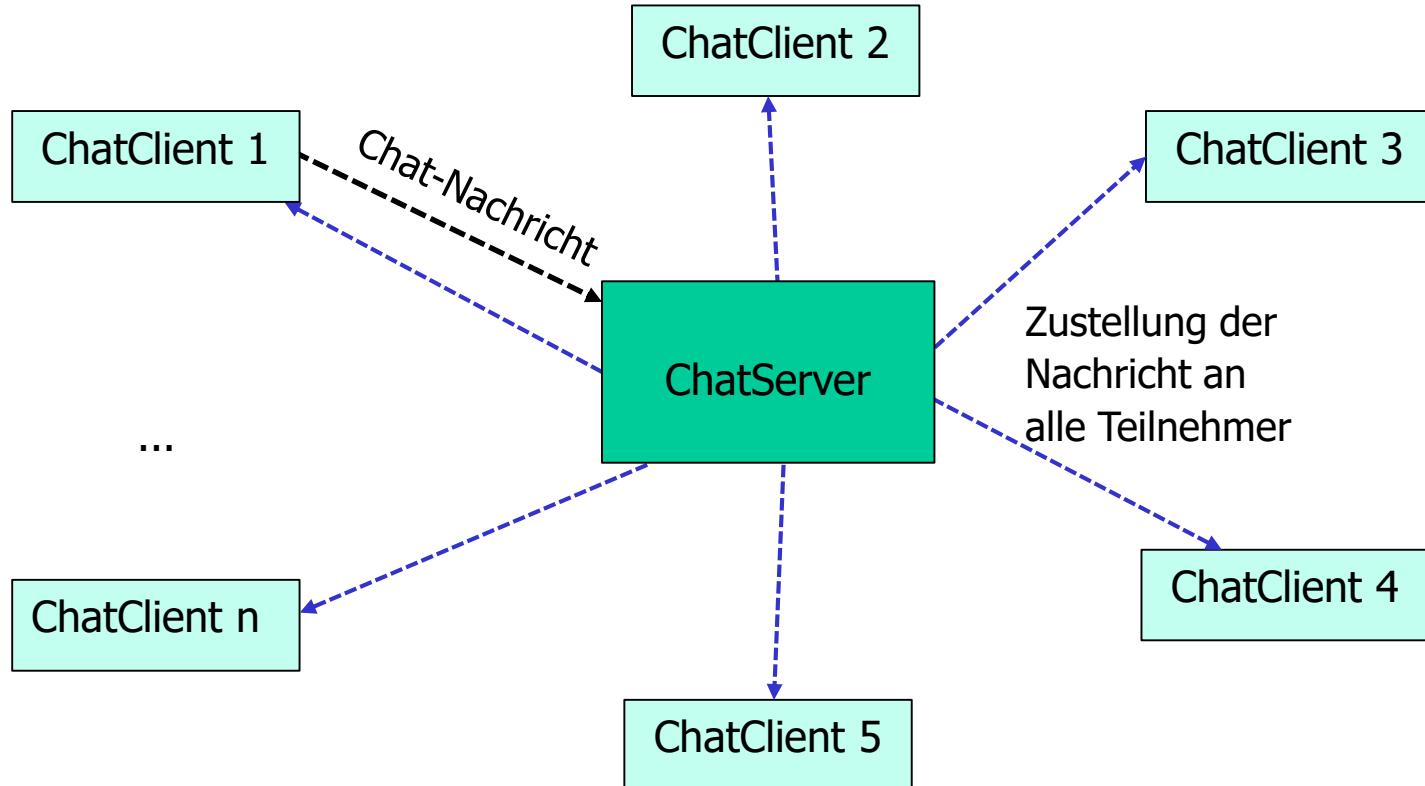
## Unsere Chat-Anwendung

---

- Implementierung einer einfachen Chat-Anwendung als vereinfachtes WhatsApp
- Teilnehmer meldet sich an und sendet Nachrichten über Chat-Client-Anwendung = Messenger
- Chat-Serveranwendung verteilt die Nachrichten an alle aktuellen Teilnehmer (wie WhatsApp-Server)
- Zwei Protokollvarianten:
  - SimpleChat: Verteilung ohne logische Bestätigung
  - AdvancedChat: Verteilung mit logischer Bestätigung

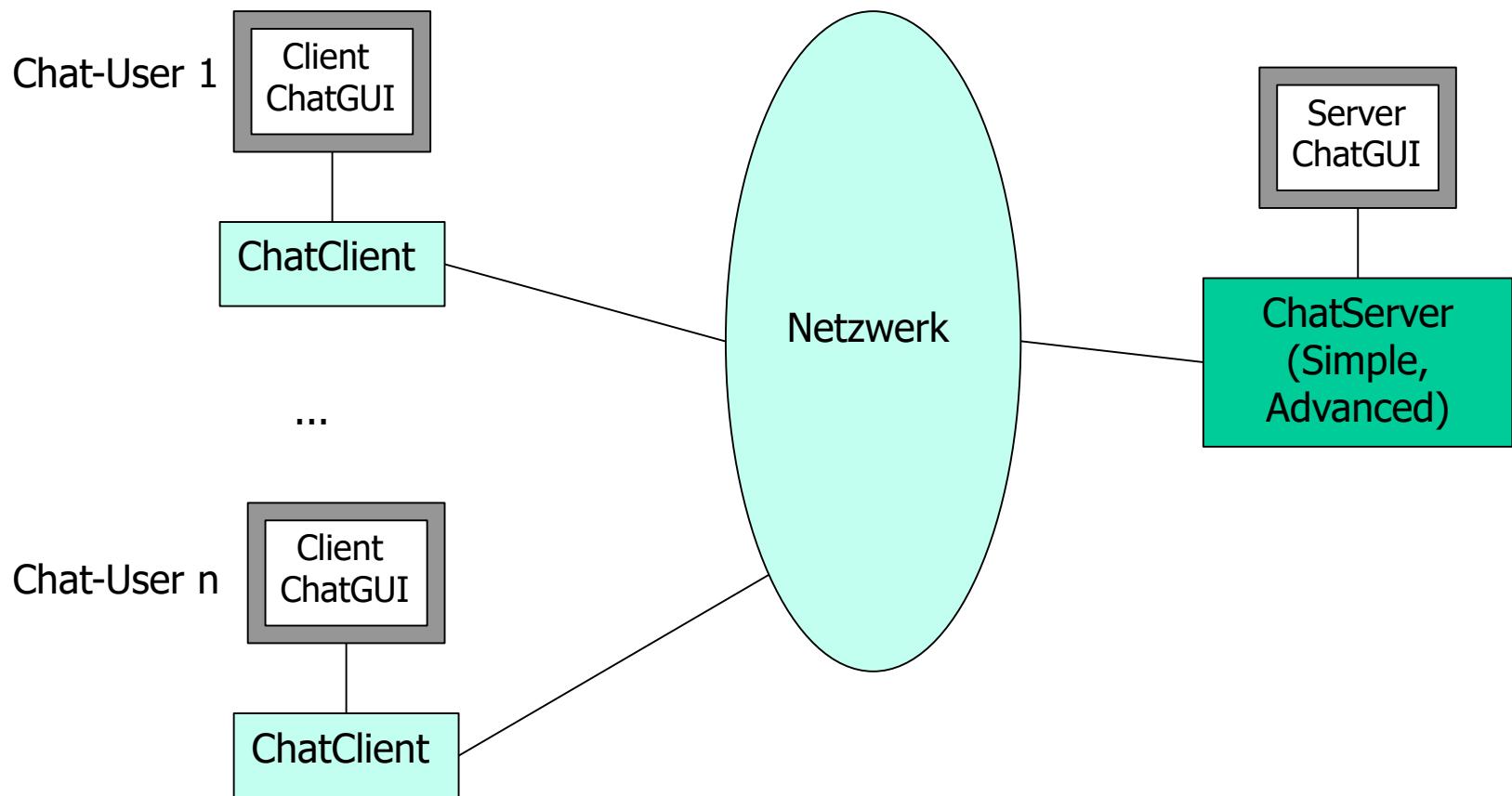
# Chat-Anwendung

- Chat-Client sendet Nachricht an alle angemeldeten Teilnehmer über einen Chat-Server

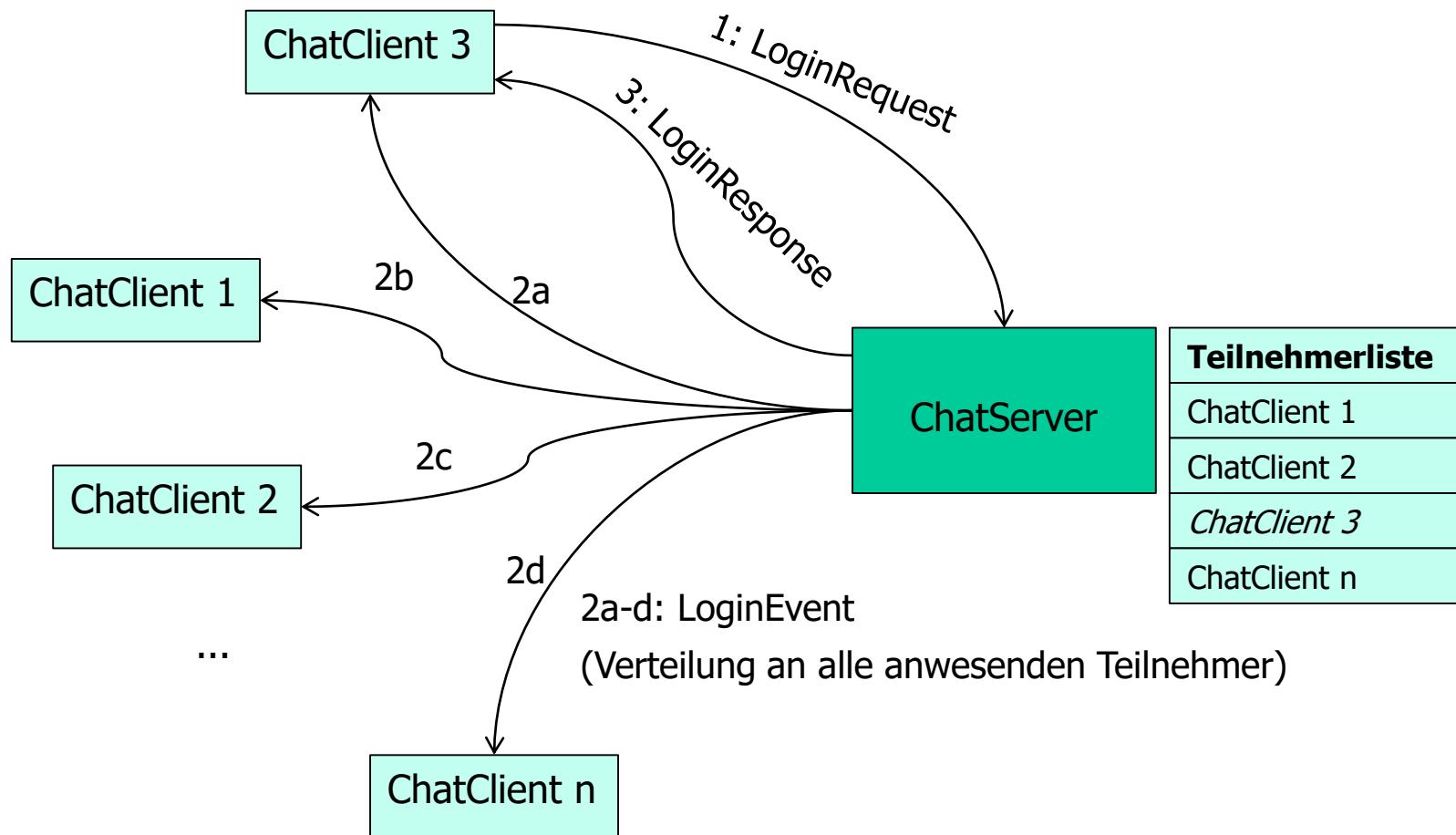


# Chat-Clients und Chat-Server: User-Interface für Client und Server

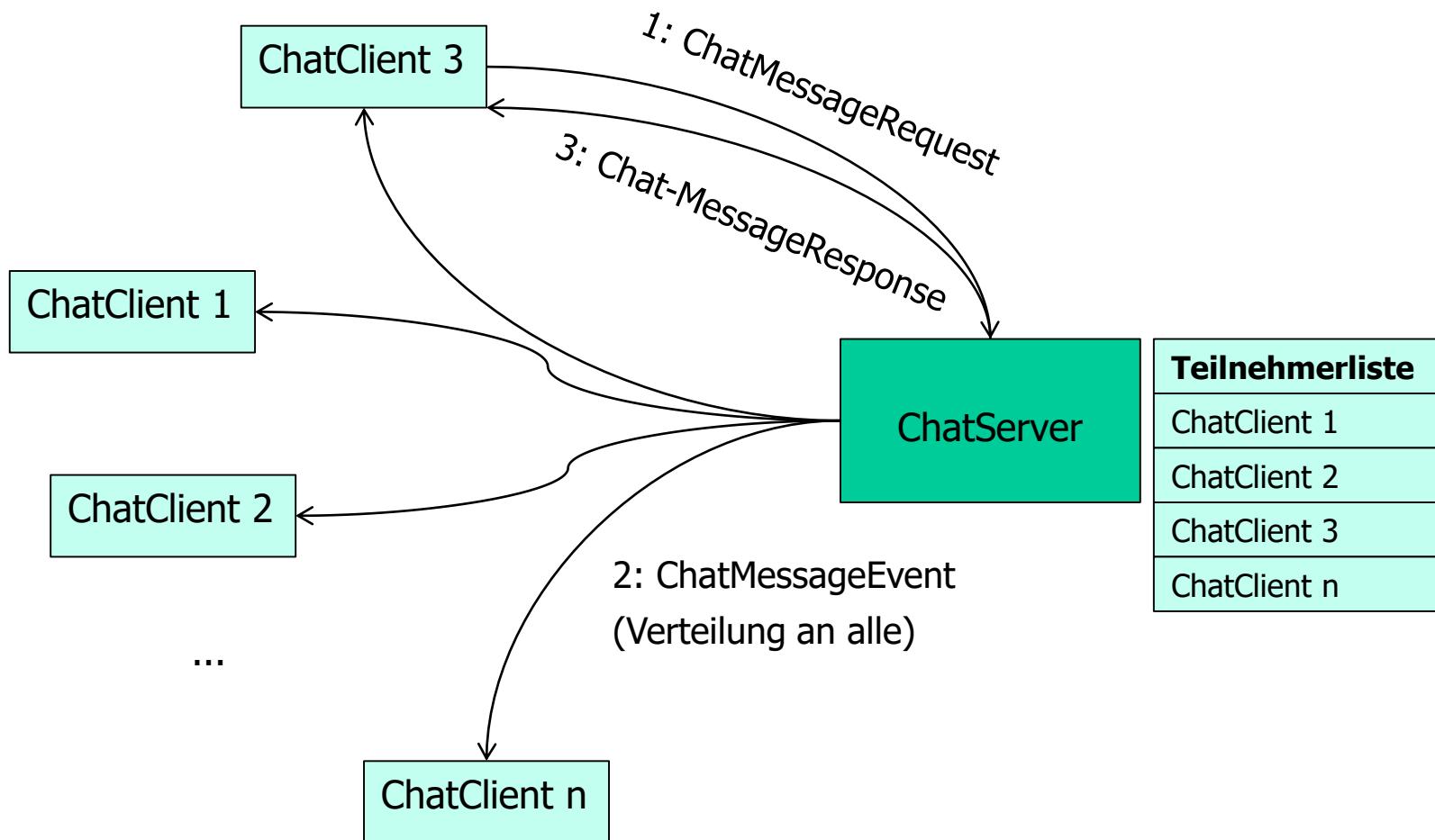
---



# Zusammenspiel von Chat-Client und Chat-Server: SimpleChat, Login-Phase



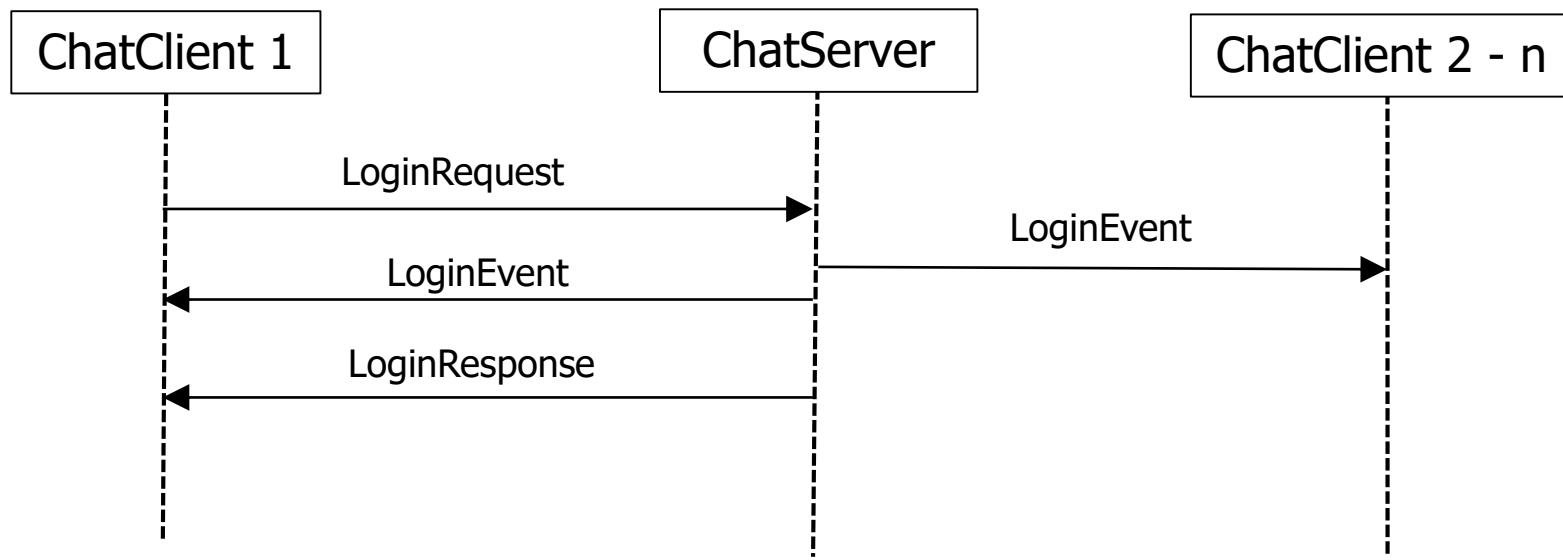
# Zusammenspiel von Chat-Client und Chat-Server: SimpleChat, Chat-Phase



$n$  Clients  $\rightarrow$   $n+2$  Nachrichten werden zur Übertragung einer Chat-Nachricht gesendet

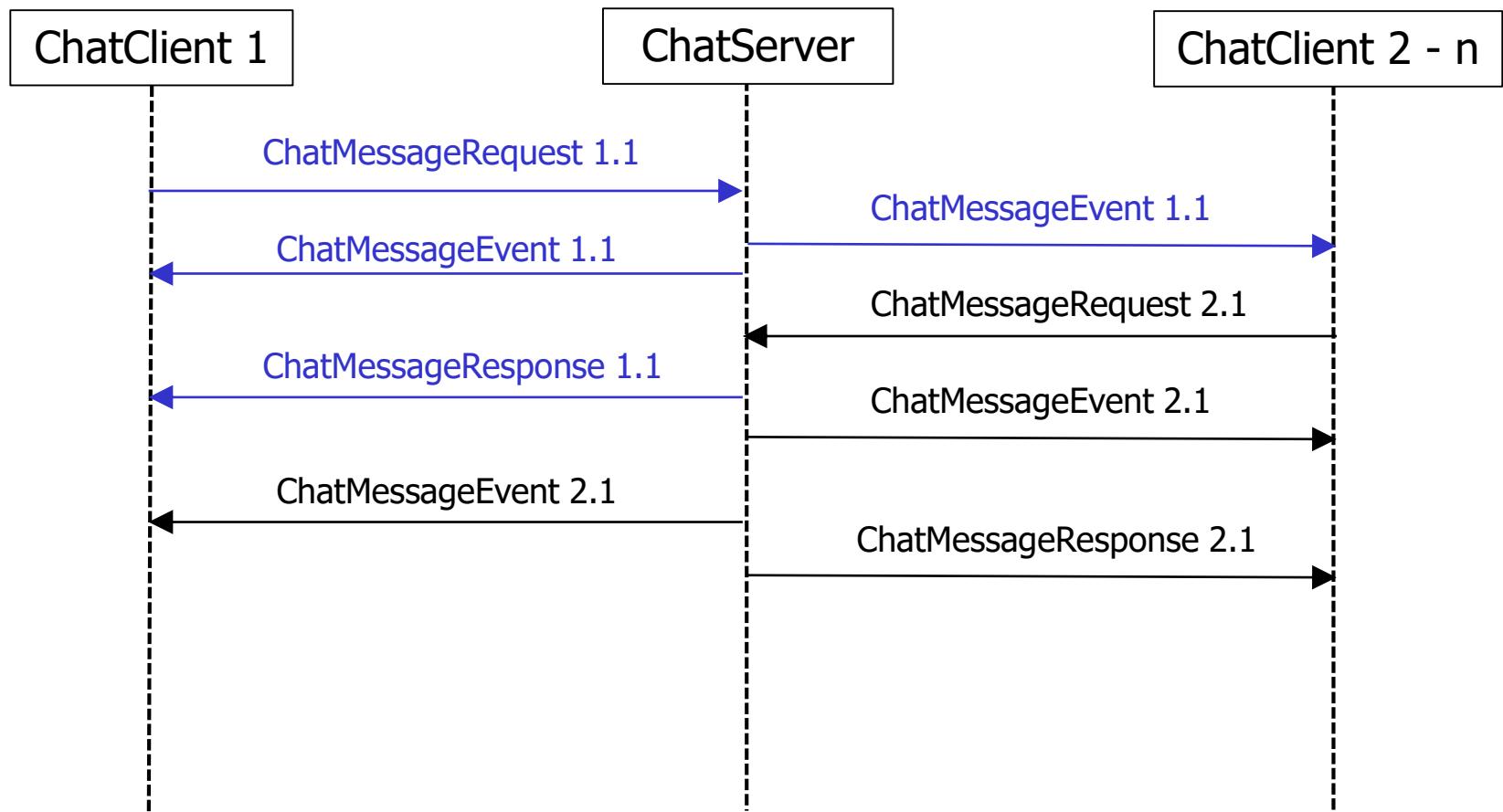
# Protokollablauf SimpleChat: Login-Phase

---



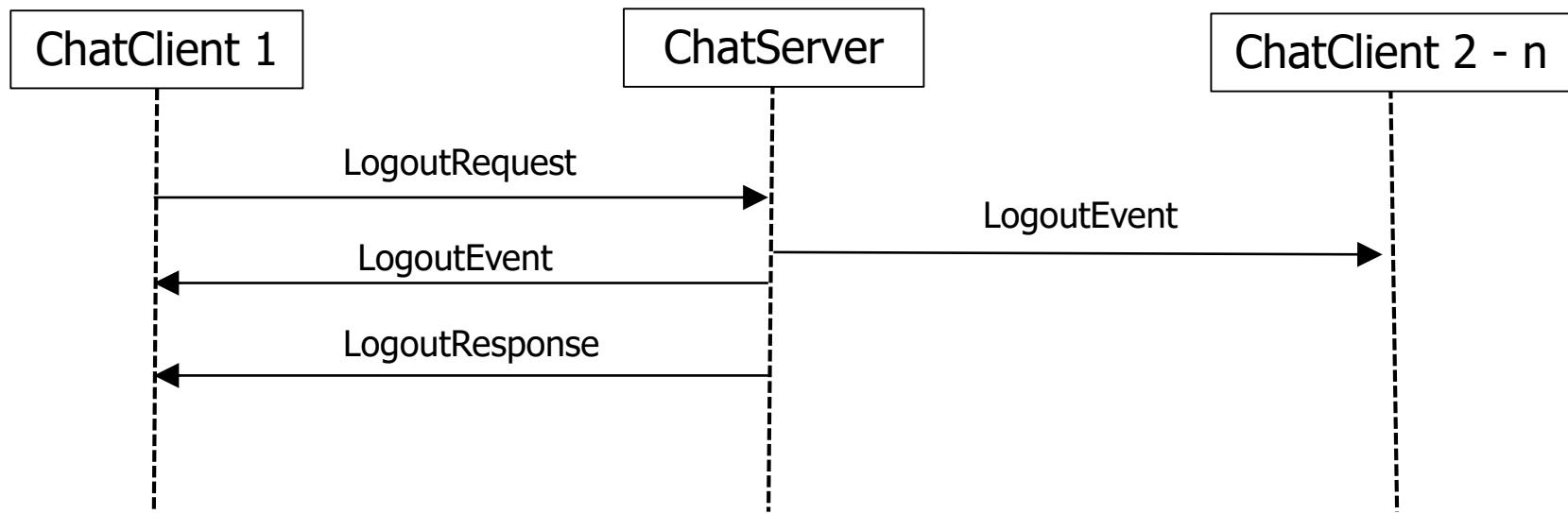
# Protokollablauf SimpleChat: Send-Phase

---



# Protokollablauf SimpleChat: Logout-Phase

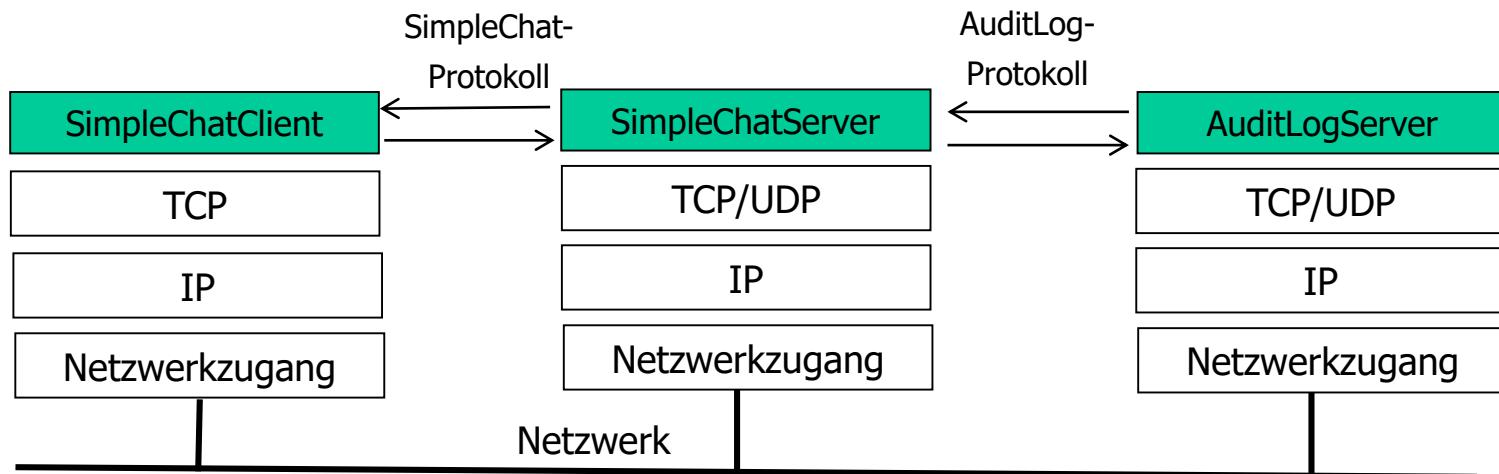
---



# Aufgabenstellung der Studienarbeit (1)

## ■ Erweiterung der Anwendung um einen AuditLog-Server

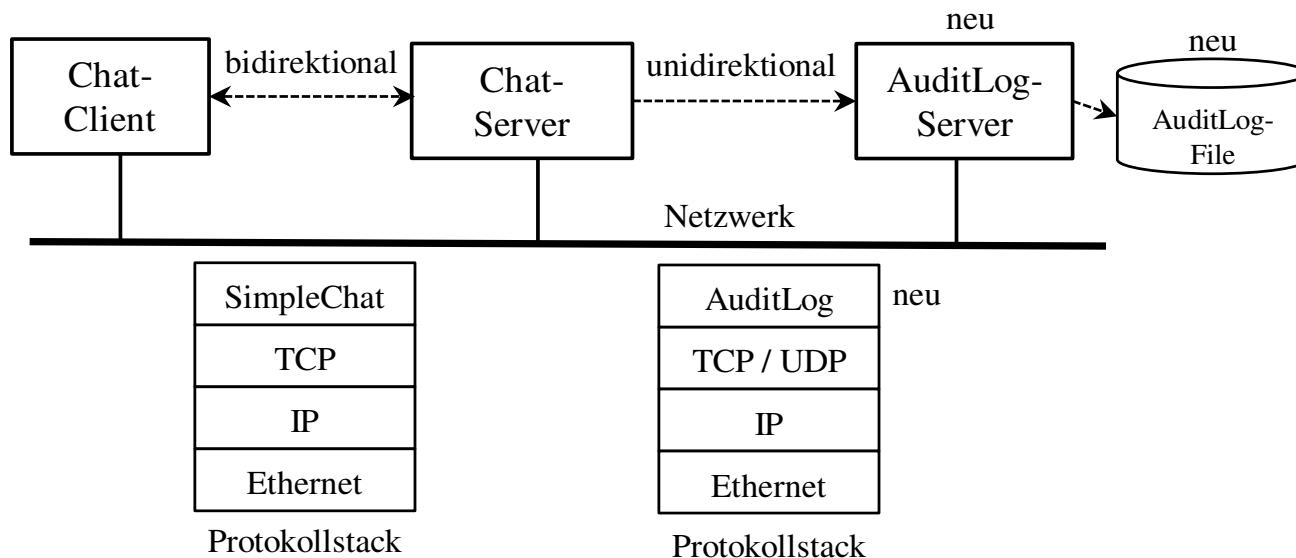
- Im Chat-Server soll jede Nachricht am Ende der Bearbeitung noch an einen AuditLog-Server gesendet werden, der alles in einem AuditLog-File protokolliert
- Implementierung mit TCP und mit UDP und Vergleich
- Näheres siehe Aufgabenstellung zur Studienarbeit



# Aufgabenstellung der Studienarbeit (2)

- **AuditLog-Protokoll ist ein Anwendungsprotokoll**

- Protokollstack wird variiert (TCP und UDP)
- Unidirektionale Kommunikation vom Chat-Server (Client) zum AuditLog-Server

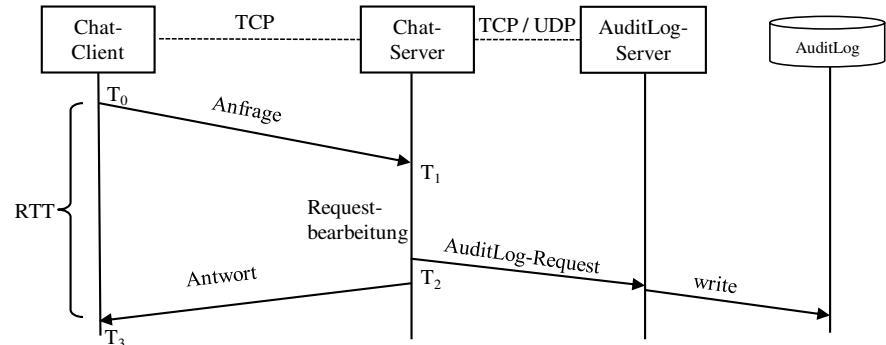


# Metriken zur Leistungsbewertung eines Protokolls: RTT

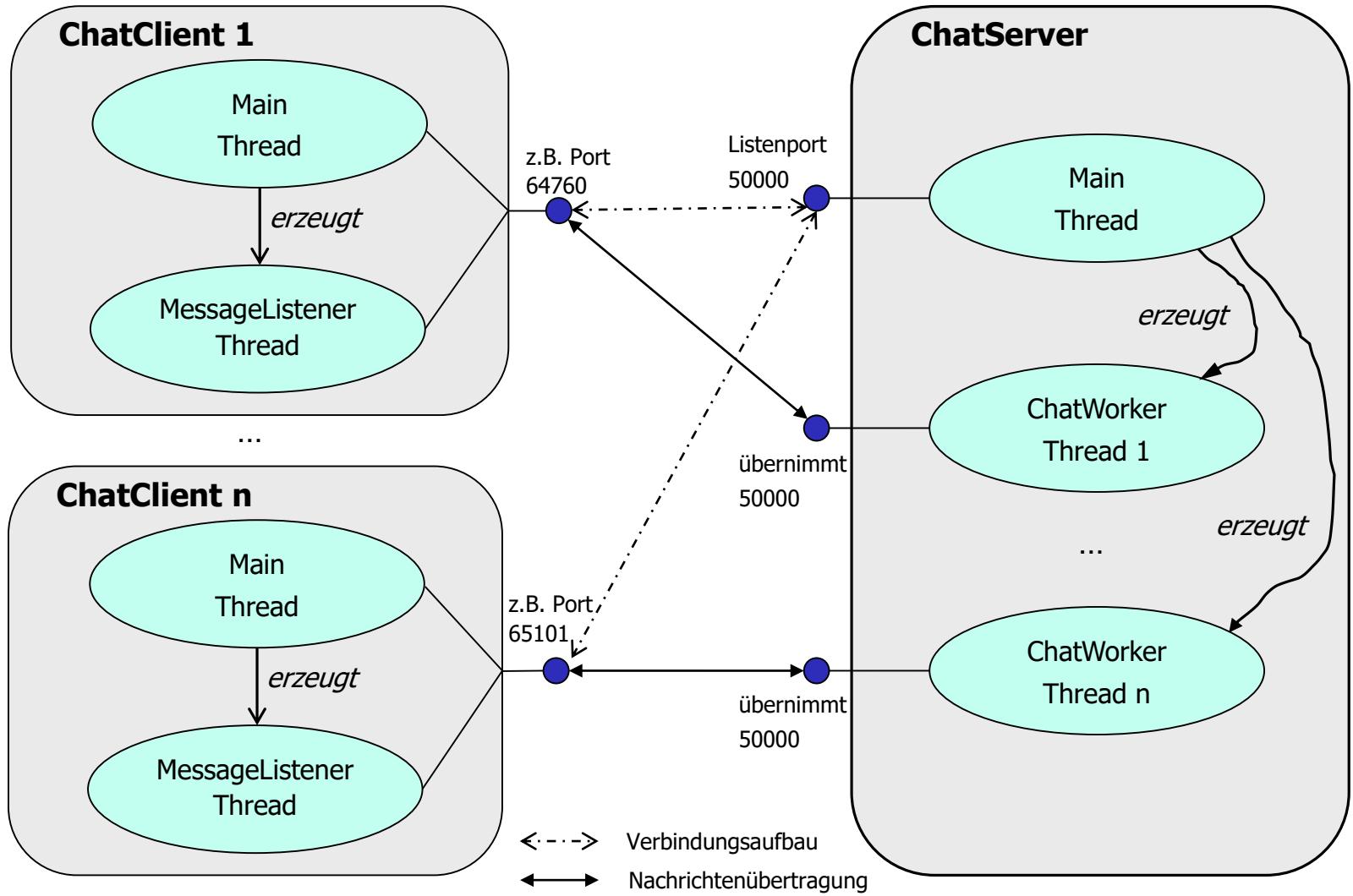
---

## ■ Definition:

- Unter der Round-Trip-Time (RTT) versteht man die Reaktions- oder Bearbeitungszeit eines Anwendungssystems für eine Anfrage
- RTT bezeichnet die Zeitspanne, die erforderlich ist, um eine Nachricht bzw. einen Request von einem Sender zu einem Empfänger zu senden und die Antwort (den Response) des Empfängers wieder im Sender zu empfangen.
- Messung in Millisekunden
- $RTT = T_3 - T_0$

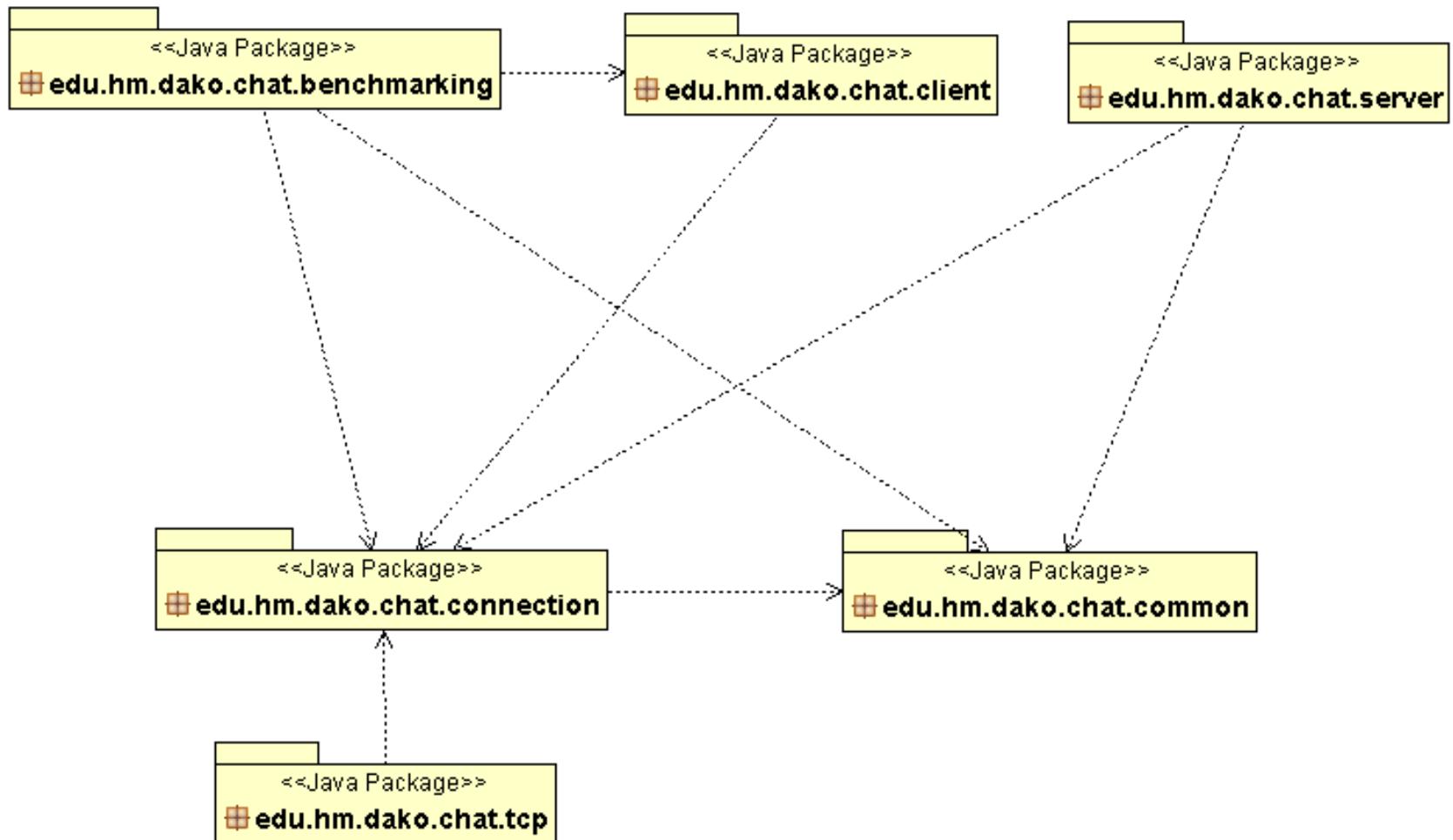


# Threadmodell

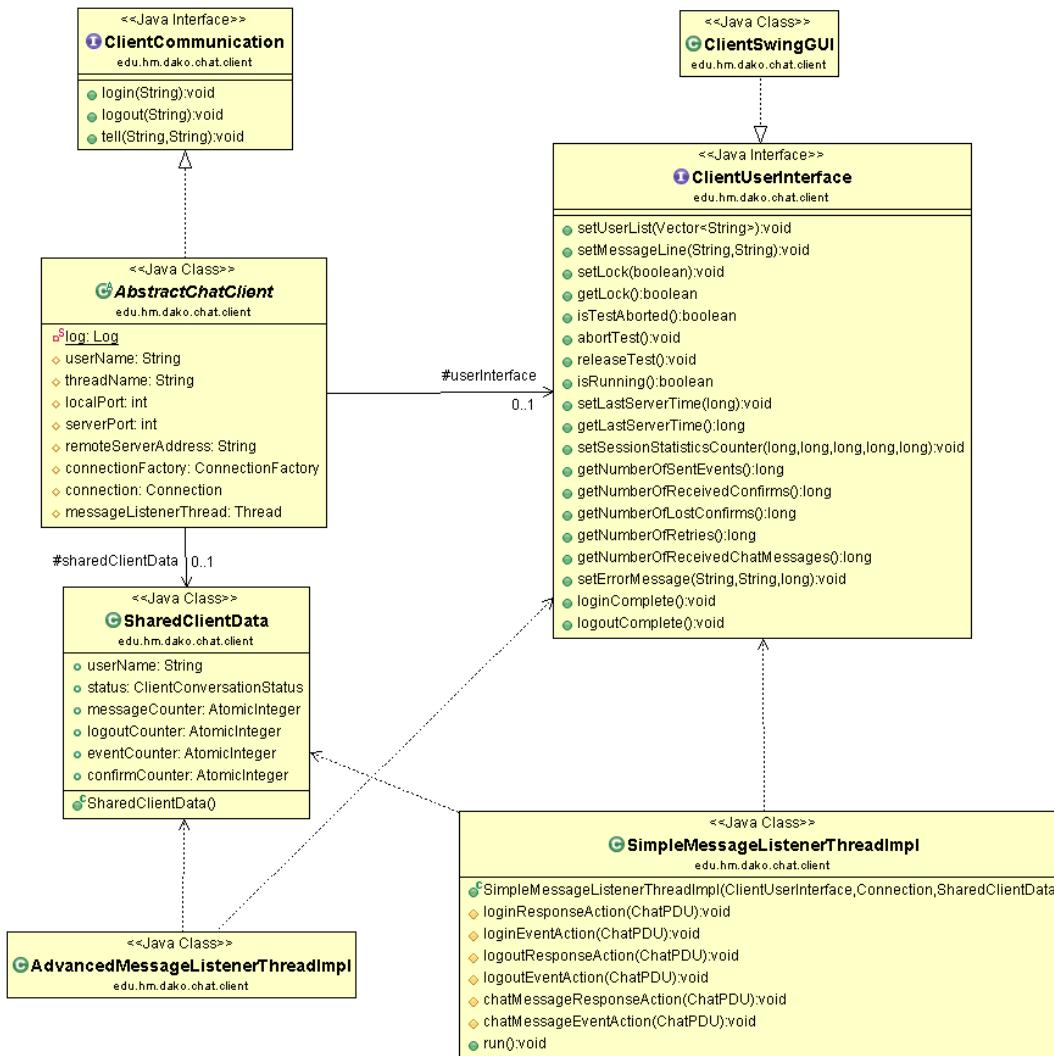


# Package-Struktur des Java-Projekts

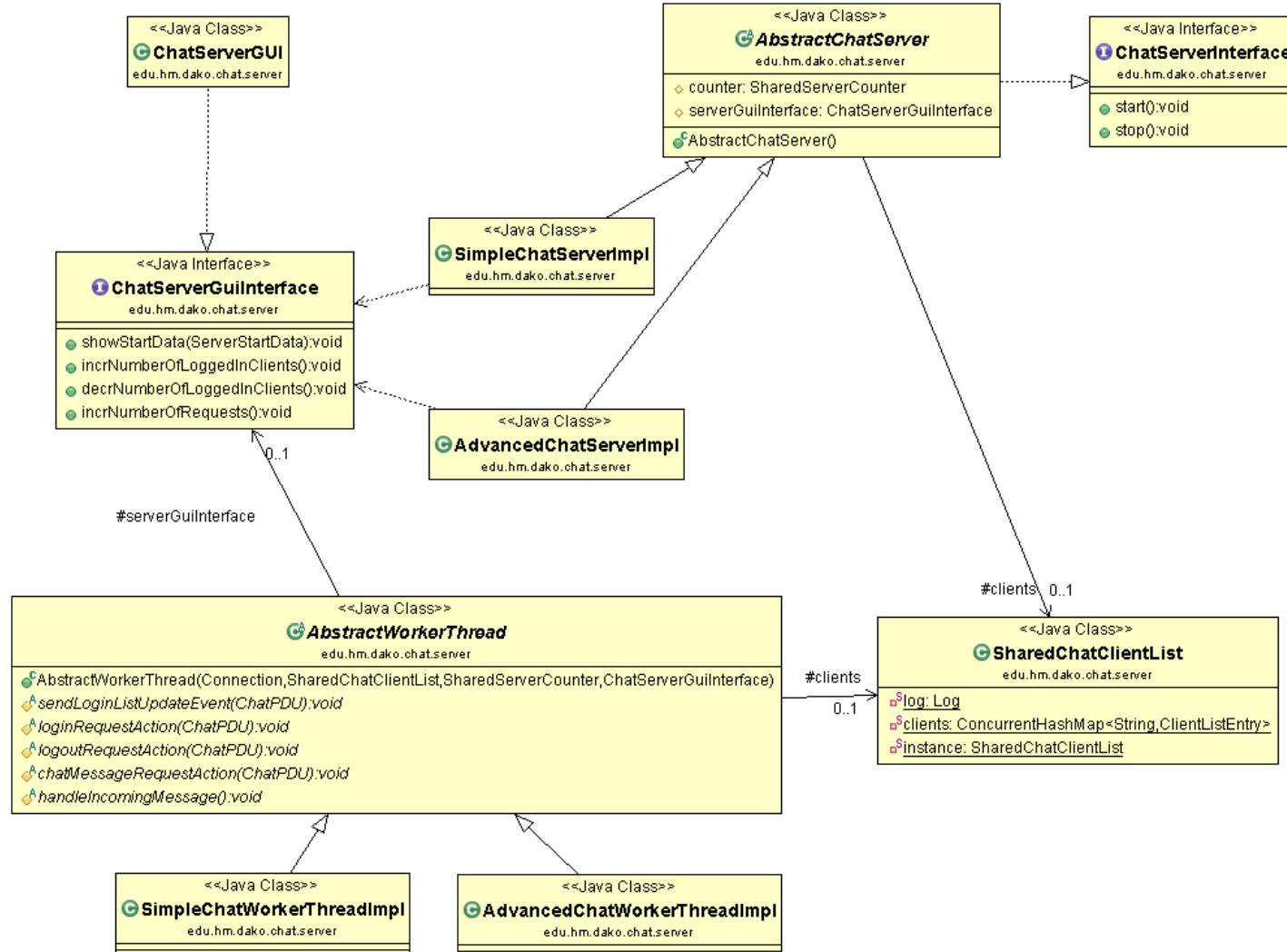
---



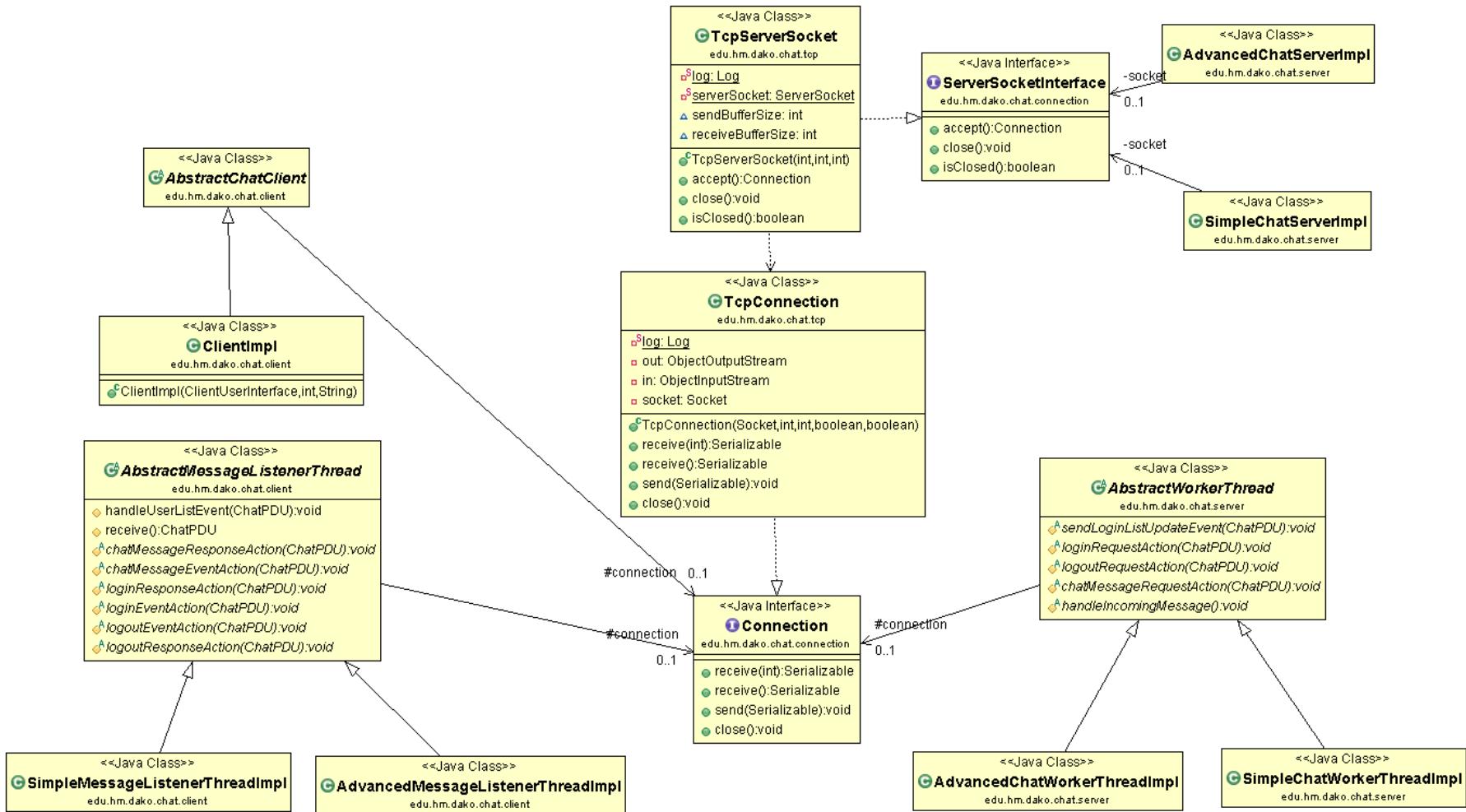
# Wichtige Objektklassen und Interfaces auf Chat-Client-Seite



# Wichtige Objektklassen und Interfaces auf Chat-Server-Seite



# Kommunikation über TCP-Sockets



# Überblick

---

- ✓ Begriffe und Referenzmodelle
- ✓ Beispiele für Anwendungsprotokolle
- ✓ Chat-Anwendung als begleitende Studie

- 1 Kommunikationssysteme und verteilte Anwendungen
- 2 **Grundlagen der Transportschicht**
- 3 Transportprotokolle TCP und UDP
- 4 Grundlagen der Vermittlungsschicht
- 5 Internet und Internet Protocol (IP)
- 6 Routing-Verfahren und -Protokolle
- 7 Internet-Steuerprotokolle und DNS
- 8 Internet Protocol Version 6 (IPv6)
- 9 Netzwerkschnittstelle

# Überblick über das Kapitel

---

## 1. Transportzugriff

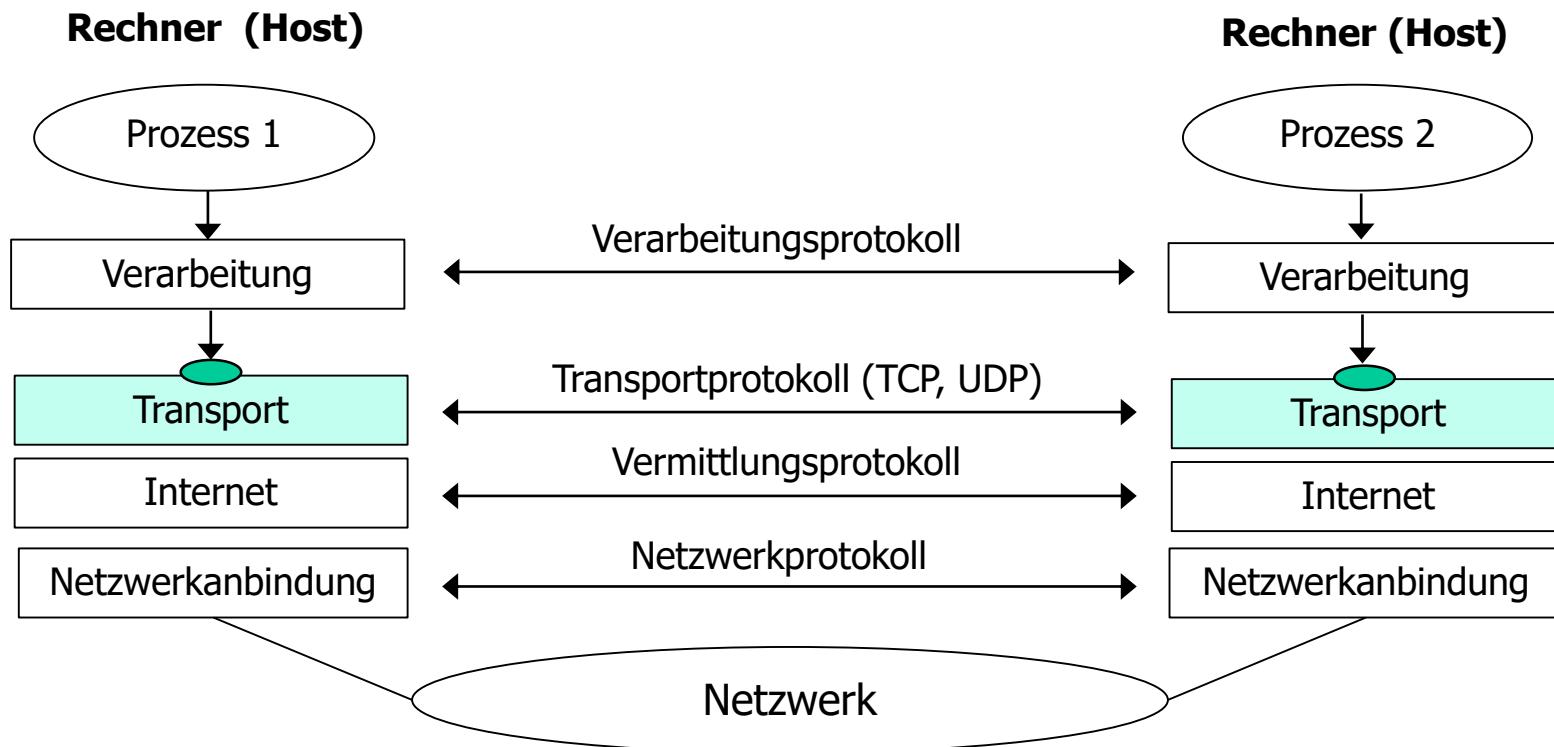
- Sockets: Kommunikationsmodell und API
- Socket API in Java

## 2. Transportorientierte Protokolle

- Verbindungsmanagement
- Datentransferphase

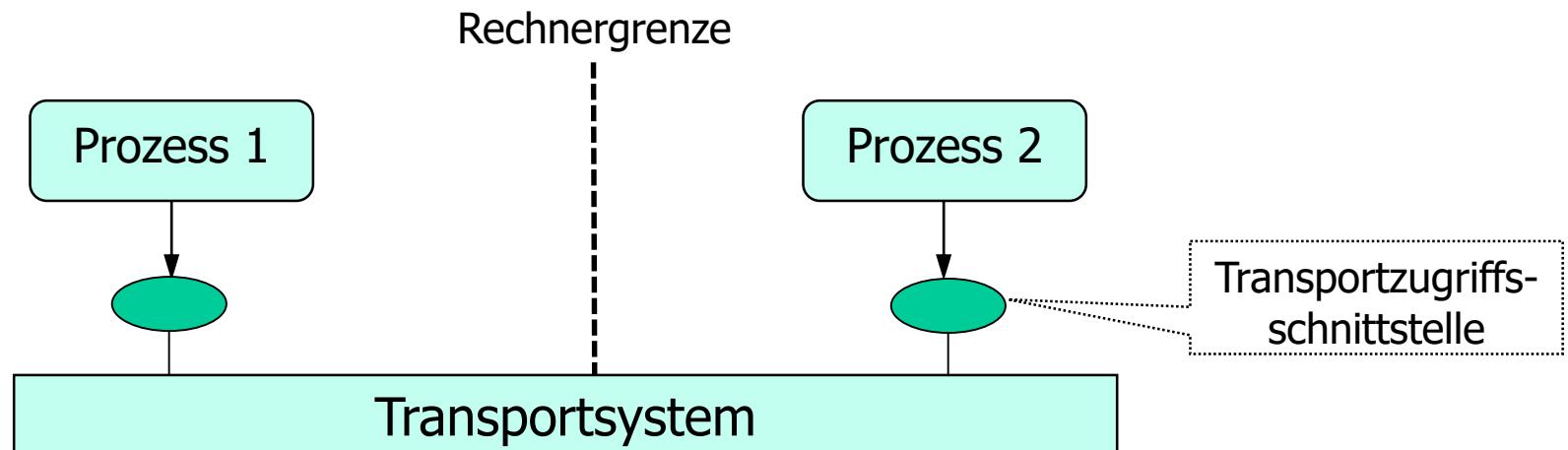
# Wiederholung: TCP/IP-Referenzmodell

## ■ Vier Kommunikationsschichten



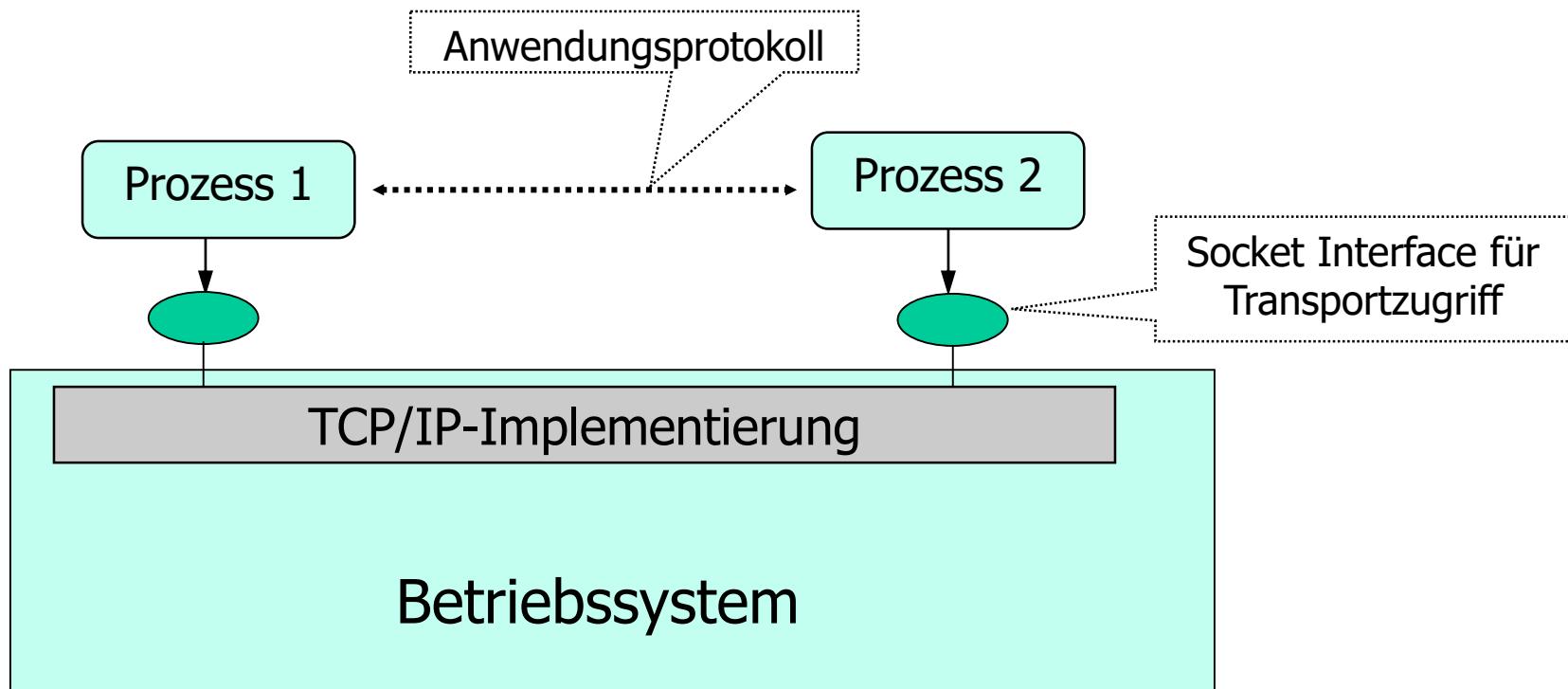
# Transportzugriffsschnittstelle

- Transportzugriffsschnittstelle ermöglicht den Anwendungsprozessen eine Ende-zu-Ende-Kommunikation



## Beispiele: Sockets: Systemeinbettung

- Sockets sind eine konkrete Implementierung einer Transportzugriffsschnittstelle



# Überblick über Sockets (1)

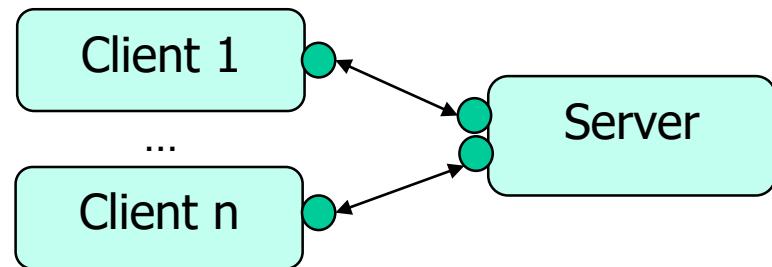
---

- Die Socket-Schnittstelle ist eine API, mit der man Kommunikationsanwendungen entwickeln kann
- Die **Originalversion** der Socket-Schnittstelle stammt von Mitarbeitern der **Firma BBN** (ARPA-Projekt, 1981)
- Sockets wurden in der Universität von Berkeley entwickelt (BSD-Version von UNIX)
  - Die **erste Version 4.1cBSD-System** für die VAX von DEC im Jahre 1982
- Sockets sind heute ein **De-facto-Standard** für Transportzugriffsschnittstellen
  - Siehe auch POSIX Standard

## Überblick über Sockets (2)

- Die Socket API **unterstützt** vor allem **Client-Server-Anwendungen**, was aus dem Programmiermodell hervorgeht:

- Aktiver Partner = Client
- Passiver Partner = Server

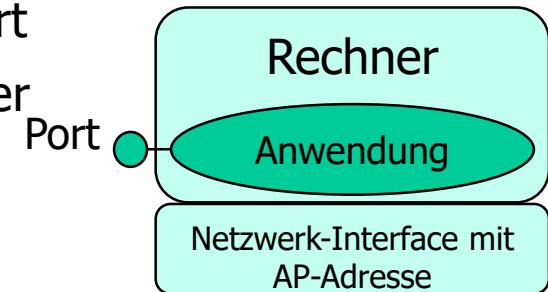


- Sockets sind **Kommunikationsendpunkte** innerhalb der Applikationen, die in der Initialisierungsphase miteinander verbunden werden
- Dabei spielt es keine Rolle, auf welchen Rechnern die miteinander kommunizierenden Prozesse laufen

# Sockets: Protokollmechanismen

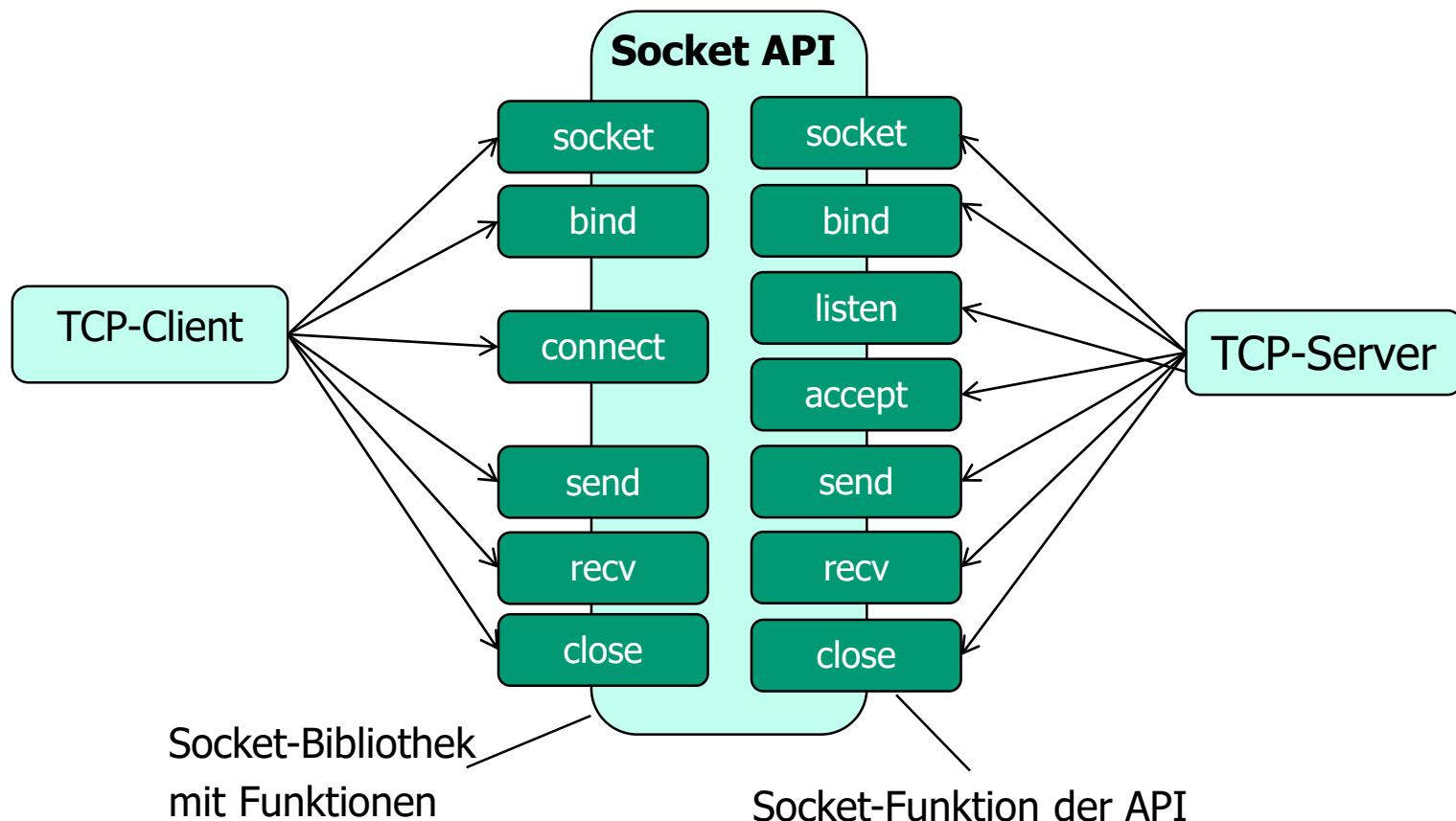
---

- TCP-Socket-Schnittstelle
  - **Verbindungsorientiert** und Stream-orientiert
  - Gesicherte Kommunikation (keine Duplikate, Ordnung, garantierte Übertragung)
  - Vollduplex-Verbindung zwischen zwei Partnern
- UDP-Socket-Schnittstelle
  - **Verbindungslos** und Nachrichten-orientiert
  - Keine Empfangsgarantie, Keine Ordnung der Pakete, Duplikate sind möglich
  - Vollduplex-Kommunikation
- Adressierung der Kommunikationspartner
  - Kommunikationsendpunkte werden über das Tupel (IP-Adresse, Portnummer) identifiziert



# TCP-Socket-Programmierung in C: Überblick

- Sockets wurden ursprünglich in der Sprache C implementiert

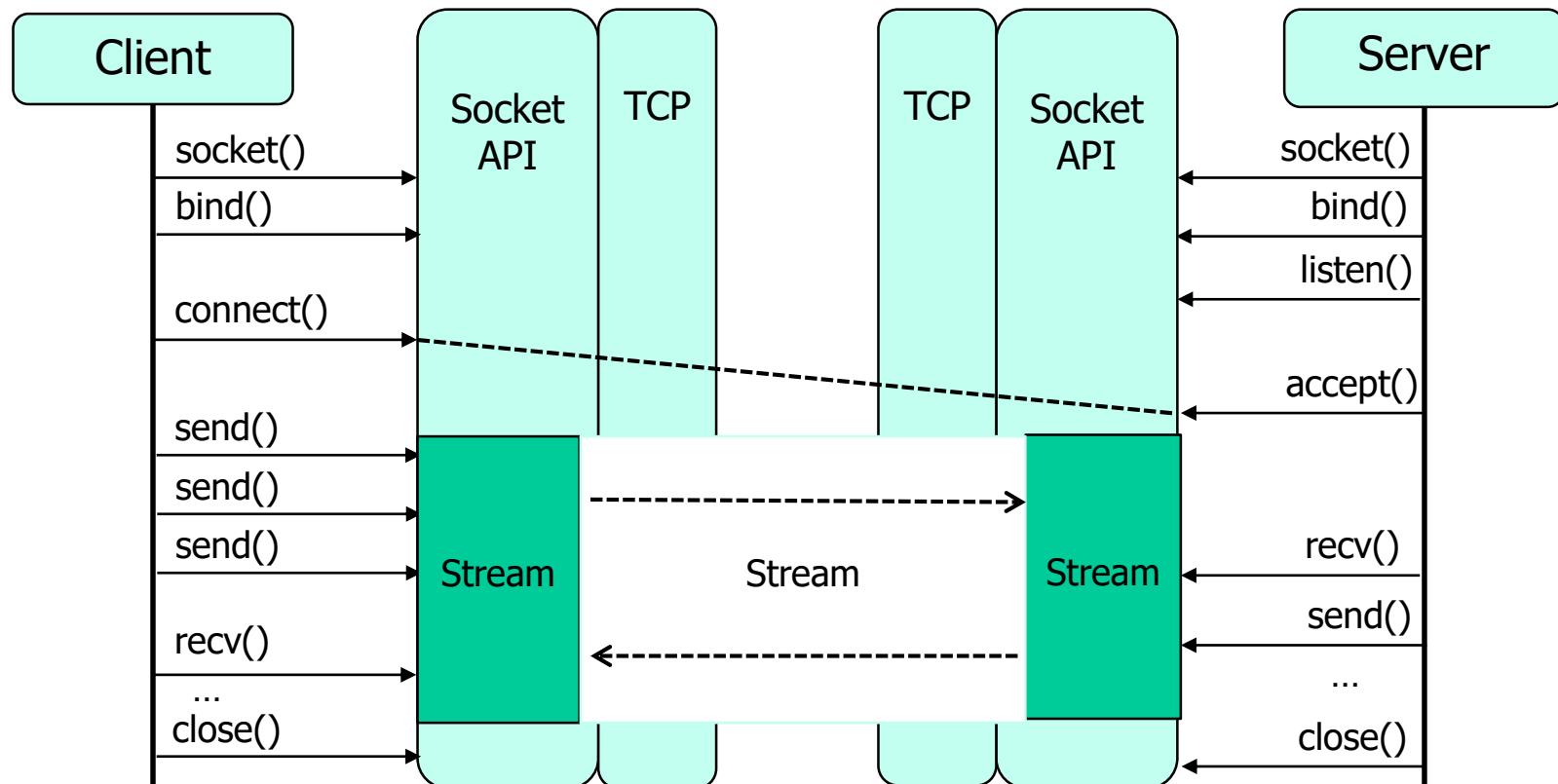


# TCP-Sockets: Die wichtigsten C-Funktionen

Socket-Funktion	Beschreibung	Nutzung durch
socket()	Initialisiert einen Socket	Client, Server
bind()	Ordnet einem Socket eine lokale Adresse zu	Client, Server
connect()	Setzt den Socket in einen passiven, d.h. auf ankommenden Verbindungswünsche wartenden Zustand	Client
listen()	Wird bei TCP-Verbindungen verwendet und gibt die nächste ankommende, aufgebaute Verbindung aus der Warteschlange zurück	Server
accept()	Wird bei TCP-Verbindungen verwendet und gibt die nächste ankommende, aufgebaute Verbindung aus der Warteschlange zurück	Server
close()	Schließt ein Socket und die dazugehörige Verbindung	Client, Server
recv()	Liest Daten aus dem spezifizierten Socket und gibt die Anzahl der tatsächlich gelesenen Byte zurück	Client, Server
send()	Sendet Daten über den spezifizierten Socket und gibt die Anzahl der tatsächlich gesendeten Byte zurück	Client, Server

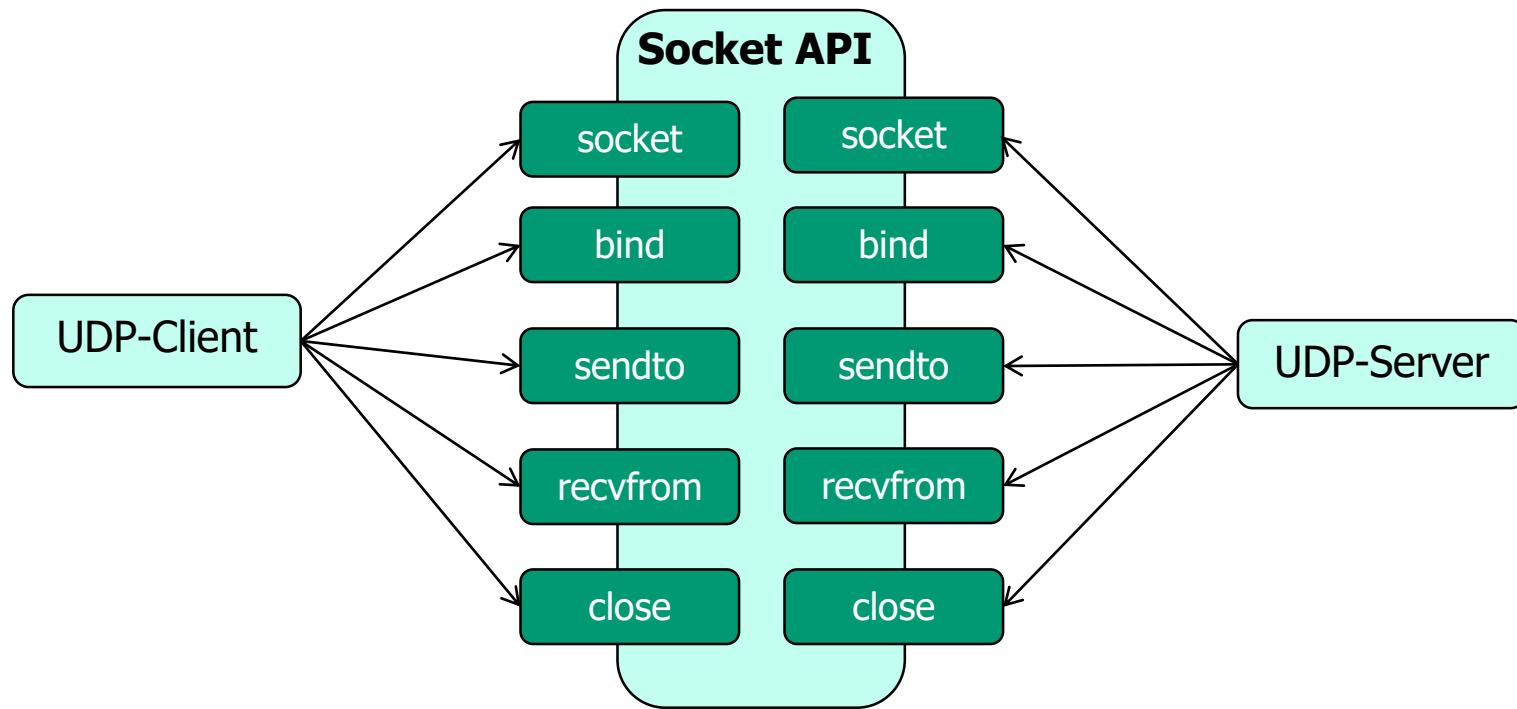
# TCP-Socket-Programmierung in C: Zusammenspiel

- Zusammenspiel: Wie TCP arbeitet wird an der Socket-Schnittstelle für den Anwendungsprogrammierer nicht sichtbar



# Datagramm-Socket-Programmierung in C: Überblick

- Kein Verbindungsauflauf notwendig

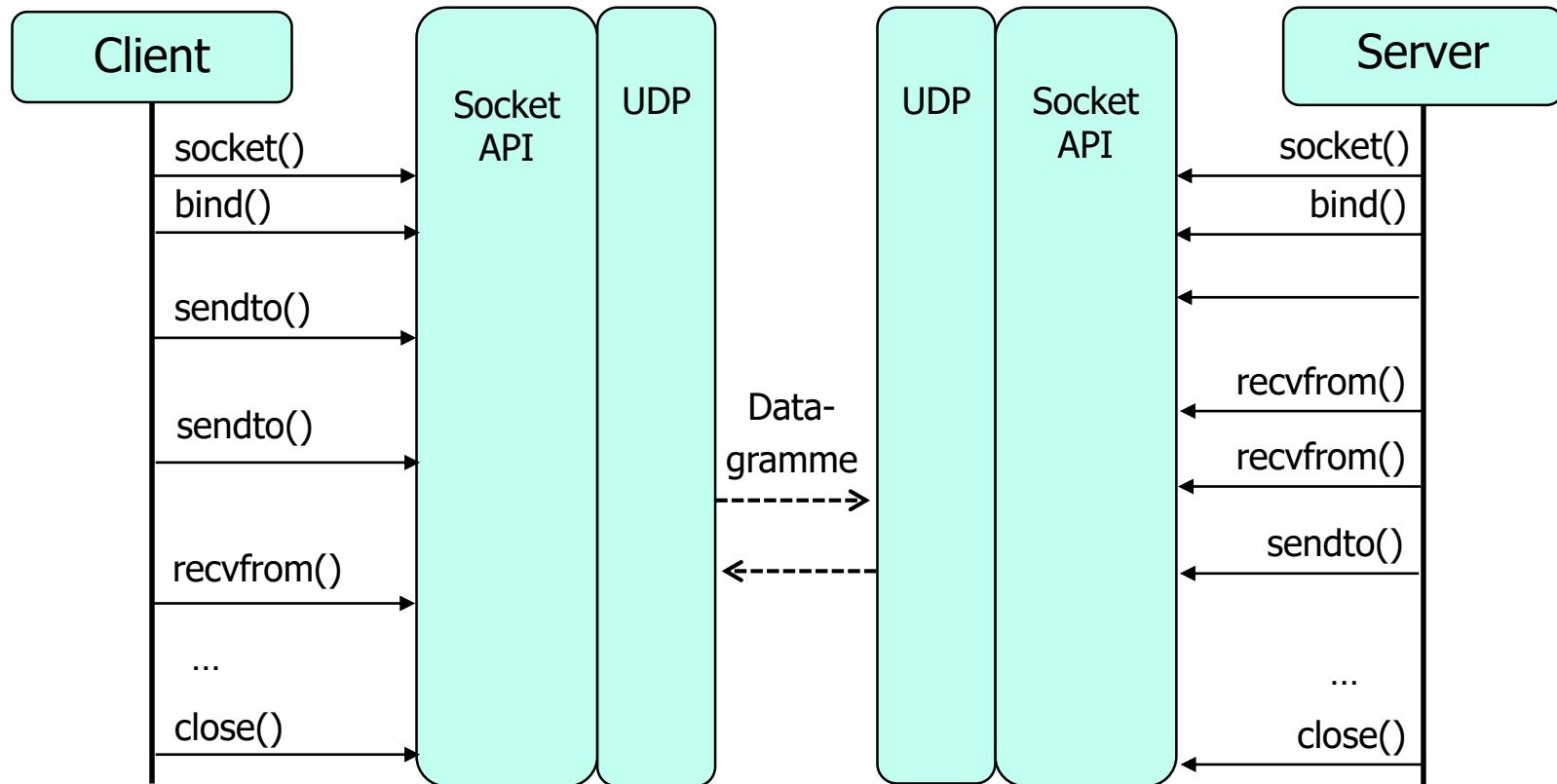


# UDP-Sockets: Die wichtigsten C-Funktionen

Socket-Funktion	Beschreibung	Nutzung durch
socket()	Initialisiert einen Socket	Client, Server
bind()	Ordnet einem Socket eine lokale Adresse zu (UDP-Port und IP-Adresse)	Client, Server
close()	Schließt ein Socket	Client, Server
recvfrom()	Empfängt eine Nachricht an einem UDP-Socket, speichert sie in einem Empfangspuffer und gibt die Anzahl der gelesenen Bytes zurück	Client, Server
sendto()	Sendet eine Nachricht und liefert die Anzahl der gesendeten Bytes zurück	Client, Server

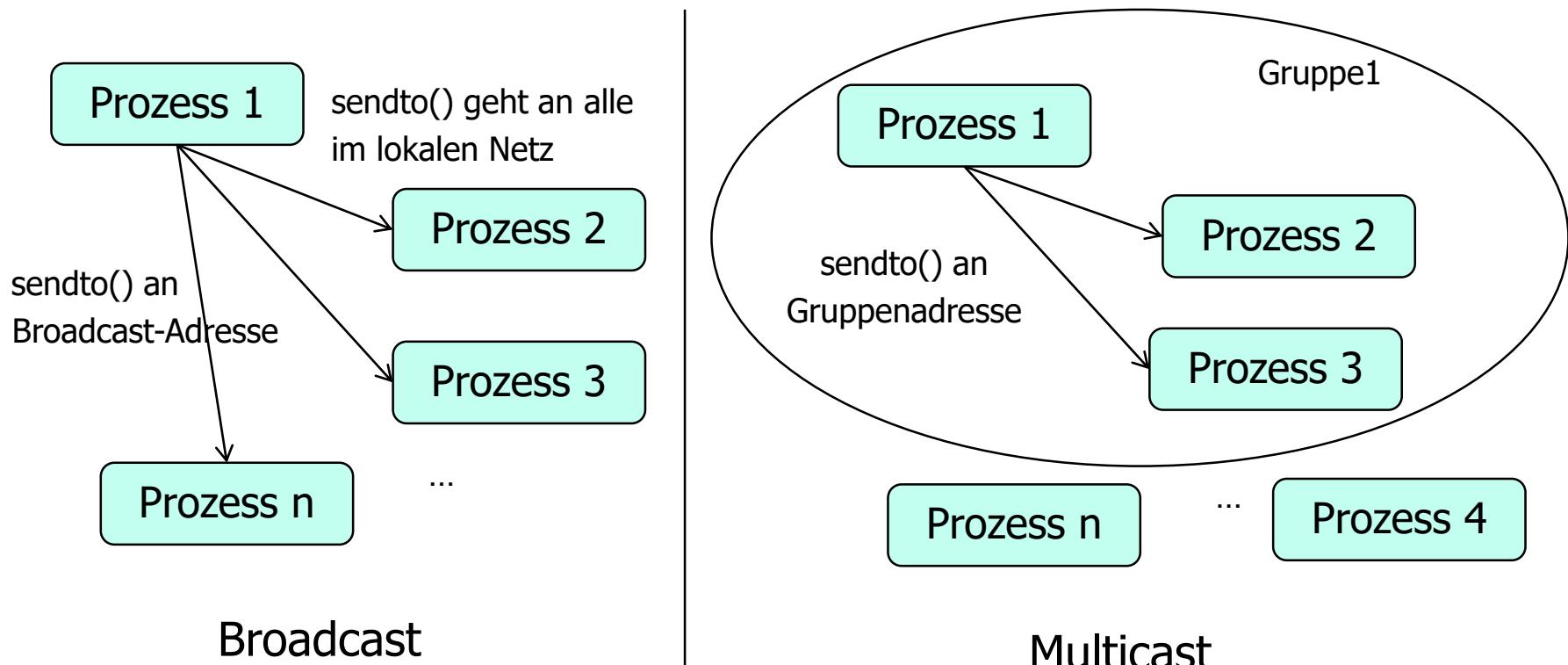
# UDP-Socket-Programmierung in C: Zusammenspiel

- Kein Verbindungsaufbau und auch kein Stream wie bei TCP-Sockets, sondern Austausch von Datagrammen



# UDP-Socket-Programmierung in C: Multicast und Broadcast

- Multicast- und Broadcast lässt sich nur mit UDP-Sockets realisieren
- Einsparung von Nachrichten → weniger Overhead



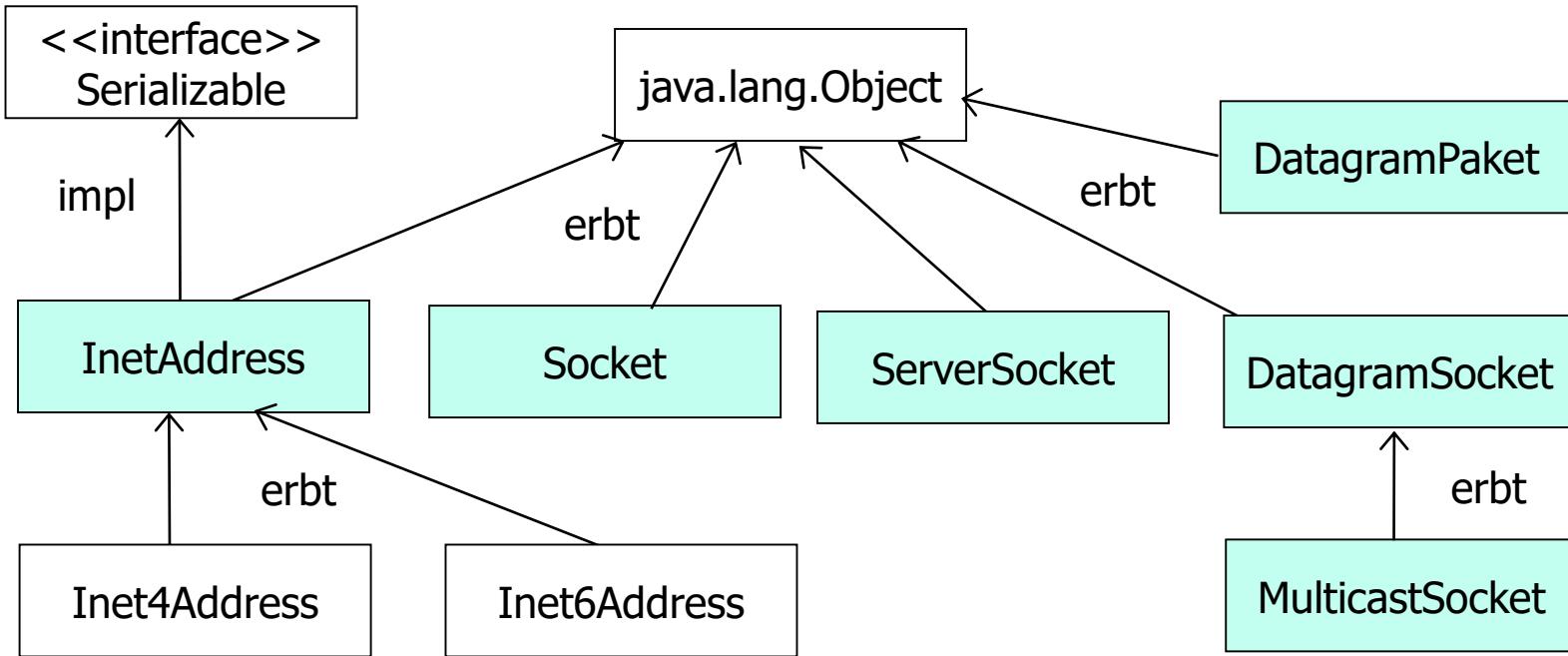
# Java-Socket-Programmierung: Wichtige Java-Klassen für TCP-und UDP-Sockets

---

- In der Java-API ist das Package **java.net** für TCP-Sockets vorgesehen. Wichtige Klassen sind:
  - InetAddress
  - Socket (Client)
  - ServerSocket
- Das Package **java.net** enthält auch Klassen zur Bearbeitung von UDP-Sockets. Wichtige Klassen sind:
  - DatagramSocket
  - DatagramPaket

# Java-Socket-Programmierung: Package `java.net`, wichtige Klassen

---

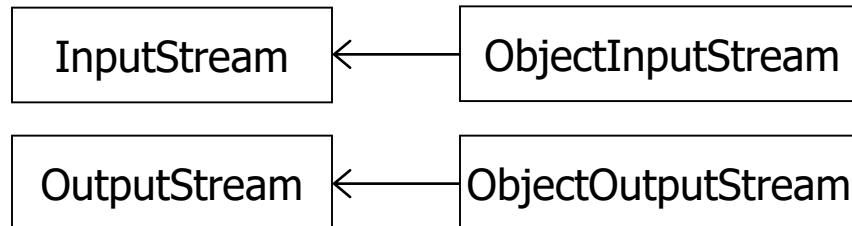


Vgl. Java™ 2 Platform Standard Edition 1.4, ..., 1.8 ...

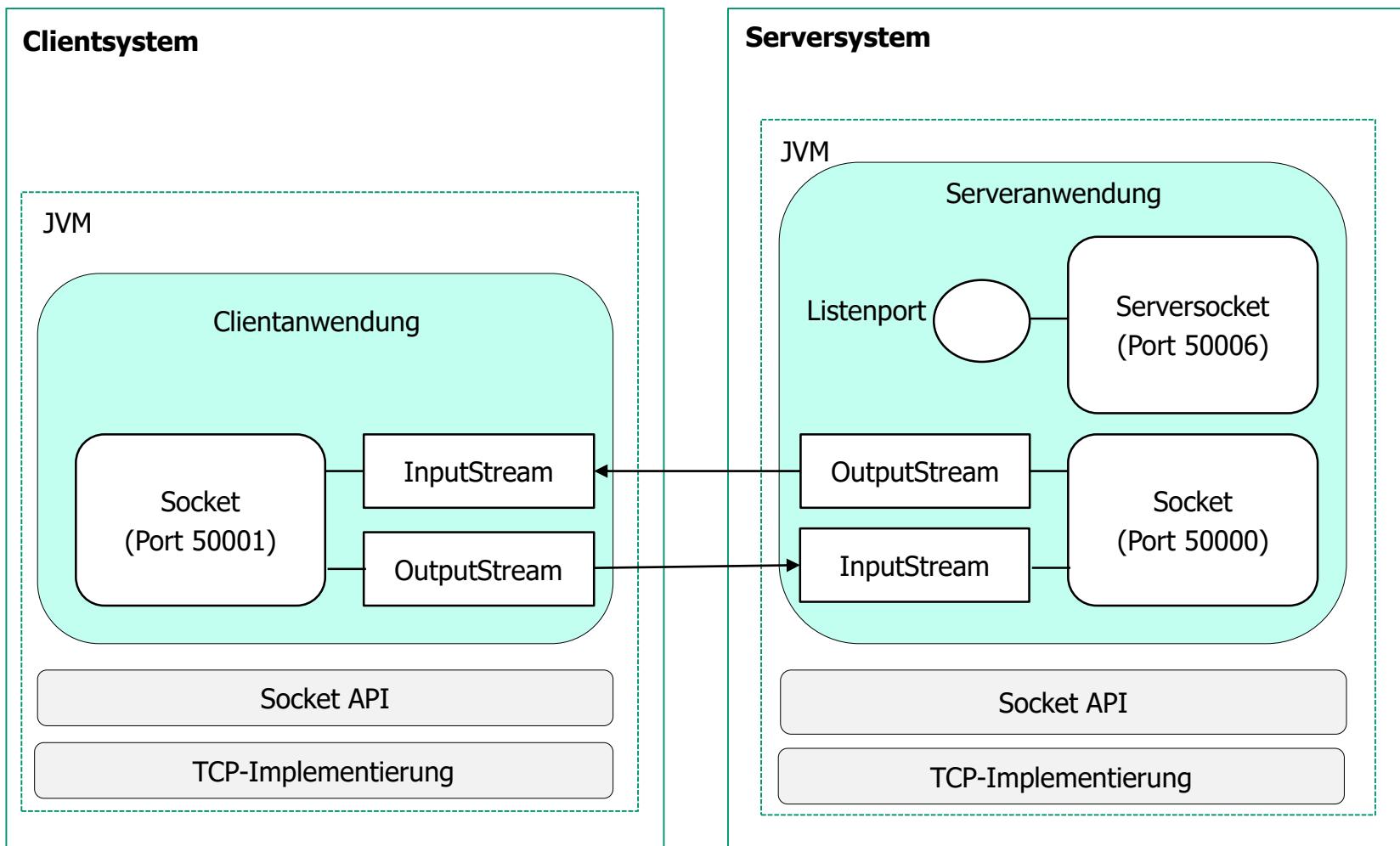
# Java-Socket-Programmierung: Streams

---

- **java.net** stellt eine höherwertige Schnittstelle für Sockets zur Verfügung
  - objektorientierte Schnittstelle
  - Implementierungsdetails gut gekapselt
- Bei TCP-Sockets werden zum Lesen und Schreiben sog. Streams verwendet:
  - InputStream
  - OutputStream
- Über diese Streams kann man wiederum **Objektströme** (ObjectInputStream und ObjectOutputStream) legen, um ganze Java-Objekte zu senden und zu empfangen

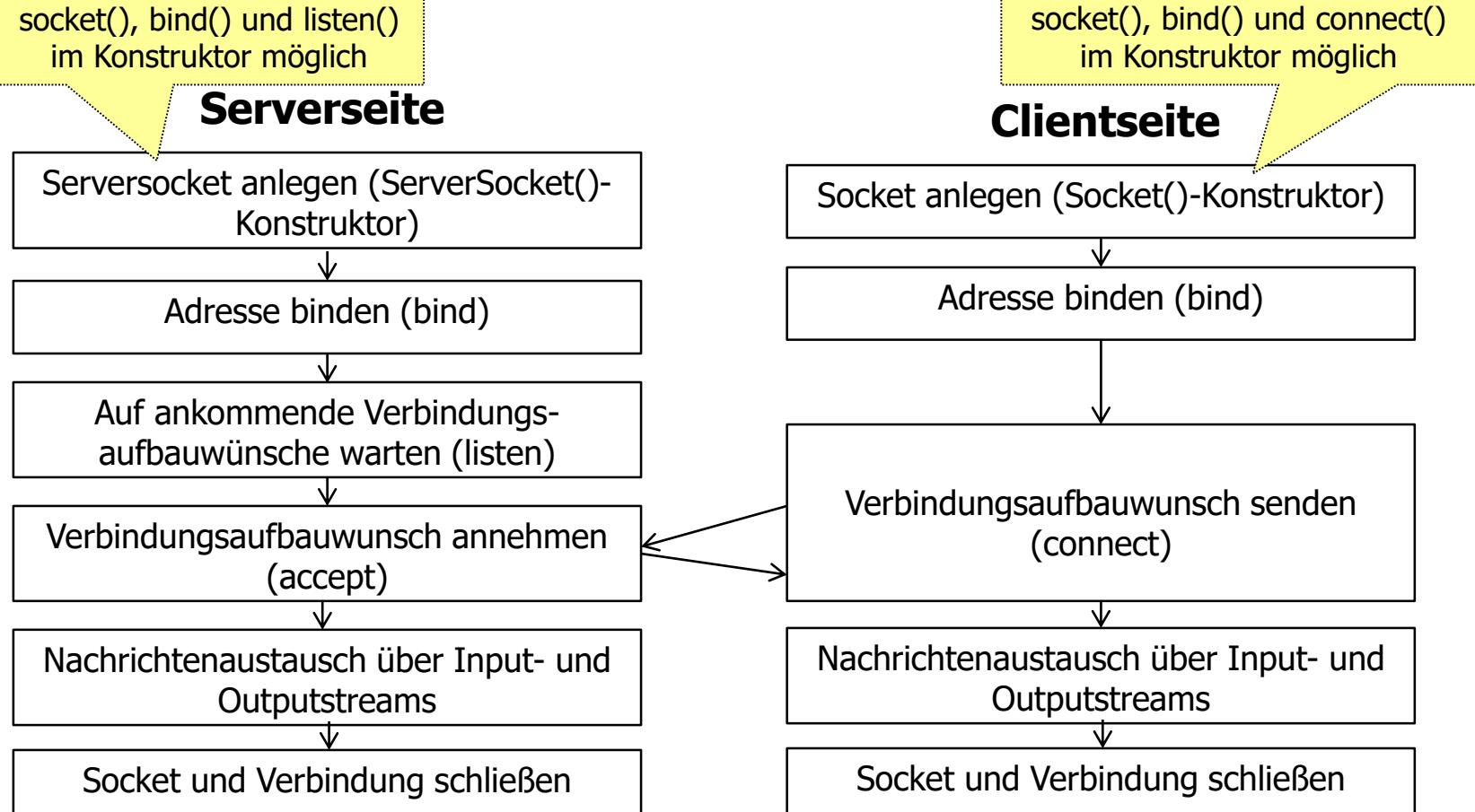


# Java-Socket-Programmierung: Modellbausteine für TCP-Sockets



# Java-Socket-Programmierung: Programmierung mit TCP-Sockets

- Verbindungsaufbau Nachrichtenaustausch und Verbindungsabbau



# Java-Socket-Programmierung: Beispielprogramm - Programmrahmen

## Server S1

```
...  
private ObjectOutputStream out;  
private ObjectInputStream in;  
ServerSocket server = new  
    ServerSocket(50000);  
  
...  
Socket verbindung = server.accept();  
out = new ObjectOutputStream  
    (verbindung.getOutputStream());  
in = new ObjectInputStream  
    (verbindung.getInputStream());  
  
...  
// Empfangen über Inputstream  
// Senden über OutputStream  
  
...  
verbindung.close();  
  
*) Ohne Exception Handling
```

socket(), bind() und  
listen() im Konstruktor

## Client

```
String serverAdresse = new String("S1");  
...  
private ObjectOutputStream out;  
private ObjectInputStream in;  
Socket verbindung =  
    new Socket(serverAdresse, 50000);  
...  
out = new ObjectOutputStream  
    (verbindung.getOutputStream());  
in = new ObjectInputStream  
    (verbindung.getInputStream());  
  
...  
// Empfangen über Inputstream  
// Senden über OutputStream  
  
...  
verbindung.close();  
  
*) Ohne Exception Handling
```

socket(), bind() und  
connect() im Konstruktor

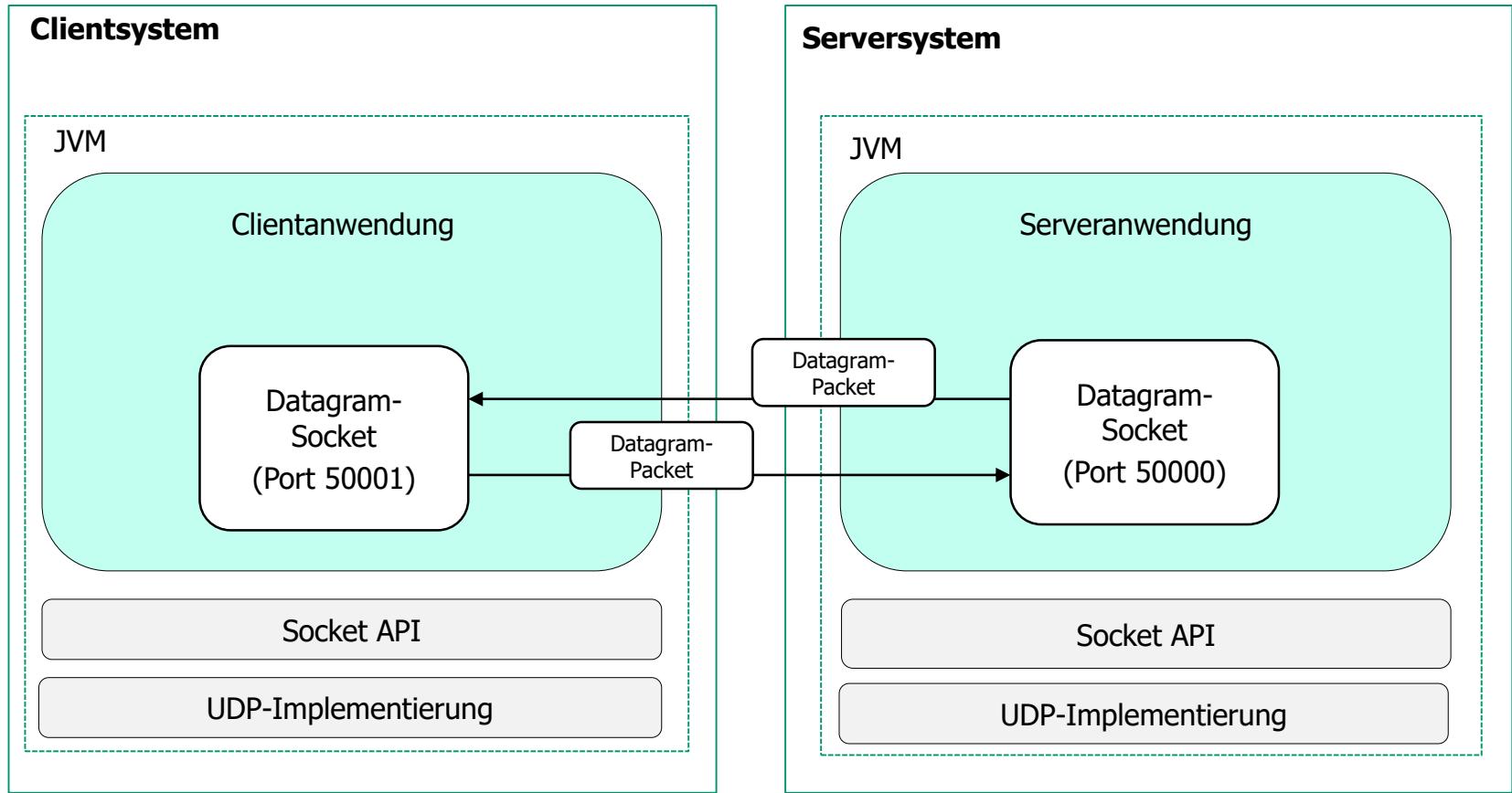
# Java-Socket-Programmierung: UDP-Sockets in Java (Datagram-Sockets)

---

- Sockets vom Typ **DatagramSocket** für das Senden von Datagrammen
  - analog TCP kann dem Konstruktor eine Portnummer vorgegeben werden
  - ohne spezifizierte Portnummer → **freie Portnummer** wird zugeordnet
- Methoden der Klasse DatagramSocket für die nachrichtenorientierte Kommunikation
  - void **send**(DatagramPacket p) zum Senden und
  - void **receive**(DatagramPacket p) zum Empfangen von Datagrammen
  - receive blockiert den Aufrufer bis ein Datagramm eingeht

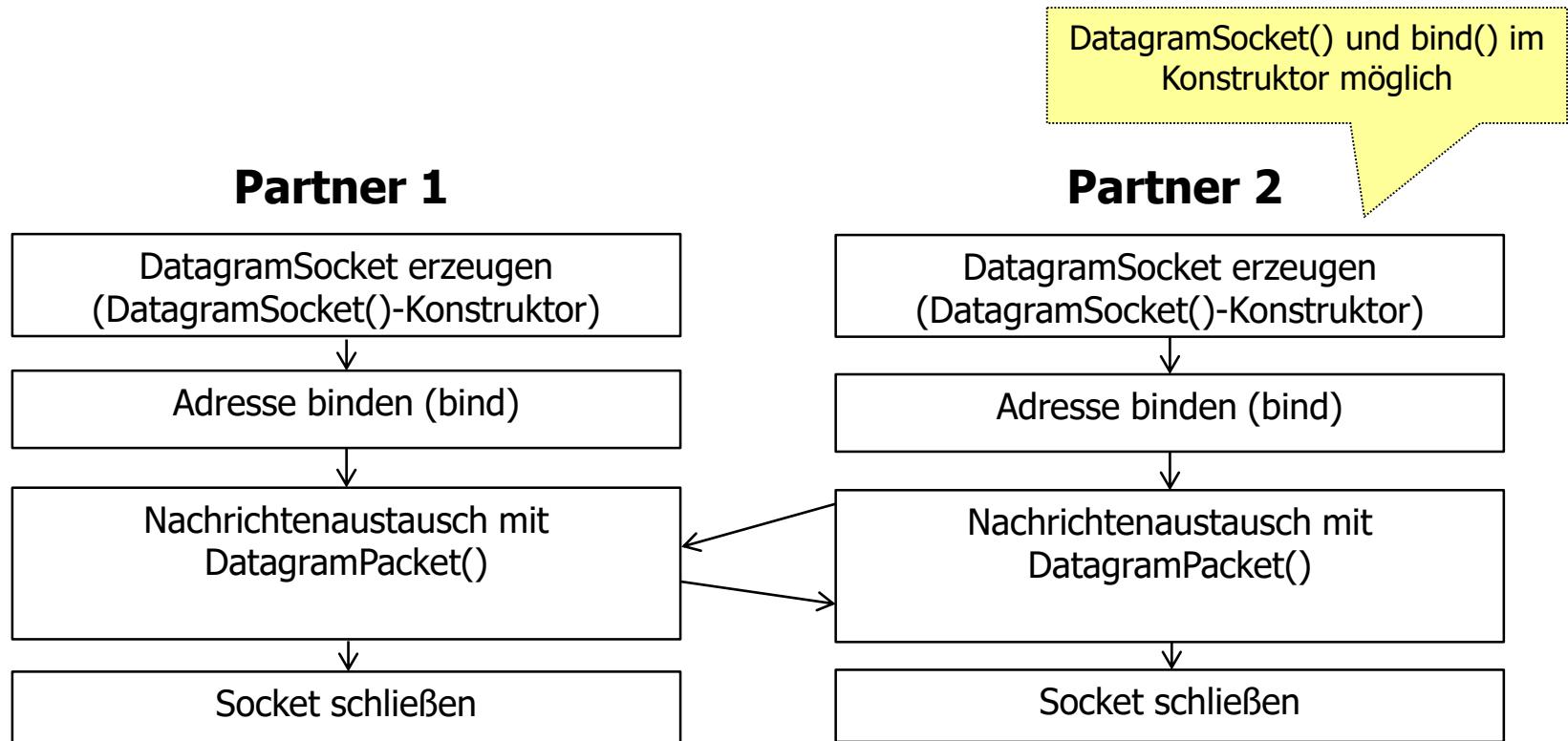
# Java-Socket-Programmierung: Modellbausteine für UDP-Datagramm-Sockets

- DatagramSocket auf beiden Seiten
- DatagramPacket zum Senden und Empfangen



# Java-Socket-Programmierung: Programmierung mit UDP-Datagramm-Sockets

- Nachrichtenaustausch ohne Verbindungsaufbau
- Datagrammlänge wird beim Senden angegeben



# Socket-Programmierung: Beispiel EchoServer (1)

---

```
package UDPEchoExample;
import java.net.*;
import java.io.*;

public class UDPEchoServer {
    protected DatagramSocket socket;
    public UDPEchoServer (int port) throws IOException
    {
        socket = new DatagramSocket (port);
    }
    public void execute () throws IOException
    {
        while (true) {
            DatagramPacket packet = receivePacket();
            sendEcho (packet.getAddress (), packet.getPort (),
                      packet.getData (),   packet.getLength ());
        }
    }
}
```

# Socket-Programmierung: Beispiel EchoServer (2)

```
protected DatagramPacket receivePacket () throws IOException {  
    byte buffer[] = new byte[65535];  
    DatagramPacket packet = new DatagramPacket (buffer, buffer.length);  
    socket.receive (packet);  
    System.out.println ("Received " + packet.getLength () + " bytes.");  
    return packet;  
}  
protected void sendEcho (InetAddress address, int port, byte data[], int length) throws  
    IOException {  
    DatagramPacket packet = new DatagramPacket (data, length, address, port);  
    socket.send (packet);  
    System.out.println ("Response sent");  
}  
public static void main (String args[]) throws IOException {  
    if (args.length != 1)  
        throw new RuntimeException ("Syntax: UDPEchoServer <port>");  
    UDPEchoServer echo = new UDPEchoServer (Integer.parseInt (args[0]));  
    echo.execute ();  
}
```

# Socket-Programmierung: Beispiel EchoClient (1)

```
package UDPEchoExample;
import java.net.*;
import java.io.*;
public class UDPEchoClient {
    protected DatagramSocket socket;
    protected DatagramPacket packet;

    public UDPEchoClient (String message, String host, int port) throws IOException {
        socket = new DatagramSocket ();
        packet = buildPacket (message, host, port);
        try {
            sendPacket ();
            receivePacket ();
        } finally {
            socket.close ();
        }
    }

    protected void sendPacket () throws IOException {
        socket.send (packet);
    }
}
```

# Socket-Programmierung: Beispiel EchoClient (2)

```
protected void receivePacket () throws IOException {
    byte buffer[] = new byte[65535];
    DatagramPacket packet = new DatagramPacket (buffer, buffer.length);
    socket.receive (packet);
    ByteArrayInputStream byteI = new ByteArrayInputStream (packet.getData (), 0,
        packet.getLength ());
    DataInputStream dataI = new DataInputStream (byteI);
    String result = dataI.readUTF ();
}
public static void main (String args[]) throws IOException {
    if (args.length != 3)
        throw new RuntimeException („EchoClient <host> <port> <message>“);
    while (true) {
        new UDPEchoClient (args[2], args[0], Integer.parseInt (args[1]));
        try {
            Thread.sleep (1000);
        } catch (InterruptedException ex) {...}
    }
}
```

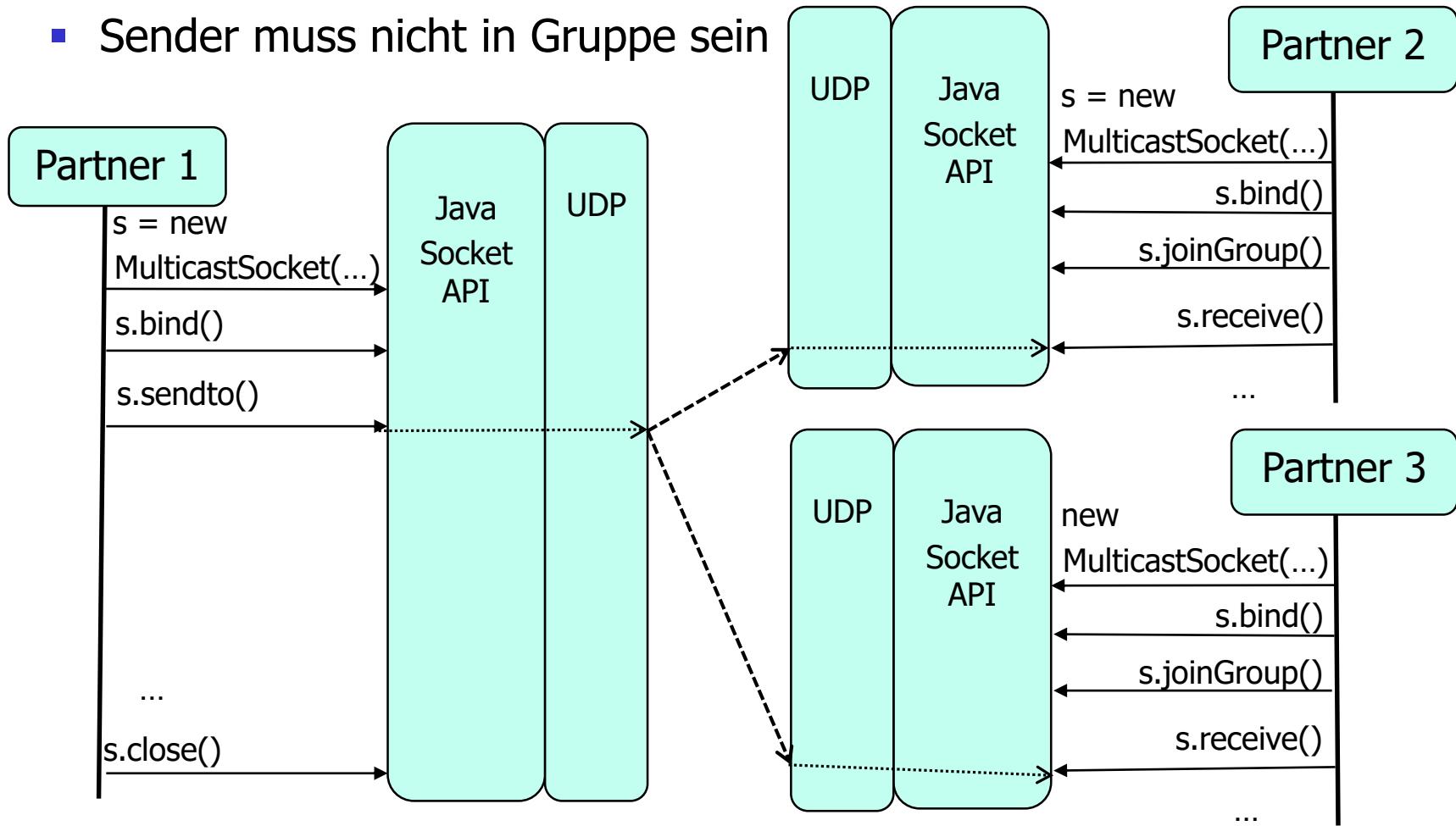
# Socket-Programmierung: Package `java.net`, `MulticastSockets`

---

- Die Klasse `MulticastSockets` dient zum Senden und Empfangen von IP-Multicast-Datagrammen
- Ein `MulticastSocket` ist ein `UDP-DatagramSocket` zur IP-basierten Gruppenkommunikation
  - Siehe IP-Adressen 224.\* - 239.\*
  - Methode **`joinGroup(InetAddress groupAddr)`** zum Anbinden an eine Gruppe
  - Methode **`leaveGroup( InetAddress mcastaddr)`** zum Verlassen der Gruppe
- Ein gesendetes Datagramm empfangen alle Gruppenmitglieder

# Java-MulticastSockets im Zusammenspiel

- An Gruppe anschließen und Nachrichten senden und empfangen
- Sender muss nicht in Gruppe sein



# Socket-Programmierung: Beispiel Multicast-Programmierung - Senden

```
package UDPMulticastExample;
import java.net.*;
import java.io.*;
public class UDPMulticastSender {
    public static void main (String args[]) {
        InetAddress group = InetAddress.getByName("224.10.1.1");
        MulticastSocket s = new MulticastSocket(7000);
        s.joinGroup(group);
        byte [] block = new byte[1024];
        // Nachricht füllen ...
        DatagramPacket packet = new DatagramPacket(block, block.length,
            group, 7000);
        s.send(packet); ...
    }
}
```

- Datagramm-Länge: 0 to 65507
- Nur Klasse D-Adressen nutzbar 224.0.0.0 bis 235.1.1.1

# Socket-Programmierung: Beispiel Multicast-Programmierung - Empfangen

```
package UDPMulticastExample;
import java.net.*;
import java.io.*;
public class UDPMulticastReceiver {
    public static void main (String args[]) {
        InetAddress group = InetAddress.getByName("224.10.1.1");
        MulticastSocket s = new MulticastSocket(7000);
        s.joinGroup(group);
        byte [] block = new byte[1024];
        DatagramPacket packet = new DatagramPacket(block, block.length);
        socket.receive(packet);

        // Nachricht bearbeiten
        ...
    }
}
```

- Nur Empfänger muss der Gruppe beitreten
- Multicast-Adresse und Port wie bei Sender

# Überblick

---

## 1. Transportzugriff

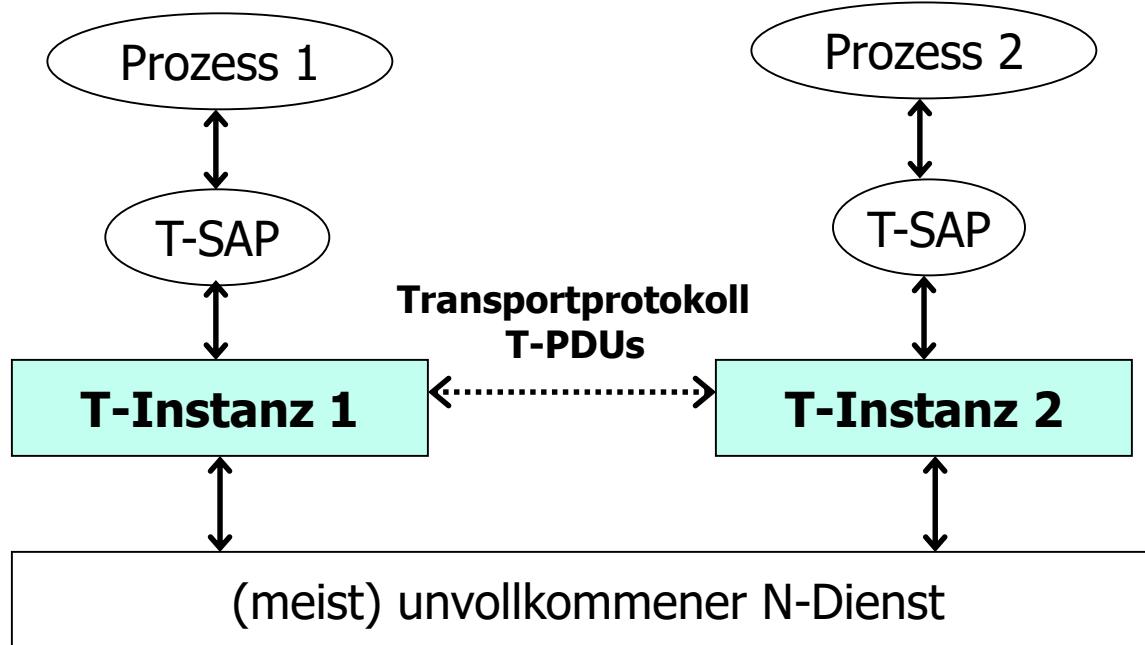
- Sockets: Kommunikationsmodell und API
- Socket API in Java

## 2. Transportorientierte Protokolle

- Verbindungsmanagement
- Datentransferphase

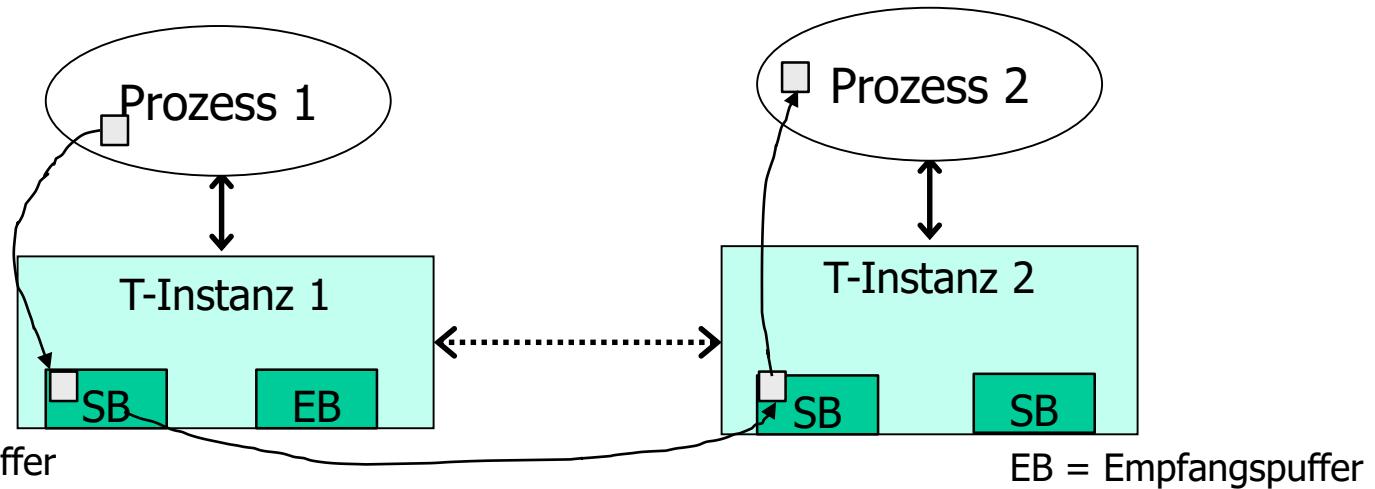
# Dienste der Transportschicht

- Logischer Ende-zu-Ende-Transport
  - **verbindungsorientierte** Transportdienste
  - **verbindungslose** Transportdienste



# Pufferung von Nachrichten

- Puffer für ankommende Nachrichten werden **in den Protokollinstanzen** (meist im Betriebssystemkern) verwaltet
- Die Instanzen **kopieren** die Nachrichten in den Adressraum der empfangenden Anwendungsprozesse
- Pufferspeicher müssen **verwaltet** werden ( $\rightarrow$  Overhead)
- Pufferspeicher **benötigen Adressraum** (Speicher)
- Pufferspeicher sind **begrenzt** ( $\rightarrow$  evtl. Verwerfen von Nachrichten, wenn sie voll sind)



# Verbindungsorientierte Transportdienste

---

- Eine Verbindung wird etabliert
- **Gemeinsamer Kontext** wird aufgebaut
- Geprägt von traditionellen Kommunikationsdiensten wie Telefonieren
- **Hohe Zuverlässigkeit**
  - Fehlerfreie und reihenfolgerichtige Auslieferung der Daten beim Empfänger
- Verbindungsorientierte Protokolle sind komplexer
  - Warum?
- Wann braucht man Verbindungen?

## Verbindungslose Transportdienste

---

- **Verlust** von Datenpaketen wird nicht bemerkt
- **Verfälschung** des Nutzdatenteils ist nicht unbedingt nachvollziehbar
- **Reihenfolgezerstörung** ist möglich
- Kein Zusammenhang bei aufeinanderfolgenden Dienstaufrufen
- T-PDUs enthalten immer die Adressinformation von Sender und Empfänger

# Protokollfunktionen in Transportprotokollen

---

- Verbindungsmanagement und Adressierung
- (Zuverlässiger) Datentransfer
- Flusskontrolle
- Staukontrolle
- Segmentierung

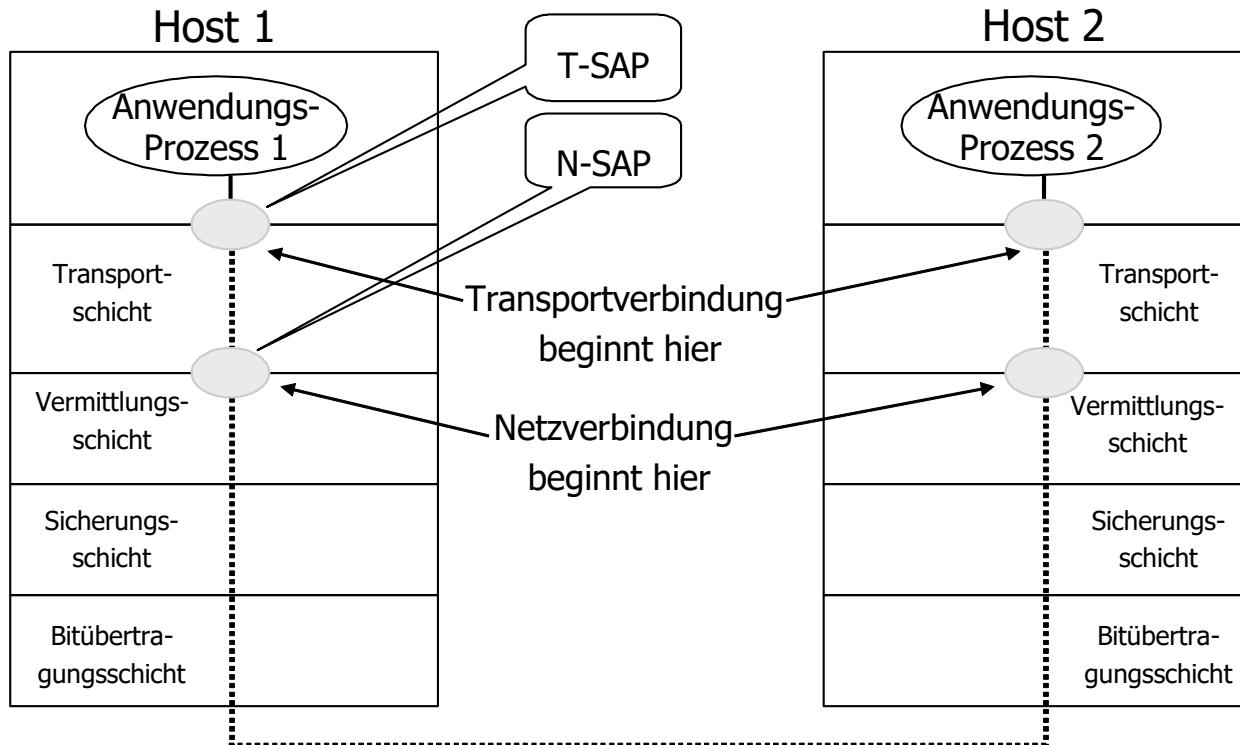
# Verbindungsmanagement und Adressierung

---

- Kommunizierende Anwendungsprozesse müssen sich kennen, d.h. die Adressen müssen bekannt sein
  - Schicht 4: Transportadresse
  - T-SAP (Transport Service Access Point)
- Transportadressen sind oft kryptisch (insbesondere bei IPv6), daher symbolische Adressen notwendig
  - Directory Service (Naming Service)

# Verbindungsmanagement und Adressierung

## T-SAP



T-SAP = Transport Service Access Point

N-SAP = Network Service Access Point

# Verbindungsauflaufbau

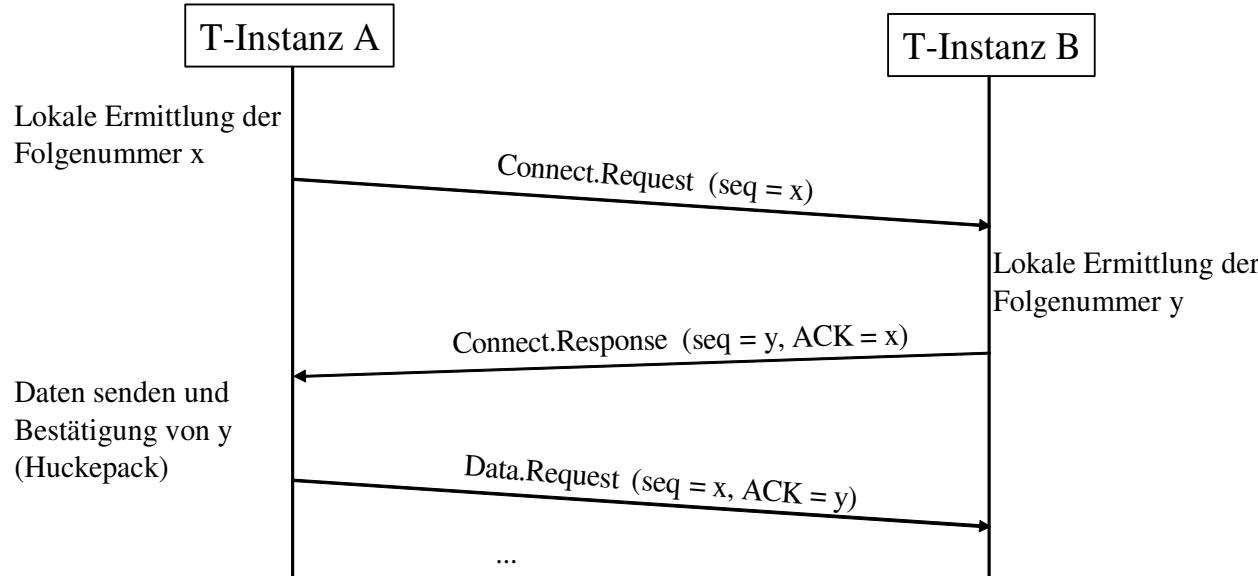
---

- Einrichten von **Connection End Points** (CEP)
  - Kontextaufbau auf beiden Seiten
- Mögliche **Fehler** sind zu **vermeiden**
  - Duplikate
  - Nachrichtenverlust
  - Reihenfolgevertauschung
- **Zwei-Wege-Handshake-Protokolle**
  - Austausch einer Sequenznummer (Folgenummer) für eine Kommunikationsrichtungen
- **Drei-Wege-Handshake-Protokolle**
  - Austausch einer Sequenznummer (Folgenummer) für beide Kommunikationsrichtungen

# Verbindungsauftbau, Drei-Wege-Handshake

## Normaler Protokollverlauf

- Instanz A und B suchen eigene Folgenummern x und y (seq) aus

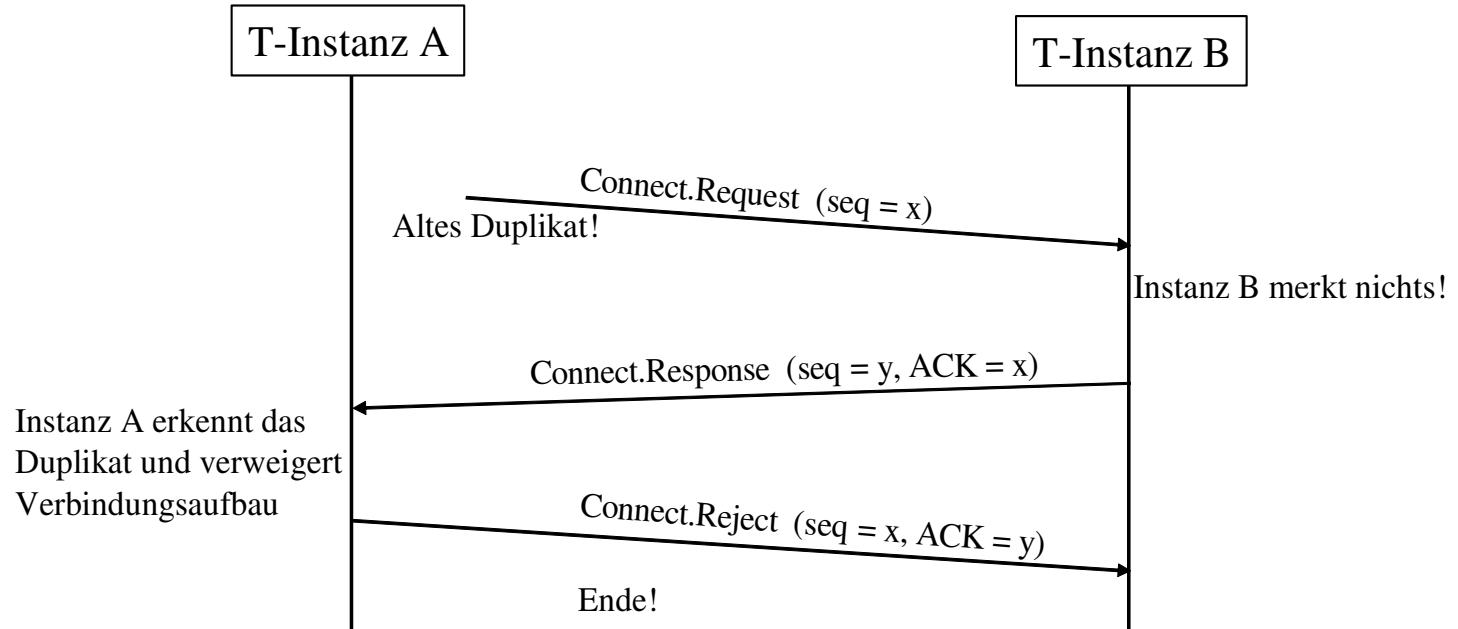


Seq = Folgenummer  
T-Instanz = Transportinstanz

Nach Tanenbaum, A.: et al.: Computer Networks,  
5. Auflage, Pearson Studium, 2011

# Verbindungsauftbau, Drei-Wege-Handshake

## Altes CR-Duplikat taucht auf



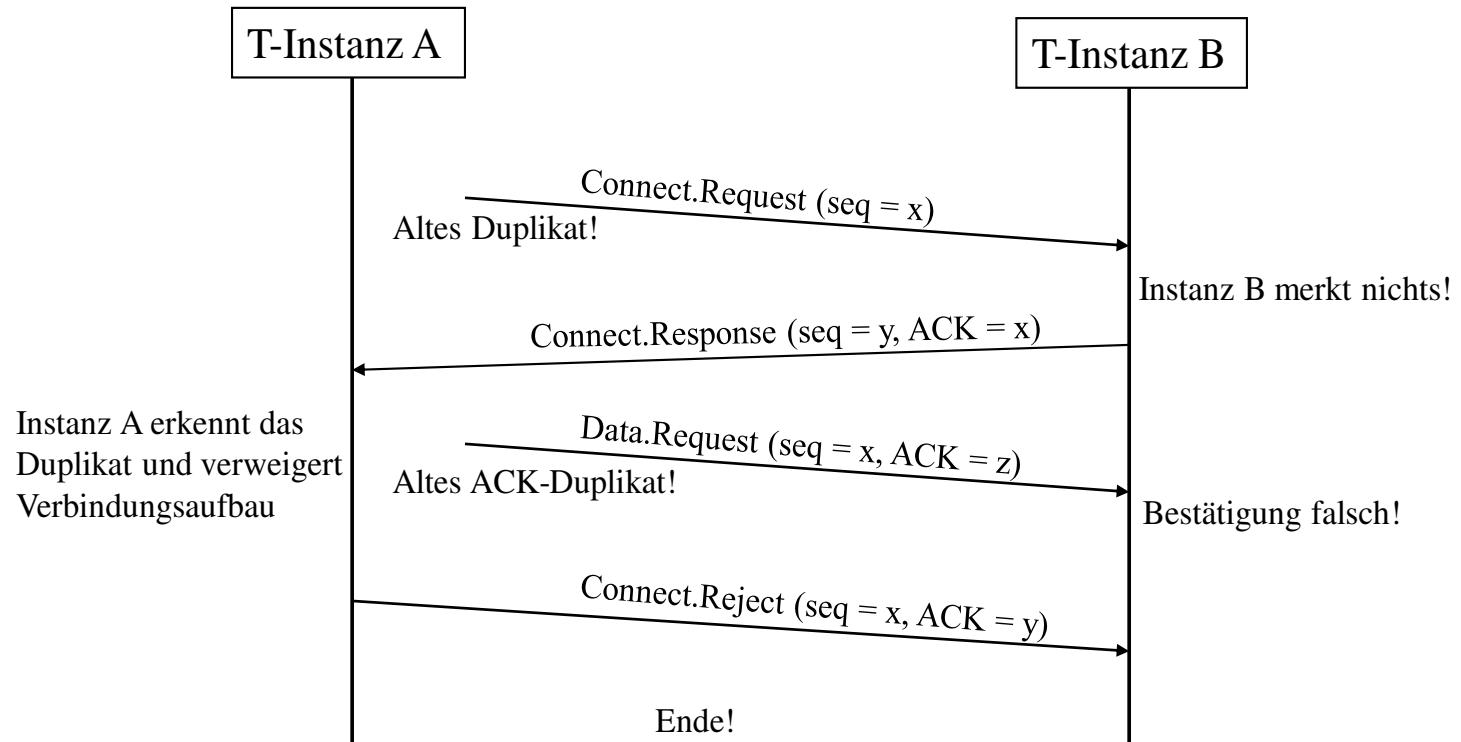
Seq = Folgenummer

T-Instanz = Transportinstanz

Nach Tanenbaum, A.: et al.: Computer Networks, 5. Auflage, Pearson Studium, 2011

# Verbindungsauftbau, Drei-Wege-Handshake

## Duplikat von Connect-Request und Duplikat von ACK tauchen plötzlich auf



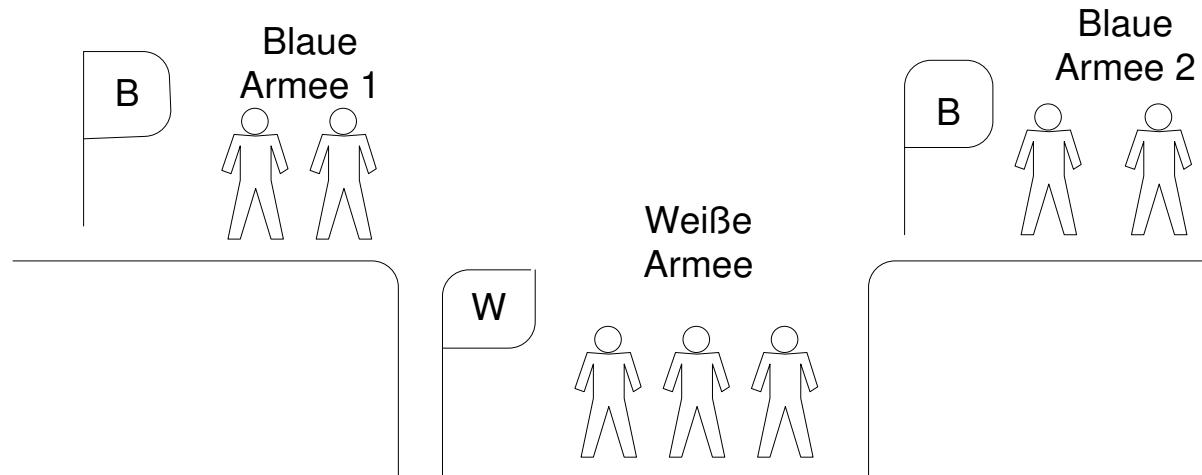
# Verbindungsabbau

---

- Anforderung:
  - Beim Verbindungsabbau dürfen keine Nachrichten verloren gehen
- Datenverlust könnte vorkommen, wenn
  - eine Seite einen Verbindungsabbau initiiert,
  - die andere aber vor Erhalt der Disconnect-Request-PDU noch eine Nachricht sendet
  - Diese **Nachricht sollte nicht verloren gehen**
- Anspruchsvolles Verbindungsabbau-Protokoll notwendig:
  - Drei- oder Mehrwege-Handshake-Mechanismus wird auch hier genutzt
  - Beide Seiten bauen ihre „Senderichtung“ ab

# Verbindungsabbau und das Zwei-Armeen-Problem

- Die Armee der Weißröcke lagert in einem Tal
- Auf beiden Anhöhen lagert ein Teil der Armee der Blauröcke
- Die Blauröcke können nur gemeinsam gewinnen und müssen ihren Angriff synchronisieren
- **Unzuverlässiger Kommunikationskanal:** Boten, die zu Fuß durch das Tal rennen müssen



Nach Tanenbaum, A.: et al.: Computer Networks, 5. Auflage, Pearson Studium, 2011

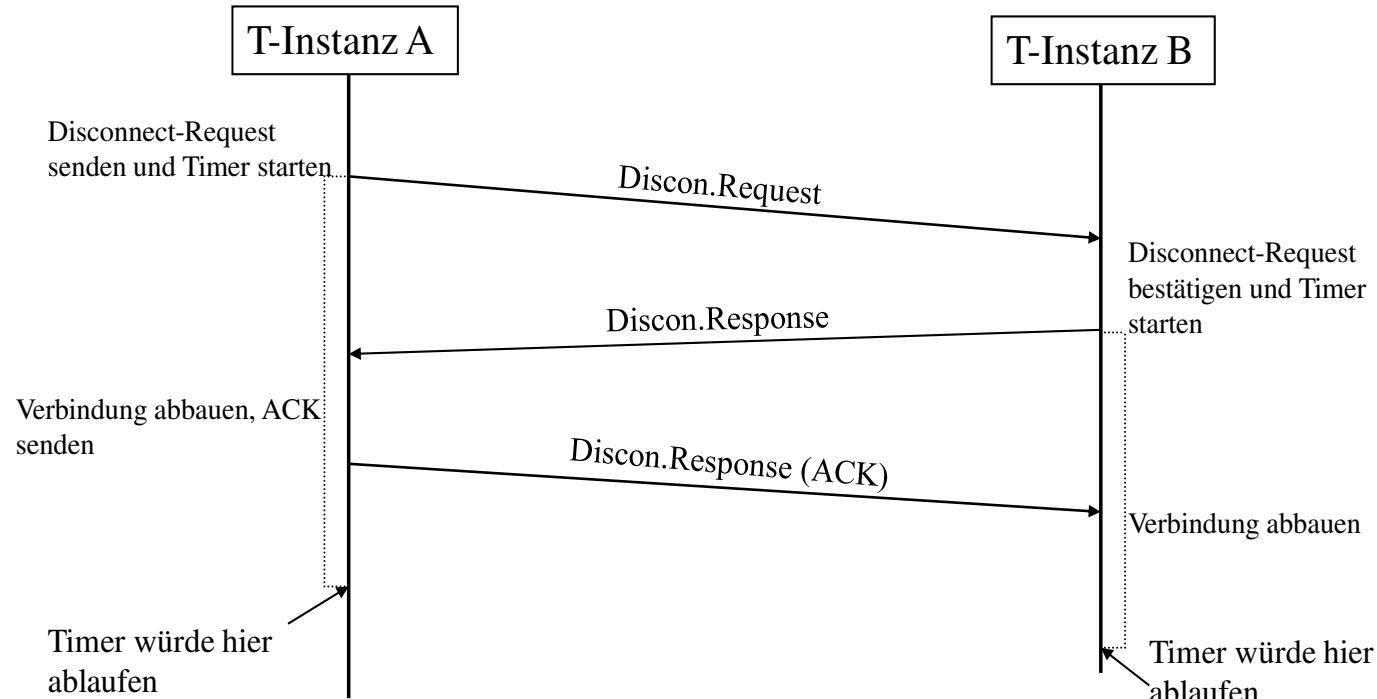
## Verbindungsabbau, Timerüberwachung

---

- Kein Protokoll ist absolut zuverlässig
- Es wird immer eine Seite geben, die **unsicher** ist, ob die letzte Nachricht angekommen ist
- Übertragen auf den Verbindungsabbau:
  - Beim Dreiwege-Handshake kann **immer** ein Disconnect-Request oder **eine Bestätigung verloren gehen**
- Praktische Lösung: **Timerüberwachung** mit begrenzter Anzahl an Nachrichtenwiederholungen
- Nicht unfehlbar, aber doch ganz zufriedenstellend

# Timerüberwachung beim Verbindungsabbau

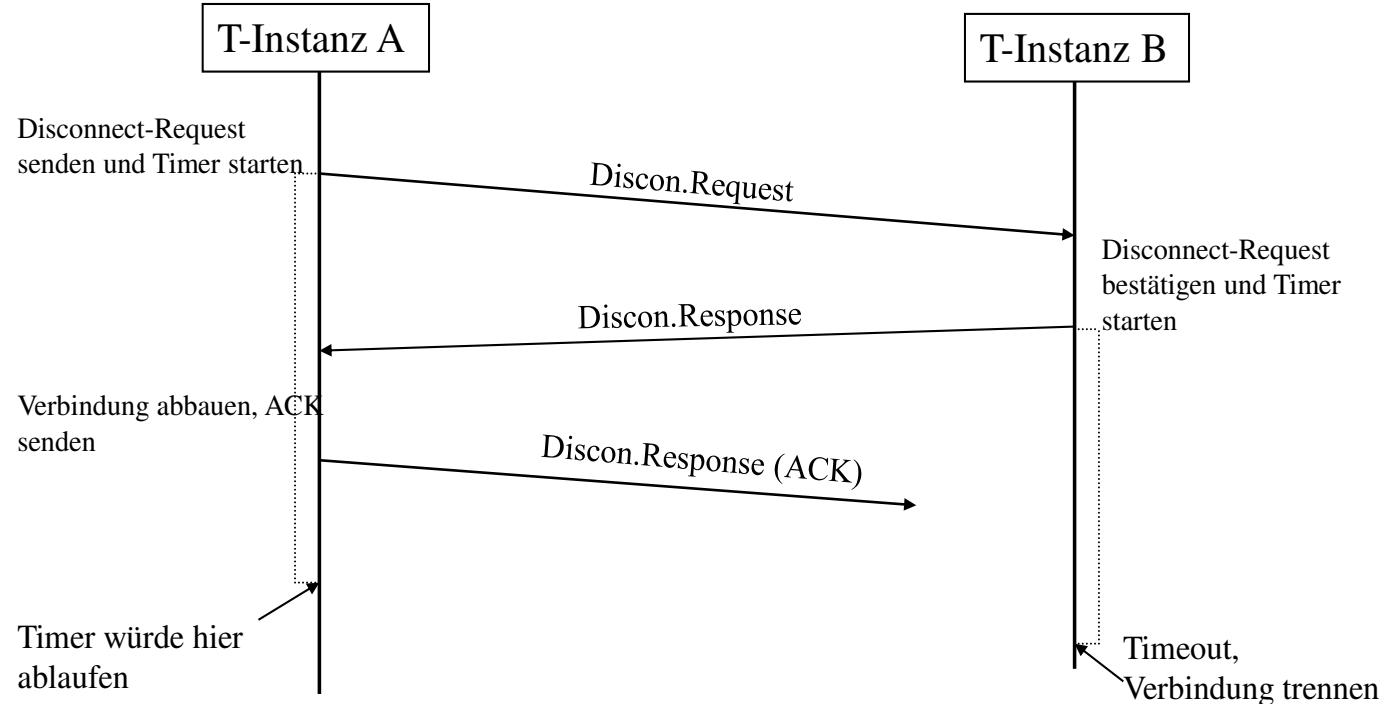
## Szenario „Normaler Ablauf“



Nach Tanenbaum, A.: et al.: Computer Networks, 5. Auflage, Pearson Studium, 2011

# Timerüberwachung beim Verbindungsabbau

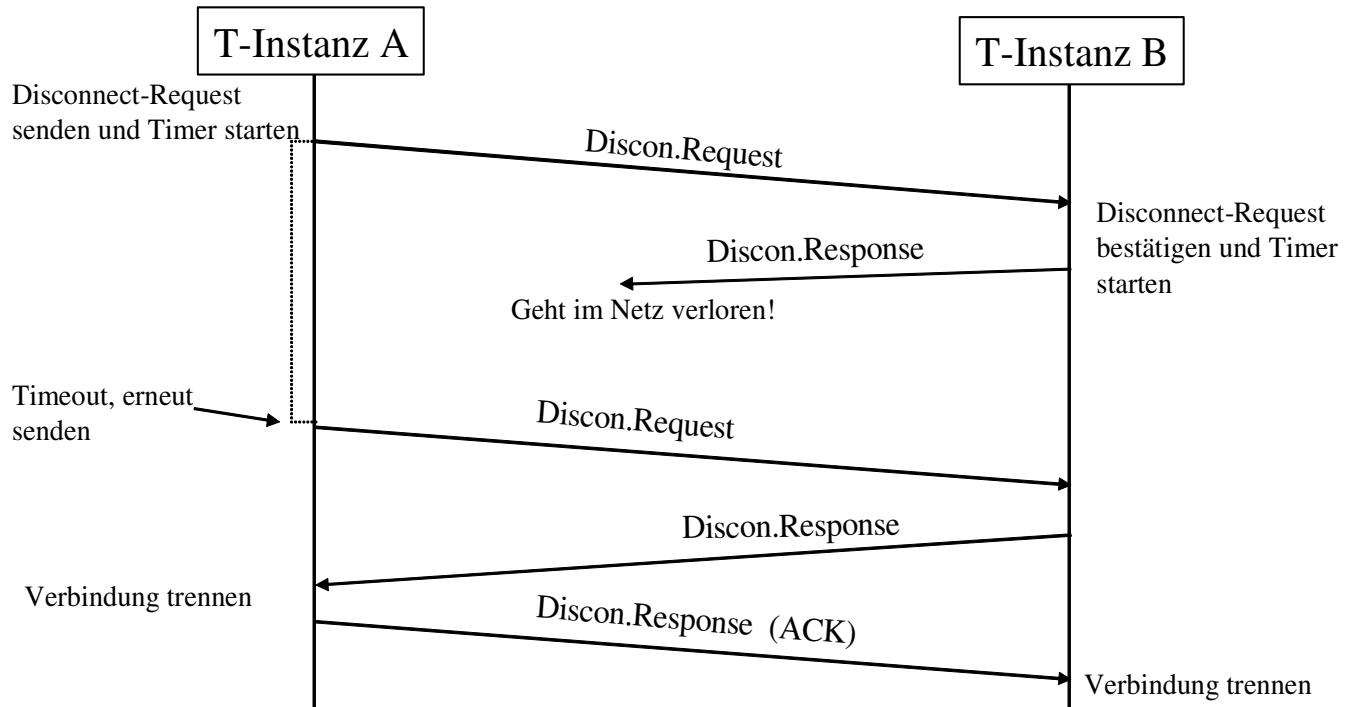
## Szenario „Timer läuft ab“



Nach Tanenbaum, A.: et al.: Computer Networks, 5. Auflage, Pearson Studium, 2011

# Timerüberwachung beim Verbindungsabbau

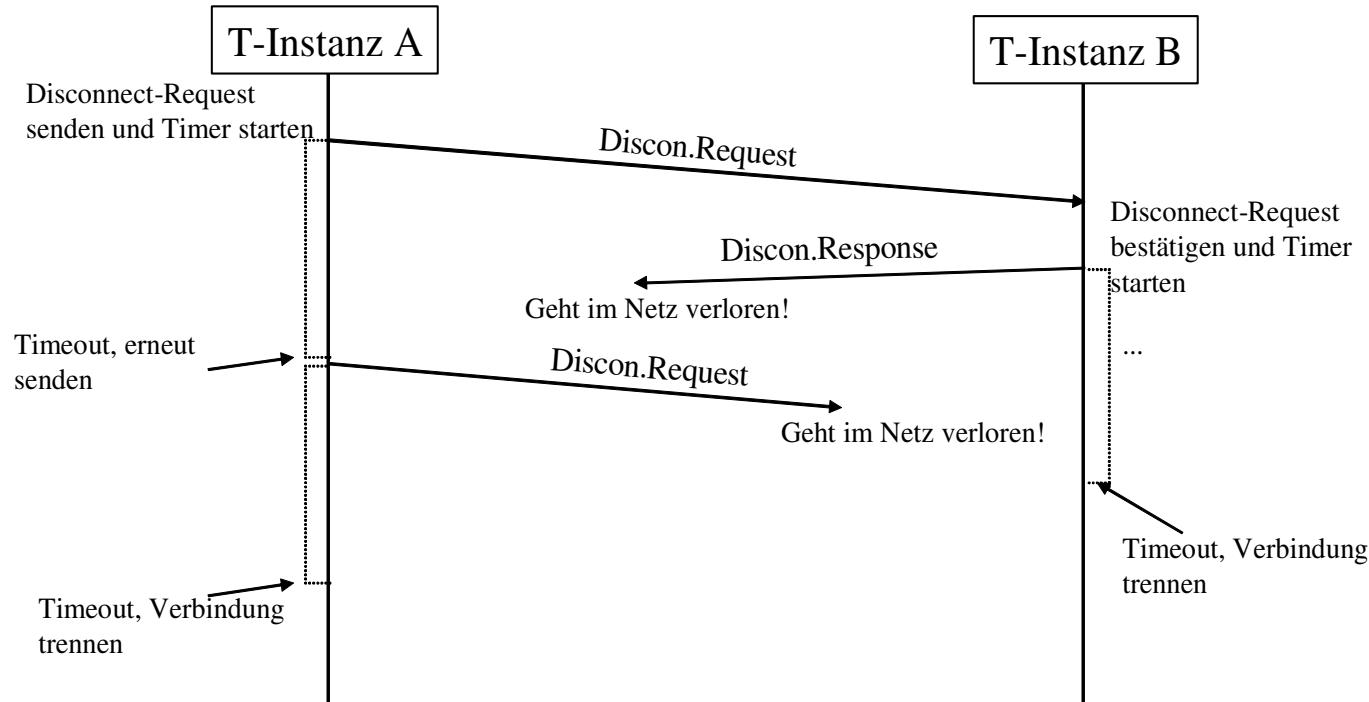
## Szenario „Disconnect-Response geht verloren“



Nach Tanenbaum, A.: et al.: Computer Networks, 5. Auflage, Pearson Studium, 2011

# Timerüberwachung beim Verbindungsabbau

## Szenario „Zwei Disconnect-PDUs gehen verloren“



Nach Tanenbaum, A.: et al.: Computer Networks, 5. Auflage, Pearson Studium, 2011

# Zuverlässiger Datentransfer

---

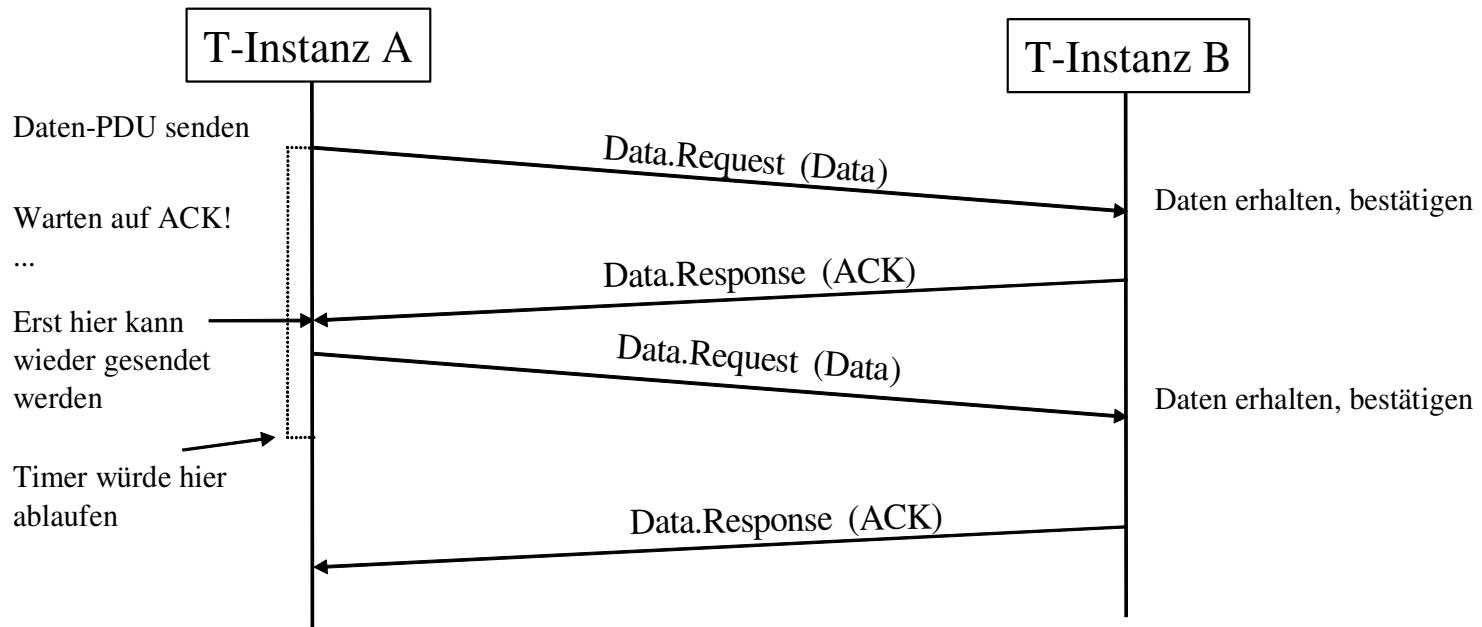
- Was heißt hier zuverlässige Datenübertragung?
  - Garantierte Ausführung ? → Nein!
  - Transaktionssicherheit → Nein!
  - Sicherstellen der Übertragung durch Quittierung und Übertragungswiederholung → **Ja!**
- Quittierungsvarianten
  - Positiv selektiv
  - Positiv kumulativ
  - Negativ selektiv
  - Kombination der Verfahren möglich
- Varianten der Übertragungswiederholung
  - Selektiv
  - Go-back-N

# Zuverlässiger Datentransfer

## Quittierungsvarianten

### ■ **Positiv selektives** Quittierungsverfahren

- Eine Quittung pro gesendeter Nachricht
- Hoher Zusatzverkehr (viele ACK-PDUs)

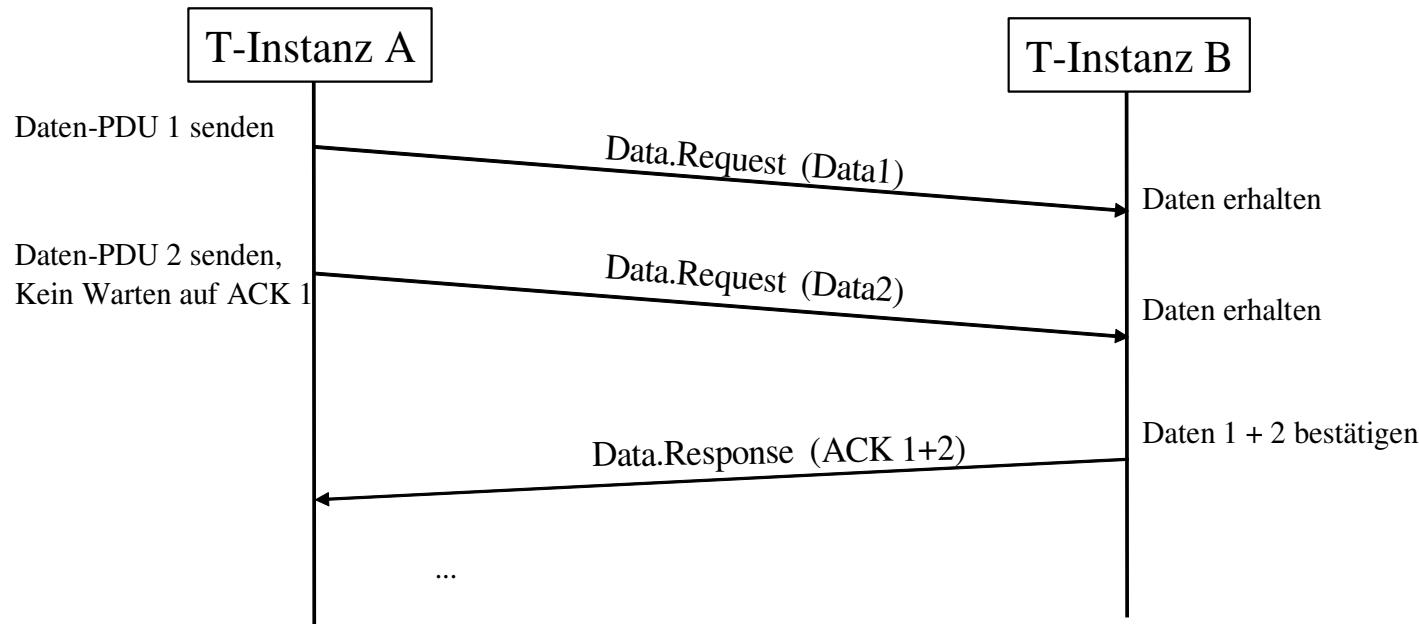


# Zuverlässiger Datentransfer

## Quittierungsvarianten

### ■ **Positiv kumulatives** Quittierungsverfahren

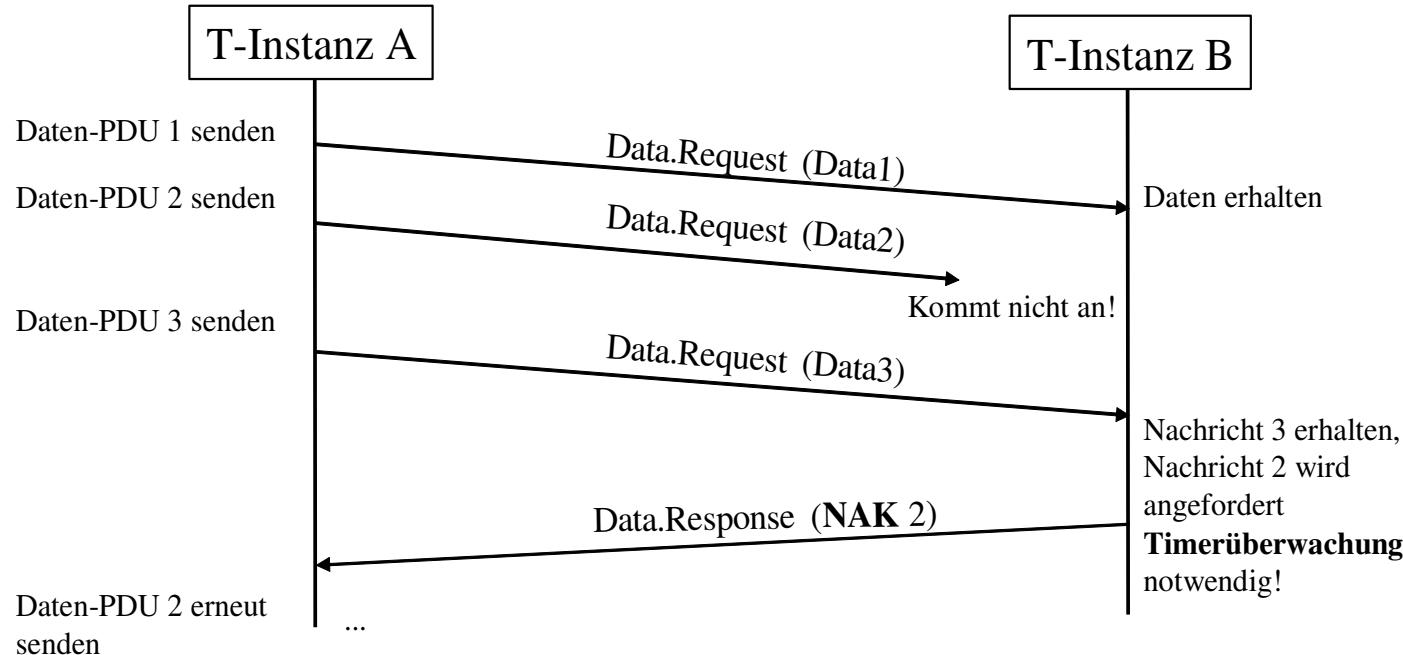
- Eine Quittung für mehrere Nachrichten
- Reduzierung der Netzlast
- Nachteil: Verspätete Information an den Sender über Datenverlust



# Zuverlässiger Datentransfer

## Quittierungsvarianten

- **Negativ selektives** Quittierungsverfahren
  - Weitere Reduzierung der Netzlast
  - Problem: Verlust von Quittungen und dessen Behandlung



# Zuverlässiger Datentransfer

## Übertragungswiederholung

---

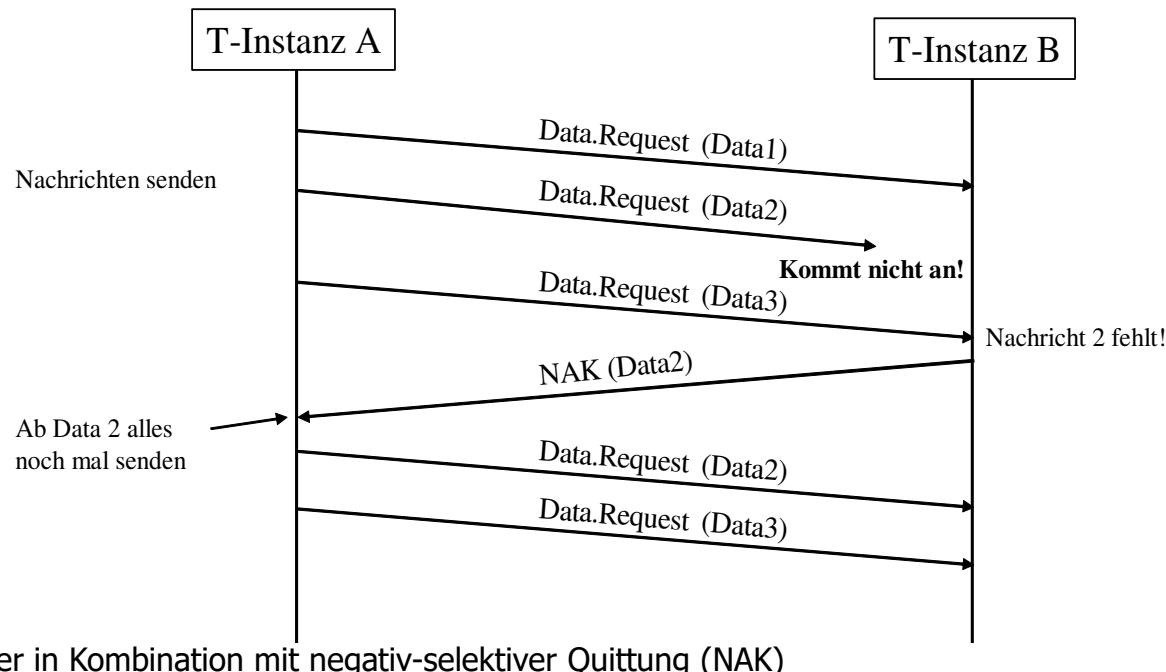
- Generell:
  - Sender muss Nachrichten über einen gewissen Zeitraum zur Übertragungswiederholung bereithalten
  - Man nennt diese Art von Protokollen auch **ARQ-Protokolle**
    - Automatic Repeat reQuest, = Automatische Wiederholungsanfrage
- **selektiv**
  - Nur die negativ quittierten Nachrichten werden wiederholt
  - Empfänger puffert die nachfolgenden Nachrichten, bis die fehlende Nachricht da ist
  - Reguläre Übertragung kann während der Wiederholung fortgesetzt werden
  - Nachteil: **Große Pufferkapazität** beim Empfänger

# Zuverlässiger Datentransfer

## Übertragungswiederholung

### ■ Go-Back-N

- Übertragungswiederholung der fehlerhaften Nachricht sowie aller nachfolgenden, die reguläre Übertragung wird unterbrochen
- Vorteil: **Geringe** Speicherkapazität beim Empfänger erforderlich.  
**Warum?**



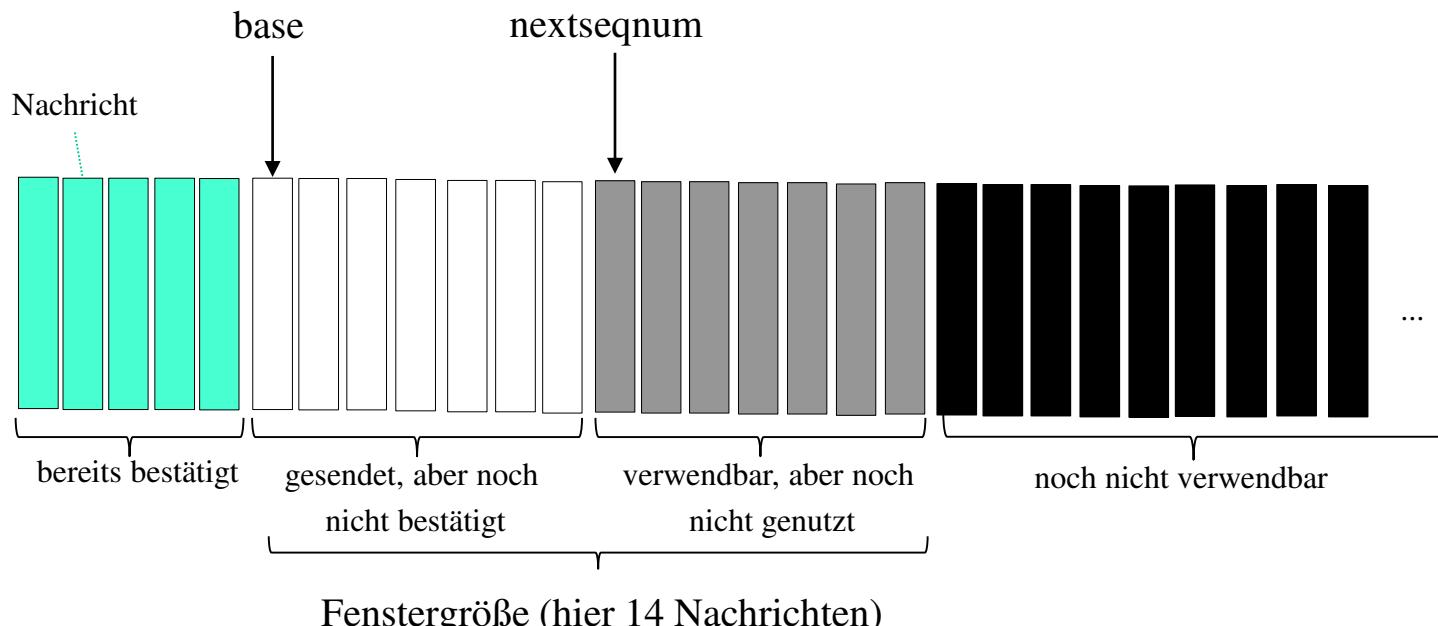
# Flusskontrolle

---

- Steuerung des Datenflusses
- Vermeidet **Überlastung des Empfängers**
- Traditionelle Verfahren sind:
  - **Stop-and-Wait**
    - Einfachstes Verfahren
    - Kopplung von Fluss- und Fehlerkontrolle
    - Nächste Nachricht wird erst nach der Quittierung gesendet
    - Fenstergröße = 1
  - **Fensterbasierte Flusskontrolle**
    - Empfänger vergibt sog. **Sendekredit**, also eine max. Menge an Nachrichten oder Bytes, die unquittiert an ihn gesendet werden dürfen
    - Empfänger kann den Sendekredit durch **positive Quittungen** erhöhen
    - Vorteil: Kontinuierlicher Datenfluss und höherer Durchsatz als bei Stop-and-Wait möglich

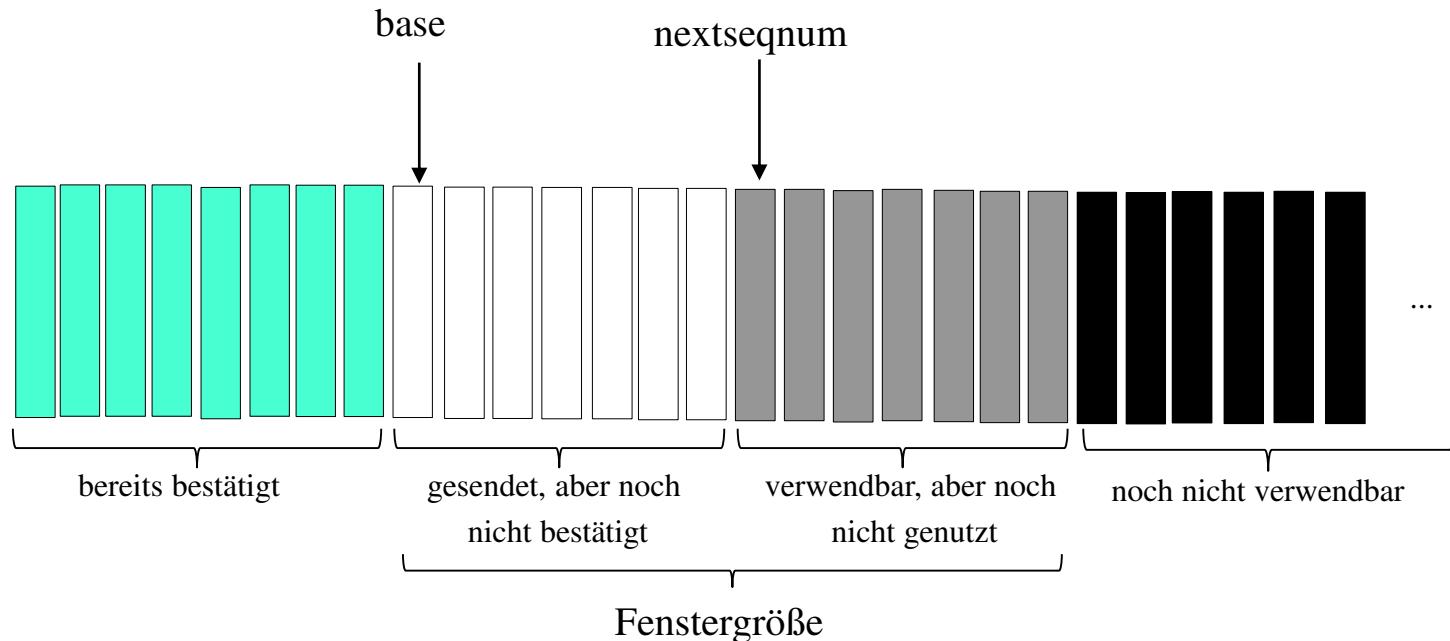
# Flusskontrolle: Fensterverwaltung

- Sliding-Window-Protokoll: Vier Folgenummernintervalle
- Bestätigung (ACK) führt zum Weiterrücken des Zeigers *base* und des Fensters



# Flusskontrolle: Fensterverschiebung bei Bestätigungen

- Drei Bestätigungen werden empfangen
- *base* und *nextseqnum* sowie das Sliding Window verschieben sich entsprechend



## Staukontrolle (Überlastkontrolle, Congestion Control)

---

- Nicht mit Flusskontrolle verwechseln
- Durch Staukontrolle sollen **Verstopfungen** bzw. Überlastungen im Netz **vermieden** werden
- Staukontrolle in der Transportschicht durch **Ende-zu-Ende-Steuerung** zwischen Endsystemen
- Staukontrolle ist ein Mechanismus mit **netzglobalen Auswirkungen**
- Beispiel später: Slow-Start-Verfahren bei TCP

# Überblick zur Transportschicht

---

- ✓ Transportzugriff
  - Sockets: Kommunikationsmodell und API
  - Socket API in Java
- ✓ Transportorientierte Protokolle
  - Verbindungsmanagement
  - Datentransferphase

- 1 Kommunikationssysteme und verteilte Anwendungen
- 2 Grundlagen der Transportschicht
- 3 **Transportprotokolle TCP und UDP**
- 4 Grundlagen der Vermittlungsschicht
- 5 Internet und Internet Protocol (IP)
- 6 Routing-Verfahren und -Protokolle
- 7 Internet-Steuerprotokolle und DNS
- 8 Internet Protocol Version 6 (IPv6)
- 9 Netzwerkschnittstelle

# Überblick über das Kapitel

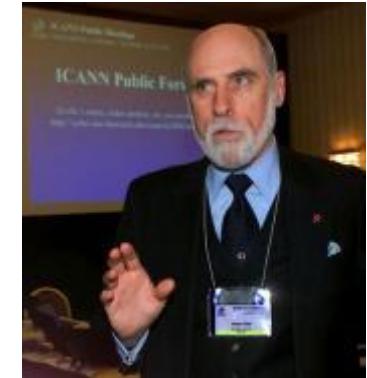
---

- 1. Einordnung und Aufgaben von TCP**
2. Der TCP-Header
3. Verbindungsauf- und -abbau, Datenübertragungsphase
4. Sliding-Window-Mechanismus
5. Optimierungen: Algorithmen von Nagle und Clark, Silly Window Syndrom
6. Staukontrolle
7. TCP-Timer
8. TCP-Zustandsautomat
9. UDP (User Data Protocol)

Robert E. Kahn



Vinton G. Cerf

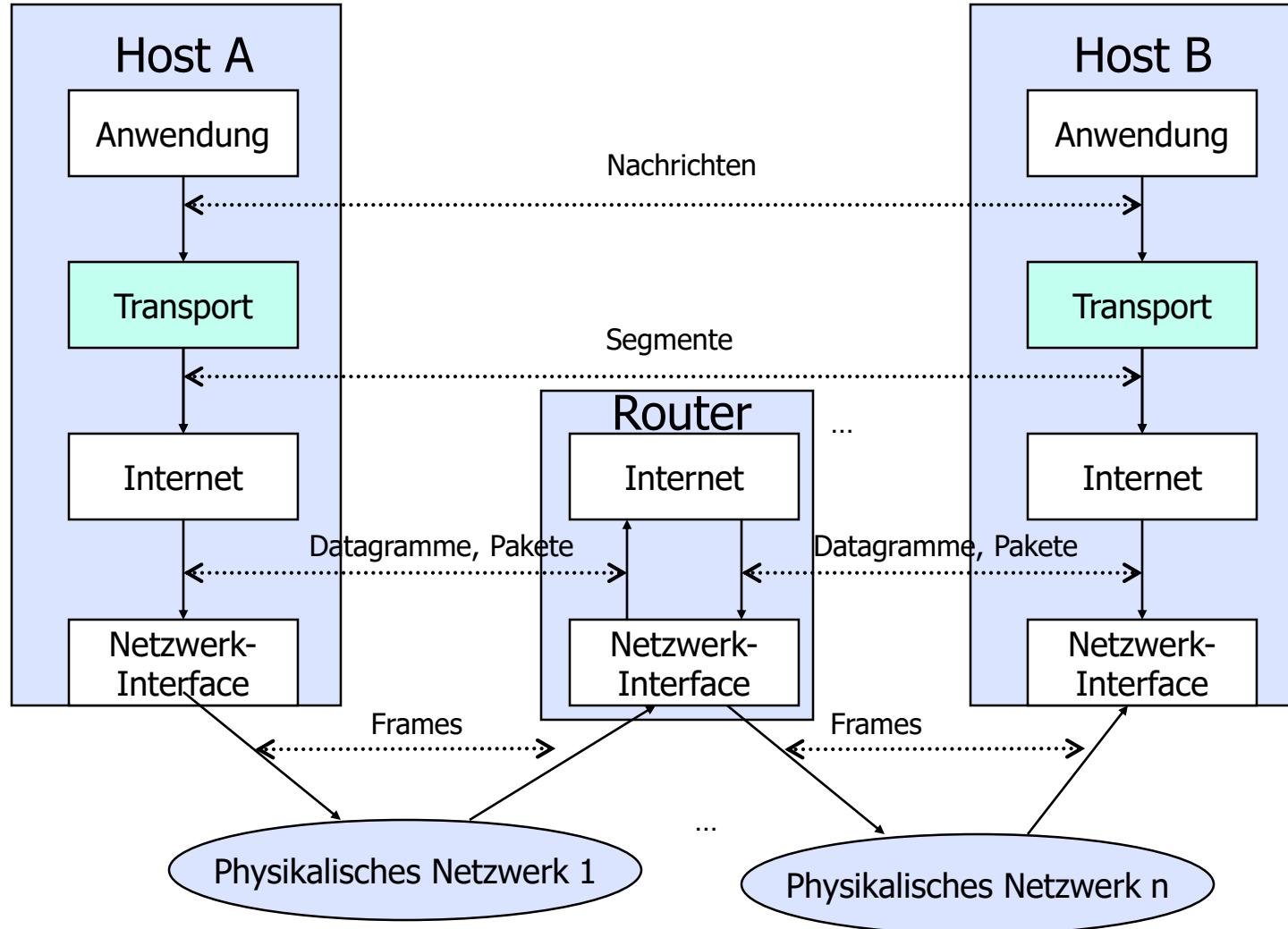


Geschichte:

Beide erhielten den Turing Award 2004

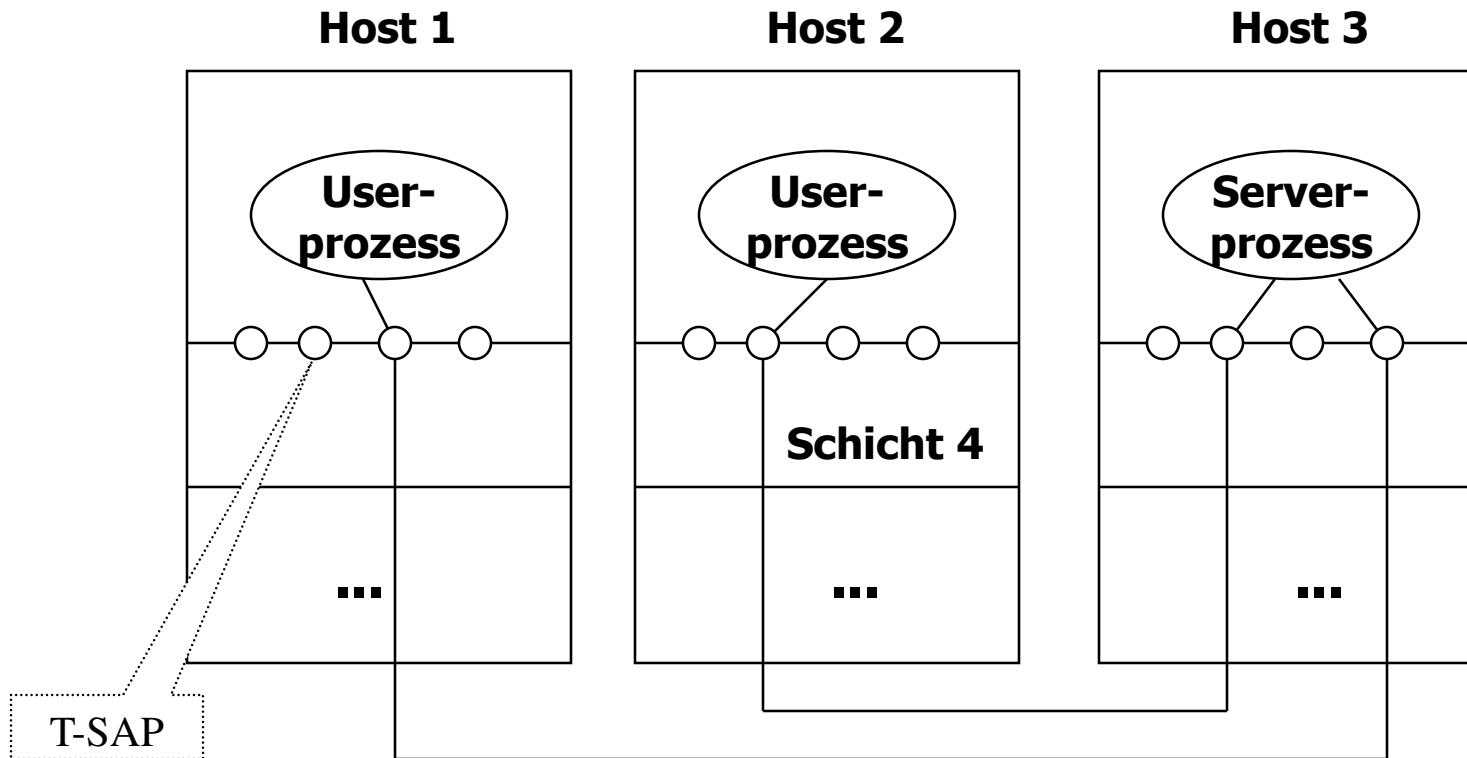
- TCP (auch IP) wurde von Robert E. Kahn und Vinton G. Cerf ab 1972 entwickelt (mehrere Jahre)
- Erste Standardisierung von TCP 1981 im RFC 793

# TCP/IP-Referenzmodell



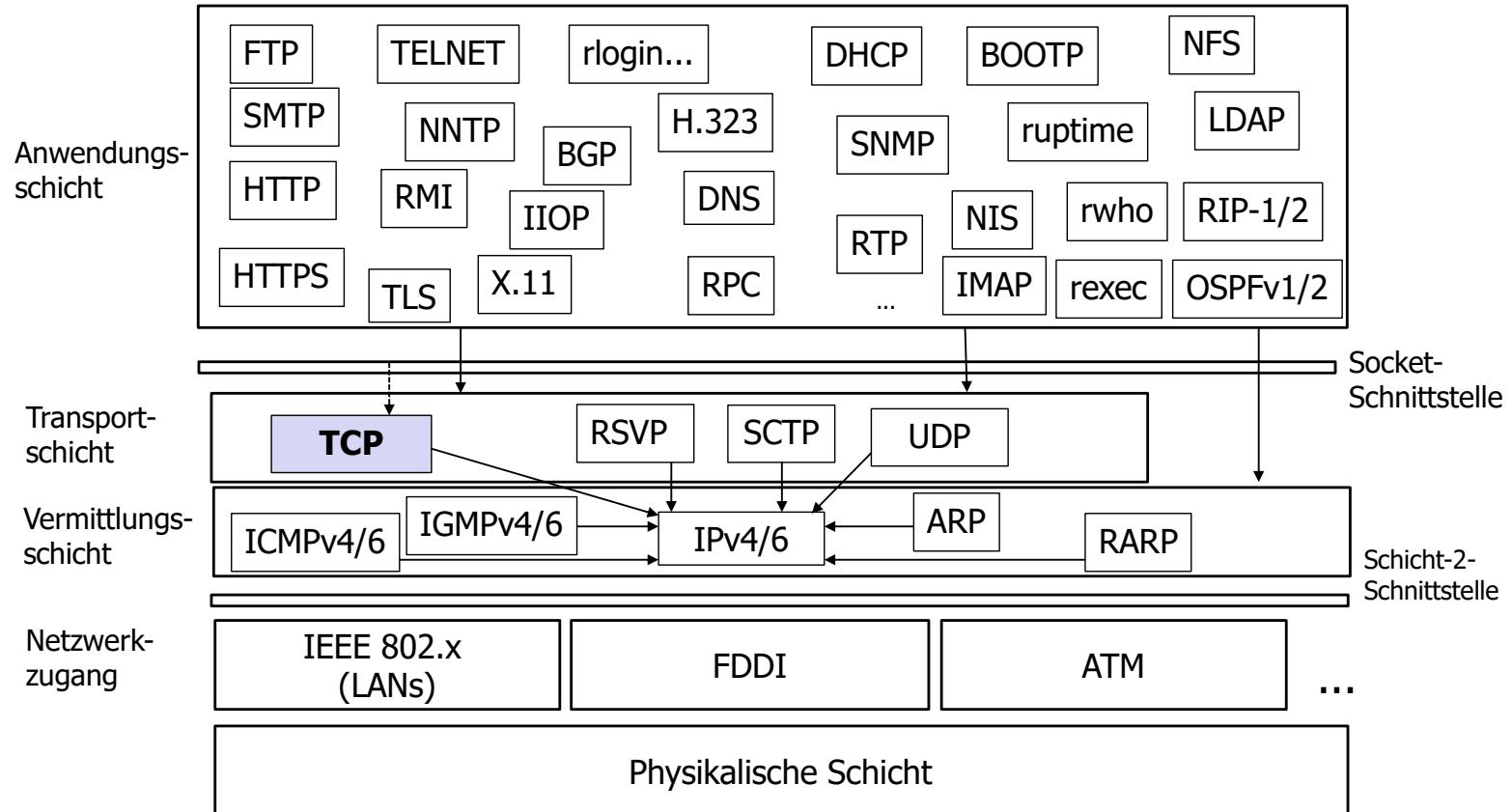
# Ende-zu-Ende-Kommunikation

- TCP ermöglicht eine **Ende-zu-Ende Beziehung** zwischen kommunizierenden Anwendungsinstanzen

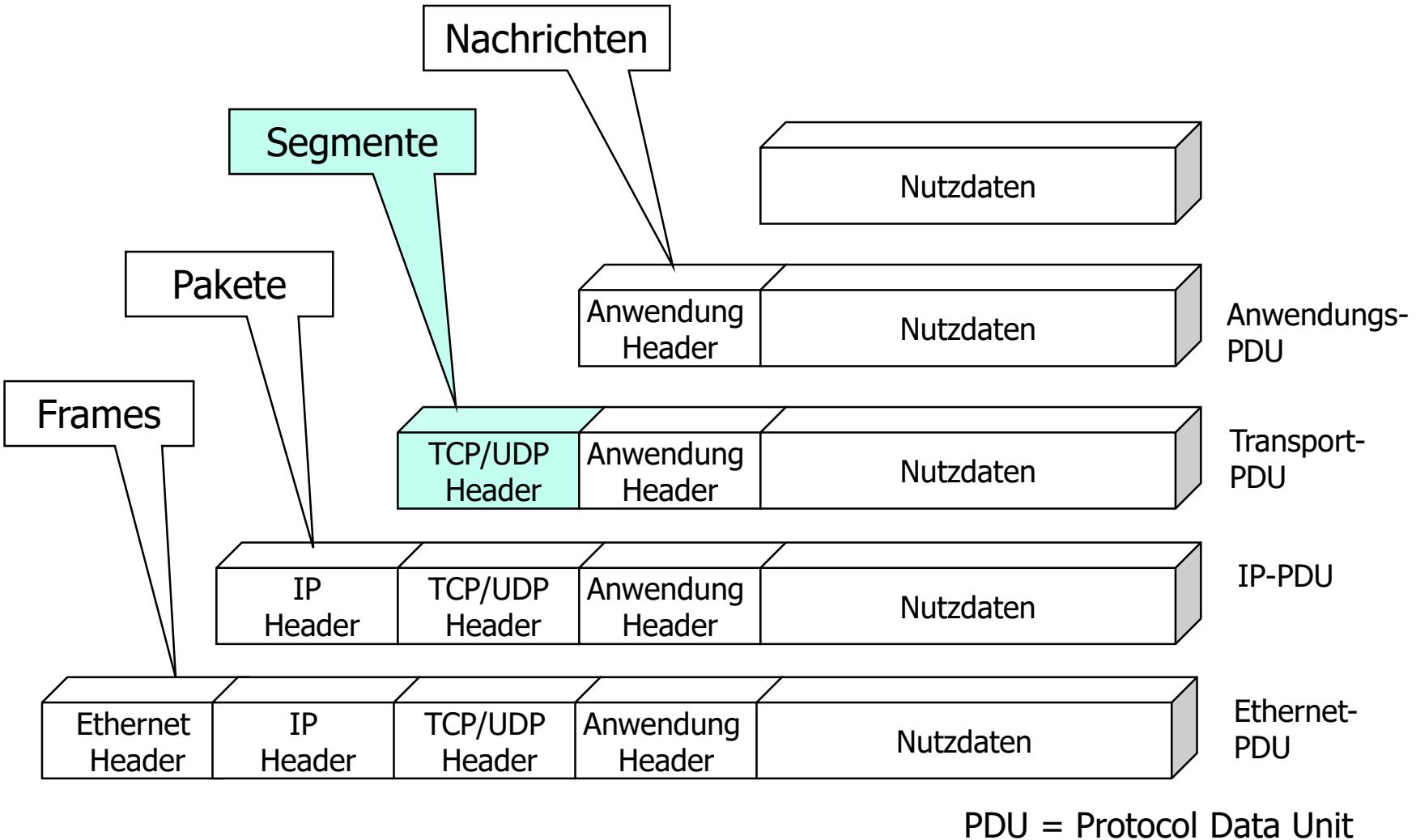


Nach Tanenbaum, A.; Wetherall, D.: Computer Networks, 5. Auflage, Pearson Studium, 2011

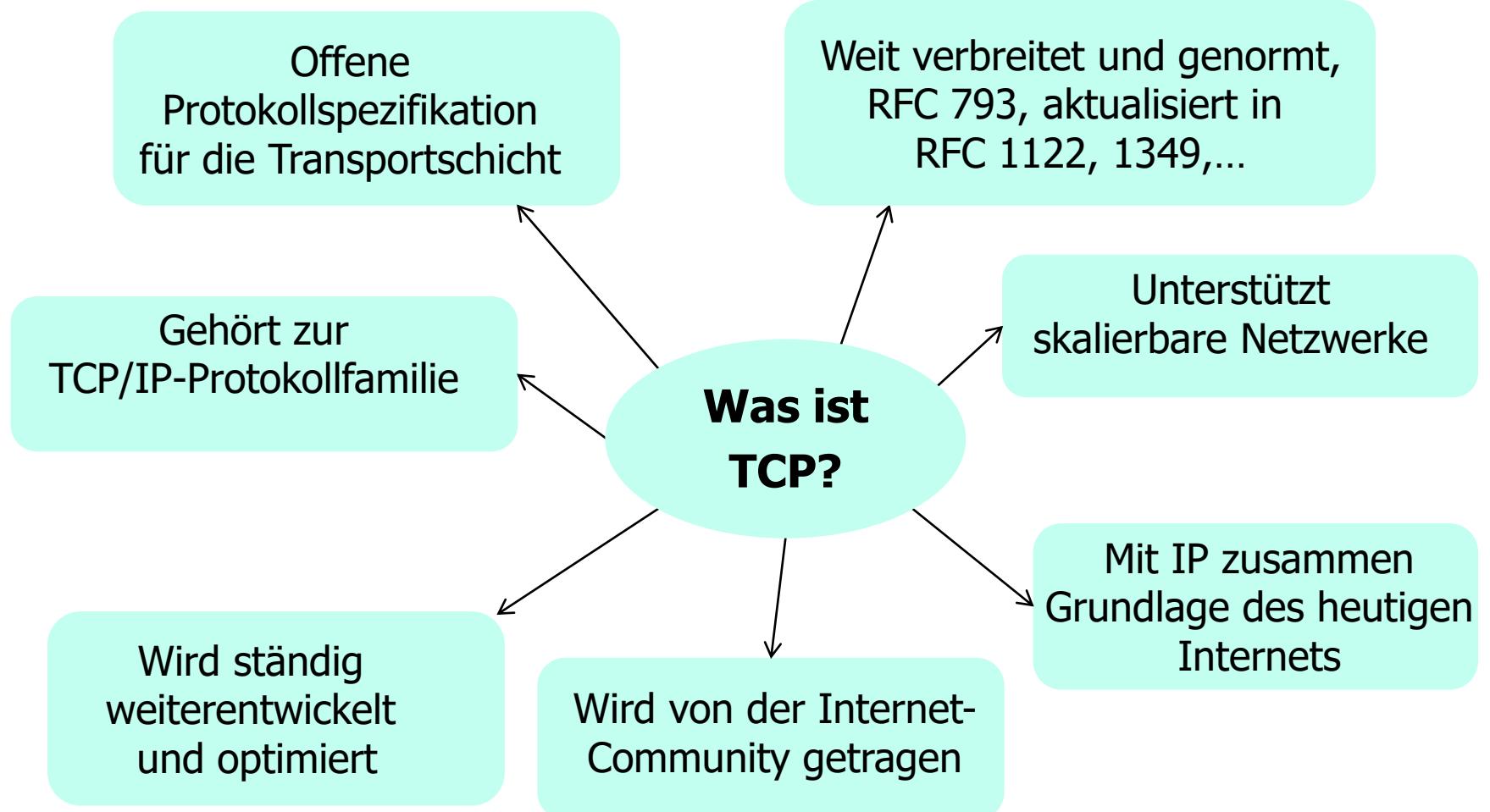
# Erinnerung: TCP/IP-Protokollfamilie



# Erinnerung: TCP/IP-Referenzmodell, Protokollkapselung

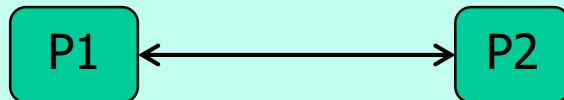


# Allgemeines zu TCP



# TCP-Aufgaben im Detail

Gesicherte Ende-zu-Ende Verbindung,  
vollduplexfähig



Reihenfolgegarantie



Segmentierung der  
Nachrichten

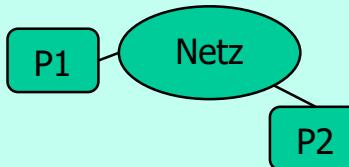


## Aufgaben von TCP

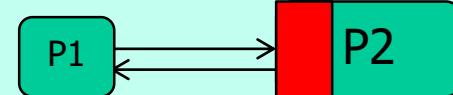
Garantierte Auslieferung  
ohne Duplikate



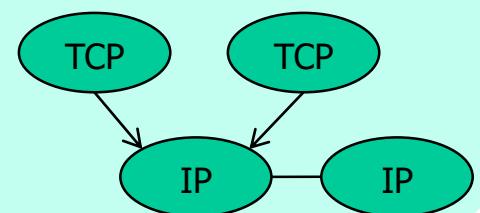
Staukontrolle



Flusskontrolle, Sliding-  
Window-Algorithmus



Multiplexing und  
Demultiplexing  
der IP-Verbindung



# TCP-Maßnahmen zur Sicherung der Übertragung

Maßnahme in TCP	Beschreibung
Dreiwege-Handshake	Verbindungsaufbau mit Synchronisation der Verbindungskontexte
Drei-/Vierwege-Handshake	Geordneter Verbindungsabbau
Verwendung von Sequenznummern	Nummerierung übertragener Bytes für die Reihenfolgeüberwachung
Verwendung einer Prüfsumme	Einfache Fehlererkennung bei der Datenübertragung
Verwendung von positiv-kumulativ ACK und implizites NAK	Quittierung mehrerer Segmente auf einmal, implizites NAK als Spezialverfahren (später mehr)
Go-Back-N oder positiv-selektiv	Nachrichtenwiederholung seit letztem Verlust, alternativ auch selektive Wiederholung
Verwendung des Sliding-Window-Verfahrens	Flusskontrolle zum Schutz des Empfängers vor Überlauf
Verwendung des Start-Slow-Verfahrens und weiterer	Staukontrollmechanismus

# Dienste, Ports und Adressierung

---

- Anwendungsprozess kommuniziert über eine Adresse, die als **Socket** bezeichnet wird
  - Tupel der Form (IP-Adresse, TCP-Portnummer)
- **Well-known Ports**
  - 16-Bit-Integer
  - Es gibt hier eine Reihe von sog. well-known Ports für reservierte Services (Portnr.  $\leq 1024$ )
  - Jeder Dienst hat eine eigene Portnummer
    - siehe Datei /etc/services unter Unix oder
    - C:\windows\System32\drives\etc unter Windows
- Ein Anwendungsprozess kann auch mehrere Verbindungen unterhalten
- Client-Server-Prinzip leicht realisierbar:
  - Server wartet an einem Port auf Connect-Requests

# Dienste, Ports und Adressierung

---

- Eine Verbindung wird durch ein Paar von Endpunkten identifiziert (**Socket Pair**)
  - Dadurch ist es möglich, dass **ein TCP-Port** auf einem Host für **viele Verbindungen** genutzt werden kann
  - Beispiel:
    - HTTP-Port 80 wird für viele Verbindungen eines HTTP-Servers verwendet. TCP-Verbindungen aus Sicht des HTTP-Servers sind:
      - ((195.214.80.76, **80**) (196.210.80.10,6000))
      - ((195.214.80.76, **80**) (197.200.80.11,6001))
      - ...
- wobei 195.214.80.76 die IP-Adresse des Servers ist.

# Vorgaben für Portnutzung

## ■ ICANN-Vorgabe für die Ports:

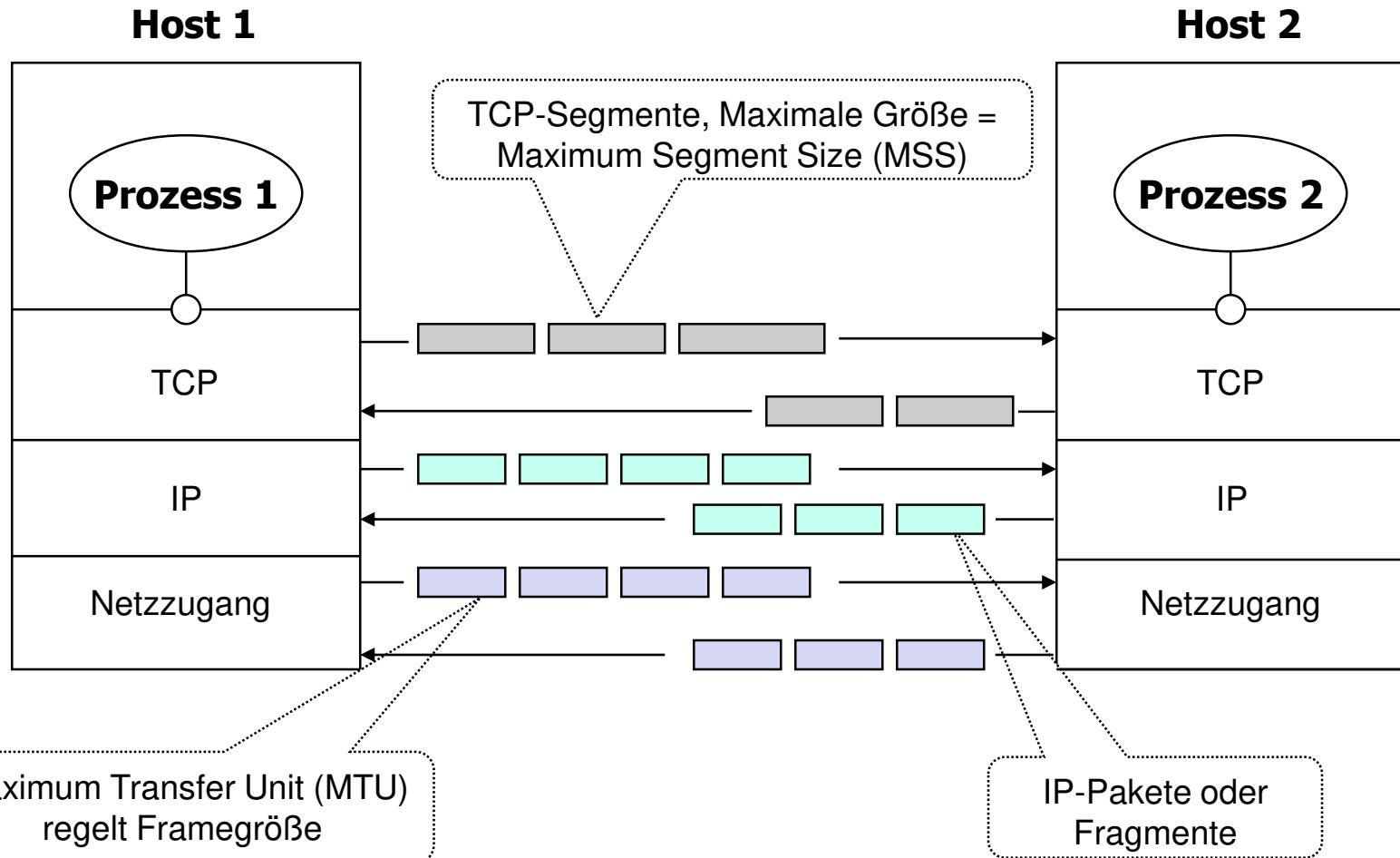
- Well known Ports (0 – 1023)
- Registered Ports (1024 – 49151)
  - Ports, die Hersteller für Anwendungen reservieren können
- Dynamische und/oder private Ports (49152 – 65535)
  - Private Ports, beliebig vergeben
- Siehe Datei <http://www.iana.org/assignments/port-numbers>

ICANN: Internet Corporation for Assigned Names and Numbers



<b>Well known TCP-Ports, Beispiele</b>	<b>Protokoll, Service</b>
23	Telnet – Remote Login
20,21	ftp – File Transfer Protocol
25	SMTP – Simple Mail Transfer Protocol
80, 443	HTTP, HTTPS

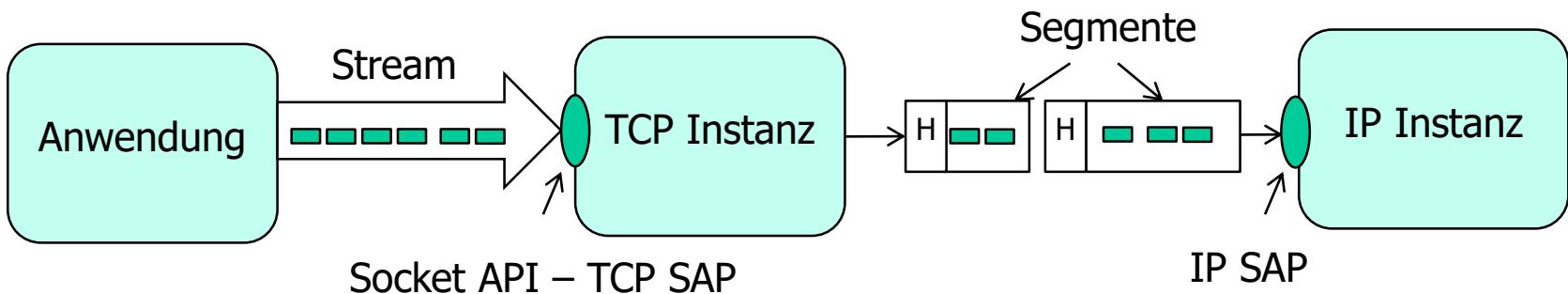
# TCP-Segmente



Byte-Strom wird in TCP-Segmente gepackt, TCP-Segmentierung != IP-Fragmentierung

# Länge der TCP-Segmente (1)

- TCP bekommt von der Anwendungsschicht über die Socket-Schnittstelle einen Datenstrom als eine Sequenz von Octets (Bytes)
- Die TCP-Instanz unterteilt diesen Datenstrom zur Übertragung in **Segmente** (TCP-Segmente)
- Die Segmente haben eine maximale Länge, die als **Maximum Segment Size (MSS)** bezeichnet wird.

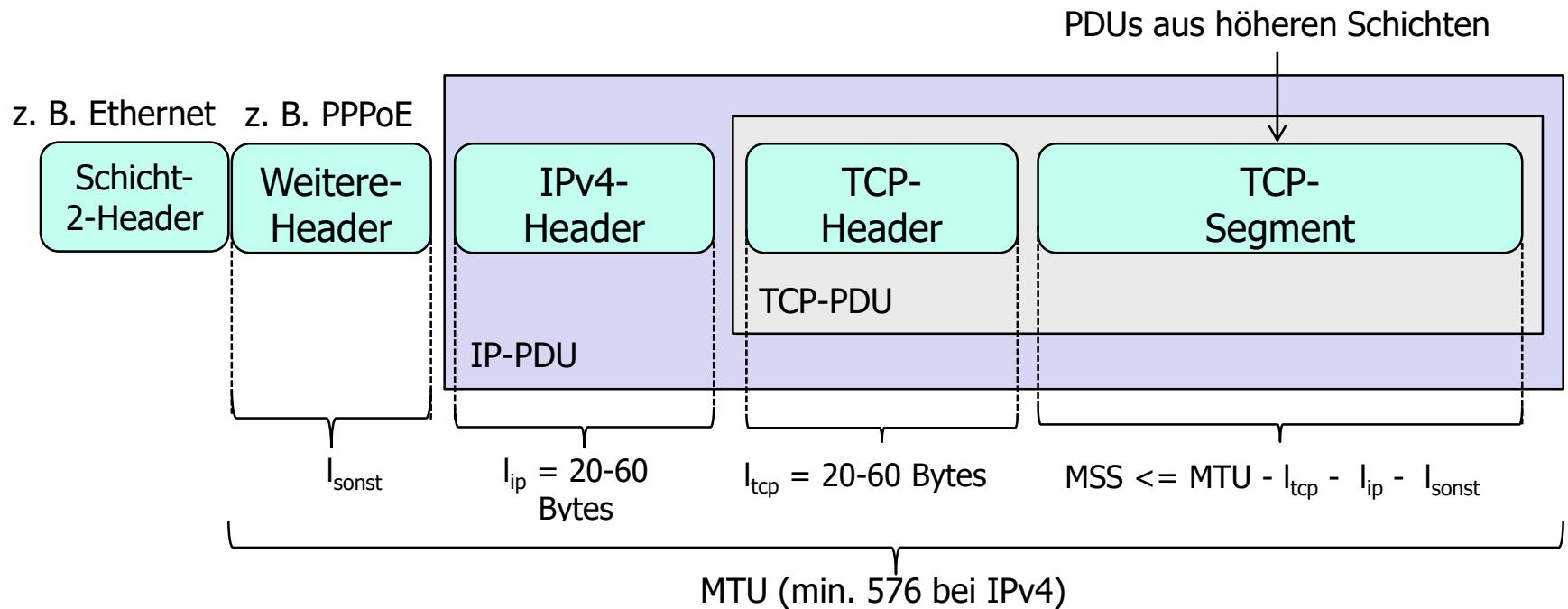


## Länge der TCP-Segmente (2)

---

- Wie viele Nutzdaten sollen in einem Segment übertragen werden, wie groß darf also die MSS sein (siehe RFC 879)?
  - Die maximale Segmentgröße (MSS) ist 64 KiB ( $2^{16}$  Bytes)
  - Hosts dürfen nicht mehr als in einem Frame nicht mehr als eine *Maximum Transfer Unit* senden (**MTU=576 Octets**), wenn sie nicht mehr über die Kommunikationsverbindung wissen
  - Mehr herausfinden über MTU Path Discovery (später)
- Wie wird die MSS festgelegt?
  - Die MSS kann optional **beim Verbindungsauftbau** vereinbart werden. Beide Seiten können einen Vorschlag machen, der kleinere Wert wird verwendet
  - Nutzung der MSS-Option im Header hat 16 Bit zur Vereinbarung der MSS
  - Nicht verwechseln mit Window-Größe für Flusskontrolle, verlängerbar über WSOpt-Option, bis 1 GB (später mehr)

## Länge der TCP-Segmente (3)



- Ursprüngliche Standardannahme für Datensegmente (oft ohne Optionen)
  - $\text{MSS} = \text{MTU} - 40$ ; (20 Bytes für IPv4-Header, 20 für TCP-Header) = **536**
  - Evtl. müssen noch weitere Protokoll-Header abgezogen werden (z. B. 8 Bytes für PPPoE-Protokoll in VDSL-Netzzugängen)
  - Besser: Kleiner, da heute oft Optionen auch in der Datenübertragung, (siehe TSopt → 12 Bytes) enthalten sind

## Länge der TCP-Segmente (4)

---

- Ergänzung zu IPv6
  - Bei IPv6 ist das zu unterstützende MTU-Minimum = 1280 Octets
  - $\text{MTU} \geq \text{MSS} + \text{TCP-Headerlänge} + \text{IPv4-Headerlänge} + \text{Länge weiterer Header}$  (z.B. 8 Bytes für PPP)  
 $\rightarrow 1280 \geq \text{MSS} + 20 + 40 + 8 \rightarrow 1212 \geq \text{MSS}$
  - IPv6 unterstützt auch Jumbograms mit 32 Bit Längenfeld  
 $\rightarrow$  TCP-Anpassung erforderlich

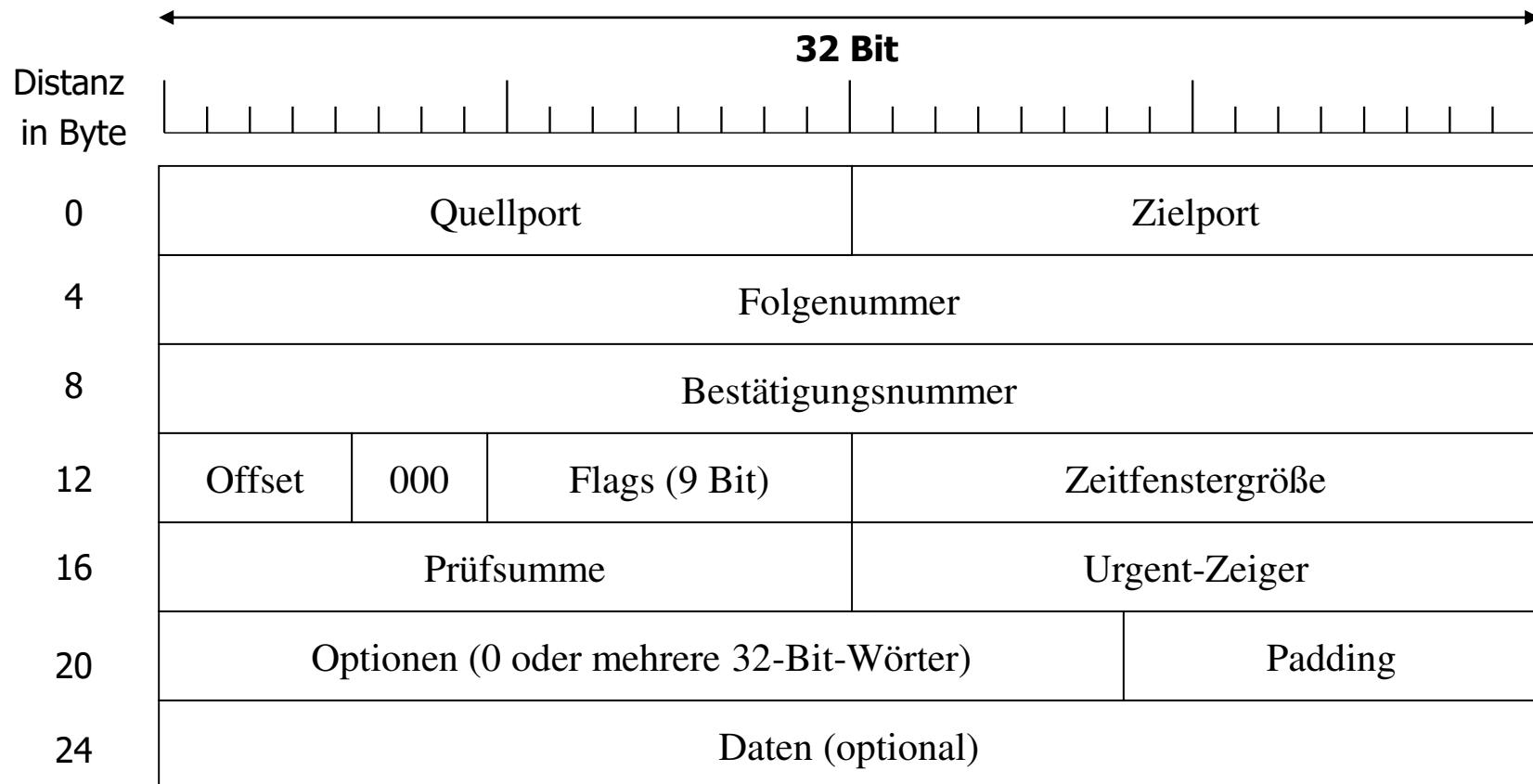
# Überblick über Transportprotokolle

---

1. Einordnung und Aufgaben von TCP
- 2. Der TCP-Header**
3. Verbindungsauf- und -abbau, Datenübertragungsphase
4. Sliding-Window-Mechanismus
5. Optimierungen: Algorithmen von Nagle und Clark, Silly Window Syndrom
6. Staukontrolle
7. TCP-Timer
8. TCP-Zustandsautomat
9. UDP (User Data Protocol)

# TCP-Header (PCI, Protocol Control Information)

---



# TCP-Header (1)

---

- **Quell- und Zielport**
  - Portnummer des Anwendungsprogramms des Senders und des Empfängers
- **Folgenummer = Sequenznummer**
  - Nächstes Byte innerhalb des TCP-Streams ( $\text{mod } 2^{32}$ )
- **Bestätigungsnummer**
  - Gibt das als nächstes erwartete Byte im TCP-Strom an und bestätigt damit den Empfang aller vorhergehenden Bytes
- **Offset**
  - Gibt die Länge des TCP-Headers in 32-Bit-Worten an
- **Reserviert**
  - Hat noch keine Verwendung
- **Optionen:** werden später behandelt

## TCP-Header (2): Flags

---

- Hier handelt es sich um Kontroll-Bits mit unterschiedlicher Bedeutung
- Hinweis: Neue TCP-Varianten definieren weitere Flags für die Staukontrolle:
  - ECE-Flag (Explicit Congestion Notification) und CWR-Flag (Congestion Window Reduced)

Flag	Bedeutung
URG	Urgent-Zeiger-Feld ist gefüllt
ACK	Bestätigung (z.B. bei Verbindungsaufbau genutzt), d.h. die ACK-Nummer hat einen gültigen Wert
PSH	Zeigt Push-Daten an, Daten dürfen beim Empfänger nicht zwischengespeichert werden, sondern sind sofort an den Empfängerprozess weiter zu leiten: Sender sendet sofort und Empfänger stellt sofort zu
RST	Dient zum <ul style="list-style-type: none"><li>- Rücksetzen der Verbindung (sinnvoll z.B. bei Absturz eines Hosts)</li><li>- Abweisen eines Verbindungsaufbauwunsches</li><li>- Abweisen eines ungültigen Segments</li></ul>
SYN	Wird genutzt beim Verbindungsaufbau
FIN	Wird genutzt beim Verbindungsabbau

# TCP-Header (3)

---

## ■ Zeitfenstergröße

- Ermöglicht es dem Empfänger, seinem Partner mitzuteilen, wie groß der vorhandene Pufferplatz (in Byte) zum Empfang der Daten noch ist (Flusskontrolle!)
- Gilt nur in Verbindung mit ACK-Flag

ACK

Zeitfenstergröße

## ■ Urgent-Zeiger

- Beschreibt die Position in den Nutzdaten (Byteversatz von der aktuellen Folgenummer ab), an der dringliche Daten vorgefunden werden können
- Gilt nur in Verbindung mit URG-Flag
- Diese Daten werden vorrangig behandelt

Siehe RFC 6093: On the Implementation of the TCP Urgent Mechanism

URG

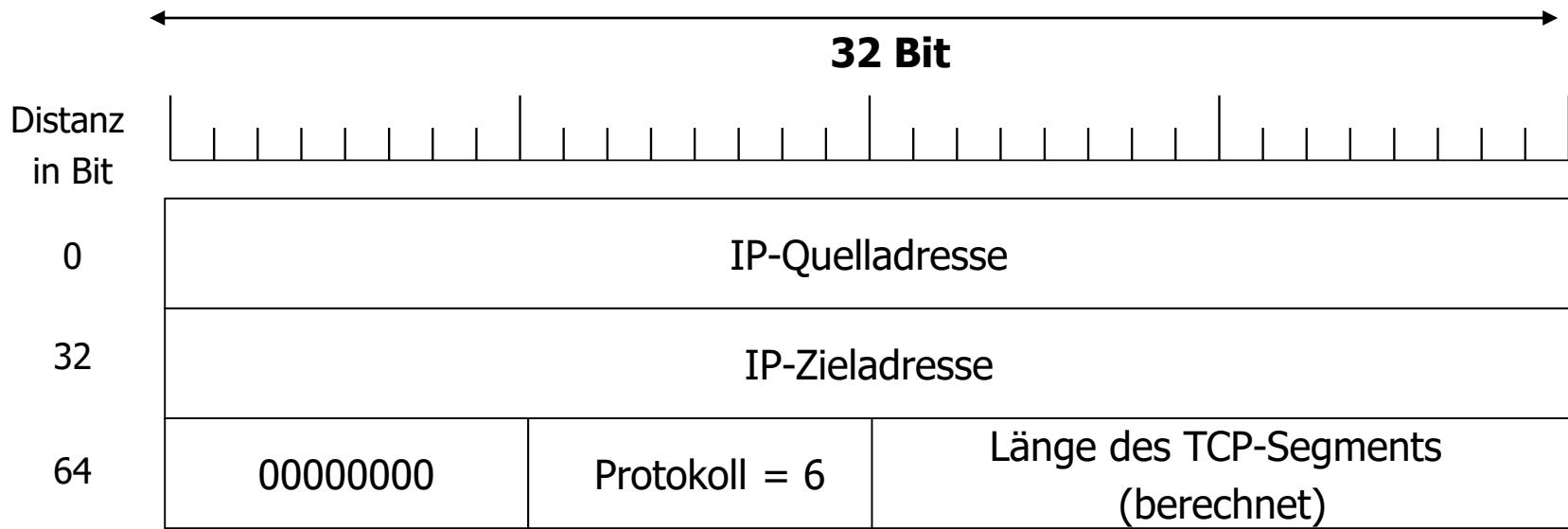
Urgent-Zeiger

# TCP-Pseudoheader

---

- **Pseudoheader:**

- Wird vor der Berechnung der Prüfsumme an das TCP-Segment (vor den TCP-Header) gehängt , aber nicht mit übertragen
- Aufbau (96 Bit):

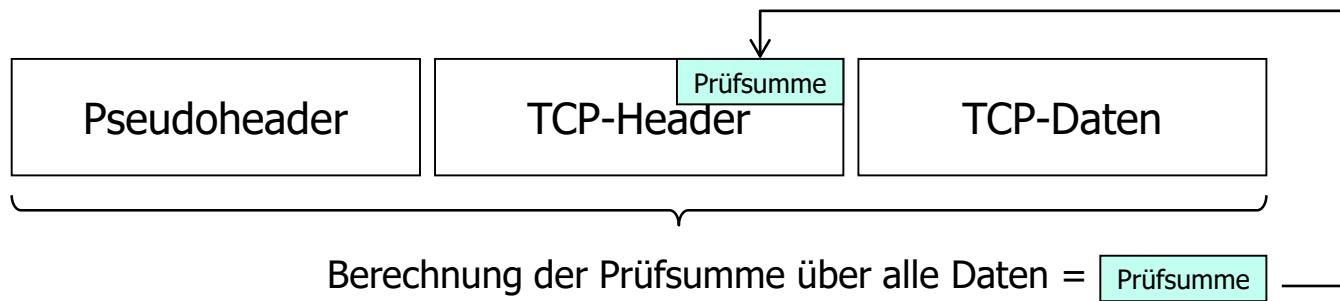


# TCP-Pseudoheader und Prüfsumme

---

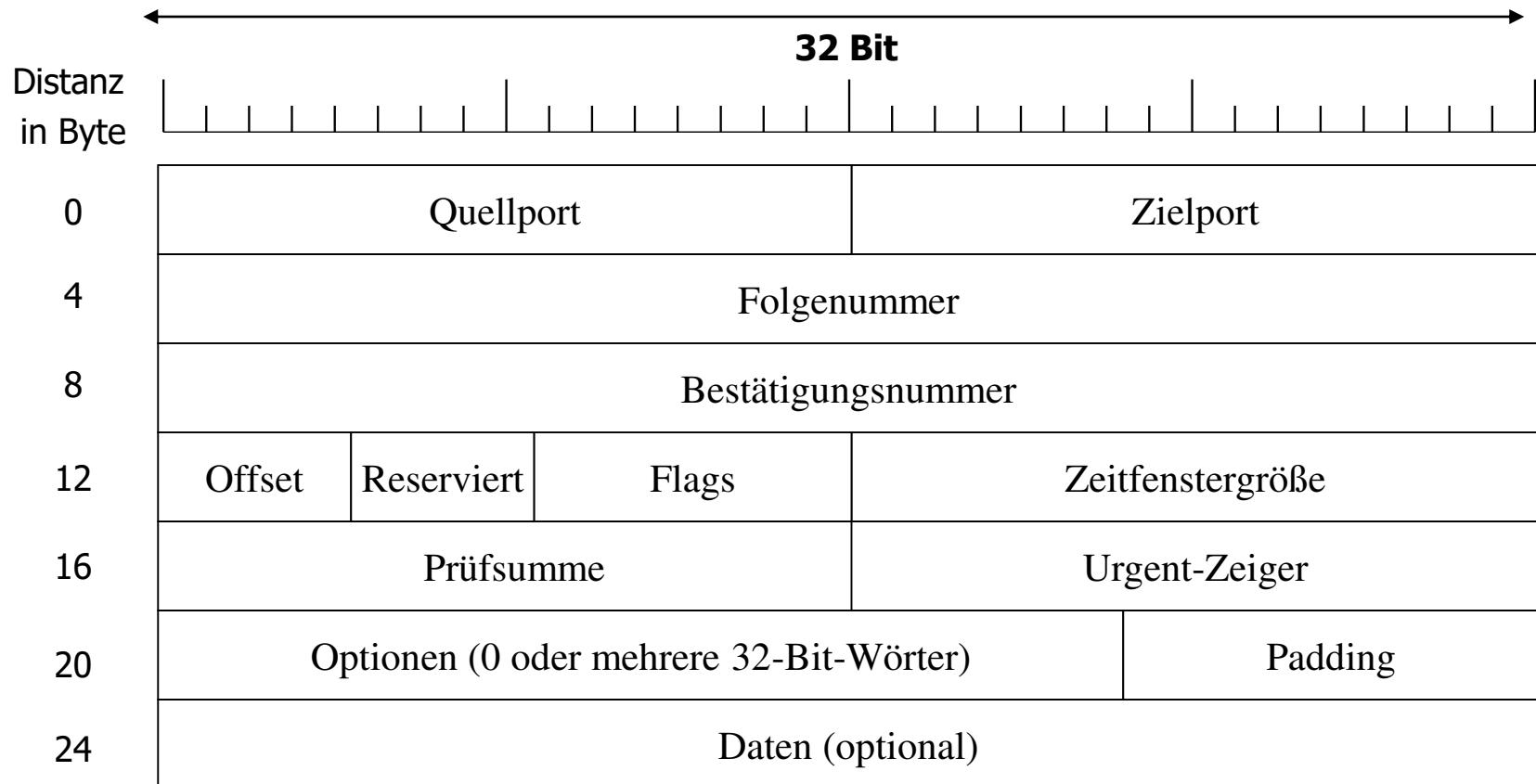
## ■ Prüfsumme

- Verifiziert das TCP-Segment (Header + Daten) inkl. eines Pseudoheaders auf Basis eines einfachen Prüfsummenalgorithmus:
  - Prüfsumme im Header auf Null setzen
  - Nutzdaten ggf. auf gerade Byteanzahl mit Nullbytes ergänzen
  - Für alle 16-Bit-Wörter des TCP-Segments (inkl. TCP-Header und sog. Pseudo-Header) die Summe berechnen
  - Danach Bildung des Einer-Kompliments der Summe ergibt die Prüfsumme



# TCP-Header (PCI, Protocol Control Information)

---



# Überblick über Transportprotokolle

---

1. Einordnung und Aufgaben von TCP
2. Der TCP-Header
- 3. Verbindungsauflauf- und -abbau, Datenübertragungsphase**
4. Sliding-Window-Mechanismus
5. Optimierungen: Algorithmen von Nagle und Clark, Silly Window Syndrom
6. Staukontrolle
7. TCP-Timer
8. TCP-Zustandsautomat
9. UDP (User Data Protocol)

Tool zum Mitschneiden und Analysieren des Nachrichtenverkehrs

→ Wireshark-Sniffer

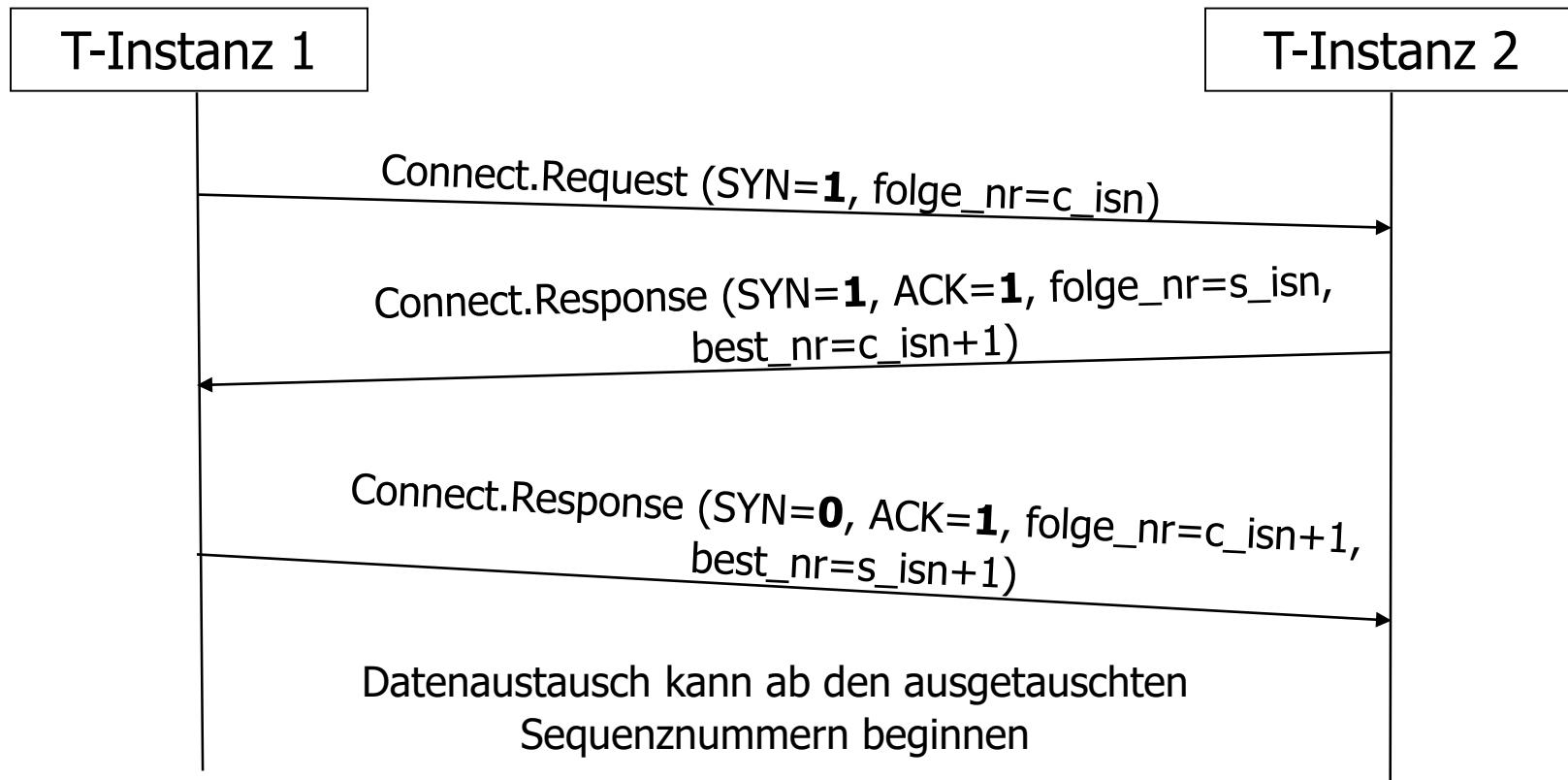
# Verbindungsauftbau: Parameter aushandeln und Dreiwege-Handshake

---

- Verwendung eines Dreiwege-Handshake-Mechanismus
  - Initiale Sequenznummern werden berechnet und ausgetauscht
- Beim Verbindungsauftbau werden die **Sequenznummer** und ggf. weitere Einstellungen (Optionen) ausgehandelt
- Kollision beim Verbindungsauftbau ist möglich:
  - Zwei Hosts versuchen gleichzeitig eine Verbindung mit gleichen Parametern zueinander aufzubauen
  - Es wird nur eine TCP-Verbindung aufgebaut

# Verbindungsaubau: Protokoll

- Normaler Ablauf



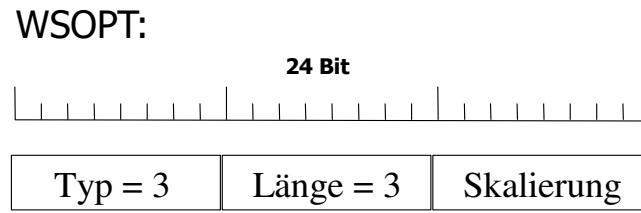
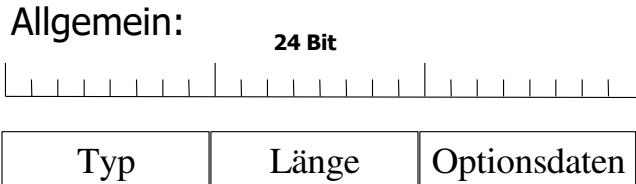
c\_isn = Initial Sequence Number des Clients (aktiver Partner, Instanz 1)

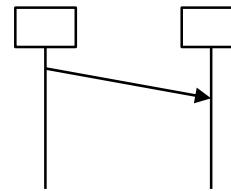
s\_isn = Initial Sequence Number des Servers (passiver Partner, Instanz 2)

# Ergänzung zum TCP-Header: Optionale Einstellungen beim Verbindungaufbau

---

- **Optionen** (insgesamt max. 40 Byte)
  - Haben eine variable Struktur, Rundung auf 32 Bit,
  - Optionen sind z.B.:
    - *MSS*: Maximum Segment Size der Verbindung einstellen
      - MSS kann maximal  $2^{16}$  Bytes sein.
      - Keine Angabe → 536 Byte als Standardgröße bei IPv4
    - *SACKOK*: Selektive Wiederholungen anstelle von *go back n* einstellen
    - *WSOPT (Windows-Scale-Option)*: Maximale Fenstergröße verlängern ( $2^{30}$  Byte max. möglich)
    - *SACK*: Selektives ACK: Liste an bestätigten Segmenten wird übertragen ...





# Wireshark-Übung – Verbindungsauftbau

## 1. Nachricht - SYN

```

▶ Frame 1: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface 0
▶ Null/Loopback
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▼ Transmission Control Protocol, Src Port: 50001, Dst Port: 50000, Seq: 0, Len: 0
    Source Port: 50001
    Destination Port: 50000
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0      (relative sequence number)
    Acknowledgment number: 0
    1011 .... = Header Length: 44 bytes (11)
    Flags: 0x002 (SYN)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0.... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...0 .... = Acknowledgment: Not set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..1. = Syn: Set
        .... .... 0 = Fin: Not set
        [TCP Flags: .....S.]
    Window size value: 65535
    [Calculated window size: 65535]
    Checksum: 0xfe34 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
    Options: (24 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), Timestamps, SACK permitted, End of Option List (EOL)
        ▶ TCP Option - Maximum segment size: 16344 bytes
        ▶ TCP Option - No-Operation (NOP)
        ▶ TCP Option - Window scale: 5 (multiply by 32)
        ▶ TCP Option - No-Operation (NOP)
        ▶ TCP Option - No-Operation (NOP)
        ▶ TCP Option - Timestamps: Tsvl 1044103970, TSecr 0
        ▶ TCP Option - SACK permitted
        ▶ TCP Option - End of Option List (EOL)

0000  02 00 00 00 45 00 00 40  23 b4 40 00 40 06 00 00  ....E..@ #.@@...
0010  7f 00 00 01 7f 00 00 01  c3 51 c3 50 94 2c 4a 3e  ..... .Q.P.,J>
0020  00 00 00 b0 2f ff  fe 34 00 00 02 04 3f d8  ..... .4....?
0030  01 03 03 05 01 01 08 0a  3e 3b c3 22 00 00 00 00  ..... >."....
0040  04 02 00 00

```

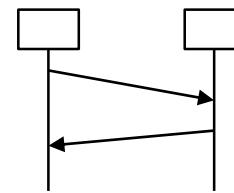
No.: 1 · Time: 2017-07-30 11:13:05.213181 · Source: 127.0.0.1 · Info: 50001 → 50000 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=32 TSeq=1044103970 TSecr=0 SACK\_PERM=1 · Destination: 127.0.0.1 · Protocol: TCP · Length: 68

Help

Close

# Wireshark-Übung – Verbindungsauftbau

## 2. Nachricht: SYN/ACK



```
▶ Frame 2: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface 0
▶ Null/Loopback
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▼ Transmission Control Protocol, Src Port: 50000, Dst Port: 50001, Seq: 0, Ack: 1, Len: 0
    Source Port: 50000
    Destination Port: 50001
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0      (relative sequence number)
    Acknowledgment number: 1      (relative ack number)
    1011 .... = Header Length: 44 bytes (11)
    ▼ Flags: 0x012 (SYN, ACK)
        000. .... .... = Reserved: Not set
        ....0 .... .... = Nonce: Not set
        .... 0.... .... = Congestion Window Reduced (CWR): Not set
        .... .0. .... = ECN-Echo: Not set
        .... ..0 .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        ▶ .... .... ..1. = Syn: Set
        .... .... .0 = Fin: Not set
        [TCP Flags: .....A..S.]
        Window size value: 65535
        [Calculated window size: 65535]
        Checksum: 0xfe34 [unverified]
        [Checksum Status: Unverified]
        Urgent pointer: 0
    ▼ Options: (24 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), Timestamps, SACK permitted, End of Option List (EOL)
        ▶ TCP Option - Maximum segment size: 16344 bytes
        ▶ TCP Option - No-Operation (NOP)
        ▶ TCP Option - Window scale: 5 (multiply by 32)
        ▶ TCP Option - No-Operation (NOP)
        ▶ TCP Option - No-Operation (NOP)
        ▶ TCP Option - No-Operation (NOP)
        ▶ TCP Option - Timestamps: TStamp 1044103970, TSecr 1044103970
        ▶ TCP Option - SACK permitted
        ▶ TCP Option - End of Option List (EOL)
        ▶ [SEQ/ACK analysis]
0000  02 00 00 00 45 00 00 40  0a 58 40 00 40 06 00 00  ....E..@ .X@. @...
0010  7f 00 00 01 7f 00 00 01  c3 50 c3 51 6b 97 09 d0  ....P.Qk...
0020  94 2c 4a 3f b0 12 ff ff  fe 34 00 00 02 04 3f d8  ,.J?... 4. ....?
0030  01 03 03 05 01 08 0a  3e 3b c3 22 3e 3b c3 22  ....>;>;"
0040  04 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....
```

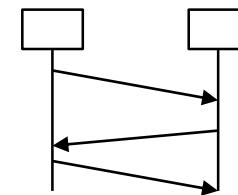
No.: 2 · Time: 2017-07-30 11:13:05.213247 · Source: 127.0.0.1 · Info: 50000 → 50001 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=32 TStamp=1044103970 TSecr=1044103970 SACK\_PERM=1 · Destination: 127.0.0.1 · Protocol: TCP · Length: 68

Help

Close

# Wireshark-Übung – Verbindungsauftbau

## 3. Nachricht: SYN/ACK



```
► Frame 3: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0
► Null/Loopback
► Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▼ Transmission Control Protocol, Src Port: 50001, Dst Port: 50000, Seq: 1, Ack: 1, Len: 0
  Source Port: 50001
  Destination Port: 50000
  [Stream index: 0]
  [TCP Segment Len: 0]
    Sequence number: 1      (relative sequence number)
    Acknowledgment number: 1    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  ▼ Flags: 0x010 (ACK)
    000. .... .... = Reserved: Not set
    ....0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... ...0... = Push: Not set
    .... .... .0.. = Reset: Not set
    .... .... ..0. = Syn: Not set
    .... .... ...0 = Fin: Not set
    [TCP Flags: .....A....]
    Window size value: 12759
    [Calculated window size: 408288]
    [Window size scaling factor: 32]
    Checksum: 0xfe28 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    ► TCP Option - No-Operation (NOP)
    ► TCP Option - No-Operation (NOP)
    ► TCP Option - Timestamps: TStamp 1044103970, TSectr 1044103970
    ► [SEQ/ACK analysis]
```

```
0000 02 00 00 00 45 00 00 34 d2 f0 40 00 40 06 00 00  ....E..4 ..@.@
0010 7f 00 00 01 7f 00 00 01 c3 51 c3 50 94 2c 4a 3f  ....Q.P..J?
0020 6b 97 09 d1 80 10 31 d7 fe 28 00 00 01 01 08 0a k....1. .(.....
0030 3e 3b c3 22 3e 3b c3 22 >;">;"
```

No.: 3 · Time: 2017-07-30 11:13:05.213256 · Source: 127.0.0.1 · Info: 50001 → 50000 [ACK] Seq=1 Ack=1 Win=408288 Len=0 TStamp=1044103970 TSectr=1044103970 · Destination: 127.0.0.1 · Protocol: TCP · Length: 56

Help

Close

# Verbindungsaubau

## Fehlerszenarien

---

- TCP muss mehrere Fehlersituationen richtig bearbeiten
  - Gleichzeitiger Verbindungsaubauversuch beider Partner darf nur zu einer Verbindung führen
  - Alte Duplikate von TCP-Segmenten werden beim Verbindungsaubau empfangen
    - Reset der Verbindung (RST-Bit) und erneuter Aufbau
  - ...

# Einschub: Konfiguration von TCP-Parametern

---

- TCP-Konfiguration abhängig vom Betriebssystem
  - Linux: z.B. in Datei **/etc/sysctl.conf** (persistente Veränderung) oder Dateien verändern in **/proc/sys/net/ipv4/** (bis zum Reboot)
  - Beispiele:
    - Wiederholung von Verbindungsaufbauversuchen durch die aktive Seite: **tcp\_syn\_retries**: 5 (Standardeinstellung)
    - Wiederholung von Bestätigungen des Verbindungsabbaus durch die passive Seite: **tcp\_synack\_retries**: 5 (Standardeinstellung)
    - ...

# Verbindungsabbau

## Ablauf

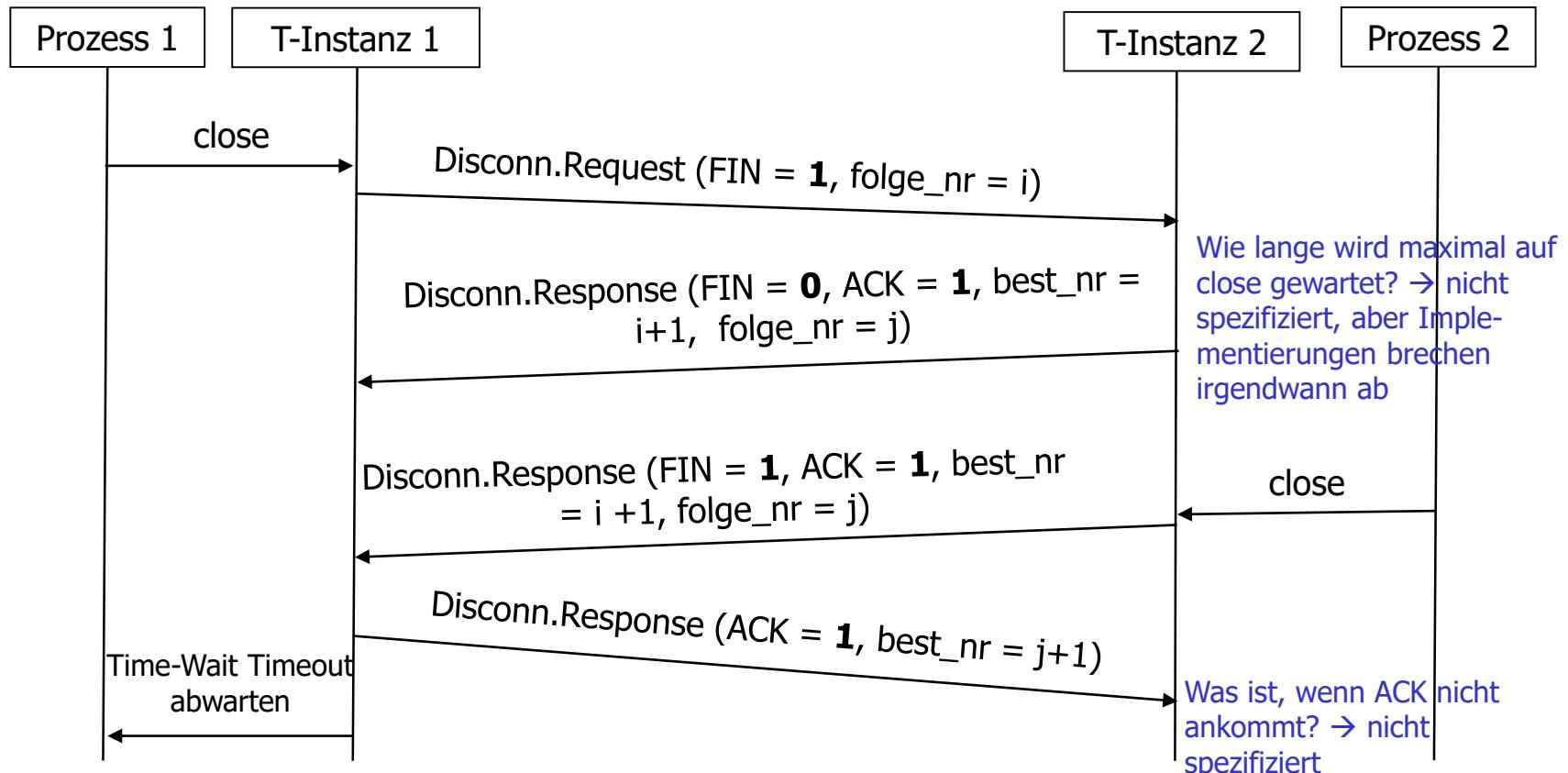
---

- Verbindungsabbau-Protokoll:
  - Vierwege-Handshake-Mechanismus
  - Jede der beiden Verbindungsrichtungen der Vollduplex-Verbindung wird abgebaut, d.h. beide Seiten bauen ihre „Senderichtung“ ab
- Ablauf:
  - Aktiv abbauender Partner sendet zunächst ein Segment mit **FIN=1**
  - Passiver Partner antwortet zunächst mit einem **ACK**
  - Wenn die Anwendung auf der Passivseite **close** aufruft, sendet die Partnerinstanz noch ein Segment mit **FIN=1, ACK=1**
  - Aktiver Partner sendet abschließend ein Segment mit **ACK=1**

# Verbindungsabbau

## Nachrichtenfluss

- Client baut die Verbindung ab (auch Server kann es)
- Alle Segmente mit Folgenummer < i bzw. j sind noch zu verarbeiten
- FIN-Segment zählt als 1 Datenbyte → Sequenznummer **+ 1**



# Verbindungsabbau

## Signalisierung beim passiven Partner

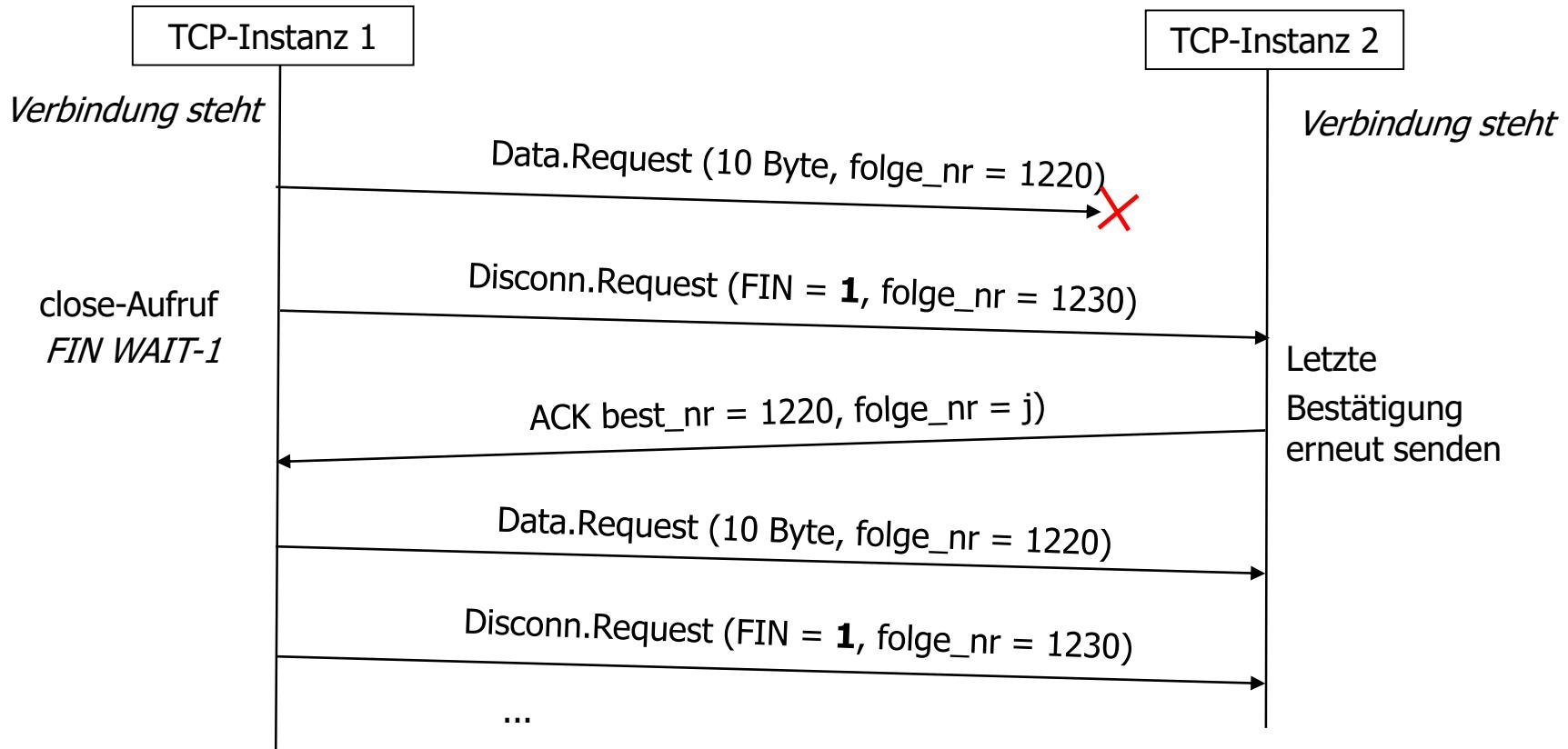
---

- Laut TCP-Spezifikation gibt es **mehrere Varianten** (genau 4) des Verbindungsabbaus
  - Es geht auch mit drei Segmenten (FIN=1 + ACK=1)
- Die **Signalisierung** des Verbindungsabbaus an die Anwendung ist im RFC nicht genau beschrieben
  - Es kann eine Weile dauern, bis die Anwendung mit *close*-Aufruf reagiert
  - Das Anwendungsprotokoll muss dies regeln
  - Anwendung kann evtl. sogar eine Benutzereingabe erfordern
  - Dies hängt vom Programm ab
- Auch **abnormale Beendigung** einer Verbindung ist möglich
  - Segment mit **RST-Bit=1** wird gesendet und der Empfänger bricht die Verbindung sofort ab

# Verbindungsabbau

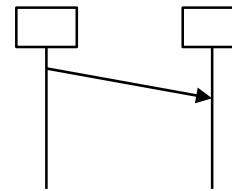
## Szenario: Fehlende Daten beim Verbindungsabbau

- Bei Ankunft eines FIN-Segments **fehlen** bei T-Instanz 2 noch Daten
- Letzte Bestätigung wird erneut gesendet, nach Ankunft der fehlenden Daten wird Verbindungsabbau erneut versucht



# Wireshark-Übung – Verbindungsabbau

## 1. Nachricht - FIN



```
▶ Frame 11: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0
▶ Null/Loopback
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▼ Transmission Control Protocol, Src Port: 50001, Dst Port: 50000, Seq: 22, Ack: 5, Len: 0
    Source Port: 50001
    Destination Port: 50000
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 22      (relative sequence number)
    Acknowledgment number: 5      (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
    ▼ Flags: 0x011 (FIN, ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0.... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... 0. .... = Urgent: Not set
        .... .1 .... = Acknowledgment: Set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        ▶ .... .... ..1 = Fin: Set
        [TCP Flags: .....A...F]
        Window size value: 12759
        [Calculated window size: 408288]
        [Window size scaling factor: 32]
        Checksum: 0xfe28 [unverified]
        [Checksum Status: Unverified]
        Urgent pointer: 0
    ▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
        ▶ TCP Option - No-Operation (NOP)
        ▶ TCP Option - No-Operation (NOP)
        ▶ TCP Option - Timestamps: TStamp 1044104029, TSectr 1044104029
    0000 02 00 00 00 45 00 00 34 65 41 40 00 40 06 00 00  ....E..4 eA@.0...
    0010 7f 00 00 01 7f 00 00 01 c3 51 c3 50 94 2c 4a 54  .........0.P.,JT
    0020 6b 97 09 d5 80 11 31 d7 fe 28 00 00 01 01 08 0a k.....1.(.....
    0030 3e 3b c3 5d 3e 3b c3 5d >;]>;.
```

Quellport 50001 als aktiv  
Verbindungsabbauender

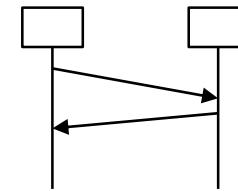
No.: 11 · Time: 2017-07-30 11:13:05.274120 · Source: 127.0.0.1 · Info: 50001 → 50000 [FIN, ACK] Seq=22 Ack=5 Win=408288 Len=0 TStamp=1044104029 TSectr=1044104029 · Destination: 127.0.0.1 · Protocol: TCP · Length: 56

Help

Close

# Wireshark-Übung – Verbindungsabbau

## 2. Nachricht – ACK



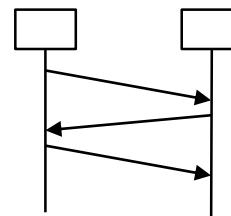
```
▶ Frame 12: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0
▶ Null/Loopback
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▼ Transmission Control Protocol, Src Port: 50000, Dst Port: 50001, Seq: 5, Ack: 23, Len: 0
    Source Port: 50000
    Destination Port: 50001
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 5      (relative sequence number)
    Acknowledgment number: 23     (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
    ▼ Flags: 0x010 (ACK)
        000. .... .... = Reserved: Not set
        ....0 .... .... = Nonce: Not set
        ....0.... .... = Congestion Window Reduced (CWR): Not set
        ....0.. .... = ECN-Echo: Not set
        ....0. .... = Urgent: Not set
        ....1 .... = Acknowledgment: Set
        ....0... .... = Push: Not set
        ....0...0.. .... = Reset: Not set
        ....0....0. .... = Syn: Not set
        ....0....0 = Fin: Not set
        [TCP Flags: .....A.....]
    Window size value: 12758
    [Calculated window size: 408256]
    [Window size scaling factor: 32]
    Checksum: 0xfe28 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
    ▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
        ▶ TCP Option - No-Operation (NOP)
        ▶ TCP Option - No-Operation (NOP)
        ▶ TCP Option - Timestamps: TStamp 1044104029, TSectr 1044104029
        ▶ [SEQ/ACK analysis]
```

```
0000 02 00 00 00 45 00 00 34 1a 3d 40 00 40 06 00 00 ....E..4 .@. @...
0010 7f 00 00 01 7f 00 00 01 c3 50 c3 51 6b 97 09 d5 .....P.Qk...
0020 94 2c 4a 55 80 10 31 d6 fe 28 00 00 01 01 08 0a ,.JU..1. .(.....
0030 3e 3b c3 5d 3e 3b c3 5d >;]>;.]
```

No.: 12 · Time: 2017-07-30 11:13:05.274152 · Source: 127.0.0.1 · Info: 50000 → 50001 [ACK] Seq=5 Ack=23 Win=408256 Len=0 TSval=1044104029 TSecr=1044104029 · Destination: 127.0.0.1 · Protocol: TCP · Length: 56

Help

Close



# Wireshark-Übung – Verbindungsabbau

## 3. Nachricht – ACK/FIN

Frame 14: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0

► Null/Loopback

► Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

▼ Transmission Control Protocol, Src Port: 50000, Dst Port: 50001, Seq: 5, Ack: 23, Len: 0

    Source Port: 50000  
     Destination Port: 50001  
     [Stream index: 0]  
     [TCP Segment Len: 0]  
     Sequence number: 5     (relative sequence number)  
     Acknowledgment number: 23     (relative ack number)  
     1000 .... = Header Length: 32 bytes (8)

▼ Flags: 0x011 (FIN, ACK)

    000. .... .... = Reserved: Not set  
     ...0 .... .... =Nonce: Not set  
     .... 0.... .... = Congestion Window Reduced (CWR): Not set  
     .... .0. .... = ECN-Echo: Not set  
     .... ..0. .... = Urgent: Not set  
     .... ...1 .... = Acknowledgment: Set  
     .... .... 0... = Push: Not set  
     .... .... .0.. = Reset: Not set  
     .... .... ..0. = Syn: Not set  
     ► .... .... .1 = Fin: Set  
         [TCP Flags: .....A...F]  
     Window size value: 12758  
     [Calculated window size: 408256]  
     [Window size scaling factor: 32]  
     Checksum: 0xfe28 [unverified]  
     [Checksum Status: Unverified]  
     Urgent pointer: 0

▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps

    ► TCP Option - No-Operation (NOP)  
     ► TCP Option - No-Operation (NOP)  
     ► TCP Option - Timestamps: TSval 1044104029, TSecr 1044104029

```

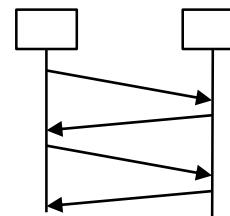
0000 02 00 00 00 45 00 00 34 f8 22 40 00 40 06 00 00  ....E..4 ."@.@
0010 7f 00 00 01 7f 00 00 01 c3 50 c3 51 6b 97 09 d5  .....P.Qk...
0020 94 2c 4a 55 80 11 31 d6 fe 28 00 00 01 01 08 0a  ..JU..1. .(.....
0030 3e 3b c3 5d 3e 3b c3 5d >;]>;.]
```

No.: 14 · Time: 2017-07-30 11:13:05.274361 · Source: 127.0.0.1 · Info: 50000 → 50001 [FIN, ACK] Seq=5 Ack=23 Win=408256 Len=0 TSval=1044104029 TSecr=1044104029 · Destination: 127.0.0.1 · Protocol: TCP · Length: 56

[Help](#) [Close](#)

# Wireshark-Übung – Verbindungsabbau

## 4. Nachricht - ACK



```
▶ Frame 15: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0
▶ Null/Loopback
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▼ Transmission Control Protocol, Src Port: 50001, Dst Port: 50000, Seq: 23, Ack: 6, Len: 0
    Source Port: 50001
    Destination Port: 50000
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 23      (relative sequence number)
    Acknowledgment number: 6      (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
    ▼ Flags: 0x010 (ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... =Nonce: Not set
        .... 0.... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: .....A.....]
    Window size value: 12759
    [Calculated window size: 408288]
    [Window size scaling factor: 32]
    Checksum: 0xfe28 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
    ▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
        ▶ TCP Option - No-Operation (NOP)
        ▶ TCP Option - No-Operation (NOP)
        ▶ TCP Option - Timestamps: TSval 1044104029, TSecr 1044104029
        ▶ [SEQ/ACK analysis]
```

```
0000 02 00 00 00 45 00 00 34 16 13 40 00 40 06 00 00  ....E..4 ..@.@
0010 7f 00 00 01 7f 00 00 01 c3 51 c3 50 94 2c 4a 55  ....Q.P.,JU
0020 6b 97 09 d6 80 10 31 d7 fe 28 00 00 01 01 08 0a  k.....1. .(.....
0030 3e 3b c3 5d 3e 3b c3 5d  >;.]>;.]
```

No.: 15 · Time: 2017-07-30 11:13:05.274417 · Source: 127.0.0.1 · Info: 50001 → 50000 [ACK] Seq=23 Ack=6 Win=408288 Len=0 TSval=1044104029 TSecr=1044104029 · Destination: 127.0.0.1 · Protocol: TCP · Length: 56

Help

Close

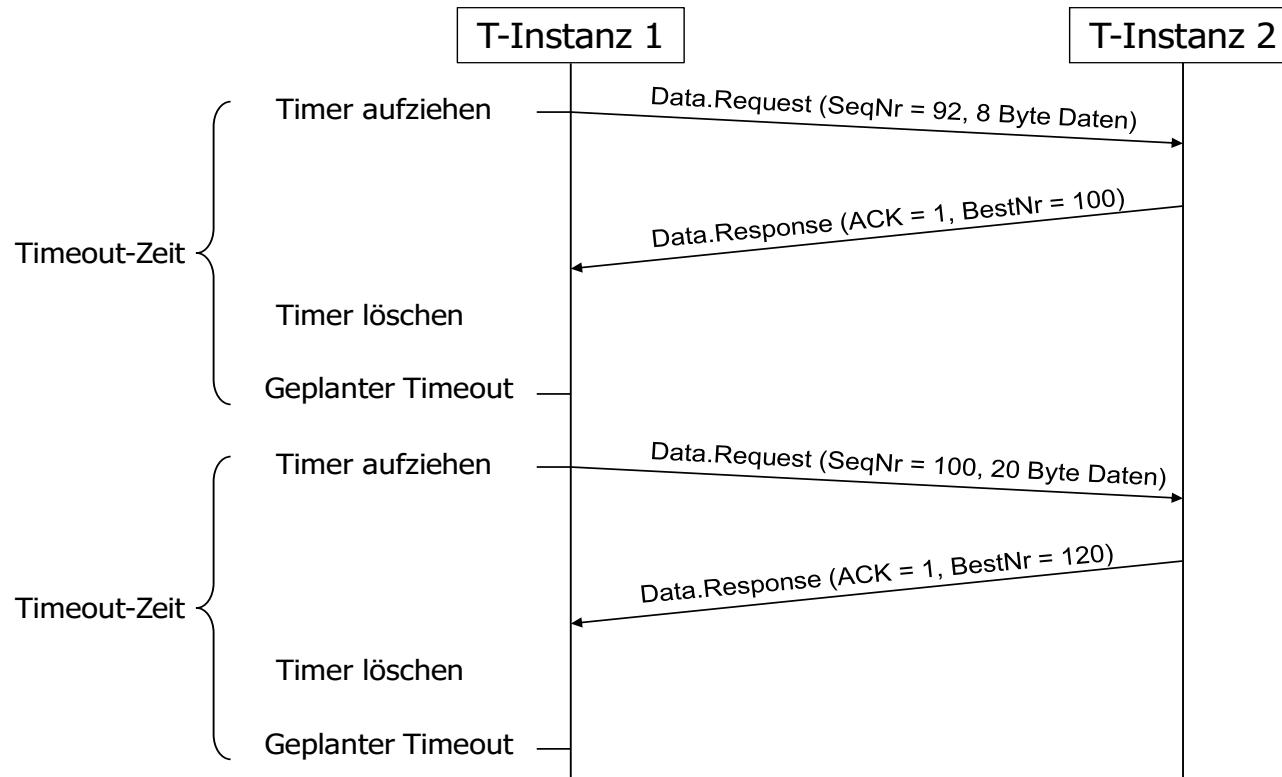
# Sequenznummern und Quittierung (1)

---

- Einsatz von Sequenznummern, die auf einzelnen Bytes des TCP-Streams, nicht auf TCP-Segmenten basieren
- Die Sequenznummer enthält die Nummer des nächsten erwarteten Bytes (Octets)
- Alle gesendeten Bytes werden vom Empfänger im Feld **Bestätigungsnummer kumulativ** quittiert
- Positiv selektive Quittierung über SACK-Option auch optional möglich
- Die Bestätigung muss von der empfangenden TCP-Instanz **nicht unbedingt sofort** gesendet werden, wenn noch Platz im Puffer ist
  - Es gibt verschiedene Optimierungsvarianten
  - Hier besteht Implementierungsfreiheit für die Hersteller von TCP/IP-Stacks

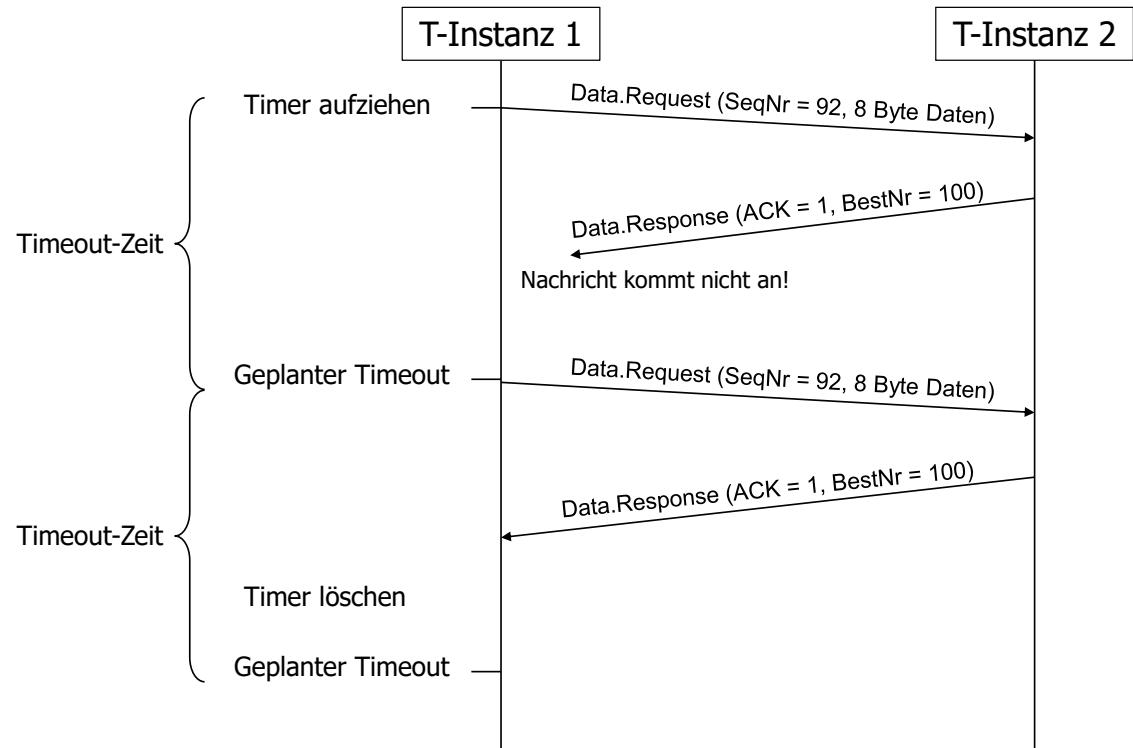
# Szenario: Erfolgreiche Übertragung

- Sende-Instanz zieht Retransmission-Timer auf (Berechnung dynamisch)
- Timer wird nach ACK gelöscht
- Evtl. Verzögerung von ACKs nicht berücksichtigt



# Szenario: Bestätigung geht verloren

- Sende-Instanz zieht Retransmission-Timer auf
  - Timer läuft bei verlorengegangener Quittung ab
  - TCP-Segment wird erneut gesendet
  - Kernelabhängig, wie oft wird wiederholt wird, z.B. unter Windows TcpMaxDataRetransmissions=5 (per default)



Nach Tanenbaum, A.; Wetherall, D.: Computer Networks, 5. Auflage, Pearson Studium, 2011

## Sequenznummern und Quittierung (2)

---

- Wann muss ein Empfänger spätestens eine Quittierung (ACK) senden?
  - Siehe hierzu RFC 5681
  - Empfänger darf nicht zu lange verzögern
  - Ein ACK **soll** gesendet werden nach dem Empfang des zweiten (TCP-Segments)
  - Spätestens nach 500 ms **muss** ein TCP-Segment bestätigt werden

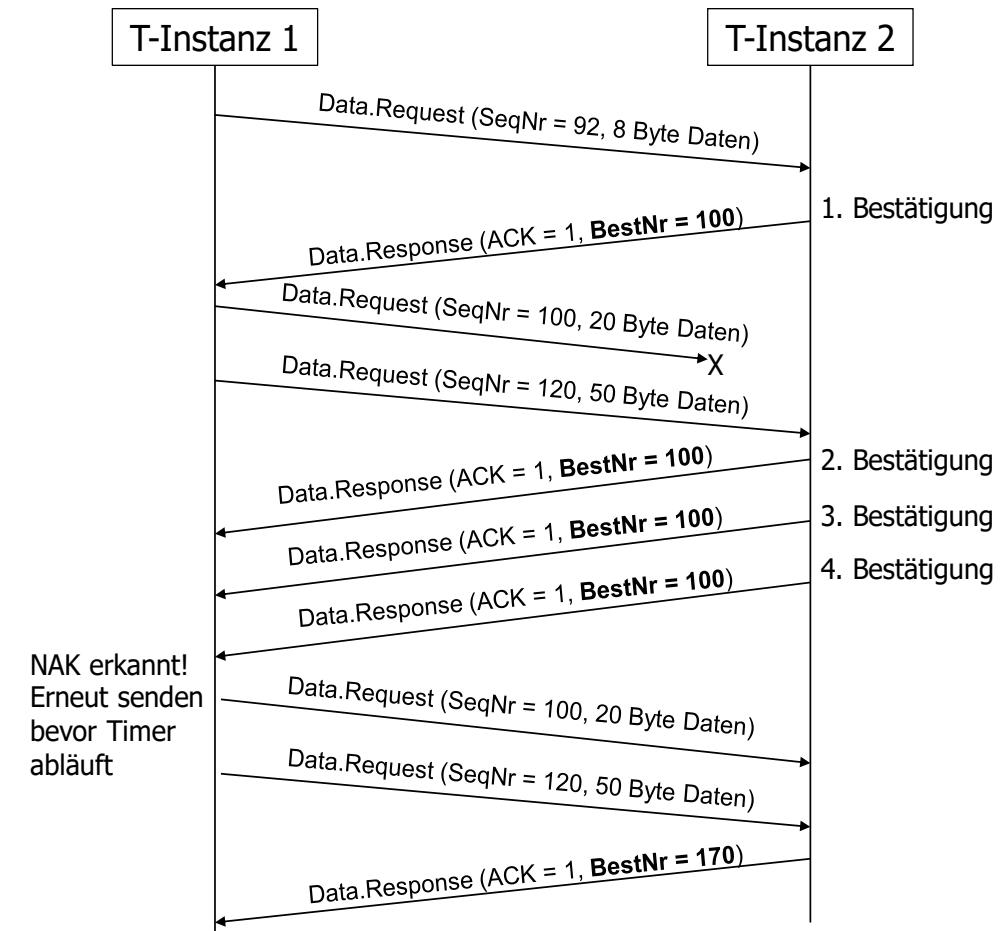
## Einschub: Push-Flag (PSH)

---

- Viele TCP-Implementierungen (TCP-Stacks) setzen das Push-Flag beim Senden eines Segments in der Regel am Ende einer logisch zusammengehörigen Nachricht, also nach jedem send-Aufruf
  - Nur die TCP-Implementierung kontrolliert das Setzen des Push-Flags, **nicht** die Anwendung
- Nachrichten werden dann **weder** im Sende-, **noch** im Empfangspuffer zwischengespeichert, sondern gleich an den Empfängerprozess ausgeliefert
  - So werden Auslieferungsverzögerungen vermieden und es können beim Empfänger logisch zusammengehörige Daten leichter ausgeliefert werden

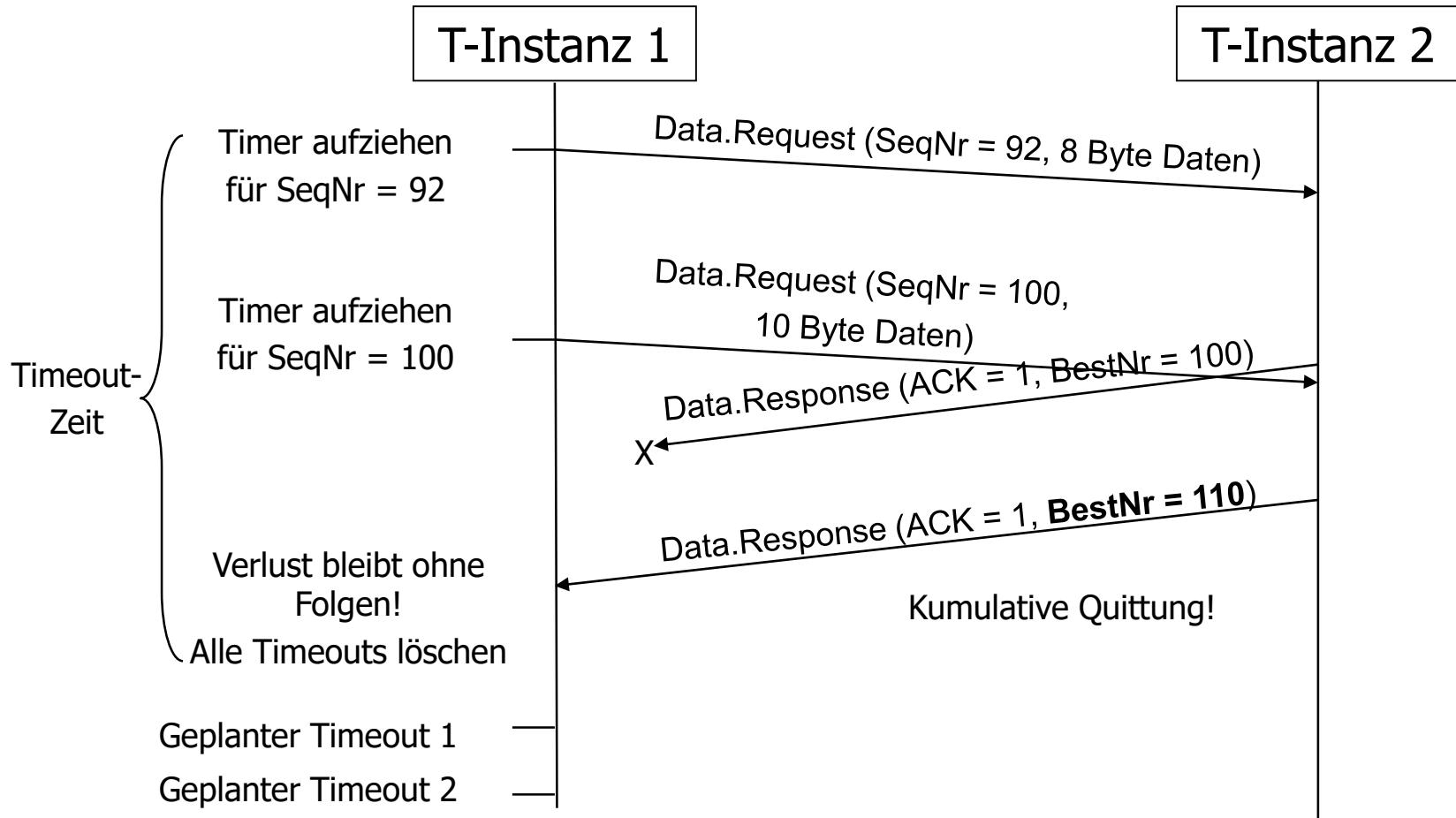
# Szenario: Implizites NAK bewirkt Sendewiederholung

- Im Beispiel wird das 1. ACK **drei mal** vom Empfänger wiederholt
- Sender erkennt Problem und sendet erneut → Problem wird vor dem Timerablauf erkannt
- Wird als **implizites NAK** bezeichnet!



# Szenario: Kumulative Quittung verhindert erneutes Senden einer verlorenen Bestätigung

- Verlust eines ACK-Segments bleibt ohne Folgen



# Mögliche Probleme beim Verbindungsaufbau

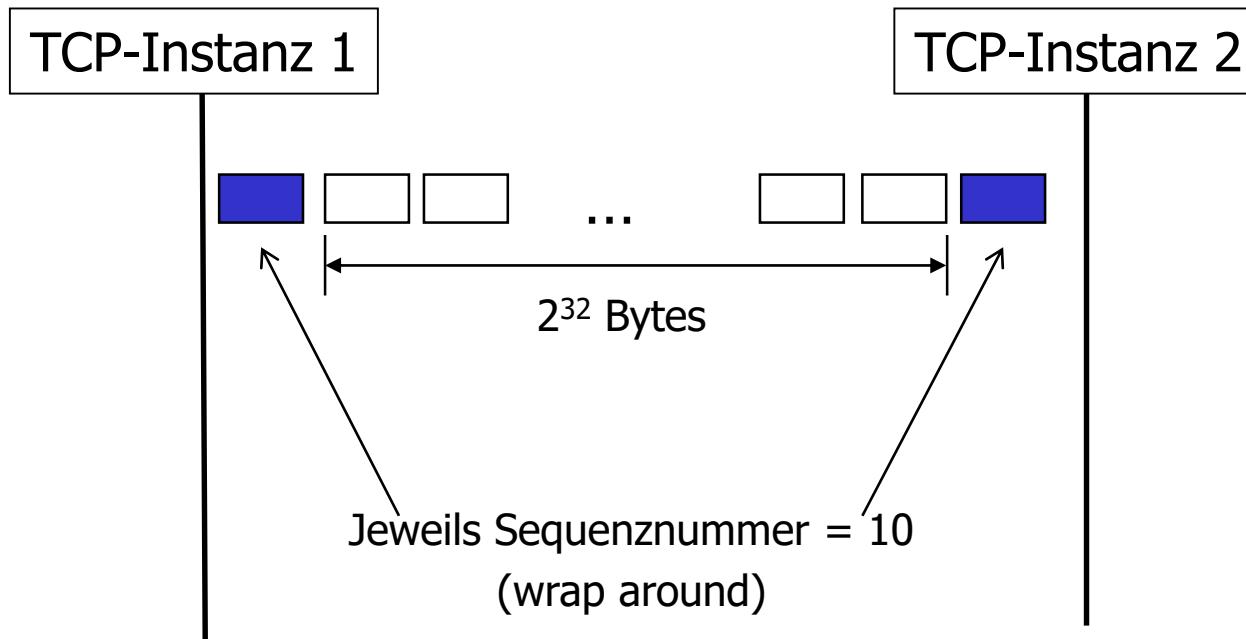
## Kollision der Sequenznummern

---

- **Sequenznummern-Anzahl** bei TCP:  $2^{32}$  (32-Bit-Feld)
  - Vergabe mod  $2^{32}$
  - Bei 64 Kbit/s kommt es frühestens nach ca. 6,2 Tagen zu einer Wiederholung
  - Bei 100 Mbit/s kommt es frühestens nach ca. 340 s zu einer Wiederholung
  - Bei 1 Gbit/s kommt es frühestens nach ca. 34 s zu einer Wiederholung
    - $2^{32} * 8 \text{ Bit} : 1.000.000.000 \text{ Bit/s} = 2^{35} : 10^9 \text{ s} \sim 34,35 \text{ s}$
- **Probleme:**
  - Wrapping der Sequenznummern: Erneuter, schneller Verbindungsauftakt nach Crash könnte zu Sequenznummern-Kollision führen
    - Ein noch altes TCP-Segment könnte bei neuer „Inkarnation“ der Verbindung (gleiche Adressparameter = gleiches Socket Pair) ankommen und nicht erkannt werden

# Kollision der Sequenznummern bei der Datenübertragung

- Sequenznummern wiederholen sich evtl. auch bei Netzen mit hoher Pfadkapazität und hoher Latenzzeit
  - Hier handelt es sich um sog. „long fat networks“
  - z.B. Hochleistungs-Satellitenübertragung



# PAWS (1)

---

- **PAWS** = Protect Against Wrapped Sequences
- **Ursprünglicher Ansatz: Minderung des Problems durch Maßnahme im Verbindungsaufbau**
  - Einsatz des Dreiwege-Handshake zum Synchronisieren der Sequenznummern beim Verbindungsaufbau
  - Generierung der initialen Sequenznummer (ISN) anhand eines (fiktiven) Zeitgebers
    - Ursprünglicher Vorschlag im RFC 793, S. 28: Zyklus des Zeitgebers von 4,55 Stunden, Zeitgeber wird alle 4 ms erhöht
- **Weitere Schutzmaßnahme: Maximale Segmentlebensdauer (MSL):**
  - Wurde im RFC 792 auf 2 Minuten festgelegt
  - Zeit von  $2 * \text{MSL}$  muss abgewartet werden, bevor nach einem Crash einer Verbindung eine neue ISN zugewiesen wird → Zustand **TIMED\_WAIT**

## PAWS (2)

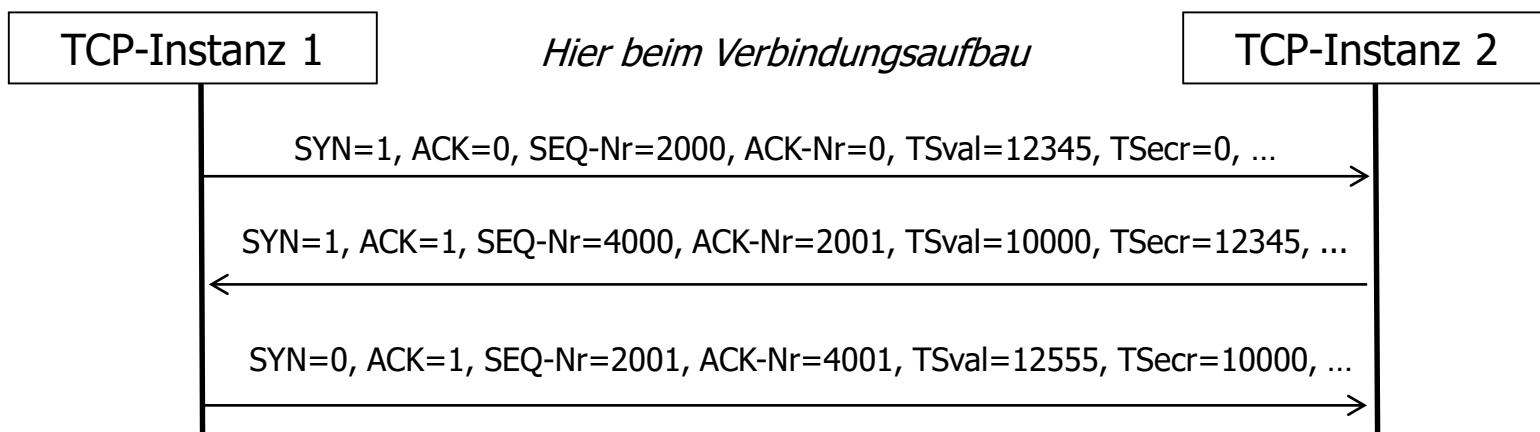
---

- **Weitere Lösung für Wrapping der Sequenznummern**
  - RFC 1323: TCP extensions für high performance
  - Nutzung einer weiteren TCP-Option **Timestamps Option TSopt** um Sequenznummern-Wrapping in schnellen Netzen zu erkennen:
    - Die TSopt-Option wird meist beim Verbindungsaufbau für die Dauer der Verbindung aktiviert
    - Zwei Timestamps (in ms) werden verwendet
      - Timestamp Value (TSval)
      - Timestamp-Echo-Reply (TSecr)
    - Das erste Feld wird von der sendenden TCP-Instanz belegt
    - Letzteres Feld ist nur bei ACK-Segmenten (ACK-Flag=1) erlaubt

# PAWS (3)

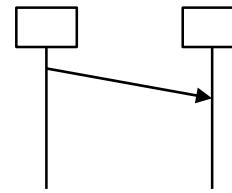
## ■ TSopt-Nutzung

- Der Sender sendet im *TSval*-Feld die aktuelle Absendezzeit gemäß seiner lokalen Uhr mit
- Der Empfänger sendet mit dem Bestätigungssegment seinen eigenen Zeitstempel zur Sendezeit im Feld *TSval* und im *Tsecr*-Feld den Zeitstempel des Senders zurück
- Der Sender kann dann die Sequenznummer eindeutig zuordnen
- Nebeneffekt: Daraus lässt sich dann die auch RTT berechnen (Variante für RTO-Algorithmus (später))



# Wireshark-Übung – Datenübertragung

## Nachricht senden



```
► Frame 9: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0
► Null/Loopback
► Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▼ Transmission Control Protocol, Src Port: 50001, Dst Port: 50000, Seq: 5, Ack: 5, Len: 17
    Source Port: 50001
    Destination Port: 50000
    [Stream index: 0]
    [TCP Segment Len: 17]
    Sequence number: 5 (relative sequence number)
    [Next sequence number: 22 (relative sequence number)]
    Acknowledgment number: 5 (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
    ▼ Flags: 0x018 (PSH, ACK)
        000. .... .... = Reserved: Not set
        ....0 .... .... =Nonce: Not set
        .... 0.... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... ...1.. = Push: Set ←
        .... ...0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ..0 = Fin: Not set
        [TCP Flags: .....AP...]
        Window size value: 12759
        [Calculated window size: 408288]
        [Window size scaling factor: 32]
        Checksum: 0xfe39 [unverified]
        [Checksum Status: Unverified]
        Urgent pointer: 0
    ▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
        ► TCP Option - No-Operation (NOP)
        ► TCP Option - No-Operation (NOP)
        ► TCP Option - Timestamps: TStamp 1044104029, TSectr 1044103972
    ► [SEQ/ACK analysis]
        TCP payload (17 bytes)
    ▼ Data (17 bytes)
        Data: 74000e544553542d4e6163687269636874
        [Length: 17]
```

Quellport: 50001  
Zielport: 50000

Sequenznummer

Push-Flag gesetzt

Nutzdaten

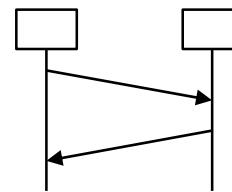
No.: 9 · Time: 2017-07-30 11:13:05.273689 · Source: 127.0.0.1 · Info: 50001 → 50000 [PSH, ACK] Seq=5 Ack=5 Win=17 TStamp=1044104029 TSectr=1044103972 · Destination: 127.0.0.1 · Protocol: TCP · Length: 73

Help

Close

# Wireshark-Übung – Datenübertragung

## Bestätigung senden



Frame 10: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0  
Null/Loopback  
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
Transmission Control Protocol, Src Port: 50000, Dst Port: 50001, Seq: 5, Ack: 22, Len: 0

Source Port: 50000  
Destination Port: 50001  
[Stream index: 0]  
[TCP Segment Len: 0]  
Sequence number: 5 (relative sequence number)  
Acknowledgment number: 22 (relative ack number)  
1000 .... = Header Length: 32 bytes (8)

(\*) Flags: 0x010 (ACK)  
000. .... .... = Reserved: Not set  
...0 .... .... = Nonce: Not set  
.... 0... .... = Congestion Window Reduced (CWR): Not set  
.... .0.. .... = ECN-Echo: Not set  
.... ..0. .... = Urgent: Not set  
.... ...1 .... = Acknowledgment: Set  
.... .... 0... = Push: Not set  
.... .... .0.. = Reset: Not set  
.... .... ..0. = Syn: Not set  
.... .... ...0 = Fin: Not set  
[TCP Flags: .....A.....]  
Window size value: 12758  
[Calculated window size: 408256]  
[Window size scaling factor: 32]  
Checksum: 0xfe28 [unverified]  
[Checksum Status: Unverified]  
Urgent pointer: 0

Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps  
TCP Option - No-Operation (NOP)  
TCP Option - No-Operation (NOP)  
TCP Option - Timestamps: TVal 1044104029, TSectr 1044104029  
[SEQ/ACK analysis]

Quellport: 50000  
Zielport: 50001

0000 02 00 00 00 45 00 00 34 8e 2d 40 00 40 06 00 00 ....E..4 .@. @...  
0010 7f 00 00 01 7f 00 00 01 c3 50 c3 51 6b 97 09 d5 ..... P.Qk...  
0020 94 2c 4a 54 80 10 31 d6 fe 28 00 00 01 01 08 0a ..JT..1. (.....  
0030 3e 3b c3 5d 3e 3b c3 5d >;]>;.]

(\*\*) Sequenznummer beginnt  
immer bei 0 (relative Angabe)

(\*) Stream-Index:  
Wireshark-Nummerierung für  
aufgezeichnete TCP-Verbindungen

No.: 10 · Time: 2017-07-30 11:13:05.273740 · Source: 127.0.0.1 · Info: 50000 → 50001 [ACK] Seq=5 Ack=22 Win=408256 Len=0 TVal=1044104029 TSectr=1044104029 · Destination: 127.0.0.1 · Protocol: TCP · Length: 56

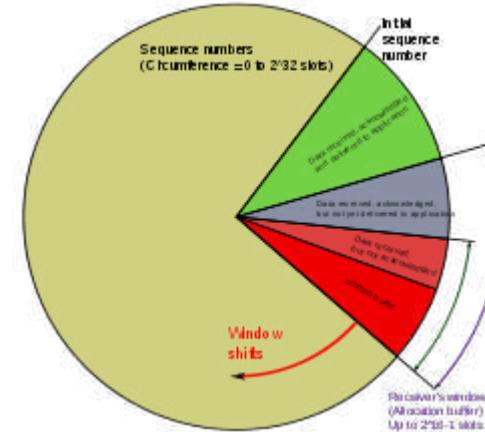
Help

Close

# Überblick über Transportprotokolle

---

1. Einordnung und Aufgaben von TCP
2. Der TCP-Header
3. Verbindungsauflauf- und -abbau, Datenübertragungsphase
- 4. Sliding-Window-Mechanismus**
5. Optimierungen: Algorithmen von Nagle und Clark, Silly Window Syndrom
6. Staukontrolle
7. TCP-Timer
8. TCP-Zustandsautomat
9. UDP (User Data Protocol)



Quelle: [https://en.wikipedia.org/wiki/Transmission\\_Control\\_Protocol](https://en.wikipedia.org/wiki/Transmission_Control_Protocol)

# Sliding Window Mechanismus

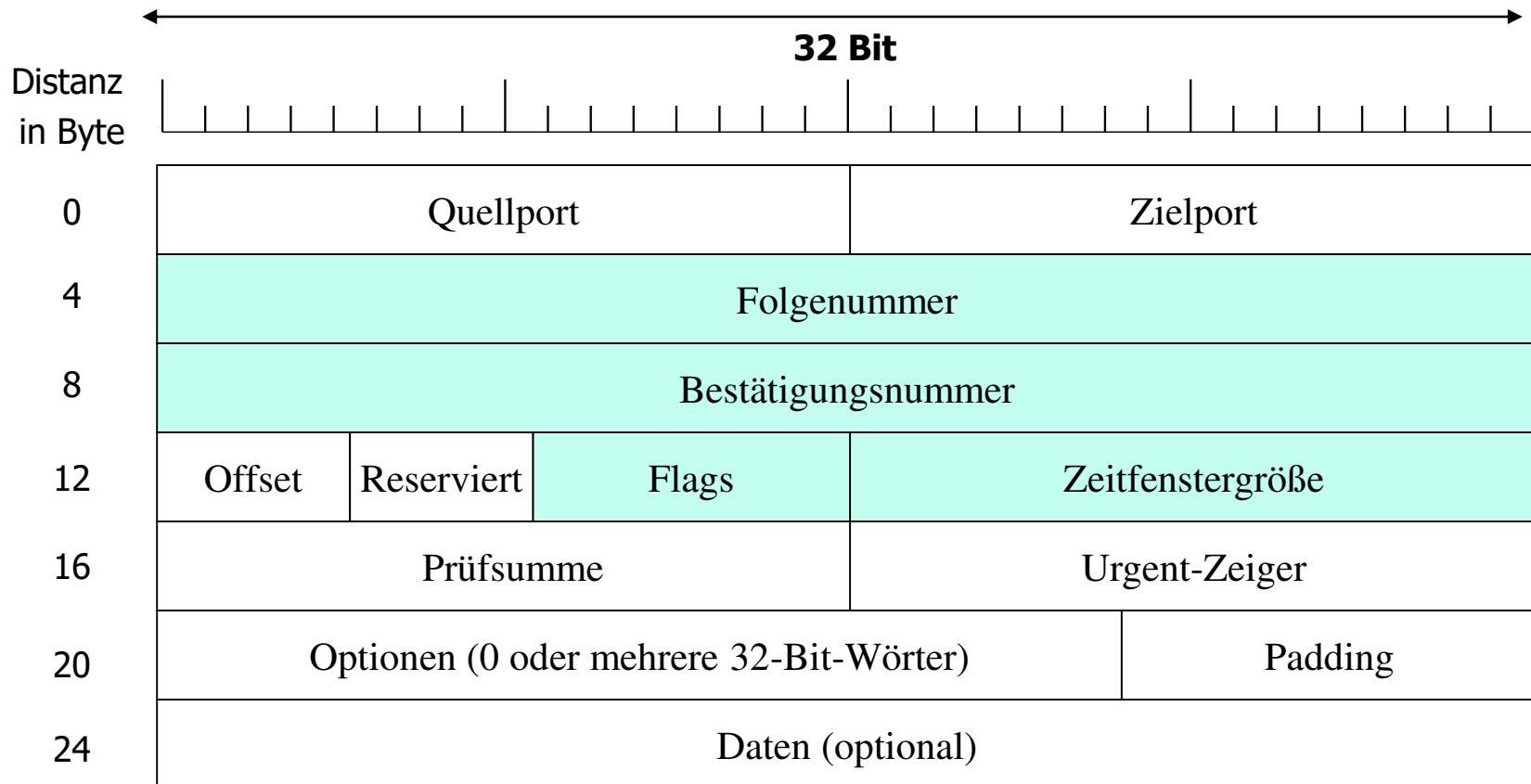
## Überblick

---

- Der Sliding Window Mechanismus erlaubt die Übertragung von mehreren TCP-Segmenten, bevor ein **ACKnowlegde** eintrifft
  - Auch ein *Piggypacking* ist erlaubt: Neue Daten können mit einer Bestätigung gesendet werden
- Bei TCP funktioniert Sliding Window auf Basis von Octets (Bytes)
- Die Octets (Bytes) eines Streams sind sequenziell nummeriert
- Flusskontrolle wird über das durch den Empfänger veranlasste Ausbremsen der Übertragung erreicht

# TCP-Header (PCI, Protocol Control Information)

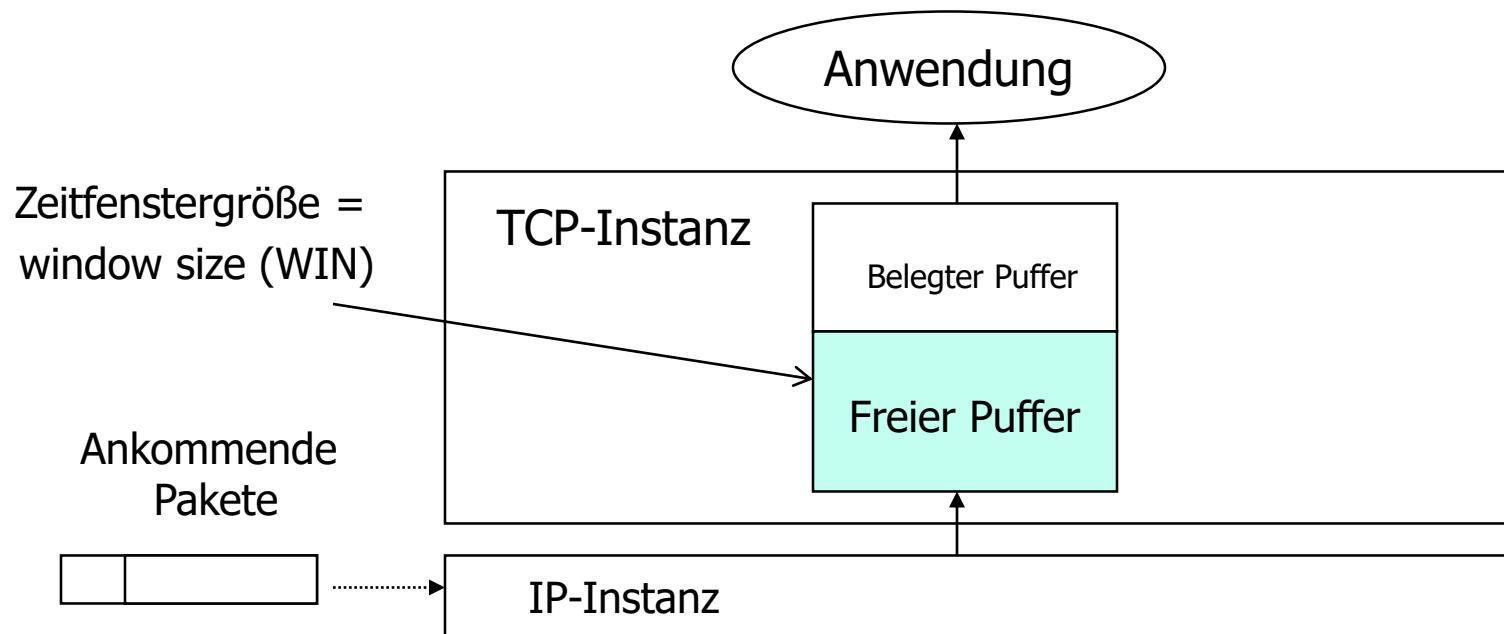
---



# Sliding Window Mechanismus

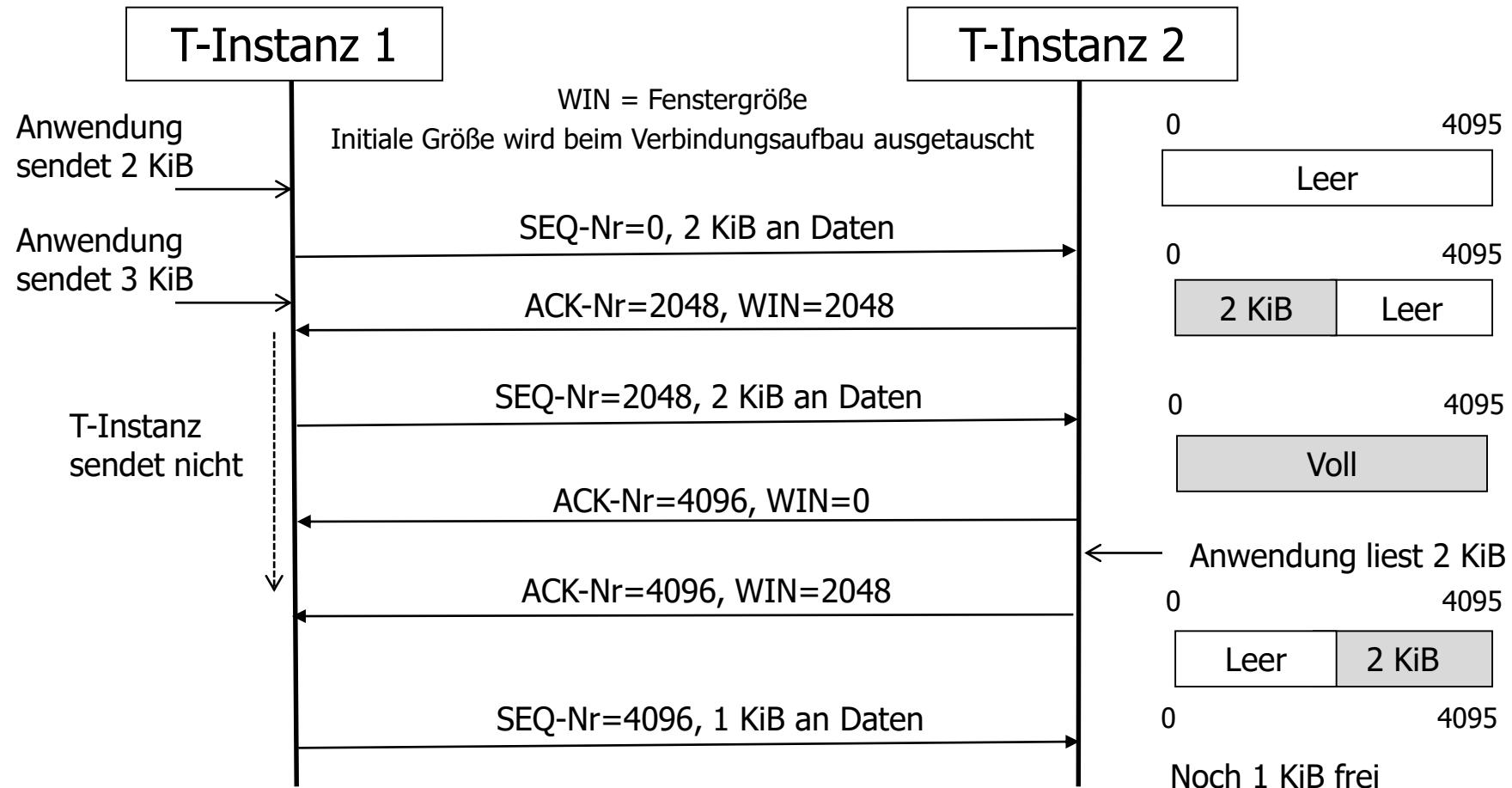
## Pufferverwaltung

- Die TCP-Instanzen reservieren beim Verbindungsaufbau Pufferspeicher für abgehende und ankommende Daten
- Empfänger informiert den Sender über den freien Platz in seinem Empfangspuffer



# Sliding-Window-Mechanismus

## Beispielablauf



Nach Tanenbaum, A.; Wetherall, D.: Computer Networks, 5. Auflage, Pearson Studium, 2011

## Einschub

### Initiale TCP-Fenstergröße

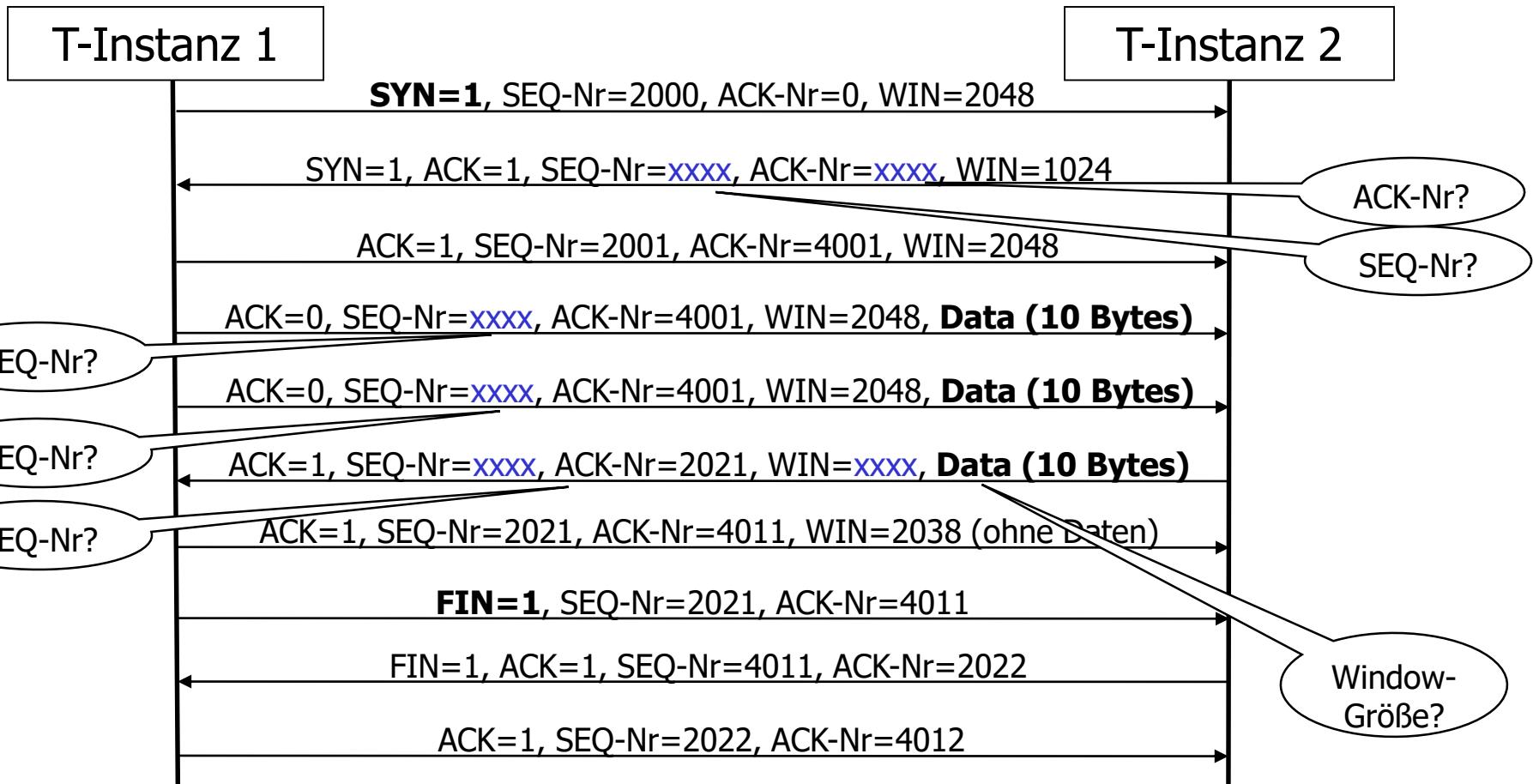
---

- Die initiale Fenstergröße für TCP wird im RFC 3390 (Increasing TCP's Initial Window) vorgegeben
- Vorher muss die MSS festgelegt werden, z.B. durch Aushandeln beim Verbindungsaufbau
- Die Berechnung ergibt sich getrennt für jede Senderichtung aus folgender Formel (gemessen in Bytes):

$$\begin{aligned} \text{Initiale Fenstergröße} \\ = \min (4 * \text{MSS}, \max (2 * \text{MSS}, 4380)) \end{aligned}$$

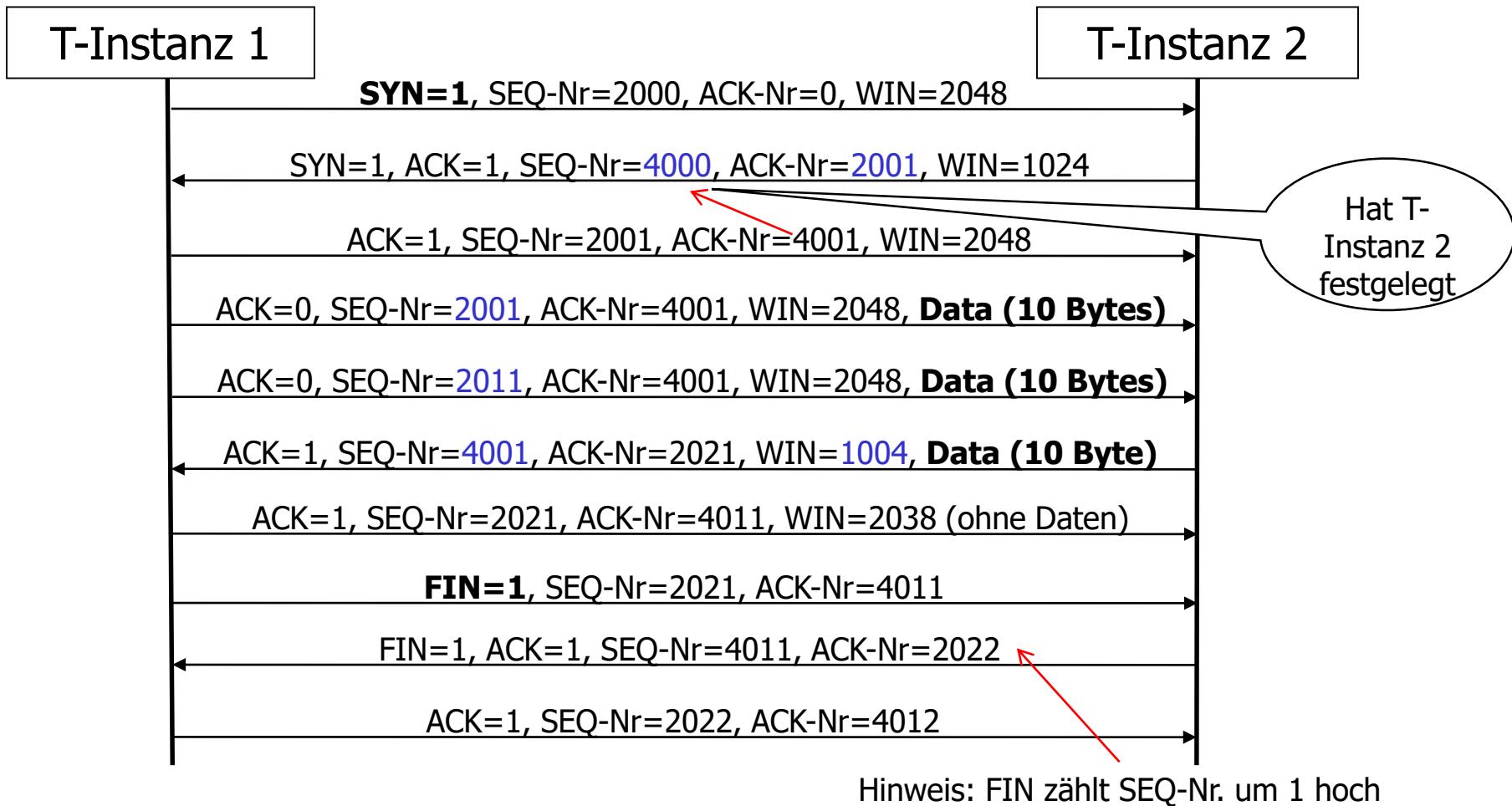
# Nachrichtenfluss: Kleine Übung

Ports vernachlässigt



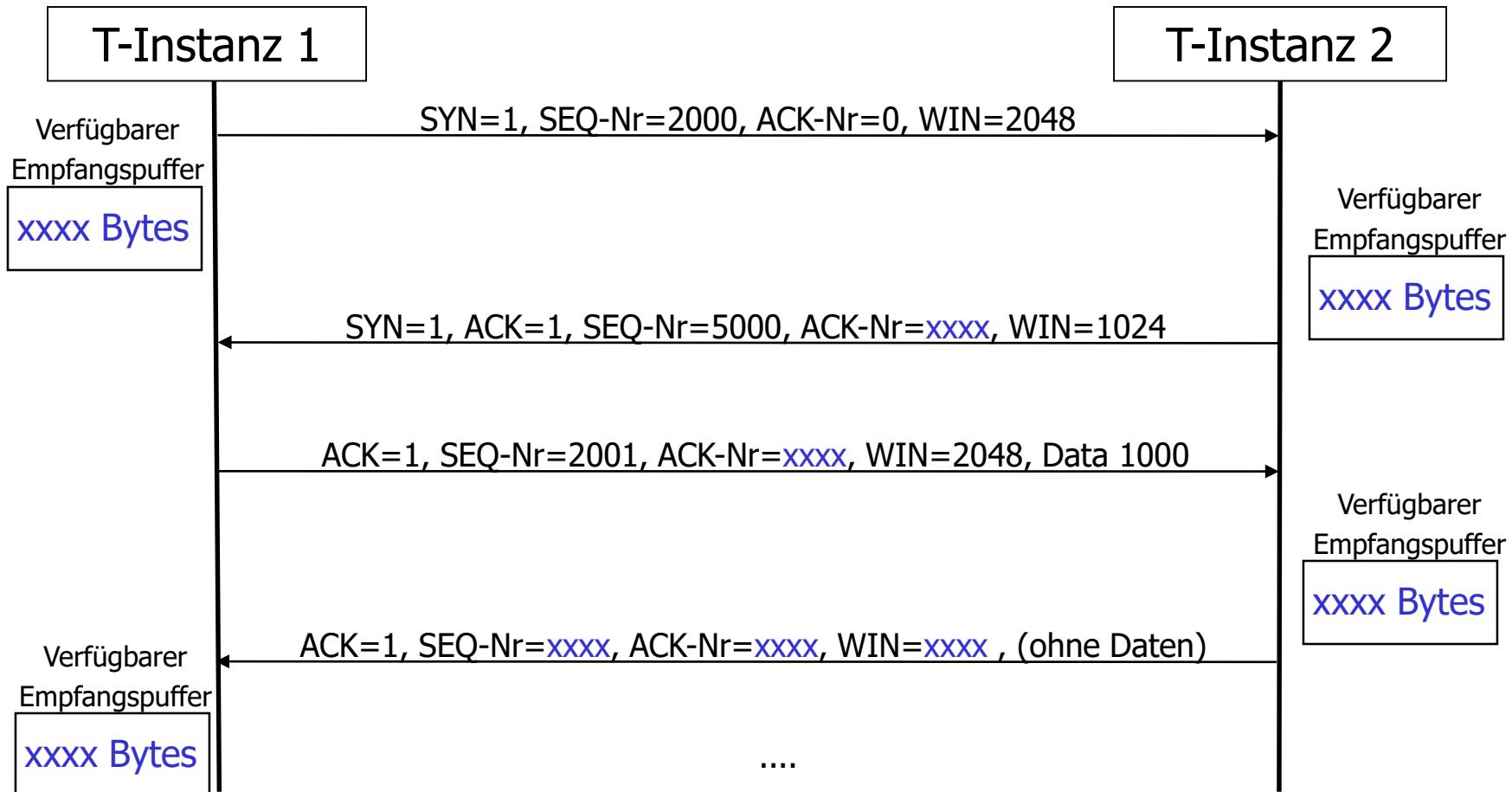
# Nachrichtenfluss: Lösung

Ports vernachlässigt



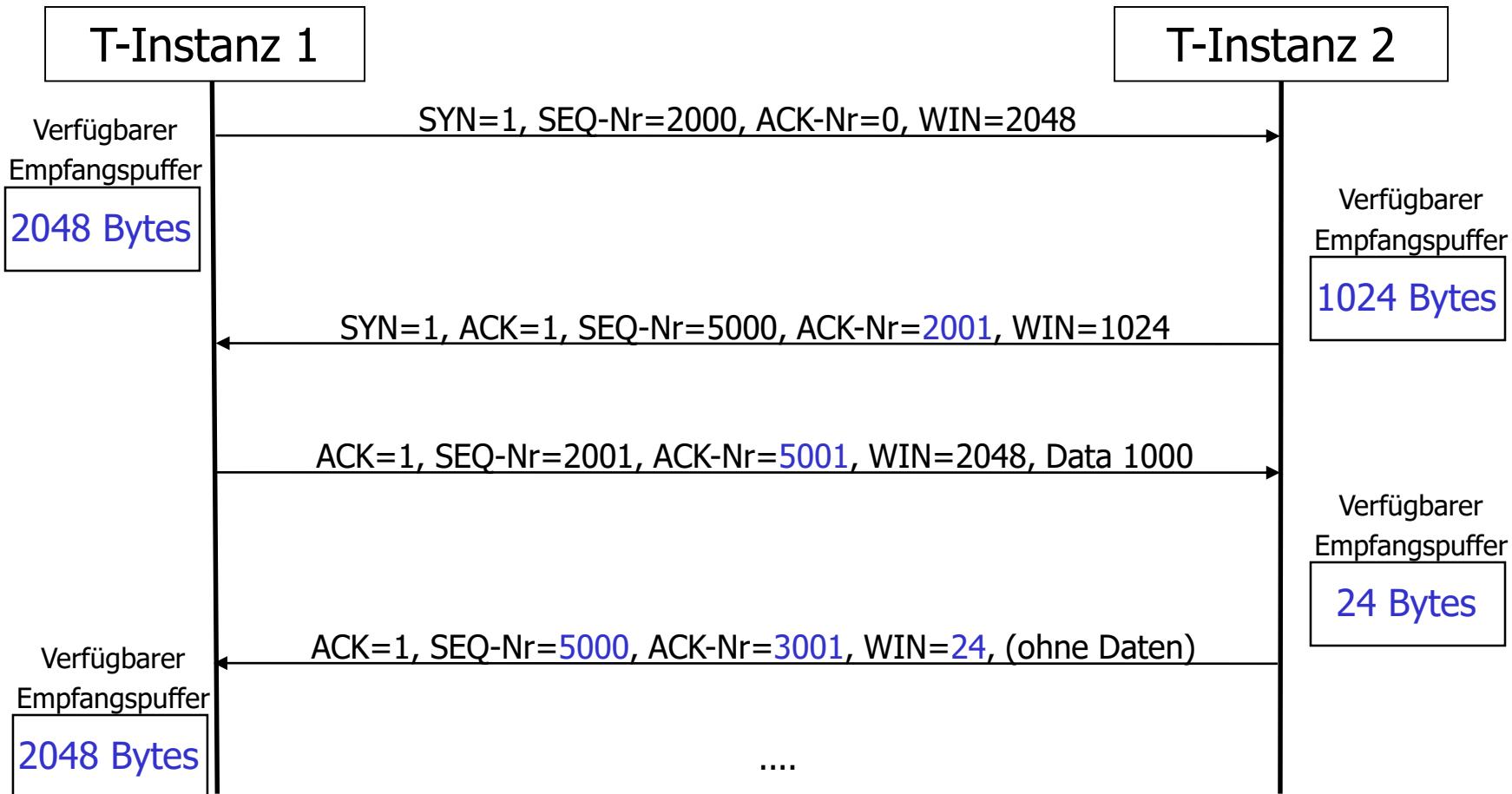
# Nachrichtenfluss: Noch eine Übung

Ports vernachlässigt



# Nachrichtenfluss: Lösung

Ports vernachlässigt



# Überblick über Transportprotokolle

---

1. Einordnung und Aufgaben von TCP
2. Der TCP-Header
3. Verbindungsauf- und -abbau, Datenübertragungsphase
4. Sliding-Window-Mechanismus
- 5. Optimierungen: Algorithmen von Nagle und Clark, Silly Window Syndrom**
6. Staukontrolle
7. TCP-Timer
8. TCP-Zustandsautomat
9. UDP (User Data Protocol)

# Algorithmus von Nagle: Optimierung des Sendeverhaltens (1)

---

## ■ Problem

- Sendeprozess sendet immer kleine Pakete
- Dies verursacht viel Overhead

## ■ Lösungsansatz von Nagle

- Der Nagle-Algorithmus versucht dies zu optimieren (RFC 1122)
- Bei einem Sendeaufruf über die Socket-Schnittstelle wird zunächst nur 1 Byte bzw. ein kleines Segment gesendet
- Danach werden Daten im Sendepuffer gesammelt, bis eine MSS erreicht ist, und dann wird erst wieder ein Segment gesendet
- Wenn aber eine ACK-PDU empfangen wird, wird der aktuelle Sendepufferinhalt sofort gesendet

# Algorithmus von Nagle

## Optimierung des Sendeverhaltens (2)

---

- **Kritik**
  - Das Nagle-Sendeverhalten ist schlecht bei Anwendungsprotokollen wie telnet oder ssh oder auch für Realtime-Anwendungen
  - Hier ist eine sofortiges Senden notwendig, damit das Anwendungsprotokoll wie gewünscht arbeitet
  - Für diese Anwendungsprotokolle ist das Nagle-Verhalten über eine Socket-Option (NO\_DELAY) abschaltbar
  - in Java: `Socket.setTcpNoDelay(boolean)`
- **Hinweis**
  - Das Setzen des PSH-Flag wäre laut TCP-Spezifikation auch eine Möglichkeit, um sofort zu senden, Socket-API lässt Flag aber nicht setzen!!

# Algorithmus von Nagle

## Pseudocode

---

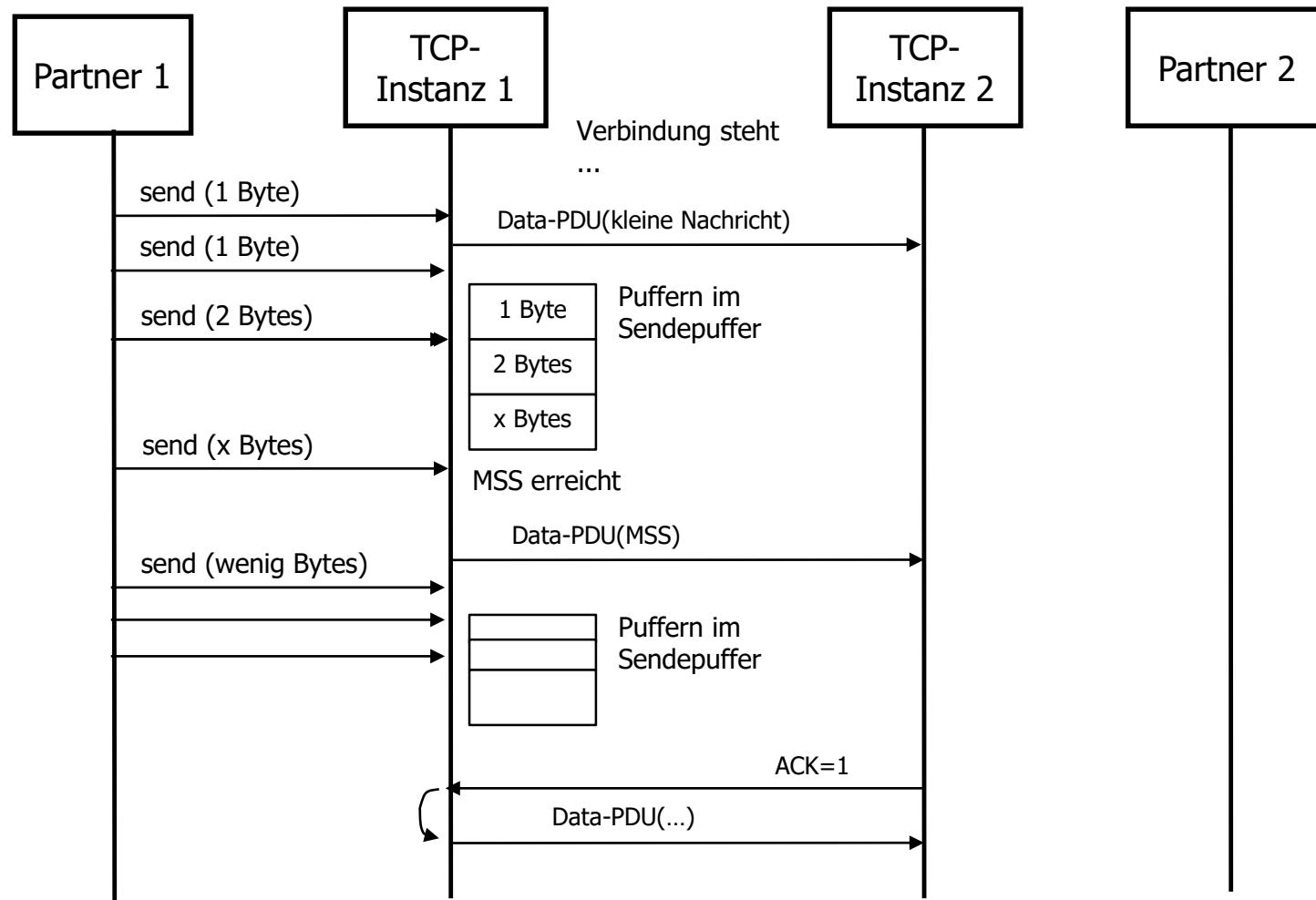
### Nagle-Algorithmus {

```
If (neue Daten zum Senden vorhanden) {  
    if (window size >= MSS) and (verfügbare Datenlänge >= MSS)  
        Sende sofort ein komplettes Segment mit Länge = MSS;  
    } else {  
        if (es sind noch Daten nicht bestätigt, ACK fehlt)  
            Lege Daten in den Sendepuffer solange bis ein ACK  
            empfangen wird;  
        } else {  
            Sende Daten sofort;  
        }  
    } // window size ...  
} // neue Daten ...  
}
```

---

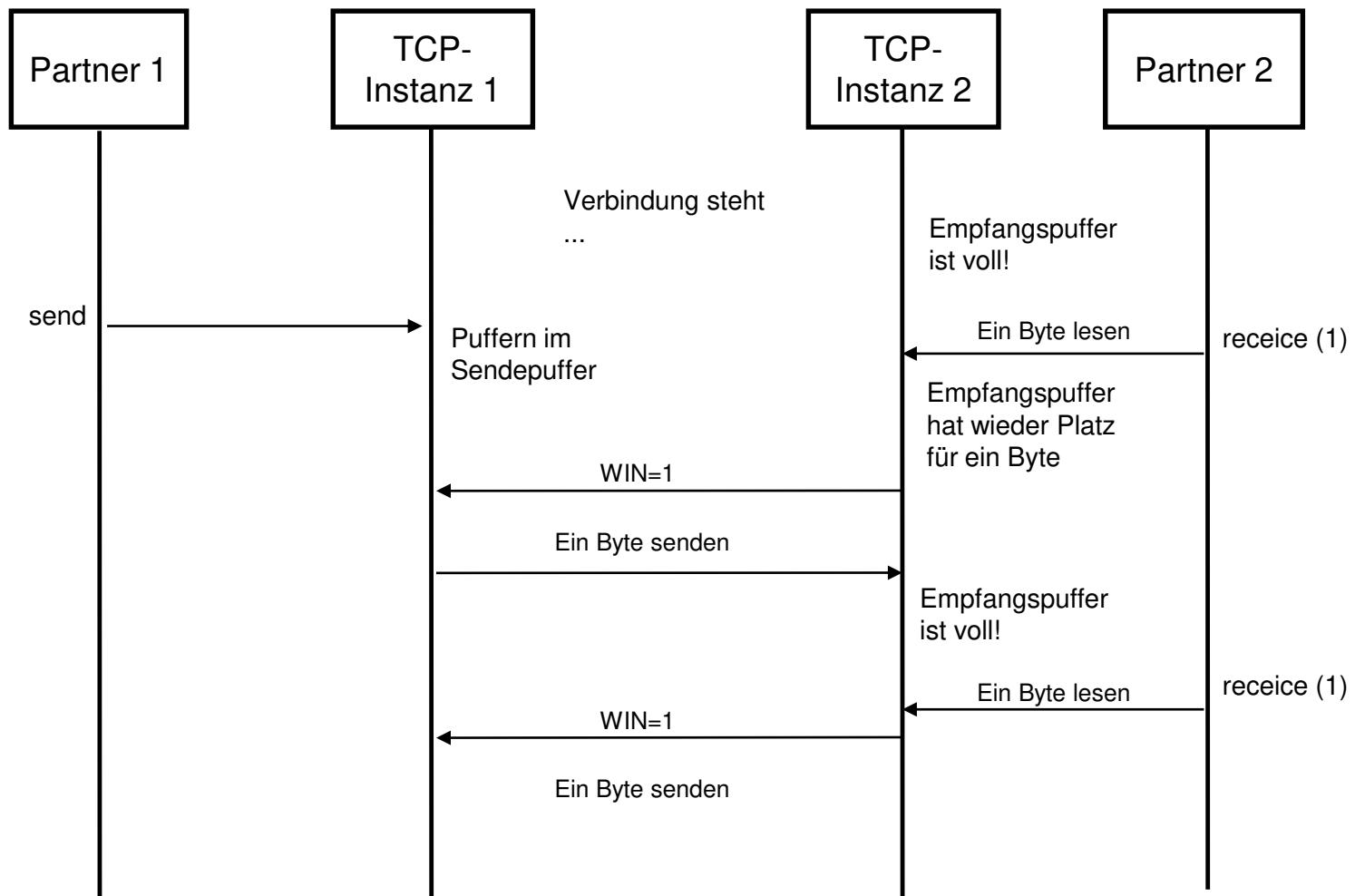
# Algorithmus von Nagle

## Ablaufbeispiel



# Silly-Window-Syndrom

## Empfangsprozess sehr langsam



# Silly-Window-Syndrom

## Lösungsansatz: Algorithmus von Clark

---

### ■ Problem

- Empfängerprozess holt Daten zu langsam und in kleinen Mengen ab und gibt immer nur einen kleinen Sendekredit

### ■ Clarks Lösung

- Es wird verhindert, dass der Sende-Instanz ständig kleine Segmente sendet, weil der Sendekredit sehr klein ist
- Wie wird es gemacht?
  - Empfangsinstanz sendet ACK-PDU mit **Window Size** (Fenstergröße) = **0 (kein Sendekredit)**
  - Es wird solange verzögert, bis der Empfänger eine Segmentlänge (MSS) gelesen hat *oder* der Empfangspuffer halb leer ist
  - Dann erst wird eine ACK-PDU mit neuer WIN-Angabe und entsprechendem Sendekredit gesendet

---

Quelle: Tanenbaum, A.; Wetherall, D.: Computer Networks, 5. Auflage, Pearson Studium, 2011

# Überblick über Transportprotokolle

---

1. Einordnung und Aufgaben von TCP
2. Der TCP-Header
3. Verbindungsauf- und -abbau, Datenübertragungsphase
4. Sliding-Window-Mechanismus
5. Optimierungen: Algorithmen von Nagle und Clark, Silly Window Syndrom
- 6. Staukontrolle**
7. TCP-Timer
8. TCP-Zustandsautomat
9. UDP (User Data Protocol)

# Grundüberlegungen zur Staukontrolle

---

- **1986** gab es im Internet massive Stausituationen
- Seit **1989** ist Staukontrolle ein wichtiger Bestandteil von TCP
  - J. Nagle: Congestion Control in IP/TCP Internetworks, in RFC 896, 1984
- **Grundüberlegungen**
  - **IP reagiert nicht** auf Überlastsituationen, ausgenommen über neue Ansätze (wie ECN = Explicit Congestion Notification)
  - Verlust eines TCP-Segmentes wird von TCP als Auswirkung einer **Stausituation** im Netz interpretiert
  - Annahme: Netze sind prinzipiell stabil, **eine fehlende ACK-PDU** nach dem Senden einer Nachricht **wird als Stau** im Netz betrachtet

## TCP-Lösungsansatz: Slow-Start-Verfahren, TCP Tahoe

---

- Der Algorithmus dafür wird als reaktives **Slow-Start-Verfahren** (RFC 1122, RFC 5681, ...) bezeichnet
  - Andere Bezeichnungen: **Slow Start and Congestion Avoidance** oder **TCP Tahoe**
  - TCP-Implementierungen **müssen** das Verfahren unterstützen
- Das Slow-Start-Verfahren kennt **zwei Phasen**:
  - Slow-Start-Phase
  - Probing-Phase (Congestion Avoidance-Phase)

# Slow-Start-Verfahren: TCP Tahoe, Idee

---

- **Idee**
  - Verfahren baut auf dem Erkennen von Datenverlusten in der Ende-zu-Ende-Kommunikation auf
  - Jede Verbindung wird separat betrachtet
  - TCP **tastet** sich an die maximale Datenübertragungsrate einer Verbindung **heran**
  - Die **Übertragungsrate** wird im Überlastfall beim Empfänger vom Sender massiv **gedrosselt**
  - **TCP-Quittungen** (ACK-PDUs) dienen als **Taktgeber** für den Sender

## Slow-Start-Verfahren: Staukontrollfenster

---

- Neben dem Empfangsfenster wird ein **Überlastfenster = Staukontrollfenster** eingeführt:
  - Jede Verbindungsseite verwaltet ein Empfangsfenster und zusätzlich ein Überlastfenster für den jeweiligen Partner im Verbindungskontext
  - Überlastfenstergröße wird dynamisch angepasst
  - Initiale Größe des Überlastfensters:
    - Ursprünglich so groß wie MSS
    - Im Laufe der Jahre Veränderung auf 10, später 15 Segmente, ...
    - Ständig in Diskussion (Implementierungsentscheidung)
- Dynamische Anpassung des Sendekredits:

Sendekredit für eine TCP-Verbindung =  $\min$  (Überlastfenstergröße für Partner, Empfangsfenstergröße für Partner)

# Slow-Start-Verfahren: Erläuterung der Slow-Start-Phase

---

## ■ **Slow-Start-Phase**

- Sender und Empfänger einigen sich auf eine erste sendbare TCP-Segmentlänge (MSS = z.B. 1024 Bytes)
- Ein Schwellwert (Threshold) wird festgelegt, initial 64 KiB
- Sender sendet zunächst ein Segment dieser Länge bzw. die implementierte Überlastfenstergröße
- Jeweils Verdoppelung der Überlastfenstergröße bei erfolgreicher Übertragung aller Segmente maximal bis zur Empfangsfenstergröße (exponentielle Steigerung)
  - Für jedes bestätigte TCP-Segment wird die Überlastfenstergröße um die Segmentlänge erhöht
- Bei Erreichen des Schwellwerts geht die Übertragung in die Probing-Phase über

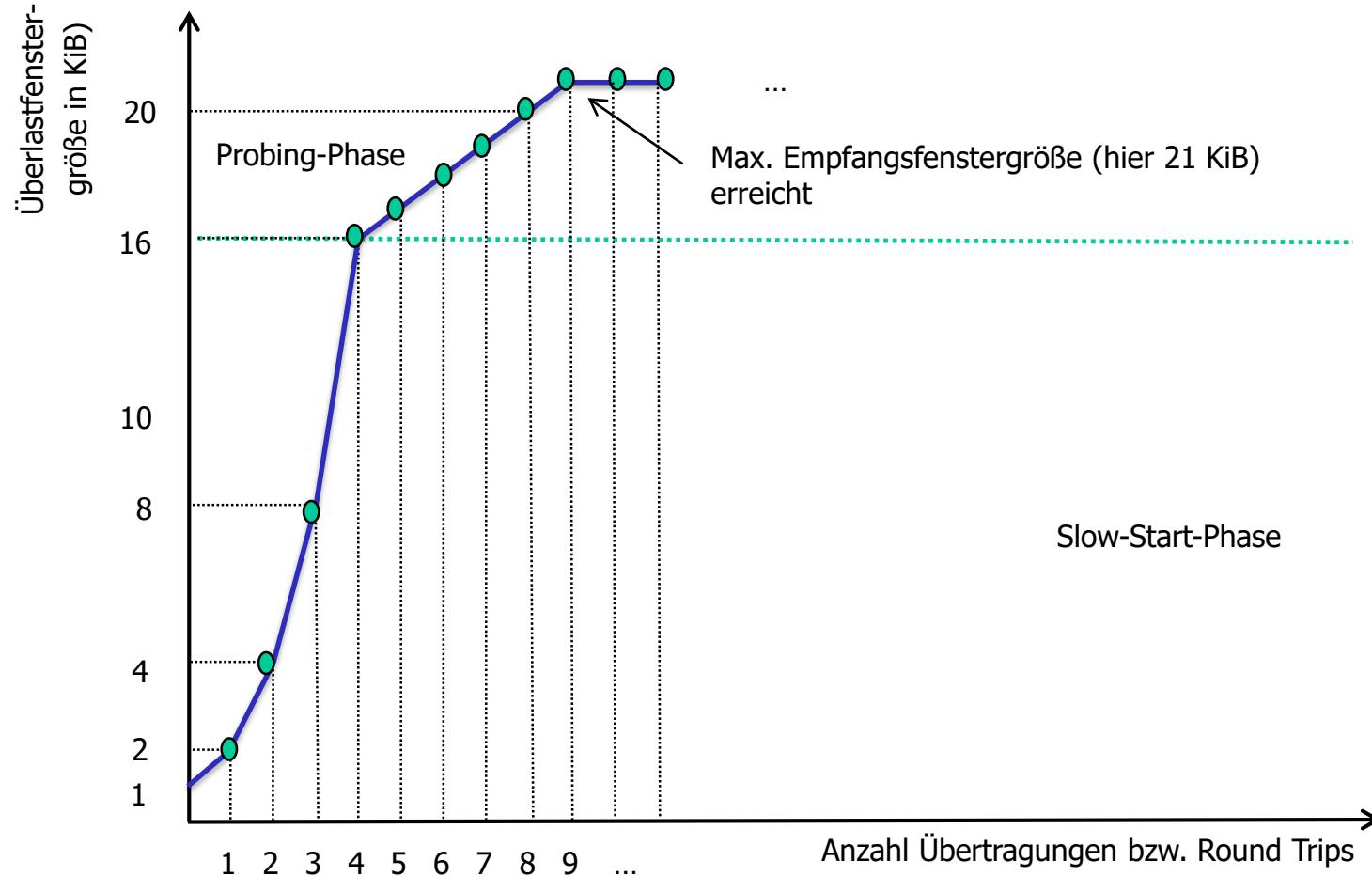
# Slow-Start-Verfahren: Erläuterung der Probing-Phase

---

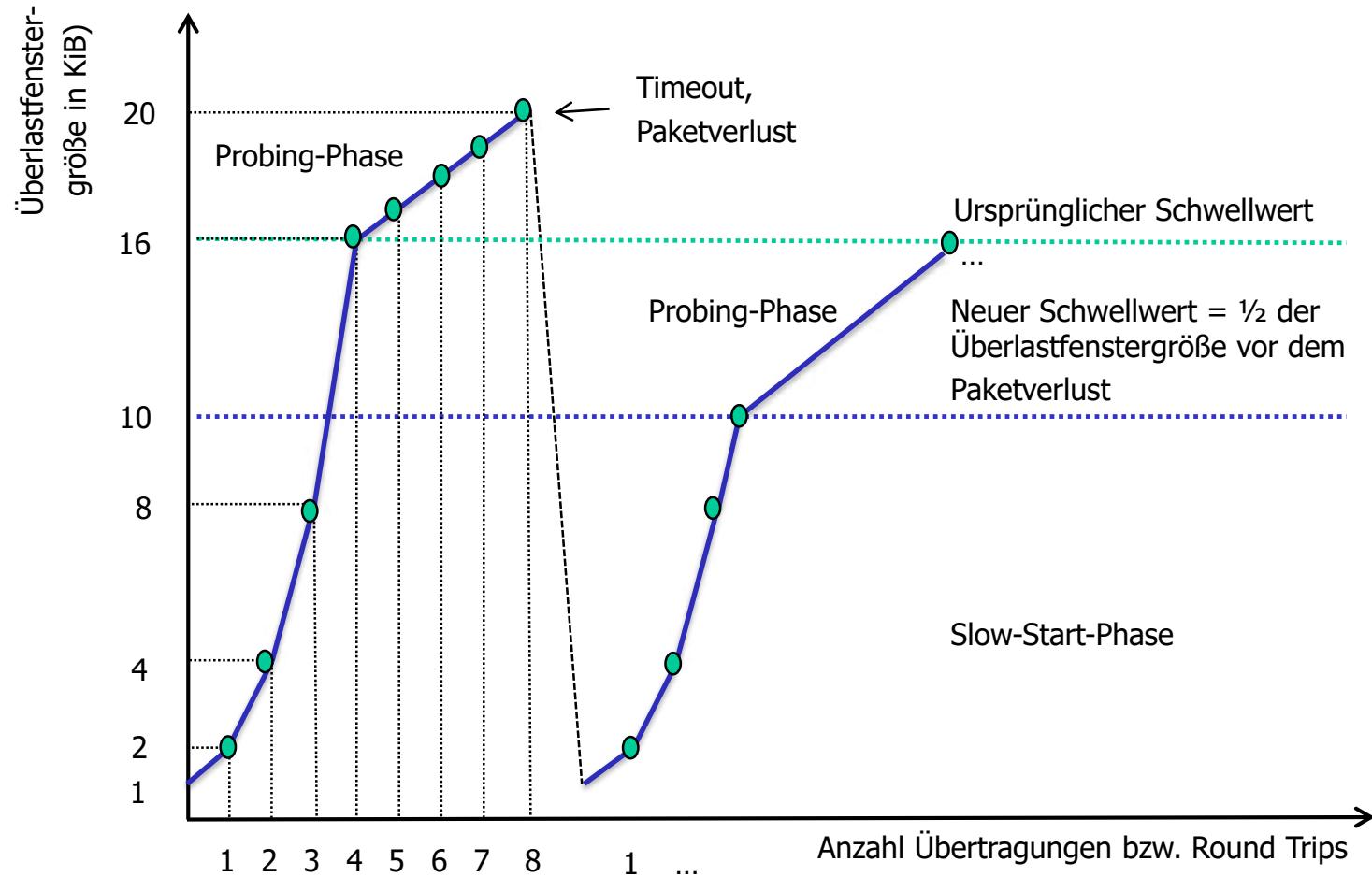
## ■ Die Probing-Phase (Congestion Avoidance-Phase)

- Bei jeder empfangenen Quittung wird die Größe des Überlastfensters „nur“ noch um eine Segmentlänge (MSS) erhöht
- Berechnung des Überlastfenster bei Ankunft der ACK-PDUs für alle übertragenen Segmente innerhalb des Überlastfensters:
  - Überlastungsfenster um eine Segmentlänge (MSS) vergrößern
  - Weiterhin gilt:  
Sendekredit für eine TCP-Verbindung =  $\min$  (Überlastfenstergröße für Partner, Empfangsfenstergröße für Partner)
- Es tritt kein Problem auf
  - Überlastfenstergröße steigt bis zur Empfangsfenstergröße und bleibt dann konstant
  - Bei Änderung des Empfangsfensters wird Sendekredit angepasst

# Slow-Start-Verfahren: Beispielhafter Ablauf bis zur Empfangsfenstergröße



# Slow-Start-Verfahren: Beispielhafter Ablauf mit Timeout



## Slow-Start-Verfahren: Beispielhafter Ablauf - Erläuterung

---

- Slow-Start-Phase: Die initiale TCP-Segmentlänge im Beispiel 1 KiB (MSS wird ausgetauscht beim Verbindungsaufbau)
- Der initialer Schwellwert ist im Beispiel 16 KiB, ab hier beginnt die Probing-Phase
- Ein Paketverlust und damit ein Retransmission-Timeout tritt im Beispiel bei Überlastfenstergröße von 20 KiB ein
- Der Schwellwert wird danach auf 10 KiB gesetzt (= 1/2 der vorherigen Überlastungsfenstergröße)
- Der Sendekredit beginnt nach dem Timeout wieder mit einer Segmentlänge von 1 KiB
- Ab dem neuen Schwellwert wird wieder in die Probing-Phase gewechselt

# Slow-Start-Verfahren: Reaktion auf fehlendes ACK (1)

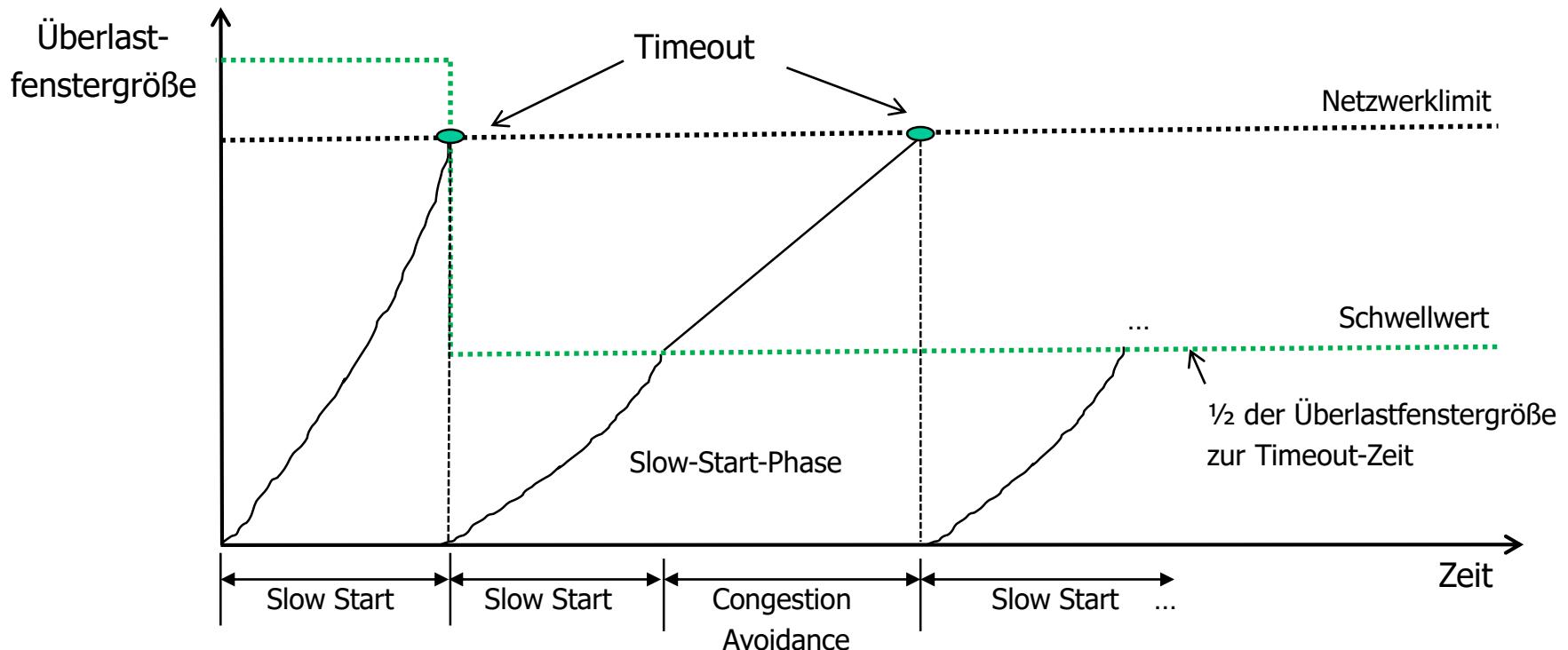
---

- **ACK wird nicht empfangen**
  - Man geht davon aus, dass ein weiterer Sender im Netz hinzugekommen ist oder dass ein Problem im Netz vorliegt
  - Mit diesem neuen Sender muss man die Pfadkapazität teilen
  - Der Schwellwert wird um die Hälfte der als letztes gesendeten Segmentanzahl (also  $\frac{1}{2}$  der aktuellen Überlastfenstergröße) reduziert
  - Die Überlastfenstergröße wird wieder auf das Minimum heruntergesetzt, z.B. 1 KiB
- **Die Timerlänge ist entscheidend!**
  - Zu lang: Evtl. Leistungsverlust wegen zu langer Wartezeiten
  - Zu kurz: Erhöhte Last durch erneutes Senden
  - Dynamische Berechnung anhand der Umlaufzeit eines Segments

## TCP-Lösungsansatz: Reaktion auf fehlendes ACK (2)

- Ablauf bei TCP Tahoe nach einem fehlenden ACK:

- Stausituation wird angenommen
- Starke Drosselung wird eingeleitet, Schwellwert wird auf  $\frac{1}{2}$  der Überlastfenstergröße reduziert



# Weitere Algorithmen: Fast Recovery nach RFC 5681 (1)

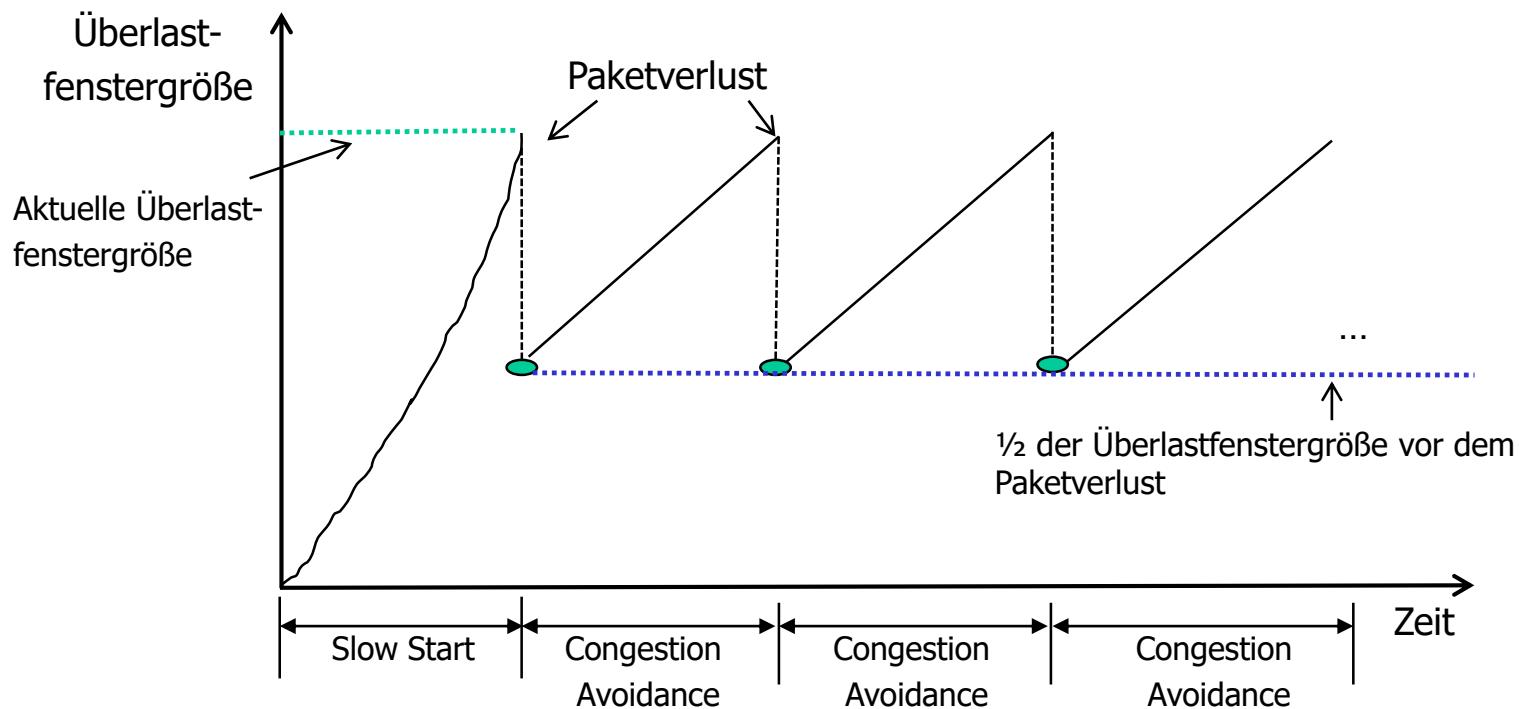
(siehe auch implizites NAK bzw. Fast Retransmit)

---

- **TCP Reno** als Weiterentwicklung zu TCP Tahoe
  - Verfährt bei Timeout wie TCP Tahoe
  - Ergänzendes dazu: **Fast Retransmit/Fast Recovery-Algorithmus**
    - Empfang von 4 Quittierungen (drei ACK-Duplikate) für eine TCP-PDU veranlasst sofortige Sendewiederholung, Timeout wird nicht abgewartet
    - Die Sendeleistung wird entsprechend angepasst
    - Der Schwellwert wird auf die Hälfte des aktuellen Staukontrollfensters reduziert
    - Da nur von einem Paketverlust und nicht von einem Stau im Netzwerk ausgegangen wird, wird die Überlastfenstergröße halbiert und mit Congestion Avoidance weitergemacht
  - Der Empfänger sendet ACK-Duplikate, wenn ein TCP-Segment ankommt, das nicht in die Reihenfolge passt

## Weitere Algorithmen: Fast Recovery nach RFC 5681 (2)

- Ablauf bei TCP Reno nach einem Paketverlust:
  - Stau ist eher unwahrscheinlich, weil andere Pakete durchkommen
  - TCP Reno halbiert das Staukontrollfenster und fährt mit Congestion Avoidance fort



# Überblick über Transportprotokolle

---

1. Einordnung und Aufgaben von TCP
2. Der TCP-Header
3. Verbindungsauf- und -abbau, Datenübertragungsphase
4. Sliding-Window-Mechanismus
5. Optimierungen: Algorithmen von Nagle und Clark, Silly Window Syndrom
6. Staukontrolle
- 7. TCP-Timer**
8. TCP-Zustandsautomat
9. UDP (User Data Protocol)

# TCP-Timer im Überblick

---

- TCP verwendet einige Timer, Beispiele:
  - **Retransmission Timer (RTO)**
    - Überwachung der TCP-Segmente ursprünglich nach Karn-Algorithmus, 1988 → adaptives Verfahren
    - Wenn Timer abläuft, wird er verdoppelt (max. 60 s) und das Segment erneut gesendet (Wiederholung)
  - **Keepalive Timer**
    - Nach Ablauf wird versucht, den Partner zu erreichen
    - Gelingt es, bleibt Verbindung bestehen
    - Beispieleinstellung unter Windows im Registry: **KeepAliveTime = 2 h**
  - **Timed Wait Timer**
    - Timer für Verbindungsabbau auf aktiver Verbindungsabbauseite (Default: 120 s), Schätzung:  $2 * MSL$  (maximum segment lifetime)
    - Sichert, dass eine neue Verbindung keine alten Daten bekommt

## Ergänzung für Interessierte: Berechnung des Retransmission Timers (1)

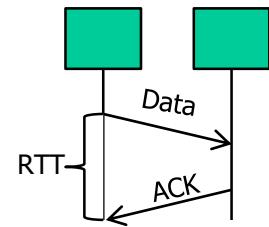
---

- Der Berechnungsalgorithmus wurde im RFC 6298 aktualisiert
- Zustandsvariablen im Verbindungskontext
  - *SRTT* (Smoothed Round Trip Time) Glättung, um Auswirkungen von Ausreißermessungen abzumildern
  - *RTTVAR* (Round-Trip Time Variation) macht eine Aussage über die Streuung der Berechnung.
- G gibt die Messgenauigkeit der lokalen Clock an
- Die erste Messung der RTT wird als R bezeichnet:

$$SRTT = R$$

$$RTTVar = R/2$$

$$RTO = SRTT + \max(G, K * RTTVar), \text{ mit } K = 4$$



## Ergänzung für Interessierte: Berechnung des Retransmission Timers (2)

---

- Bei jeder Folgeberechnung wird der neue RTO-Wert mit  $R'$  als jeweils zuletzt gemessene RTT, mit  $\alpha = 1/8$  und  $\beta = 1/4$ :

$$RTTVarneu = (1 - \beta) * RTTVaralt + \beta * |SRTT - R'|$$

$$SRTT = (1 - \alpha) * SRTT + \alpha * R'$$

$$RTO = SRTT + \max(G, K * RTTVarneu)$$

- Falls sich ein RTO-Wert  $< 1$  ergibt, soll RTO auf 1 gesetzt werden
- Als maximaler RTO-Wert ist 60 vorgesehen.
- Falls die Berechnung von  $RTO = 0$  ergibt:

$$RTO = SRTT + \max(G, K * RTTVar)$$

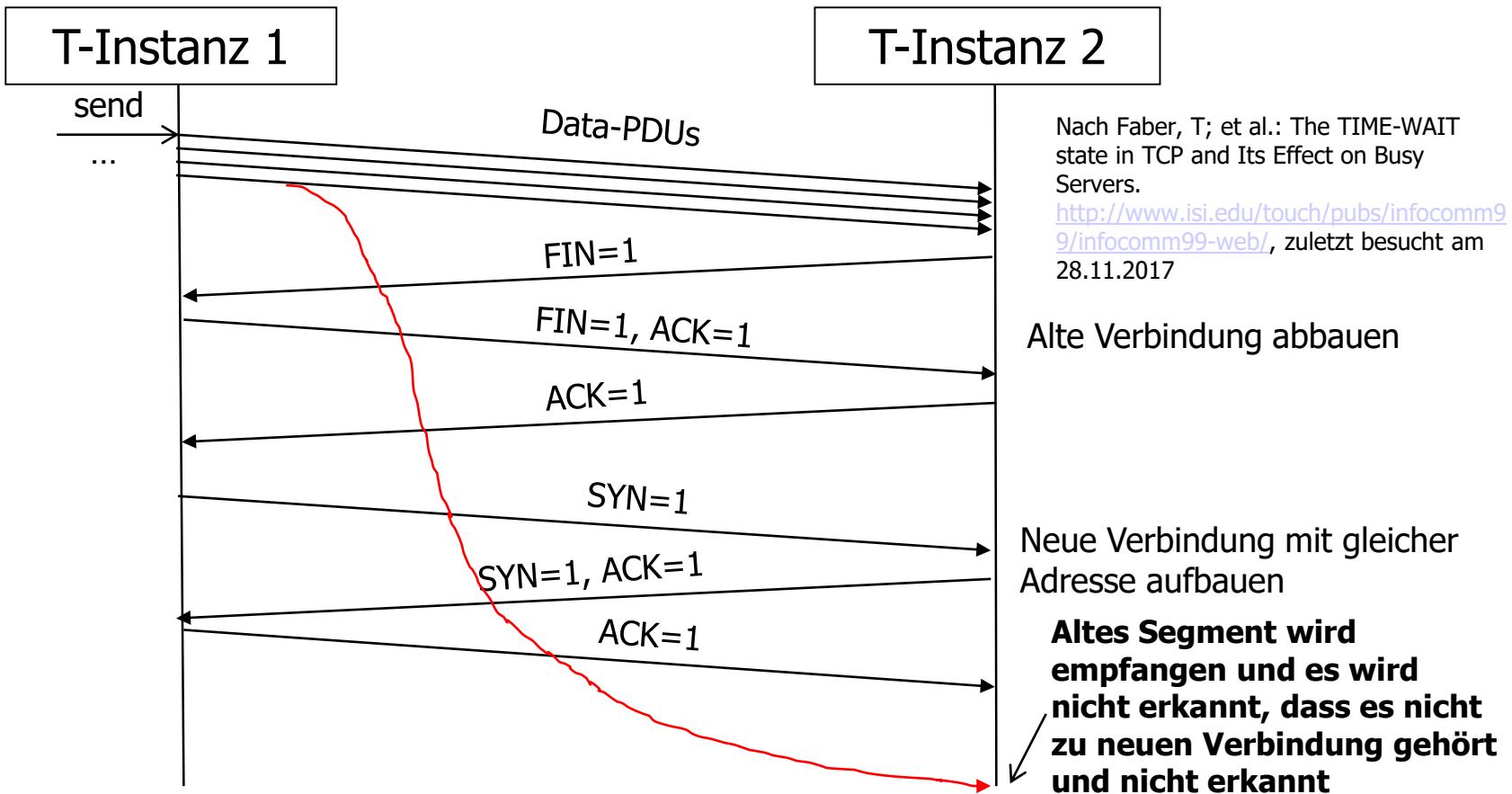
## Ergänzung für Interessierte: Berechnung des Retransmission Timers (3)

---

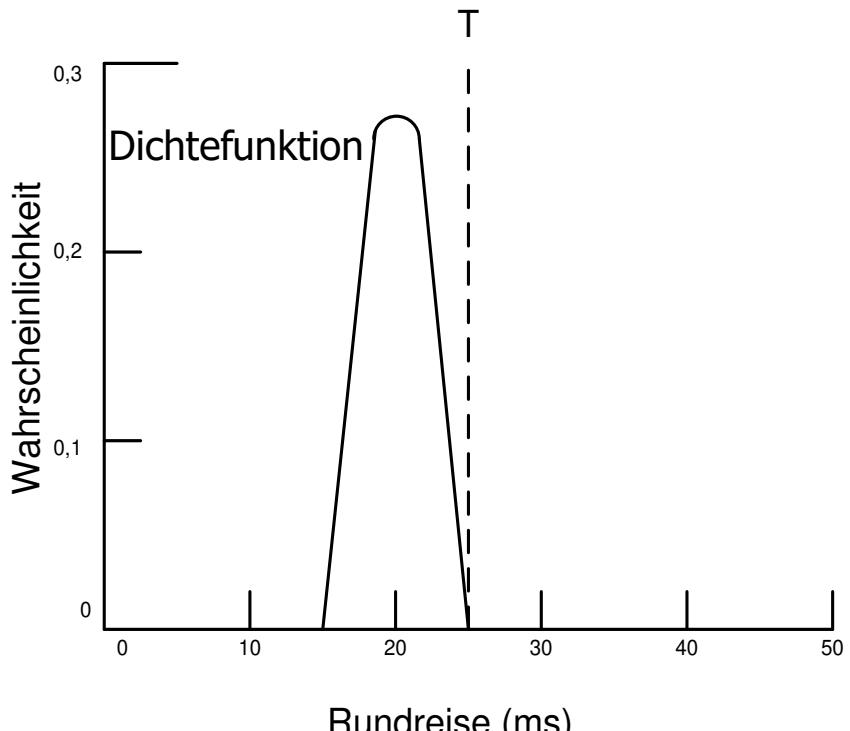
- Wenn eine Bestätigung länger als RTO ausbleibt, muss die TCP-Instanz wie folgt reagieren:
  - Der Wert von RTO muss unter Beachtung der Obergrenze von 60 Sekunden verdoppelt werden
  - Zunächst ist das erste nicht bestätigte TCP-Segment erneut zu senden
  - Wenn es sich um eine verlorene Bestätigung für einen Verbindungsauftakt handelt (SYN-Flag gesetzt), so muss RTO für die Datenübertragungsphase auf 3 Sekunden gesetzt werden
  - Die Zustandsvariablen SRTT und RTTVAR sind zu löschen und der nächste RTO-Wert ergibt sich aus der folgenden Messung

## Ergänzung zu TIMED\_WAIT

- Beobachten mit Sysinternals Windows-Tool: **TcpView**
- Problem, das vermieden werden soll:

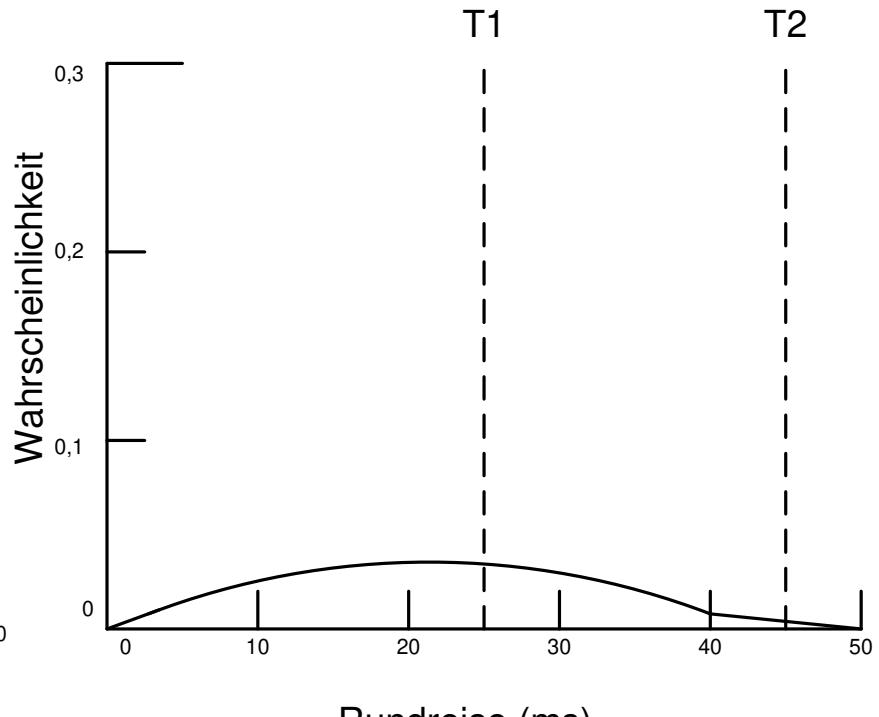


# TCP-Timer: Grundproblem



(a)

Optimale Länge des Timers  $T$



(b)

Schlechte Timer  $T_1$  (zu kurz)  
und  $T_2$  (zu lang)

Nach: Tanenbaum, A.; Wetherall, D.: Computer Networks, 5. Auflage, Pearson Studium, 2011

# Überblick über Transportprotokolle

---

1. Einordnung und Aufgaben von TCP
2. Der TCP-Header
3. Verbindungsauf- und –abbau, Datenübertragungsphase
4. Sliding-Window-Mechanismus
5. Optimierungen: Algorithmen von Nagle und Clark, Silly Window Syndrom
6. Staukontrolle
7. TCP-Timer
- 8. TCP-Zustandsautomat**
9. UDP (User Data Protocol)

# Exkurs: Endliche Zustandsautomaten (1)

---

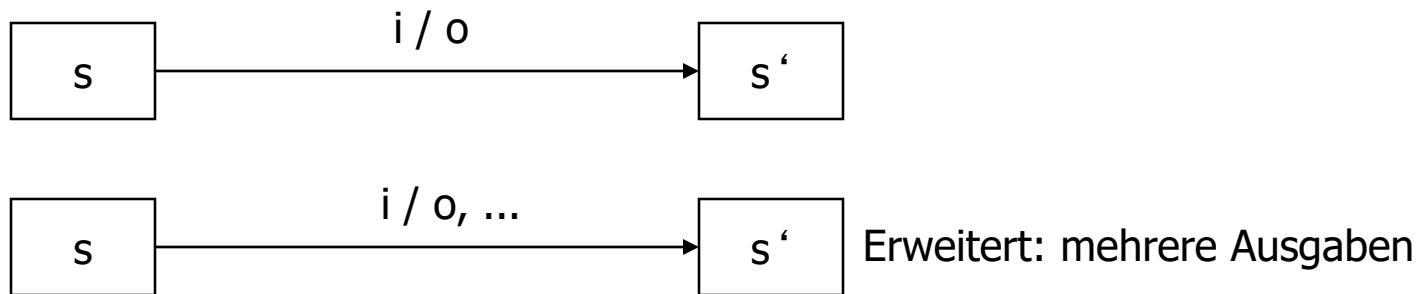
## ■ Finite State Machine (FSM)

- Deterministischer endlicher Automat mit Ausgabe (siehe Mealy-Automat): Zustandsveränderung hängt vom aktuellen Zustand und vom Input ab
- Verwendet man gerne zur groben Beschreibung des Verhaltens von Protokollinstanzen
- Ein derartiger Zustandsautomat lässt sich als 6-Tupel der Form  $\langle S, I, O, T, s_0, F \rangle$  beschreiben:
  - $S$  - endliche, nicht leere Menge von **Zuständen**
  - $I$  - endliche, nicht leere Menge von **Eingaben**
  - $O$  - endliche, nicht leere Menge von **Ausgaben**
  - $F$  - endliche, nicht leere Menge von **Endzuständen**,  $F \subseteq S$
  - $T \subseteq S \times (I \cup \{\tau\}) \times (O \cup \{\tau\}) \times S$  – eine **Zustandsüberführungsfunktion**
    - $\tau$  bezeichnet eine leere Eingabe oder leere Ausgabe
  - $s_0 \in S$  – **Initialzustand** des Automaten

## Exkurs: Endliche Zustandsautomaten (2)

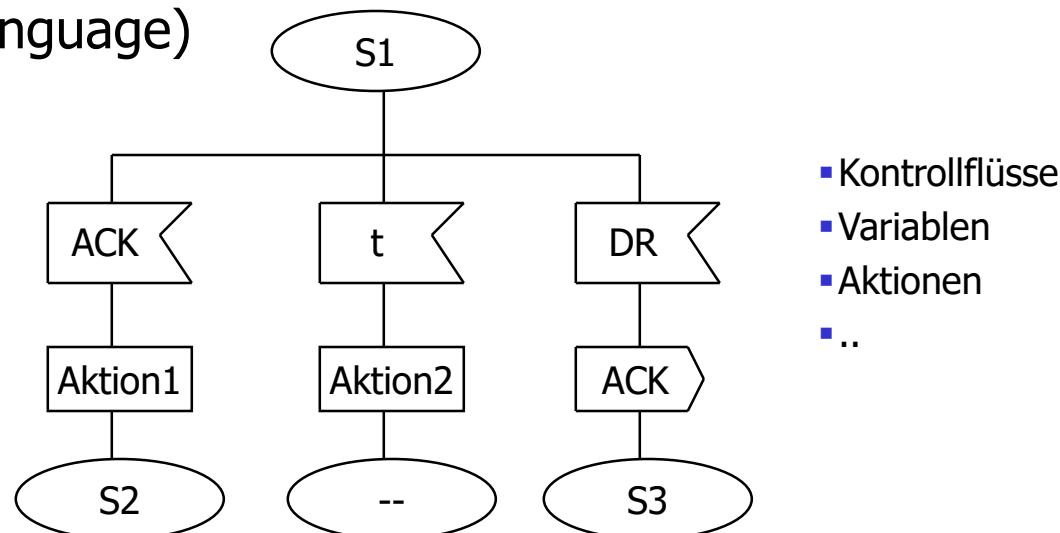
---

- Eine Transition (Zustandsübergang)  $t \in T$  ist definiert durch das Quadrupel  $\langle s, i, o, s' \rangle$  wobei
  - $s \in S$  der aktuelle Zustand,
  - $i \in (I \cup \{\tau\})$  eine Eingabe,
  - $o \in (O \cup \{\tau\})$  eine zugehörige Ausgabe und
  - $s' \in S$  der Folgezustand ist
- Grafische Darstellung eines Zustandsübergangs

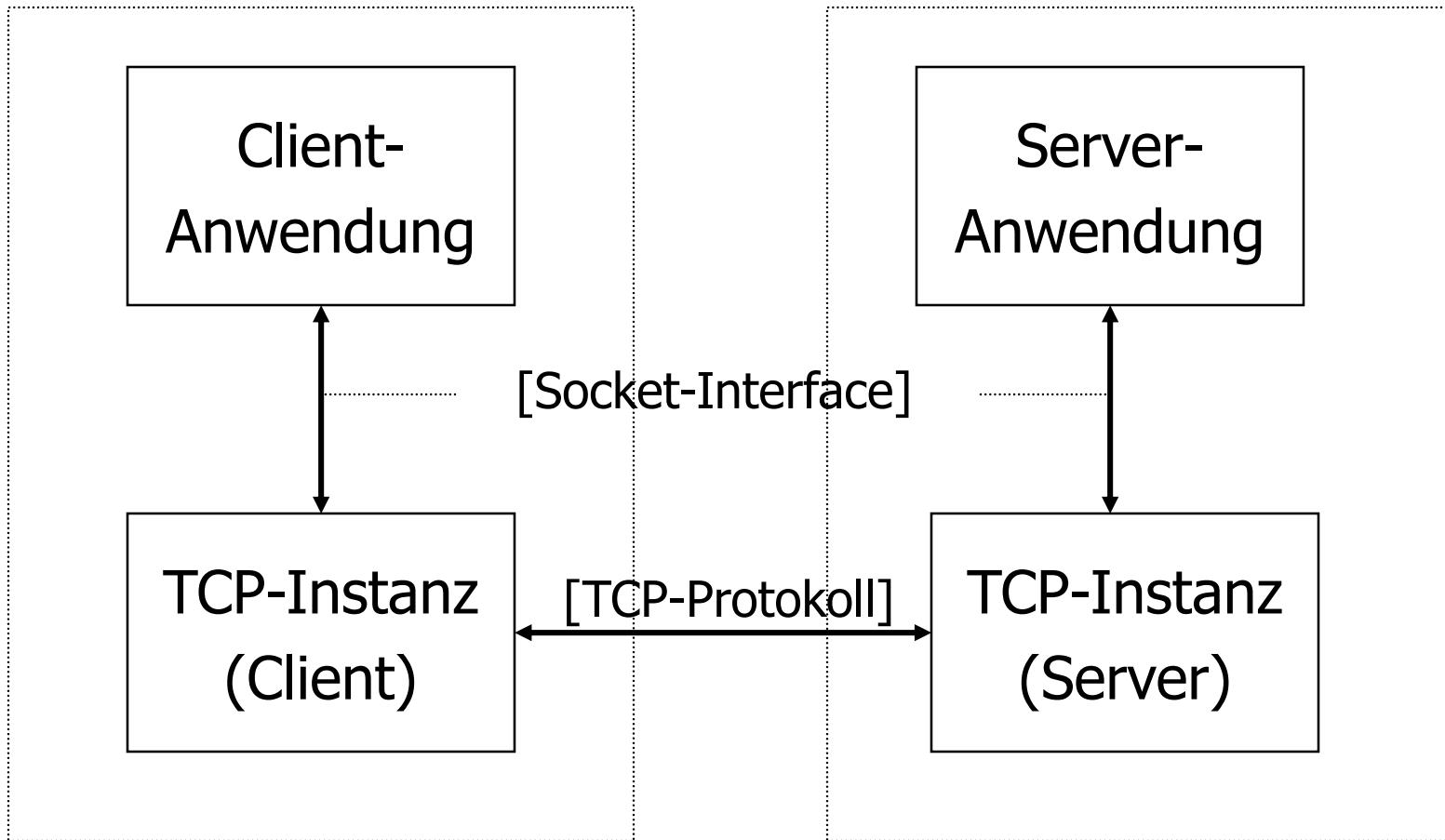


## Exkurs: Endliche Zustandsautomaten (3)

- **Nachteil** von FSM: Keine weiteren Zustandsinformationen z.B. in Variable modellierbar
- Daher in der Praxis oft Nutzung **erweiterter endlicher Automaten** (EFSM) für die Detailspezifikation, um Zustandskontakte noch besser zu beschreiben → nutzt weitere Variable neben Zustandsvariable
- Modellierung z.B. in der Sprache **SDL** (Specification and Description Language)
- Beispiel:



## Zustandsautomat allgemein



Je ein Zustandsautomat **pro Transportverbindung** wird  
in jeder beteiligten TCP-Instanz verwaltet

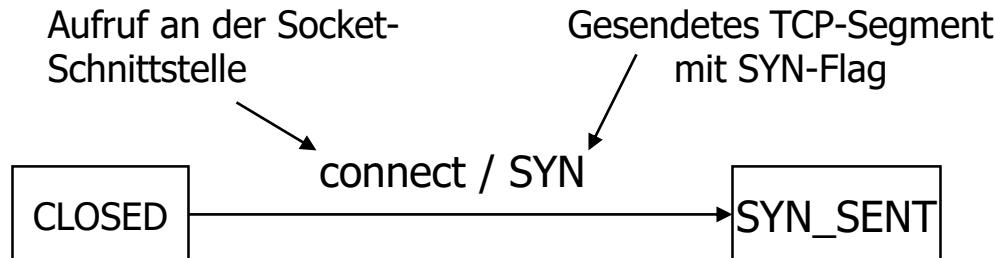
# Zustände im TCP-Zustandsautomat (nur Verbindungsmanagement)

<b>Zustand</b>	<b>Beschreibung</b>
CLOSED	Keine Verbindung aktiv oder anstehend
LISTEN	Der Server wartet auf eine ankommende Verbindung
SYN_RCVD	Ankunft einer Verbindungsanfrage und Warten auf Bestätigung
SYN_SENT	Die Anwendung hat begonnen, eine Verbindung zu öffnen
ESTABLISHED	Zustand der normalen Datenübertragung
FIN_WAIT_1	Die Anwendung möchte die Übertragung beenden, Close-Aufruf wurde bereits abgesetzt
FIN_WAIT_2	Die andere Seite ist einverstanden, die Verbindung abzubauen, Bestätigung (ACK) gesendet
TIME_WAIT	Warten, bis keine Segmente mehr kommen
CLOSING	Beide Seiten haben versucht, gleichzeitig zu beenden
CLOSE_WAIT	Die Gegenseite hat den Abbau eingeleitet, warten auf Close-Aufruf der lokalen Anwendung
LAST_ACK	Warten, bis letzte Bestätigung (ACK) für Verbindungsabbau angekommen ist

# TCP als FSM

---

- Ein TCP-Zustandsautomat für das TCP-Verbindungsmanagement lässt sich als 6-Tupel  $\langle S, I, O, T, s_o, F \rangle$  beschreiben:
  - $S = \{\text{CLOSED}, \text{LISTEN}, \text{SYN\_RCVD}, \dots\}$
  - $I = \{\text{connect}, \text{send}, \text{close}, \text{SYN}, \text{ACK}, \text{FIN}, \dots\}$
  - $O = \{\text{SYN}, \text{ACK}, \text{FIN}, \dots\}$
  - $F = \{\text{CLOSED}\}$
  - $s_o = \text{CLOSED} \in S$
- Hinweis: Wir beschreiben im Weiteren nur die Transitionen des Verbindungsaufbaus und des Verbindungsabbaus
- Beispiel einer Transition:



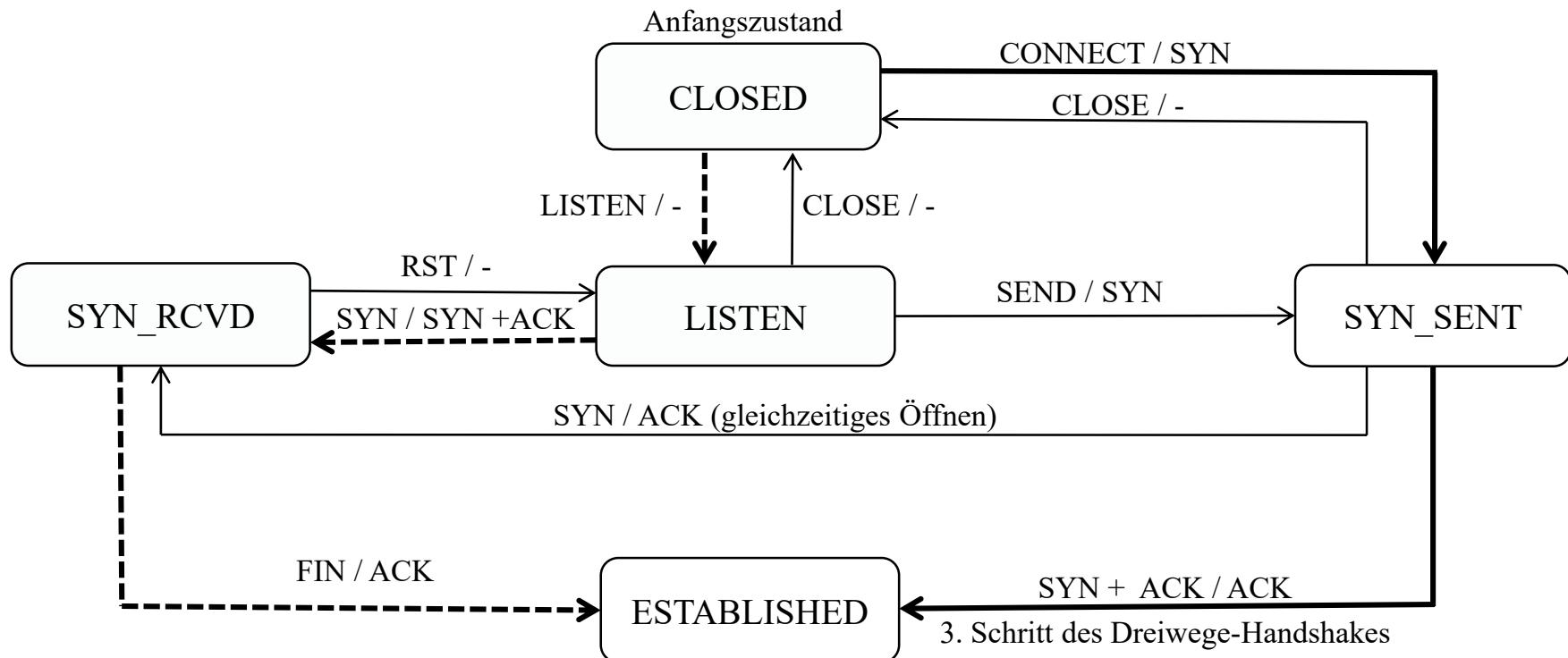
# Finite-State-Machine-Modell

## TCP-Verbindungsauftbau

**fette Linie:** normaler Pfad des Clients

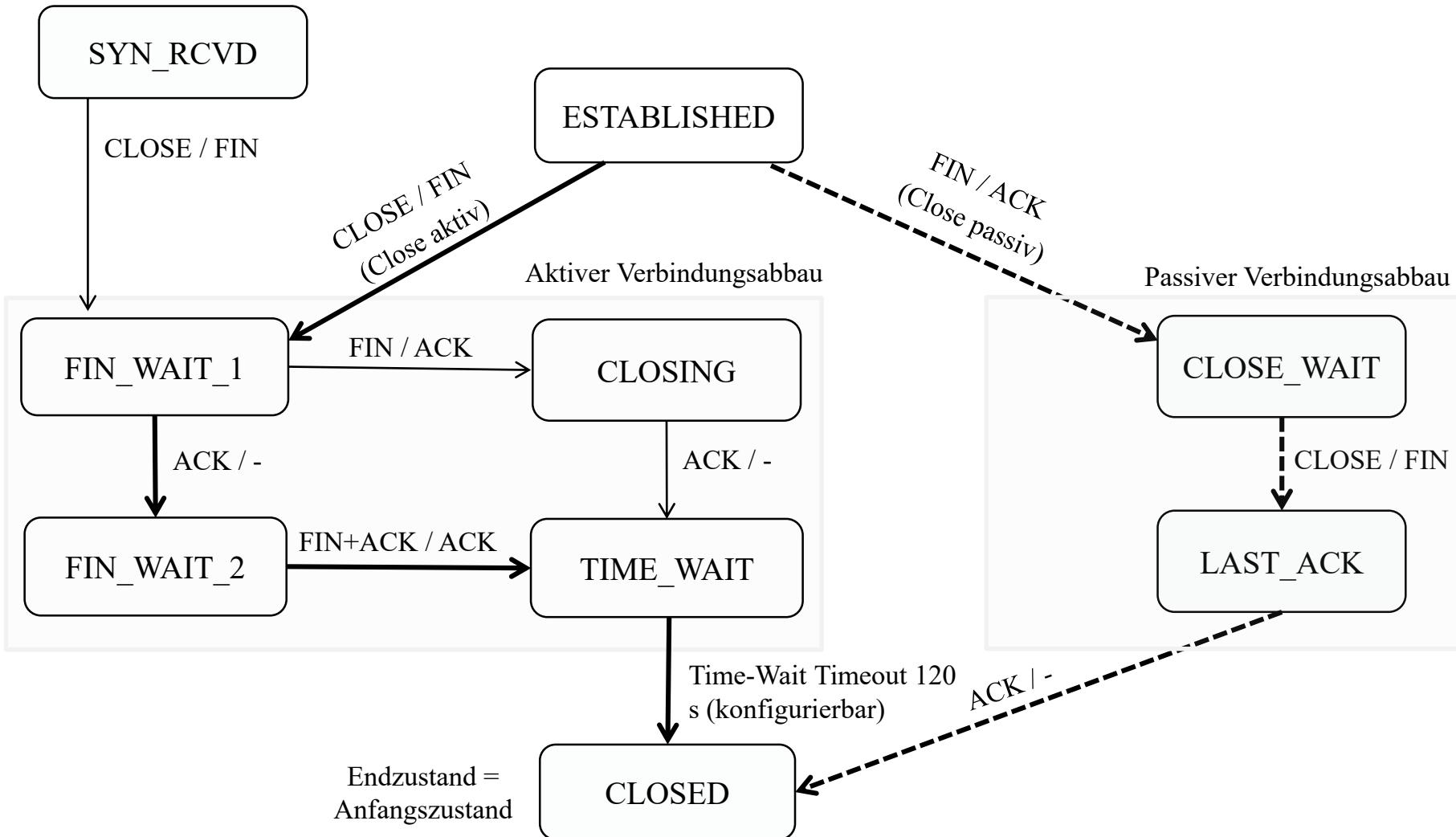
**fette gestrichelte Linie:** normaler Pfad des Servers

**feine Linie:** ungewöhnliche Ereignisse



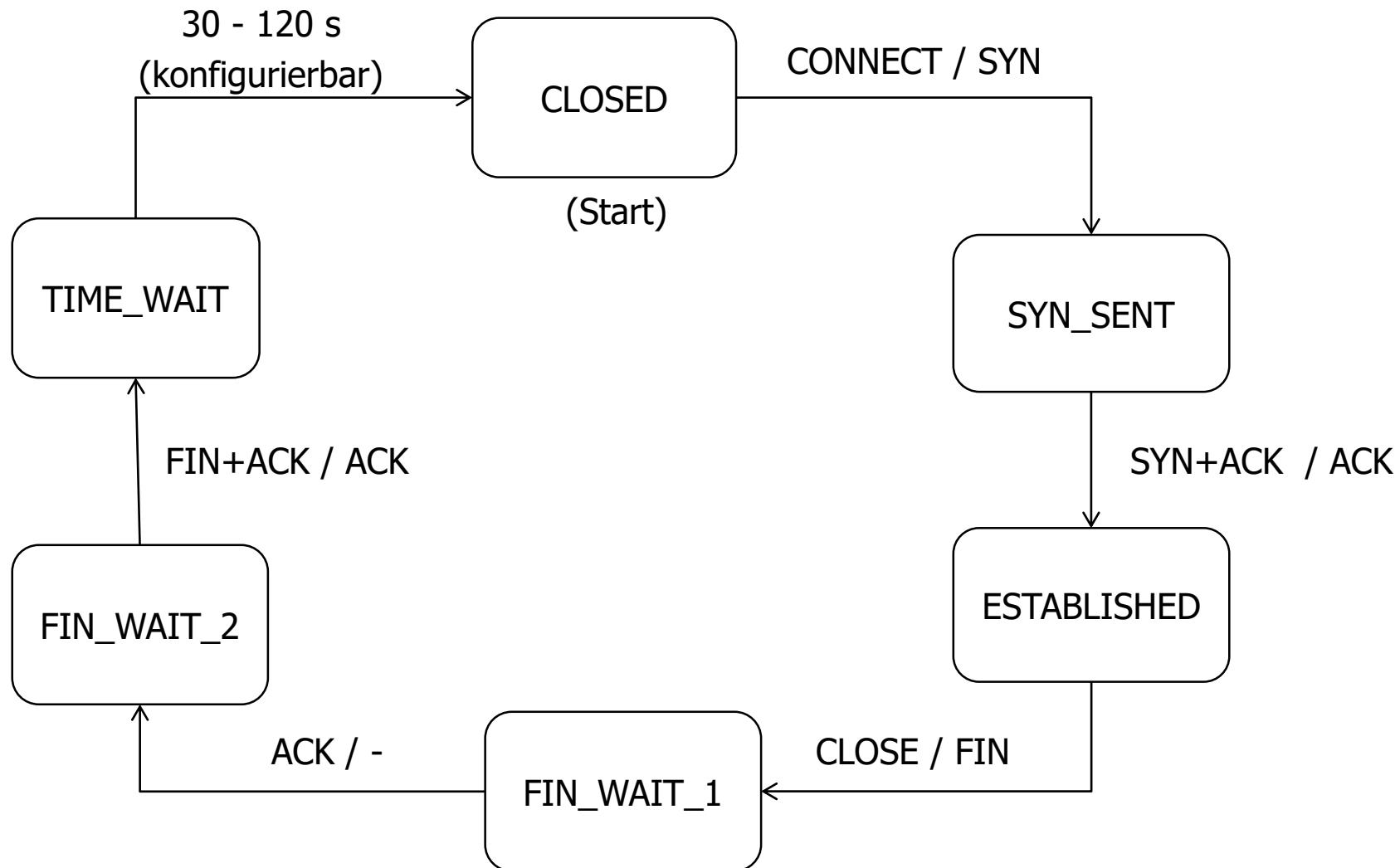
# Finite-State-Machine-Modell

## TCP-Verbindungsabbau

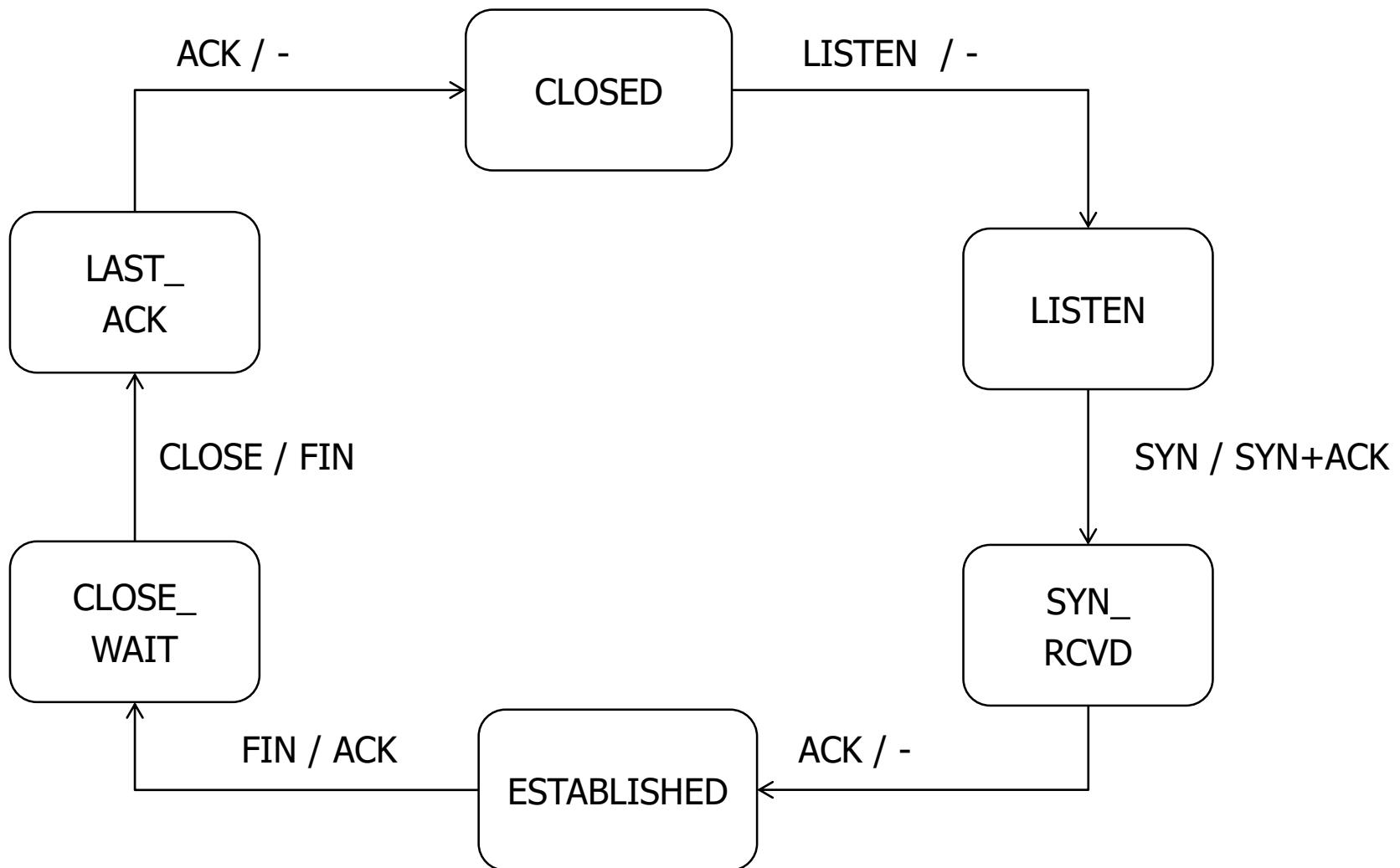


# Finite-State-Machine-Modell

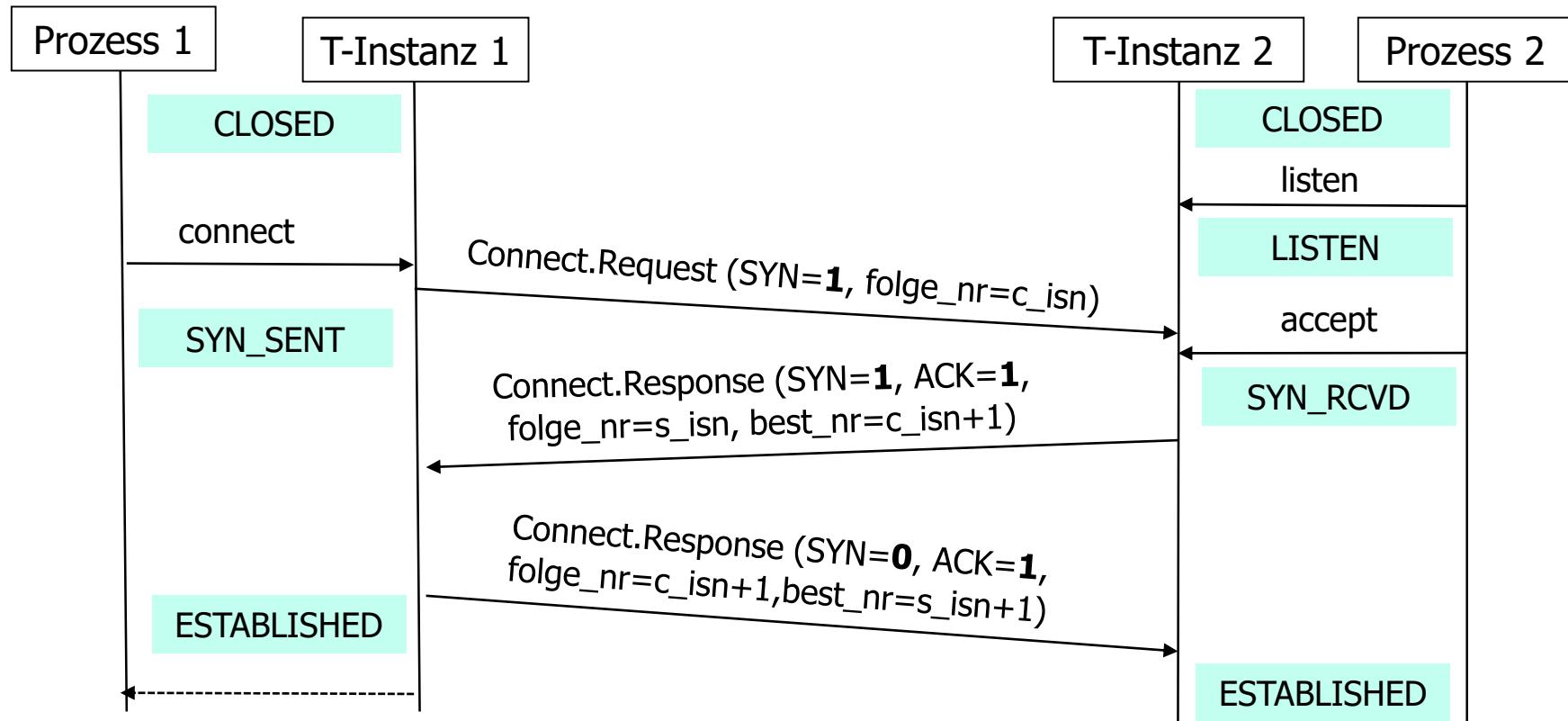
## Aktiver Partner (vereinfacht)



# Finite-State-Machine-Modell Passiver Partner (vereinfacht)



# Verbindungsauftbau mit Zustandsveränderungen

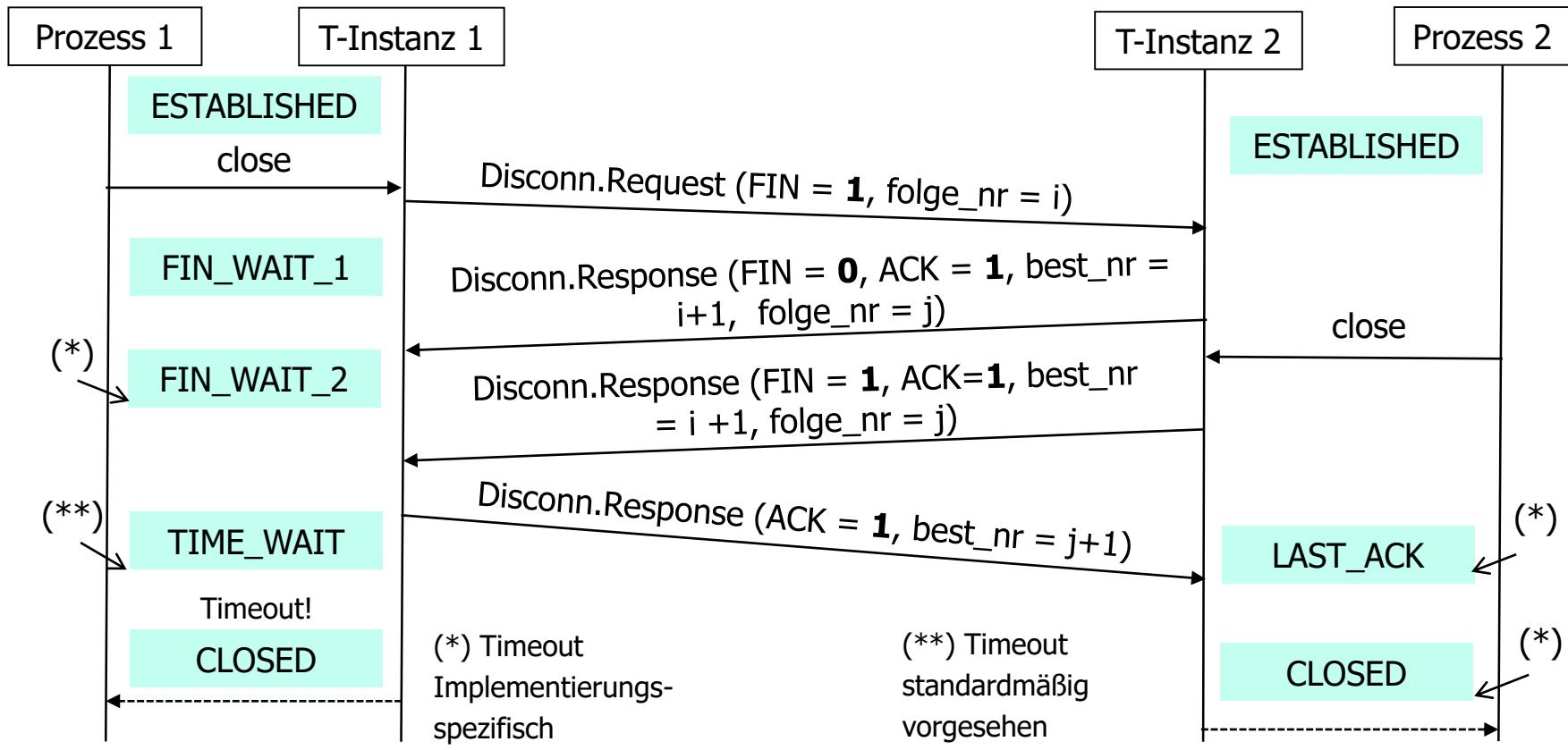


c\_isn = Initial Sequence Number des Clients (aktiver Partner, Instanz 1)

s\_isn = Initial Sequence Number des Servers (passiver Partner, Instanz 2)

# Verbindungsabbau mit Zustandsveränderungen

- Client baut die Verbindung ab (auch Server kann es)
- Alle Segmente mit Folgenummer < i bzw. j sind noch zu verarbeiten



## Für Interessierte: TIME\_WAIT und CLOSE\_WAIT

---

- Die Wartezeit, bis ein TCP-Port nach einem Verbindungsabbau wieder verwendet werden darf, wird per Systemkonfiguration (z.B. im Windows-Registry oder in der Linux-Kernelkonfiguration) festgelegt
- Vor dem Binden einer Adresse kann die Socket-Option **SO\_REUSEADDR** genutzt werden, um die Wartezeit zu unterbinden. Ein noch belegter Port kann damit gleich wieder genutzt werden
- **Java API:** In der Klasse *Socket* kann die Option SO\_REUSEADDR gesetzt werden
  - `public void setReuseAddress(boolean on) throws SocketException;`
- **Linux:** Es gibt einen CLOSE\_WAIT-Timeout und auf aktiver Seite einen Timeout für den FIN\_WAIT\_2-Zustand, siehe Kommando `sysctl` oder `/etc/sysctl.conf`

# Übung

---

- Wozu dienen folgende TCP-Zustände:
  - SYN\_RECV
  - SYN\_SENT
  - TIME\_WAIT
  - CLOSE\_WAIT
- Schneiden Sie mit dem Tool Wireshark einen TCP-Verbindungsaufbau mit
- Betrachten Sie mit dem Kommando **netstat** oder mit dem Tool **TcpView** die Zustände diverser TCP-Verbindungen, die auf Ihrem Rechner laufen
  - Web-Browser
  - Chat-Anwendung ...

# Einschub für Interessierte: Zustandsautomaten implementieren

---

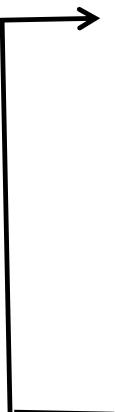
- Wie implementiert man einen Zustandsautomaten?
  - Welches Prozess-/Threadmodell verwendet man?
    - Ein bekanntes Modell: Server-Modell
  - Auftretende Ereignisse (Nachrichten, Timeouts, Dienstaufrufe) müssen verarbeitet werden und führen zu Zustandsänderungen und Aktionen
- Mehrere Varianten
  - Bei jedem Ereignis über Switch-Case-Konstruktionen prüfen, welche Aktion ausgeführt werden soll und ob das Ereignis zulässig ist
  - Nutzung des State Patterns zur Vermeidung von ewigen Switch-Case-Konstruktionen
- State Pattern (Objektorientierter Ansatz)
  - Jeder Zustand wird als eigene Objektklasse implementiert, die von einer Basisobjektklasse erbt (im Weiteren nicht betrachtet)

# Einschub für Interessierte: Zustandsautomaten implementieren - Server-Modell (1)

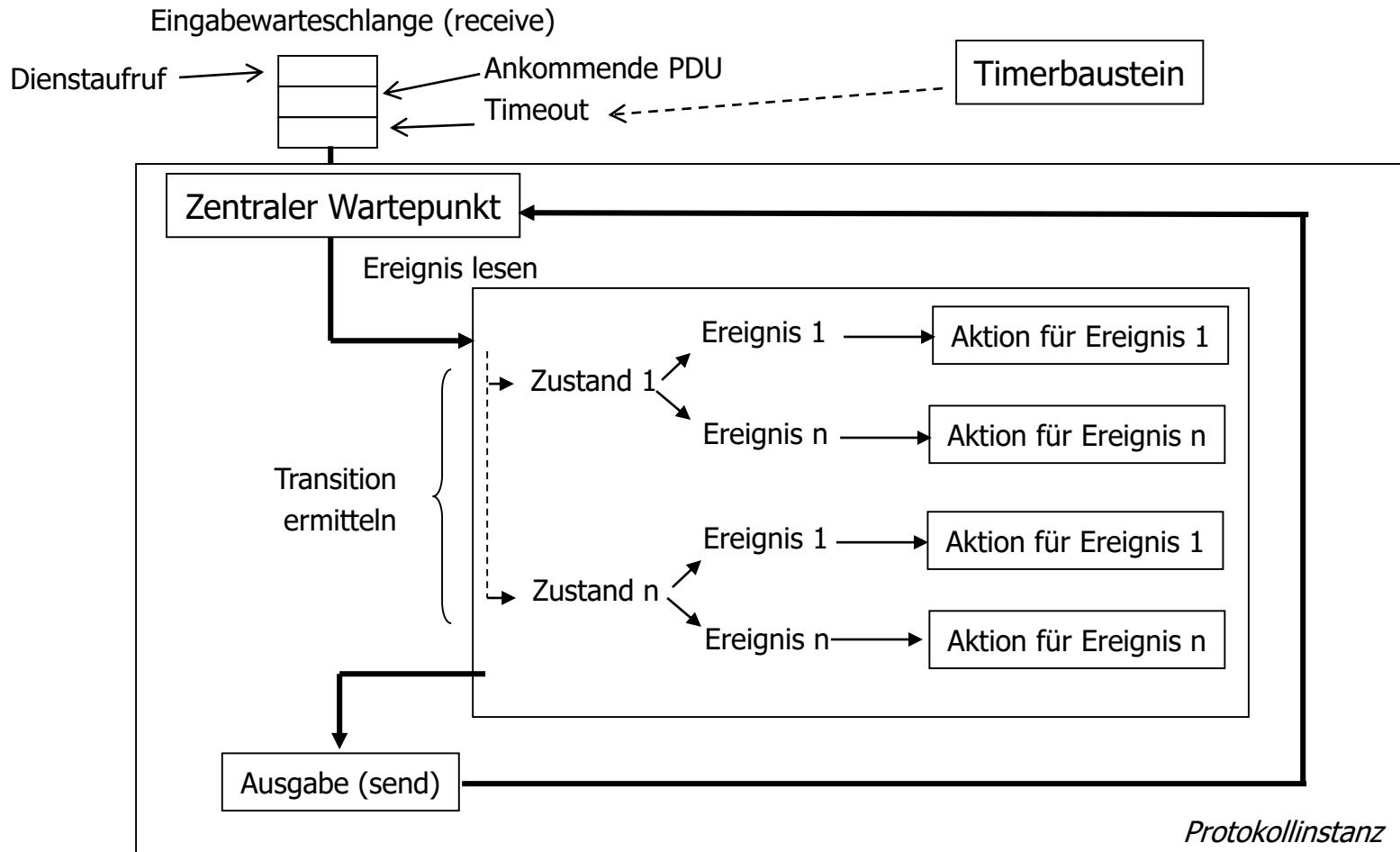
---

## ■ Das Server-Modell

- Protokollinstanz wird als zyklisch arbeitender sequentieller Prozess/Thread umgesetzt
- Eingabewarteschlange für auftretende Ereignisse wird an einem Ereigniswartepunkt abgearbeitet
- Ereignisse werden zyklisch in einer „Endlosschleife“ abgearbeitet:
- Grober Algorithmus:

- 
- Ereignis aus der Warteschlange lesen
  - Analyse des eingehenden Ereignisses (Nachricht)
  - Aktion auf Basis des aktuellen Zustandes auswählen
  - Ggf. Fehleranalyse und Fehlerreaktion
  - Aktion ausführen, Transition durchführen (in Zustand gehen) und Ausgabe erzeugen
  - Und dann wieder von vorne: Nächstes Ereignis aus der Warteschlange lesen

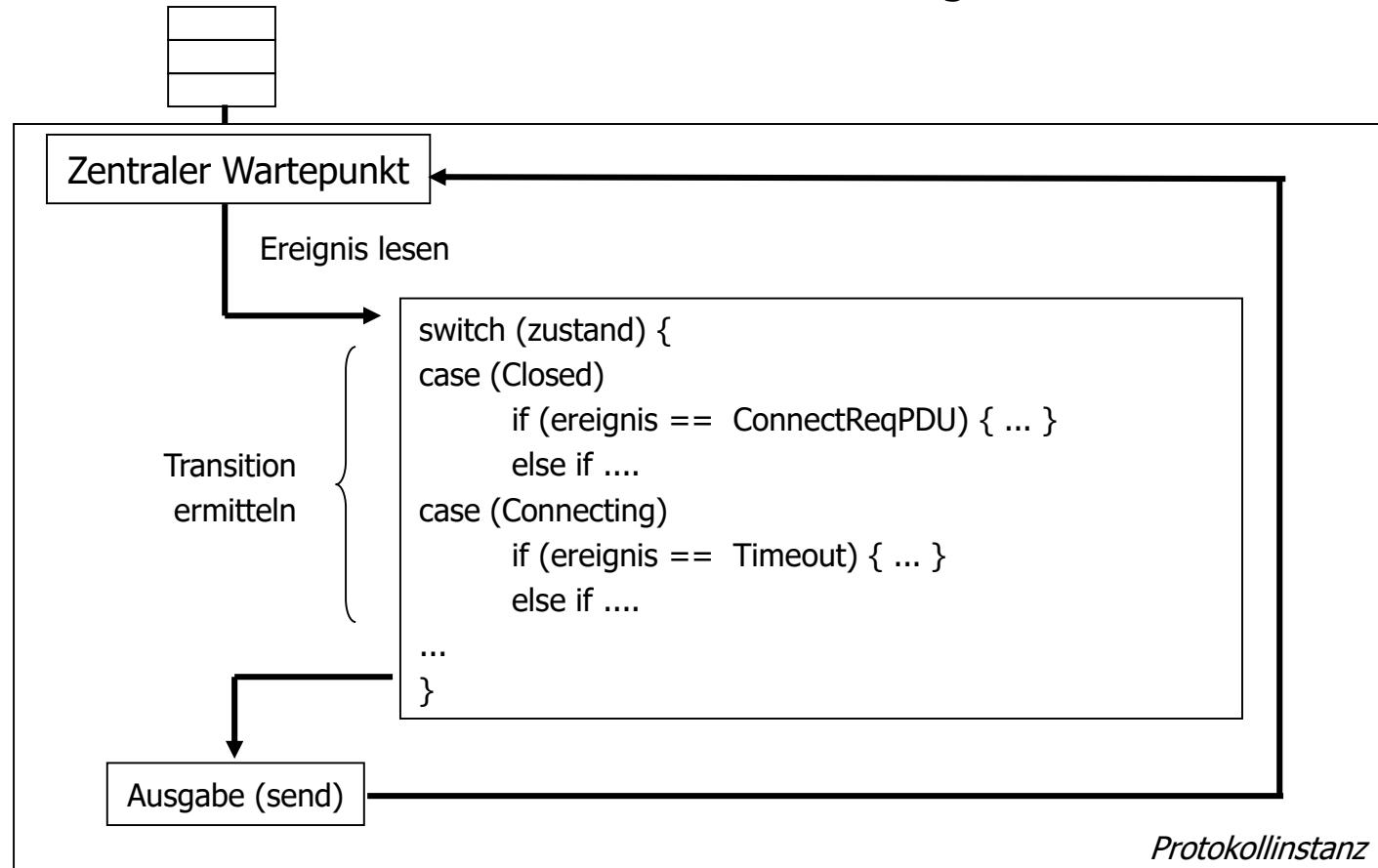
# Einschub für Interessierte: Zustandsautomaten implementieren - Server-Modell (2)



# Einschub für Interessierte: Zustandsautomaten implementieren - Server-Modell (3)

Eingabewarteschlange (receive)

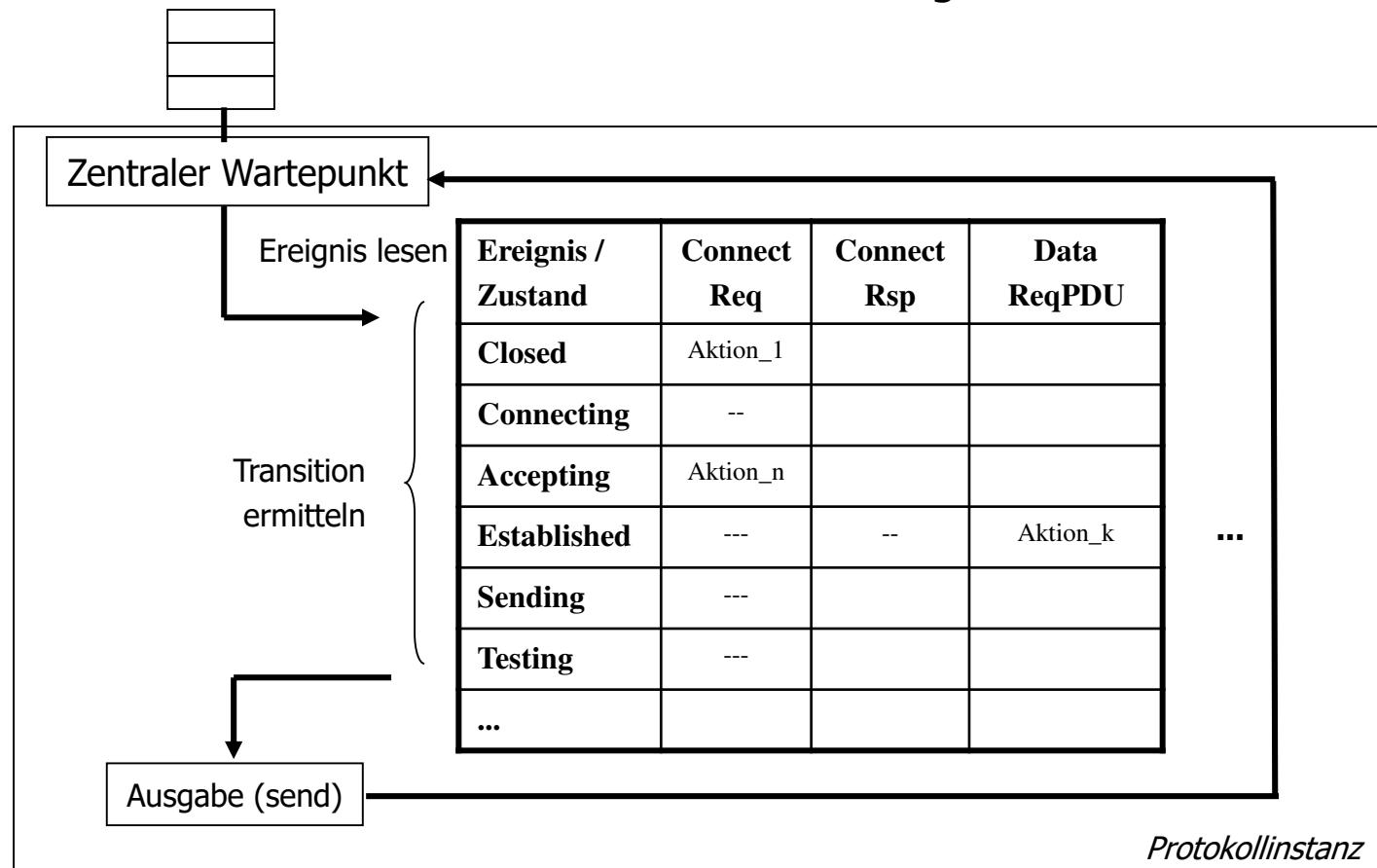
- Programmierte Auswahl



# Einschub für Interessierte: Zustandsautomaten implementieren - Server-Modell (4)

Eingabewarteschlange (receive)

- Tabellengesteuerte Auswahl



# Einschub für Interessierte: Zustandsautomaten implementieren - Server-Modell (5)

---

## ■ Kritik am Server-Modell

- Positiv
  - Einfacher Entwurf, da Zustandsautomat direkt abgebildet wird
  - Einfach zu implementieren
  - Grundgerüst leicht aus Modellierungssprache zu generieren
- Negativ
  - Bei vielen Zuständen unübersichtlich, da viele Switch-Case und If-Konstruktionen zur Ermittlung der nächsten Transition notwendig sind

# Überblick über Transportprotokolle

---

1. Einordnung und Aufgaben von TCP
2. Der TCP-Header
3. Verbindungsauf- und –abbau, Datenübertragungsphase
4. Sliding-Window-Mechanismus
5. Optimierungen: Algorithmen von Nagle und Clark, Silly Window Syndrom
6. Staukontrolle
7. TCP-Timer
8. TCP-Zustandsautomat
- 9. UDP (User Data Protocol)**

# Einordnung und Aufgaben

---

- **Unzuverlässiges**, verbindungsloses Transportprotokoll
  - **Keine Empfangsbestätigung** für Pakete
  - UDP-Nachrichten **können** ohne Kontrolle **verloren gehen**
  - Eingehende Pakete werden **nicht in einer Reihenfolge sortiert**
  - Maßnahmen zur Erhöhung der Zuverlässigkeit müssen im Anwendungsprotokoll ergriffen werden, z.B.
    - ACK und Warten mit Timeout
    - Wiederholtes Senden bei fehlendem ACK
- **Vorteile** von UDP gegenüber TCP
  - Bessere Leistung möglich, aber nur, wenn TCP nicht nachgebaut werden muss
  - Multicast- und Broadcast wird unterstützt

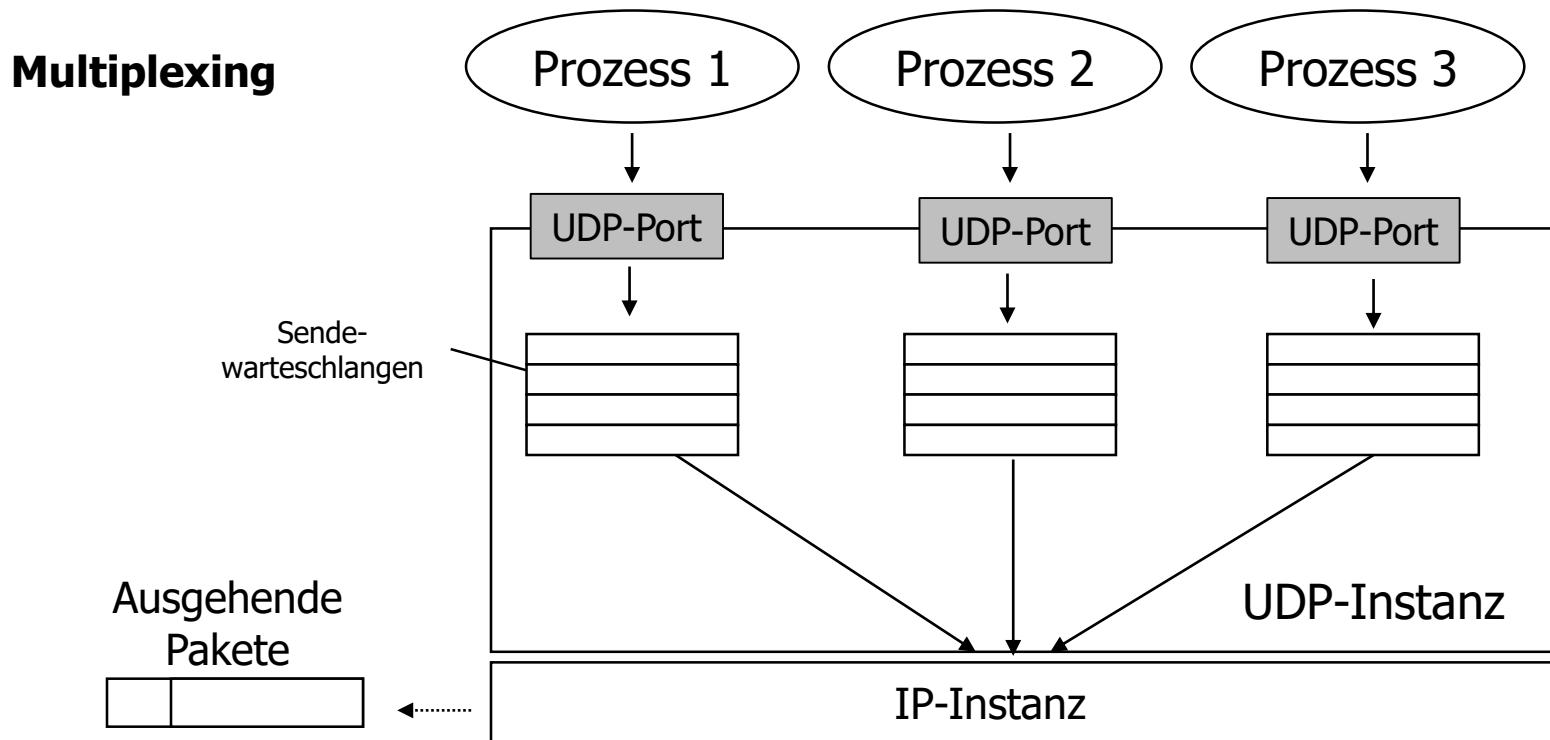
## Einordnung und Aufgaben

---

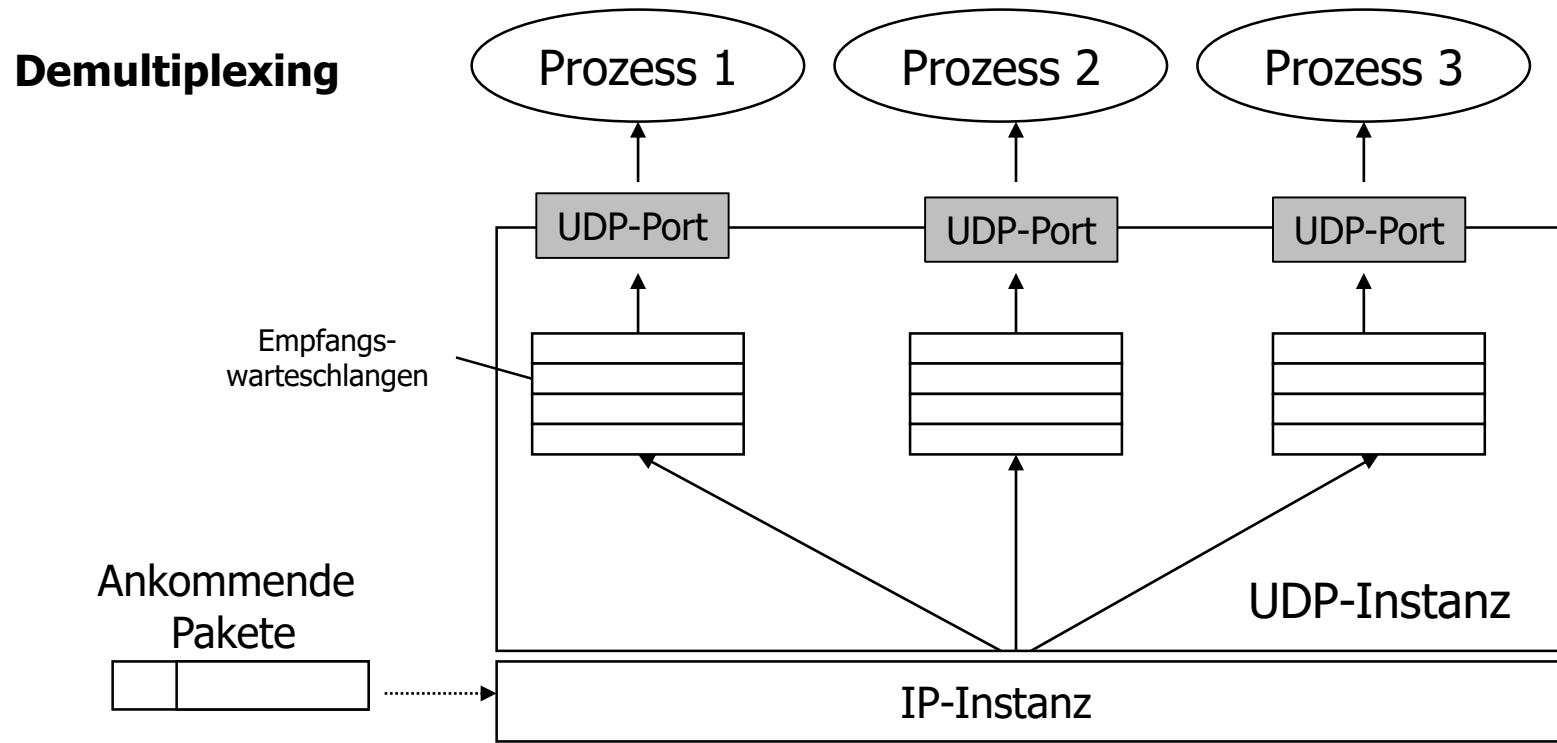
- Bei UDP ist **keine explizite Verbindungsauftaubungsphase** erforderlich und entsprechend auch **kein Verbindungsabbau**
- Userprozess erzeugt ein UDP-Socket und kann Nachrichten senden und empfangen
- Nachrichten werden bei UDP als **Datagramme** bezeichnet

# UDP als einfacher Multiplexer/Demultiplexer (1)

- UDP erweitert IP nur um eine **Multiplexing/Demultiplexing**-Funktionalität

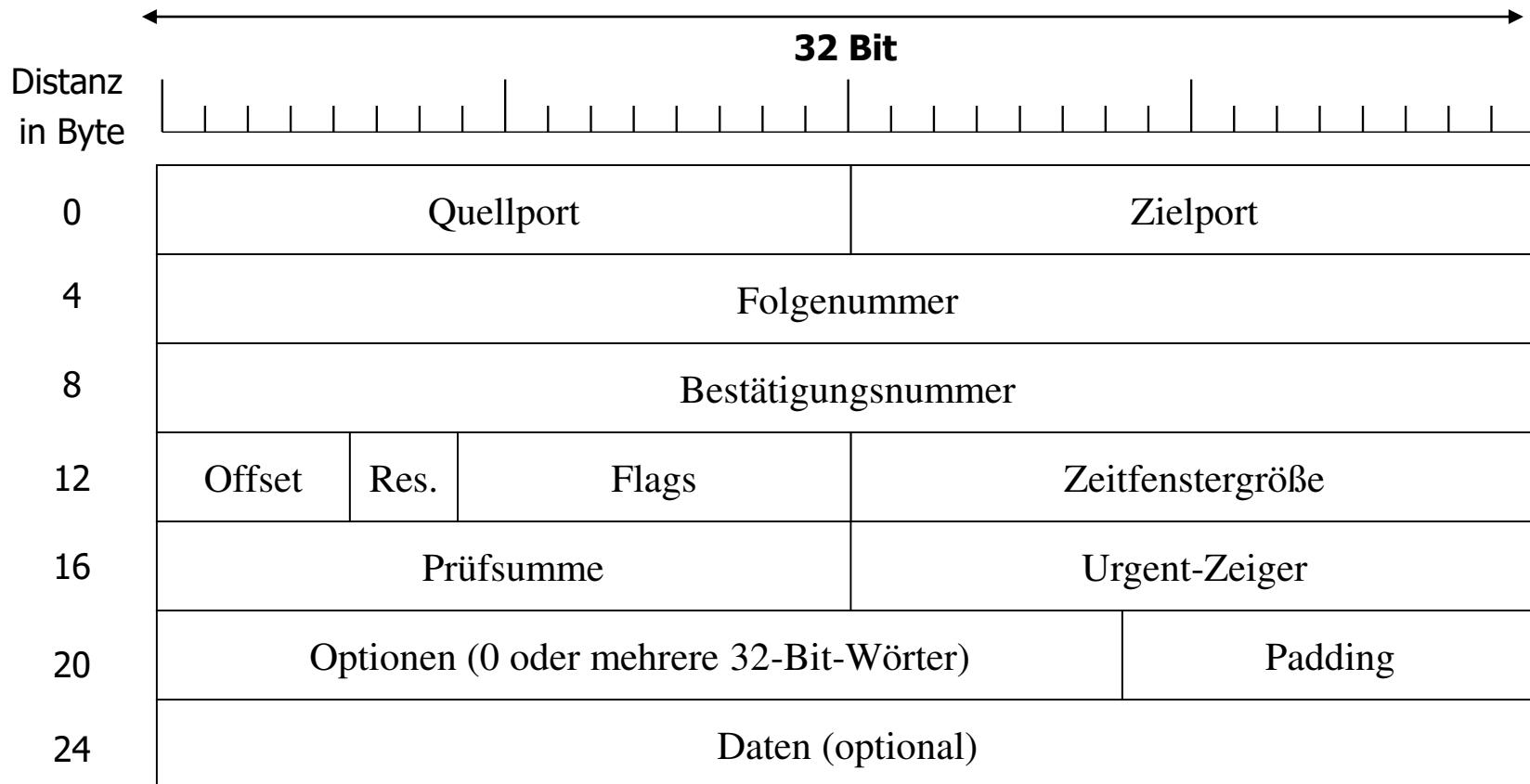


# UDP als einfacher Multiplexer/Demultiplexer (2)



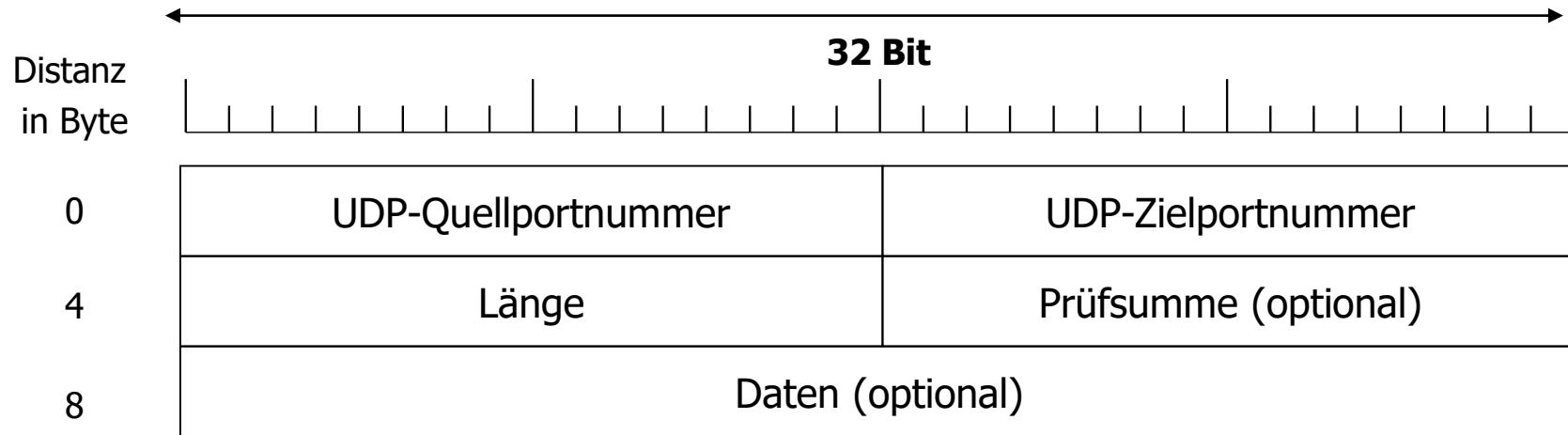
# TCP-Header zum Vergleich

---



# UDP-Header

---



- **UDP-Ziel-Portnummer**
  - Nummer des empfangenden Ports
- **UDP-Quell-Portnummer**
  - Nummer des sendenden Ports
- **Länge**
  - Größe des UDP-Segments inkl. Header in Byte
- **Prüfsumme (optional)**
  - Prüft das Gesamtsegment (Daten + Header) einschließlich eines Pseudoheaders
- **Daten**
  - Nettodaten der Nachricht

## Einige well-known UDP-Portnummern

---

<b>UDP- Portnummer</b>	<b>Protokoll, Service</b>
53	DNS – Domain Name Service
520	RIP – Routing Information Protocol
161	SNMP – Simple Network Management Protocol
69	TFTP – Trivial File Transfer Protocol

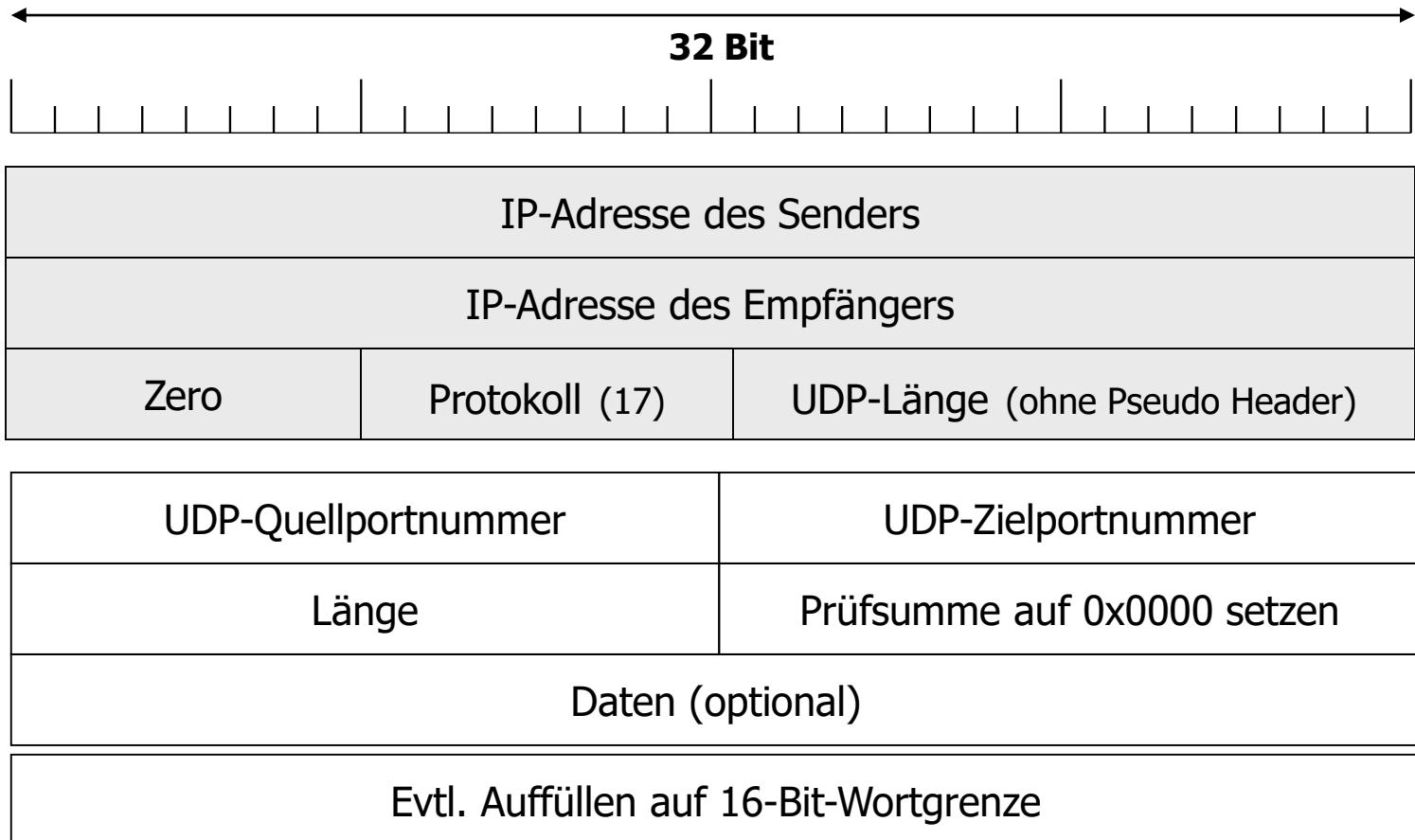
## Begrenzte Nachrichtenlänge bei UDP

---

- Die Länge eines UDP-Segments ist minimal 8 Bytes und maximal  $2^{16} - 1 = 65.535$  Bytes
- Nettodatenlänge =  $2^{16} - 1 - 8 = 65.527$  Bytes
- Darüber hinaus ist es sinnvoll, die UDP-Segmente nicht länger als in der möglichen IP-Paketlänge zu versenden, da sonst fragmentiert werden muss
  - Besser: MTU berücksichtigen

# UDP-Pseudoheader

---



Pseudoheader ist grau unterlegt

---

# Pseudoheader und Prüfsummenberechnung (1)

---

- Der Sender muss vor dem Senden einer UDP-PDU folgendes unternehmen:
  - Vor dem Berechnen der Prüfsumme wird der Pseudoheader ermitteln und für die Prüfsummenberechnung vor die UDP-PDU gestellt
  - Die ganze PDU inkl. Pseudoheader wird auf eine durch 16 Bit teilbare Größe aufgefüllt (gerade Anzahl an Bytes)
  - Die Prüfsumme im UDP-Header auf 0x0000 setzen
  - Berechnung der Prüfsumme wie bei TCP über Addition der 16-Bit-Wörter und der Bildung des Einerkomplements der Summe (RFC 1071, ...)
  - Falls Prüfsumme 0x0000, dann invertieren zu 0xFFFF
  - Ermittelte Prüfsumme in UDP-Header eintragen

## Pseudoheader und Prüfsummenberechnung (2)

---

- Der Empfänger muss nach dem Empfang einer UDP-PDU folgendes unternehmen:
  - die IP-Adressen aus dem ankommenden IP-Paket lesen
  - Der Pseudoheader muss zusammengebaut werden
  - Die Prüfsumme muss ebenfalls berechnet werden (vorher Prüfsumme **nicht** auf 0x0000 setzen)
  - Ergibt die Prüfsummenberechnung 0xFFFF so passt die Übertragung
- Wenn die beiden Prüfsummen identisch sind, dann muss das Datagramm seinen Zielrechner erreicht haben
  - Ziel des Pseudoheaders ist es somit, beim Empfänger herauszufinden, ob das Paket den richtigen Empfänger gefunden hat

# Beispiel zur Prüfsummenberechnung

---

- Für folgende 16-Bit-Wörter soll eine UDP-Prüfsumme berechnet werden:

1000 0110 0101 1110 (1)

1010 1100 0110 0000 (2)

0111 0001 0010 1010 (3)

1000 0001 1011 0101 (4)

- Berechnung:

1000 0110 0101 1110 (1)

1010 1100 0110 0000 (2) +

0011 0010 1011 1110 → 1 (Übertrag)

                  1

0011 0010 1011 1111

0111 0001 0010 1010 (3) +

1010 0011 1110 1001 (Kein Übertrag)

1000 0001 1011 0101 (4) +

0010 0101 1001 1110 → 1 (Übertrag)

                  1

0010 0101 1001 1111 = Summe (0x259F)

1101 1010 0110 0000 (Einerkomplement der Summe) = 0xCA60

**0b1101100101100000** oder 0xCA60 ist die Prüfsumme, die in das Prüfsummenfeld des UDP-Headers eingetragen wird

---

# Prüfsummenberechnung und Fehlererkennung

---

- Einbitfehler werden erkannt
- Zweibitfehler werden nicht erkannt:

<b>Richtig</b>	<b>Vertauscht</b>
0010	00 <b>0</b> (1 Bit vertauscht)
<u>0001</u> +	<u>00<b>1</b></u> (1 Bit vertauscht) +
0011	0011

- Ebenso werden dreifache Bitfehler nicht erkannt:

<b>Richtig</b>	<b>Vertauscht</b>
0011	00 <b>0</b> 1 (1 Bit vertauscht)
0010	001 <b>1</b> (1 Bit vertauscht)
<u>0000</u> +	<u>000<b>1</b></u> (1 Bit vertauscht) +
0101	0101

- Das Verfahren ist sehr schwach
- Es sollte eigentlich nach der Prüfphase durch ein CRC-Verfahren ersetzt werden → das wurde aber nie realisiert!

# Überblick über Transportprotokolle

---

1. Einordnung und Aufgaben von TCP
2. Der TCP-Header
3. Verbindungsauf- und –abbau, Datenübertragungsphase
4. Sliding-Window-Mechanismus
5. Optimierungen: Algorithmen von Nagle und Clark, Silly Window Syndrom
6. Staukontrolle
7. TCP-Timer
8. TCP-Zustandsautomat
9. UDP (User Data Protocol)

# Rückblick

---

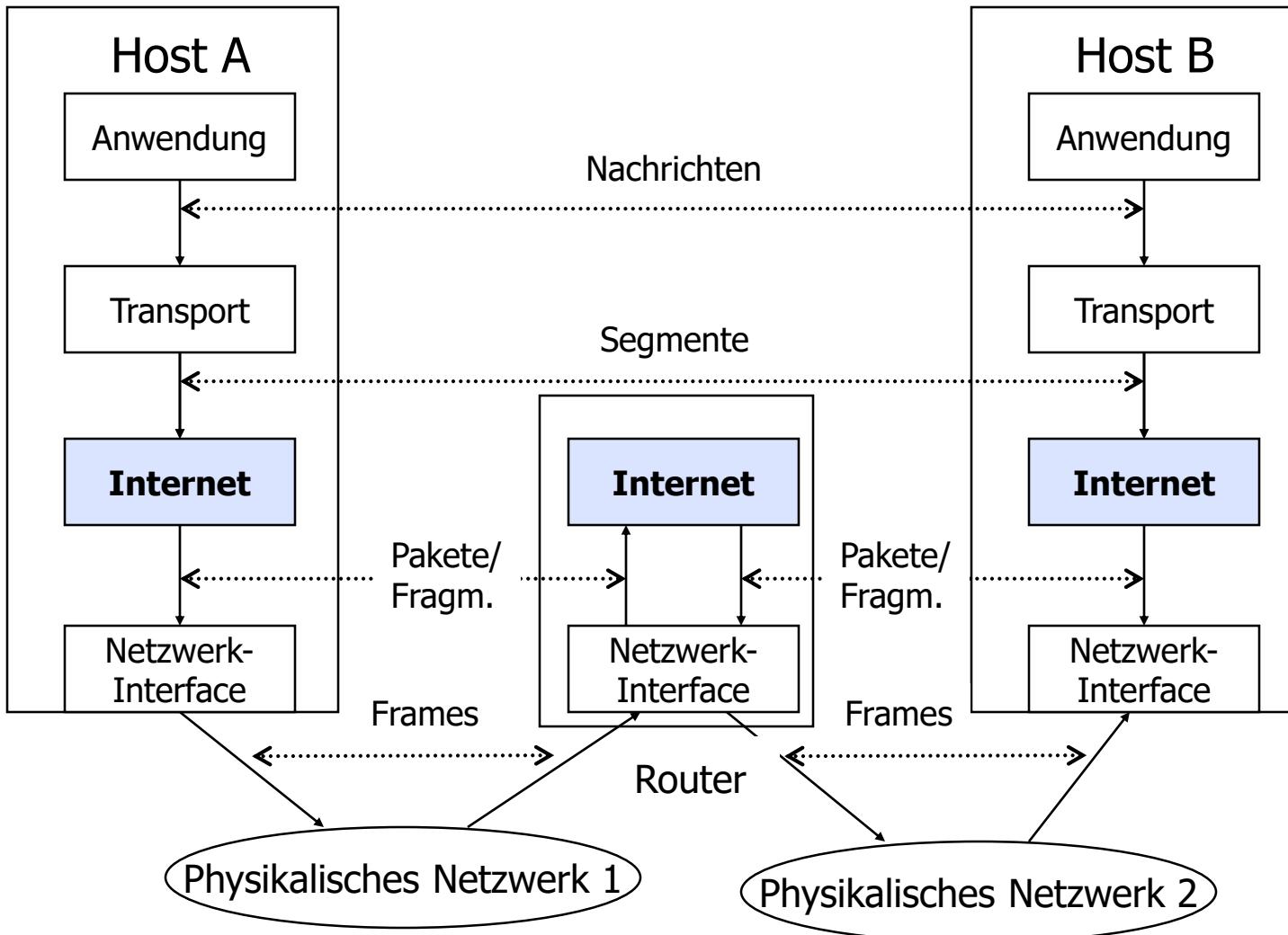
- ✓ Einordnung und Aufgaben von TCP
- ✓ Der TCP-Header
- ✓ Verbindungsauflauf- und –abbau, Datenübertragungsphase
- ✓ Sliding-Window-Mechanismus
- ✓ Optimierungen: Algorithmen von Nagle und Clark, Silly Window Syndrom
- ✓ Staukontrolle
- ✓ TCP-Timer
- ✓ TCP-Zustandsautomat
- ✓ UDP (User Data Protocol)

- 1 Kommunikationssysteme und verteilte Anwendungen
- 2 Grundlagen der Transportschicht
- 3 Transportprotokolle TCP und UDP
- 4 **Grundlagen der Vermittlungsschicht**
- 5 Internet und Internet Protocol (IP)
- 6 Routing-Verfahren und -Protokolle
- 7 Internet-Steuerprotokolle und DNS
- 8 Internet Protocol Version 6 (IPv6)
- 9 Netzwerkschnittstelle

## **1. Aufgaben und Dienste**

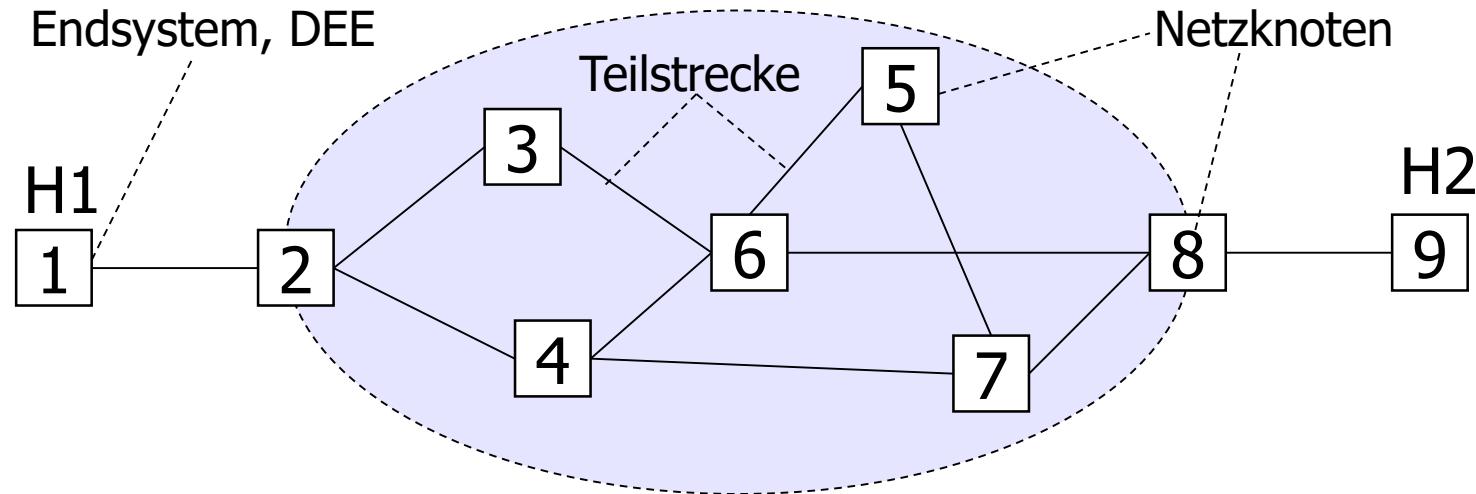
2. Vermittlungsverfahren
3. Wegewahl, Routing

# TCP-Referenzmodell



# Aufbau eines Vermittlungsnetzes

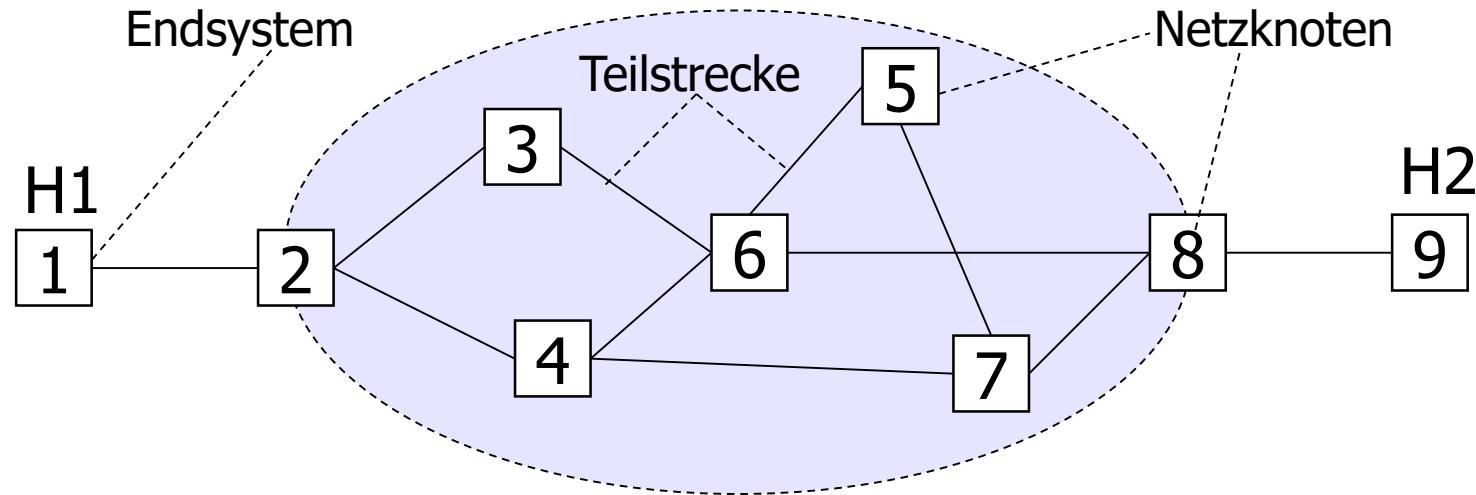
---



- **Netzknoten** sind über **Teilstrecken** miteinander verbunden
- **Endsysteme** (H1, H2) sind mit Netzknoten verbunden
- Netzknoten verwaltet zwei oder mehr Verbindungen
- Endsystem hat meist nur eine Verbindung zu einem Netzknoten

# Aufgaben (1)

---



- Nachricht soll von H1 (Host) zu H2 übertragen werden
- Voraussetzung ist eine eindeutige **Adressierung**
- Aufgabenstellung ist vergleichbar mit der Zustellung einer Postkarte

## Aufgaben (2)

---

- Die Endsysteme kommunizieren über einen oder mehrere **Netzknotenrechner** (kurz: Netzknoten, Knoten, Router)
- Die Übertragungswege werden von Knoten zu Knoten bereitgestellt (**Teilstrecken**)
- Die **Fehlersicherung** findet auf den Teilstrecken (Schicht zwei) statt
- Die Schnittstelle zum nächsten Netzknotenrechner aus Sicht eines privaten Netzes ist meist die **Netzbetreiberschnittstelle**

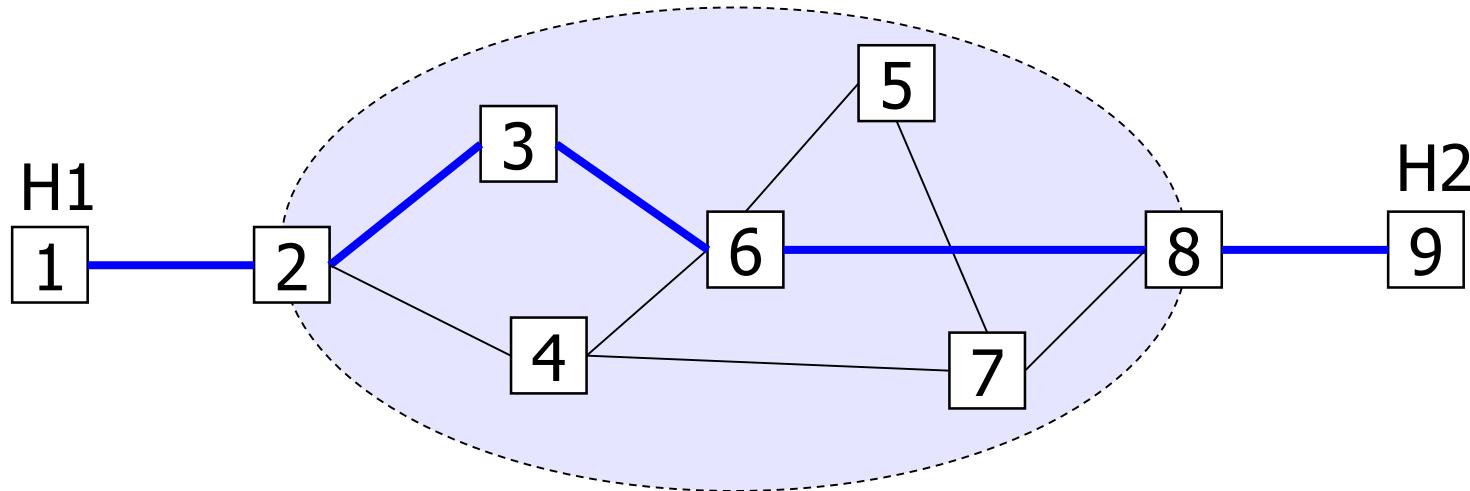
## Aufgaben (3)

---

- Zu den Aufgaben gehören:
  - Wegewahl (auch Routing genannt)
  - Multiplexen und Demultiplexen
  - Staukontrolle (Congestion Control)
  - Fragmentierung/Defragmentierung
- Oft diskutiert:
  - Ist ein verbindungsloser oder verbindungsorientierter Dienst an der Schnittstelle zur Transportschicht besser? (vgl. Tanenbaum)
  - Frage: Ist das Internet, verbindungslos oder verbindungsorientiert?

## Aufgaben (4)

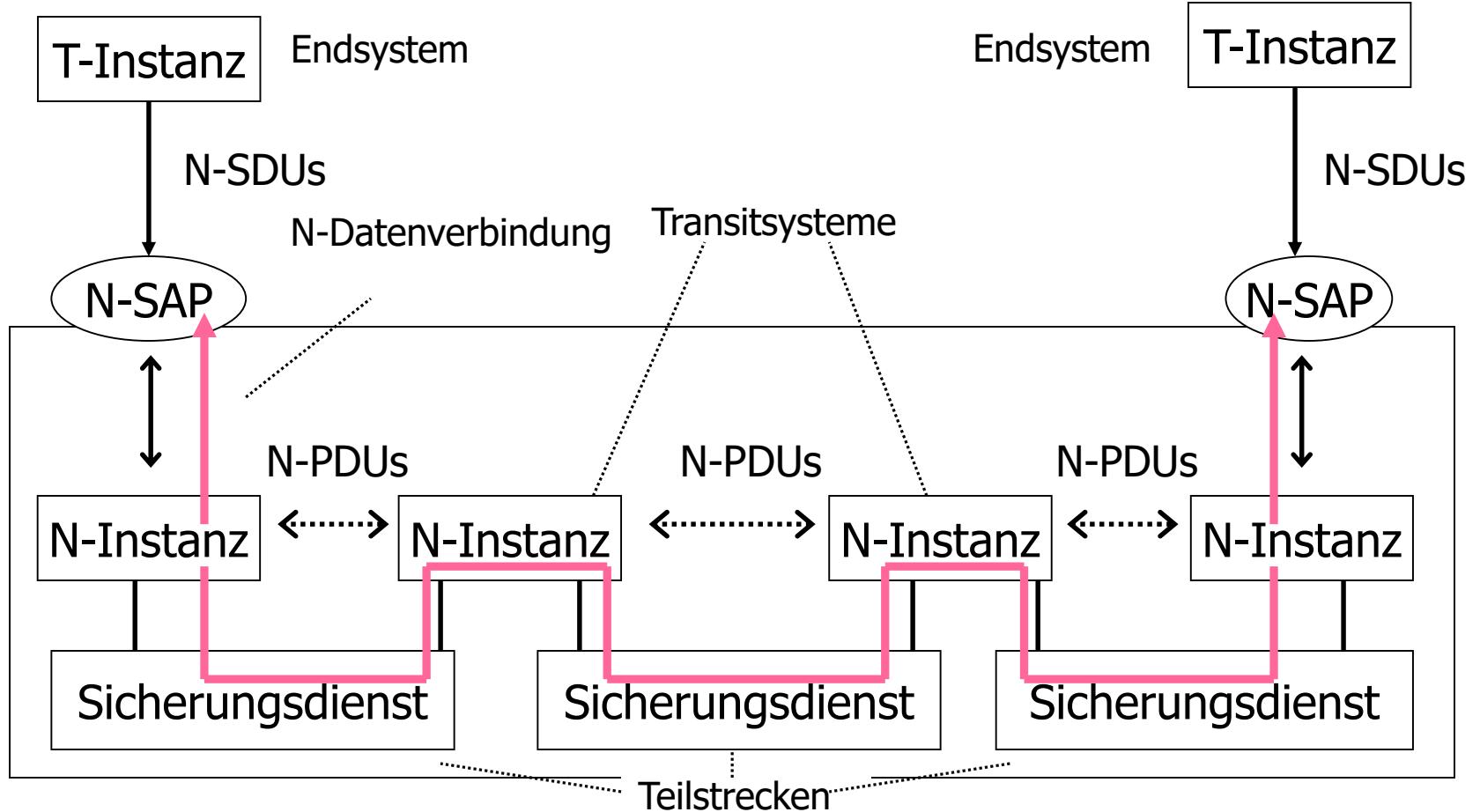
---



- Den **Gesamtvorgang** der **Verbindungsherstellung**, des **Haltens** und des **Abbauens** einer „Verbindung“ bezeichnet man als **Vermittlung** (engl. Switching)
- Beispiel: Verbindung von H1 (1) zu H2 (9) über die Netzknoten 2, 3, 6, 8

# Dienste der Vermittlungsschicht

Vgl.: Gerdzen, P., Kommunikationssysteme 1

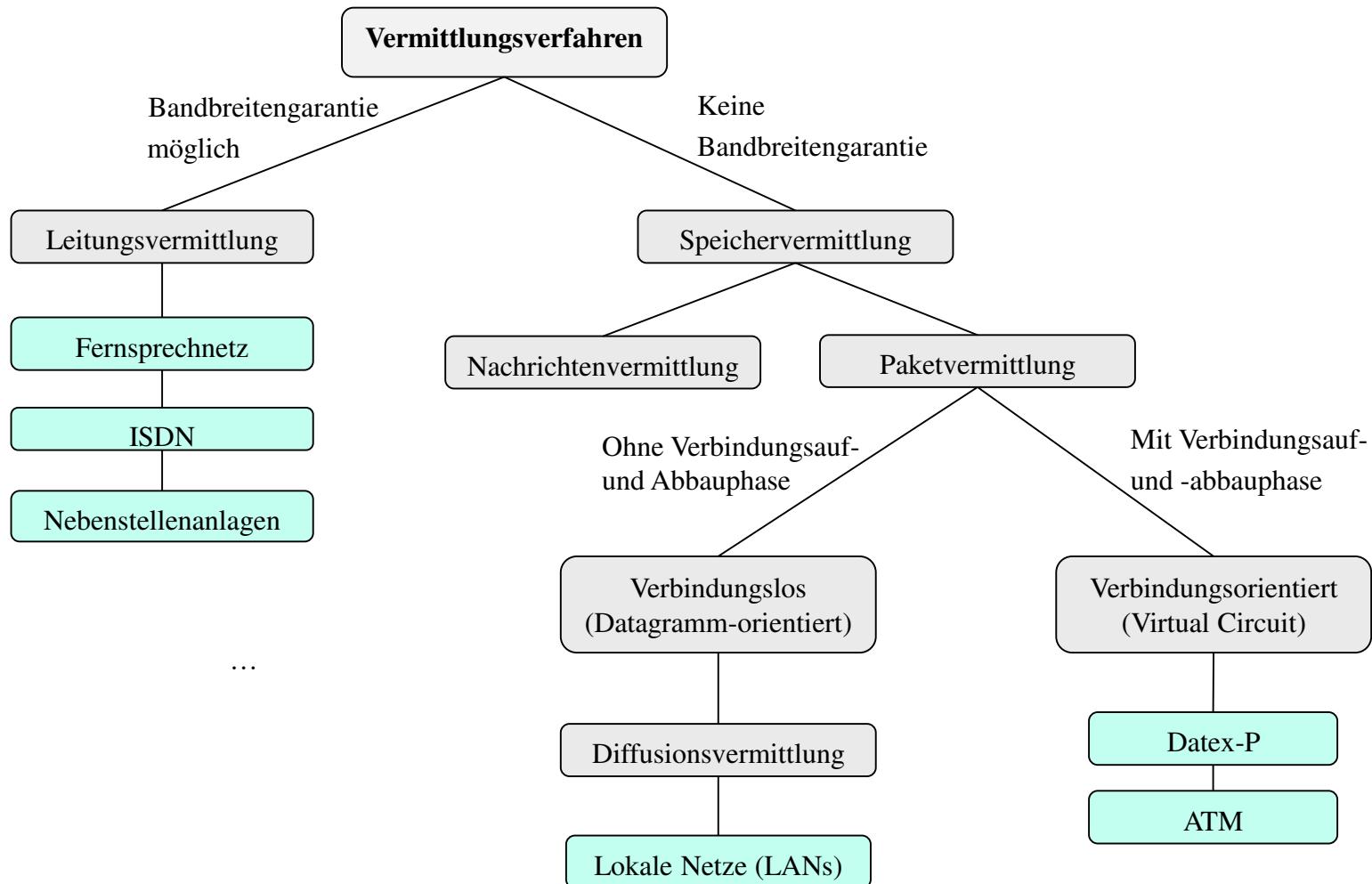


# Überblick

---

1. Aufgaben und Dienste
- 2. Vermittlungsverfahren**
3. Wegewahl, Routing

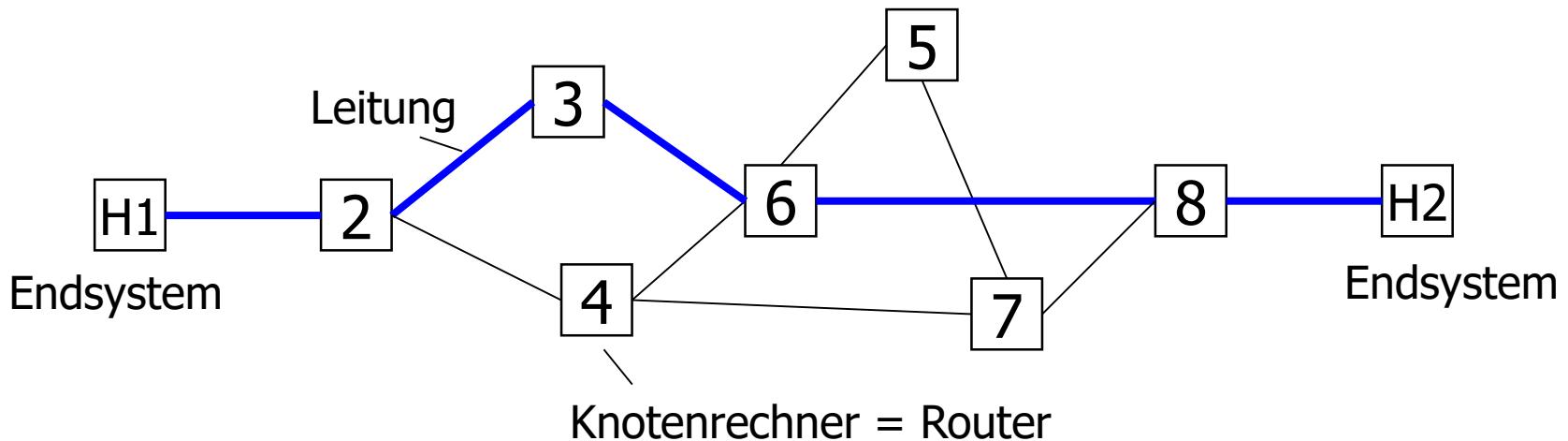
# Vermittlungsverfahren



# Leitungsvermittlung (1)

## ■ Merkmale:

- Über die gesamte Strecke wird **ein physikalischer Verbindungs weg** durch das Netzwerk geschaltet
- Klassisches Switching-Verfahren, auch **circuit switching** oder **Durchschaltevermittlung** genannt



## Leitungsvermittlung (2)

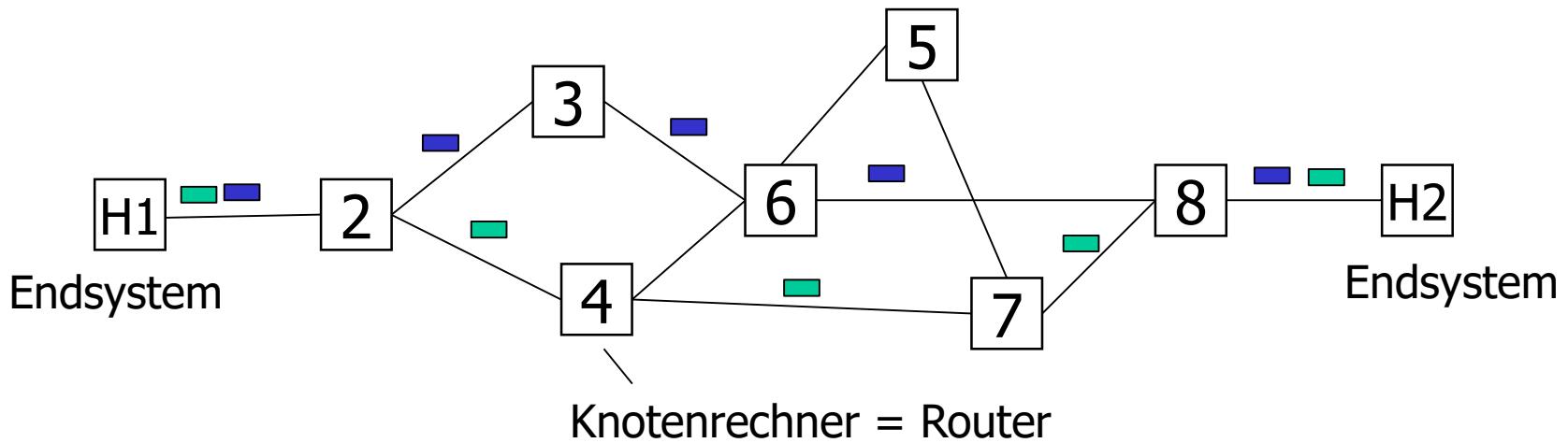
---

- **Bewertung:**
  - Es wird eine **feste Bandbreite garantiert** und zwar unabhängig von dem, was tatsächlich übertragen wird
  - Es kann sein, dass Bandbreite unnötig reserviert wird
  - Blockierungen (Ablehnung eines Verbindungswunsches), wenn kein Verbindungs weg mehr frei ist
  
- **Beispielnetze:**
  - Analoges Fernsprechnetz
  - Digitales ISDN

# Paketvermittlung (1)

## ■ Merkmale:

- Komplette Nachricht mit Zieladresse wird ins Netz gesendet
- Netz überträgt die Nachricht evtl. über mehrere Knoten mit Zwischenspeicherung
- Nachricht wird ggf. in einzelne Pakete (N-PDUs) zerlegt und versendet



## Paketvermittlung (2)

---

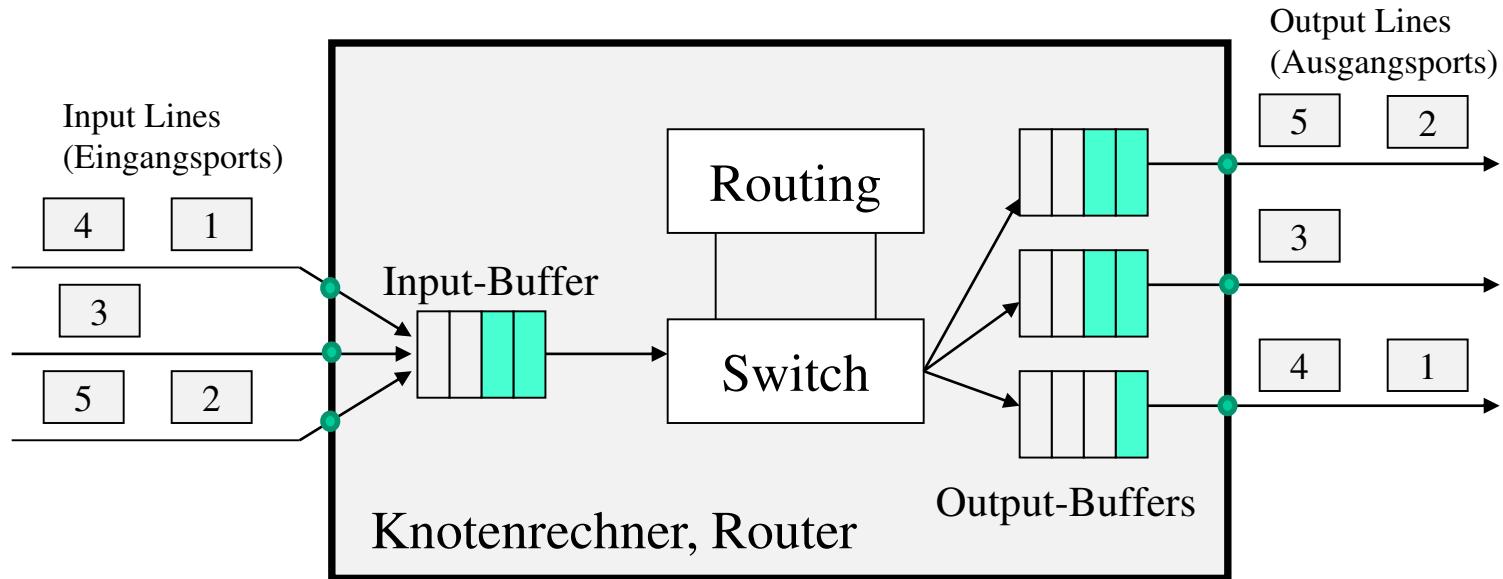
- **Bewertung:**
  - Paketvermittlung ist für die Datenübertragung effizienter
  - Aber keine garantierte Bandbreite, dafür aber auch keine Blockierungen
  
- **Beispielnetze:**
  - Internet
  - Breitband-ISDN auf Basis von ATM (sehr kurze Pakete, Zellen genannt)

# Paketvermittlung (3)

## Knotenrechner, Router

### ■ Prinzip der Paketvermittlung in einem Knotenrechner

Vgl.: Gerdzen, P., Kommunikationssysteme 1



## Paketvermittlung (4)

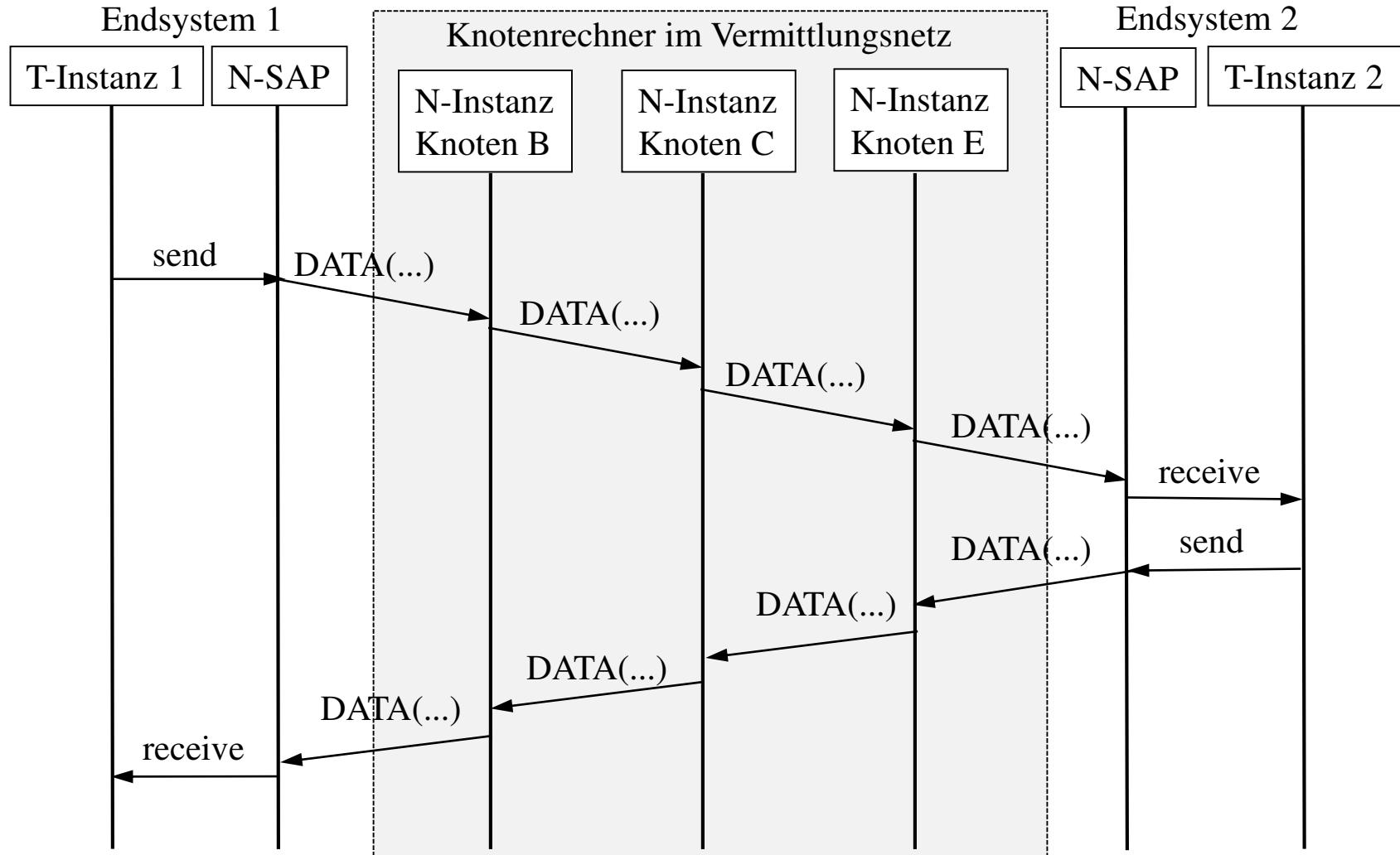
### Nutzung von Datagrammen

---

- Datagramme (N-PDUs) werden bei einer einfachen Paketvermittlung ohne vorhergehenden Verbindungsaufbau verwendet
- Jedes Datagramm enthält die Quell- und die Zieladresse
- Die Knoten **ermitteln** für jedes Datagramm einen **optimalen Weg**
- Wird auch als **verbindungslose** Vermittlung bezeichnet
- Nur ein einfacher data-Dienst zum Senden von Datagrammen erforderlich

# Paketvermittlung (5)

## Datagramm-Vermittlung – typischer Ablauf



## Paketvermittlung (6)

### Diffusionsvermittlung

---

- Einfache Form der Datagramm-Vermittlung ist die **Diffusionsvermittlung**, auch als Broadcastvermittlung bezeichnet
- Hier sendet jeder Knoten die empfangenen Pakete an alle Nachbarknoten weiter
  - mit Ausnahme des sendenden Knotens!
- Sinnvoll bei Netzen mit geringer Knotenzahl
- Klassische Vermittlungsform **in LANs**
  - Siehe z.B. Ethernet-LAN

## Nutzung von Virtual Circuits (1)

---

- Virtual Circuits werden auch „**scheinbare Verbindungen**“ genannt
- Reduzierung des aufwändigen Routens bei jedem Paket durch verbindungsorientiertes Verfahren
- Verbindung bleibt für die Dauer der Datenübertragung erhalten
- Kein physikalisches Durchschalten der Verbindung, sondern Nutzung von Verbindungsinformationen in den Knoten
  - VC-Ids in allen Knoten

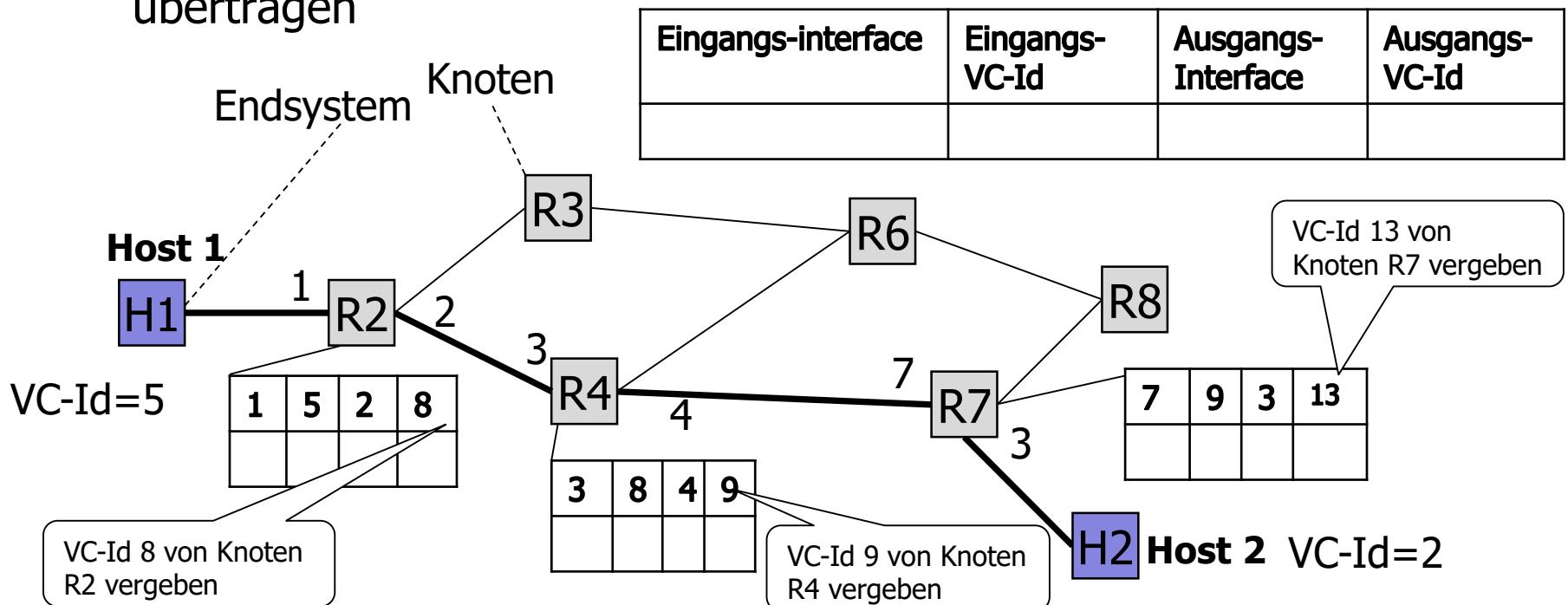
## Nutzung von Virtual Circuits (2)

---

- **Drei Phasen** der Kommunikation mit entsprechenden Diensten:
  - Verbindungsaufbau (connect-Dienst)
  - Datenübertragung (data-Dienst)
  - Verbindungsabbau (disconnect-Dienst)
- Die **Verbindung** zwischen zwei Endsystemen wird **schrittweise über Teilstrecken** aufgebaut
  - Knoten müssen in der Verbindungsaufbauphase Informationen über das Mapping von eingehenden Paketen zu Ausgangsteilstrecken speichern
  - Verbindungstabellen in den Knoten erforderlich
  - Virtuelle Verbindung ist eine „feste Route“ zwischen zwei Endsystemen

# Beispiel für den Aufbau einer virtuellen Verbindung

- VC-Ids werden zwischen den Knoten ausgetauscht, um die virtuelle Verbindung zu kennzeichnen
- In jedem Knoten wird eine Tabelle verwaltet
- VC-Ids werden in Nachrichten von Knoten zu Knoten mit übertragen



# Überblick

---

1. Aufgaben und Dienste
2. Vermittlungsverfahren
- 3. Wegewahl, Routing**

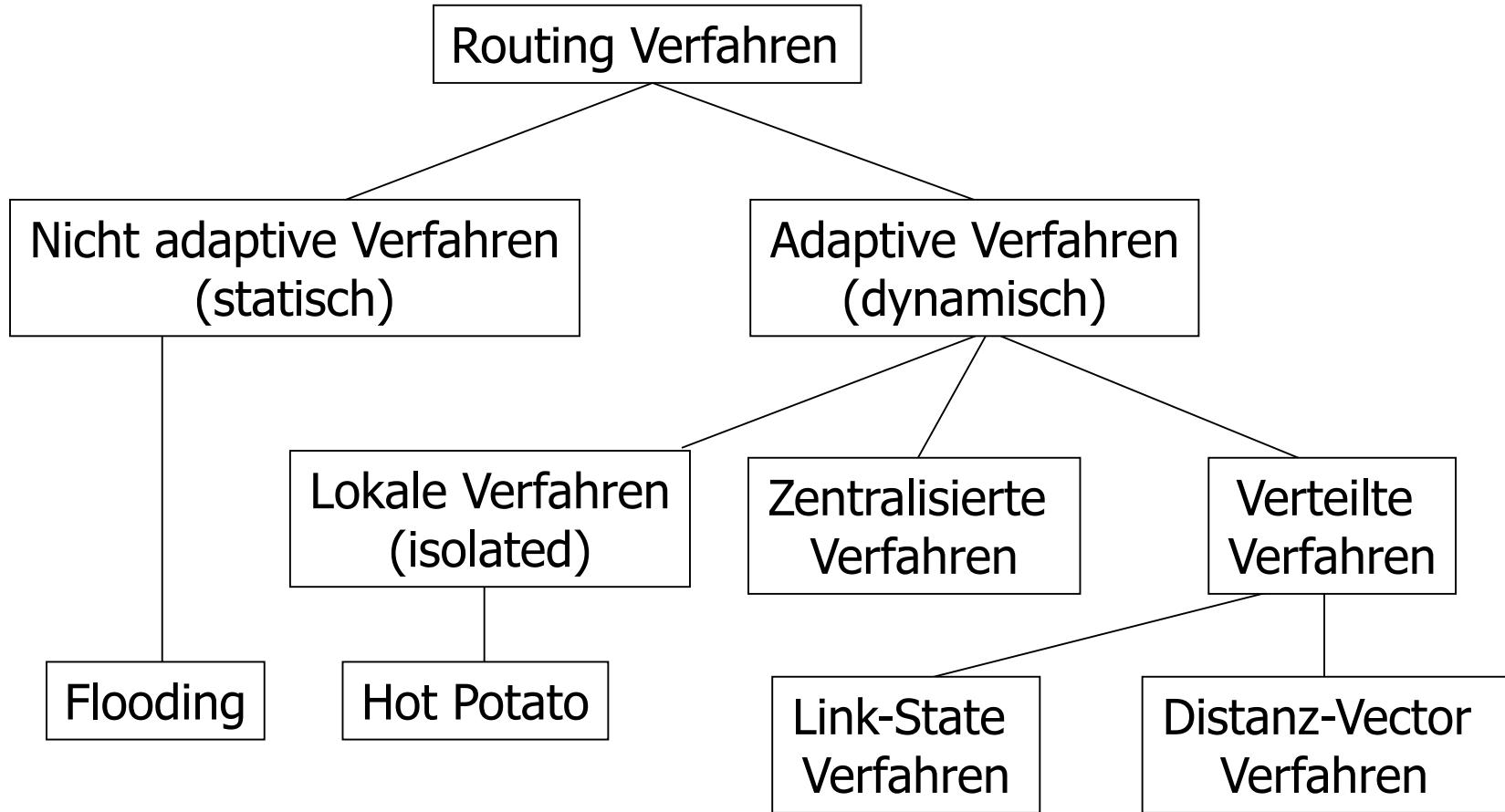
# Routing: Überblick

---

- Die Wegewahl (Verkehrslenkung, Routing) ist eine der wesentlichen Aufgaben der Schicht-3-Instanzen besonders für die paketorientierte Vermittlung
- Ziel ist es, **den „optimalen“ Weg** zwischen den Endsystemen zu wählen
- Notwendig bei alternativen Wegen zwischen den Endsystemen
- Verschiedene **Routing-Kriterien** und **-Algorithmen** sind möglich:
  - Suche der geringsten Entfernung
  - Möglichst geringe Anzahl von Hops (Anzahl der zu durchlaufenden Knoten)
  - Geringste Netzlast

# Routing – Klassifikation der Verfahren

---



# Routing - Verfahren

---

- **Statische** Algorithmen:
  - Statische Routing-Tabellen, die bei der Knotenkonfigurierung eingerichtet werden (vor Beginn des Betriebs)
- **Dynamische** (adaptive) Algorithmen:
  - Verkehrsmessungen
  - Routing-Tabellen werden dynamisch angepasst (Metriken)
  - Optimierungskriterien können sich dynamisch verändern und werden im Algorithmus berücksichtigt
  - Möglichkeiten:
    - **Isoliertes Routing**: Jeder Knoten trifft Entscheidungen alleine
    - **Zentrales** Routing über einen zentralen Knoten (Routing-Kontroll-Zentrum), Zentrale ermittelt und überträgt alle Routing-Tabellen
    - **Dezentrales** Routing mit Routing-Funktionalität in jedem Knoten

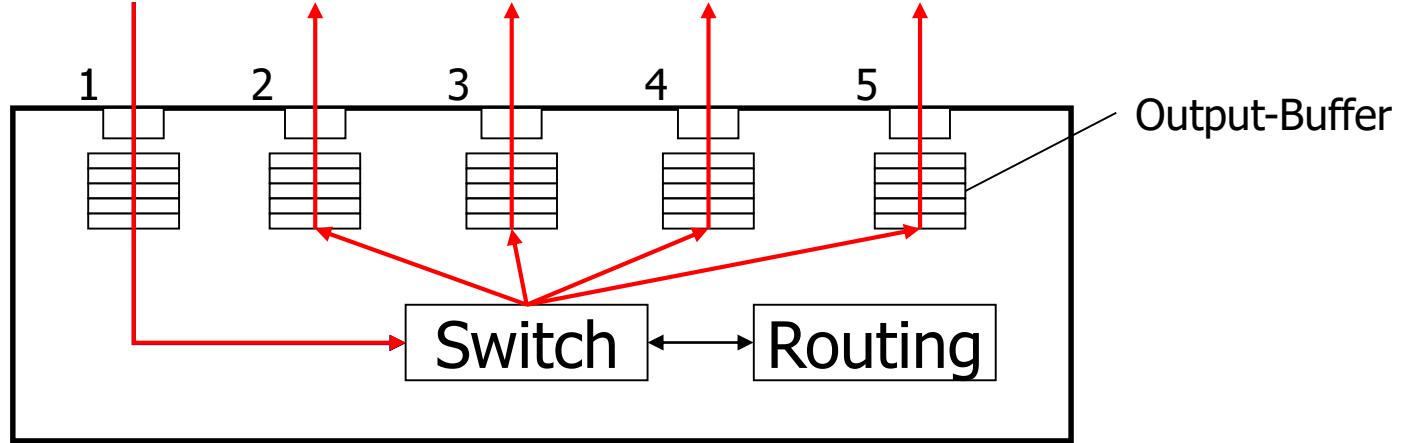
# Routing - Einige Algorithmen

---

- **Statische** Algorithmen:
  - Statische Vorgaben der Pfade durch „Shortest-Path-Routing“
  - Flooding
- **Dynamische** (adaptive) Algorithmen (heute üblich in modernen Netzen):
  - Distance-Vector-Routing
    - ursprünglicher Algorithmus im ARPANET
    - Protokollbeispiel: RIP
  - Link-State-Routing
    - Einführung Ende der 70er für größere Netze
    - Protokollbeispiel: OSPF

## Routing-Beispiel: Flooding (statisch)

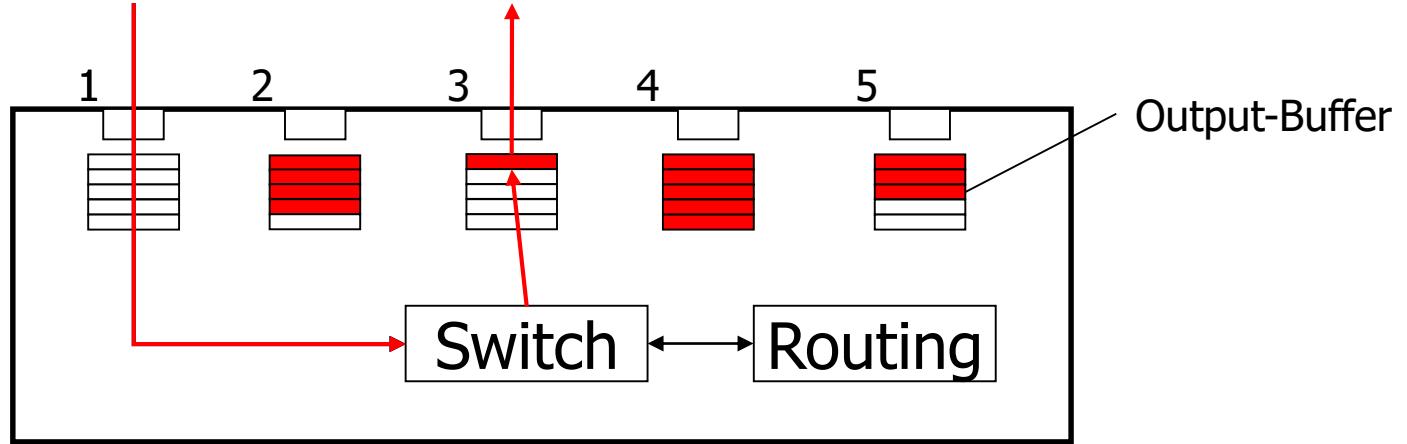
---



- Eingehende Pakete werden über alle Teilstrecken weiter versendet
- Pakete werden **nicht über die eingehende Leitung** und **nur einmal** weiter versendet
- Statischer und sehr **einfacher** Routing-Algorithmus
- Viele doppelte Pakete und somit **ineffizient**

## Routing-Beispiel: Hot Potato (dynamisch / lokal)

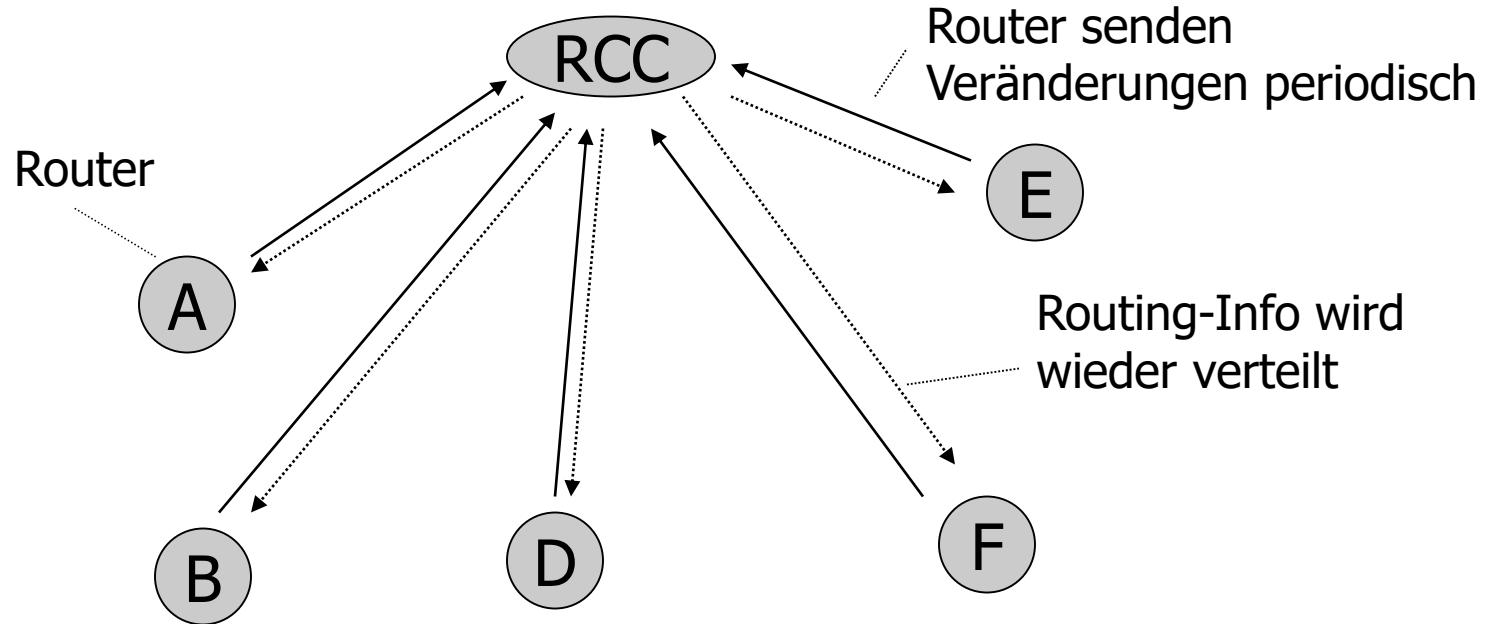
---



- Eingehende Pakete werden **so schnell wie möglich** zum nächsten Netzknoten gesendet
- Es wird der Ausgang mit dem **am geringsten belegten** Output-Buffer gewählt
- Dynamischer und **sehr einfacher** Routing-Algorithmus
- Wird in seiner reinen Form nicht verwendet

## Routing – Zentralisiertes Routing

- Es gibt ein Routing Control Center (RCC)
- Verfahren ist nicht fehlertolerant (Engpass) aber konsistent, jedoch Gefahr der veralteten Informationen

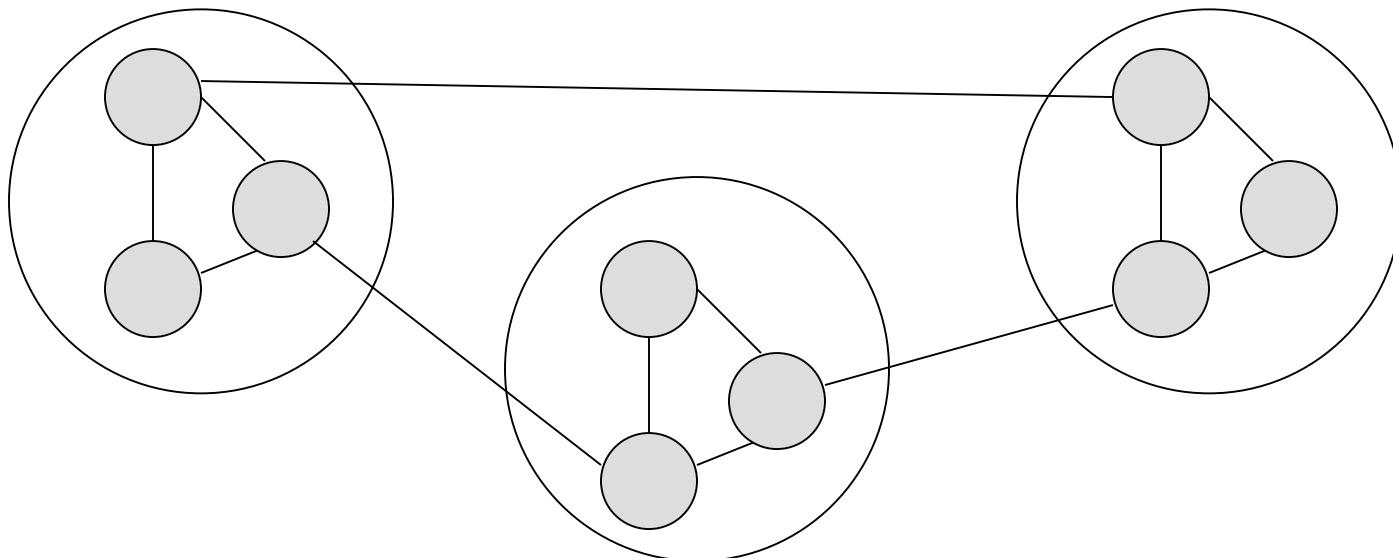


- Ist das Internet zentral oder dezentral organisiert?

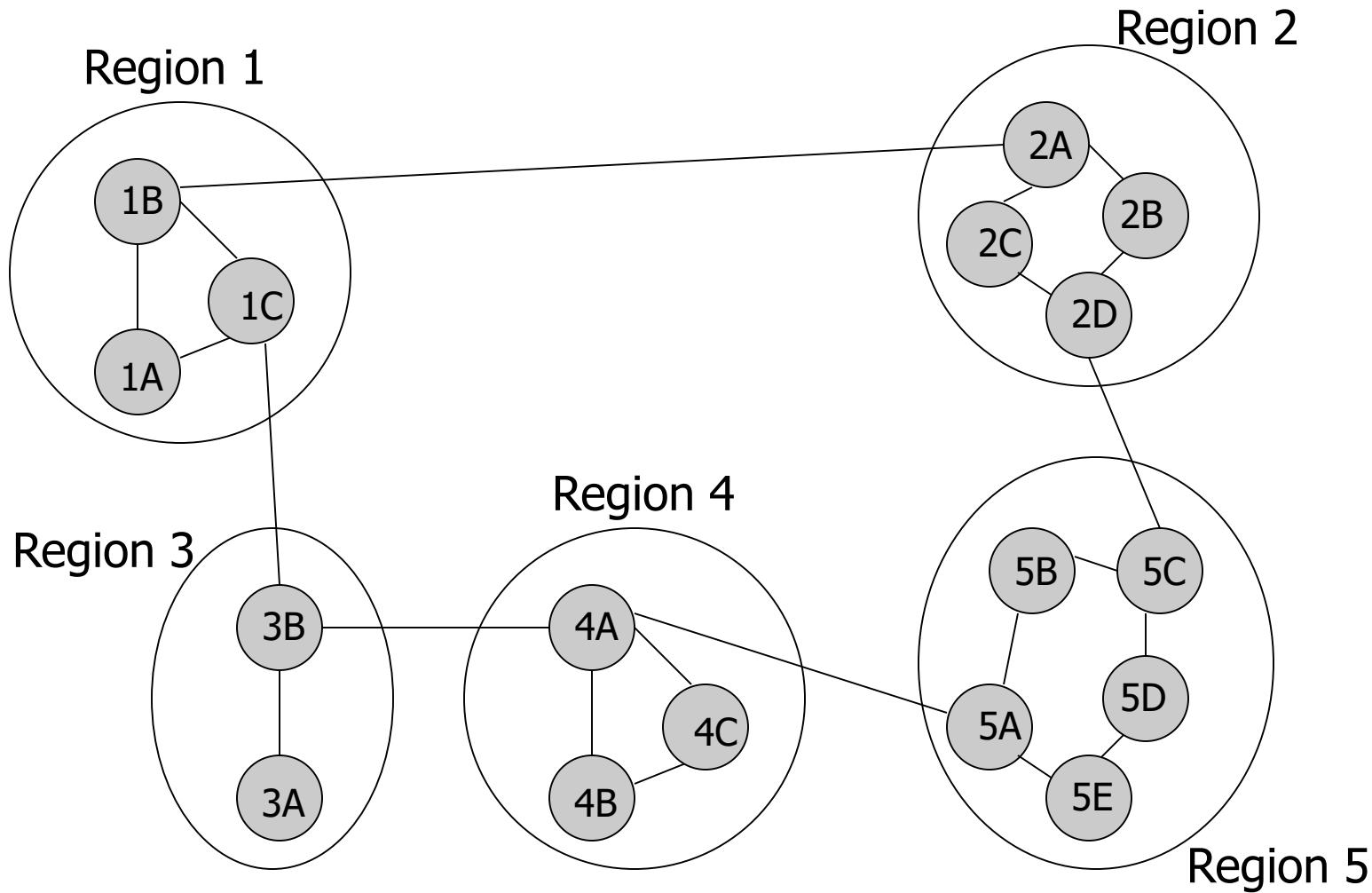
# Routing – Hierarchisches Routing

---

- Große Netze erfordern (zu) große Routing-Tabellen, wenn alle Routen eingetragen werden
  - Lange Suchzeiten
- Verringerung der Routing-Tabellen durch hierarchische Netzorganisation
  - Z.B mit folgenden Hierarchiestufen: Regionen – Cluster – ...



# Routing – Hierarchisches Routing, Beispiel (1)



# Routing – Hierarchisches Routing, Beispiel (2)

**Routing-Tabelle für 1A (vorher)**

Ziel	Leitung	Teilstr.
1A	--	--
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

**Routing-Tabelle für 1A (nachher)**

Ziel	Leitung	Teilstr.
1A	--	--
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

- Reduktion von 17 auf 7 Einträge!

Nach Tanenbaum, A.; Wetherall, D.: Computer Networks,  
5. Auflage, Pearson Studium, 2011

# Distance-Vector-Routing (1)

---

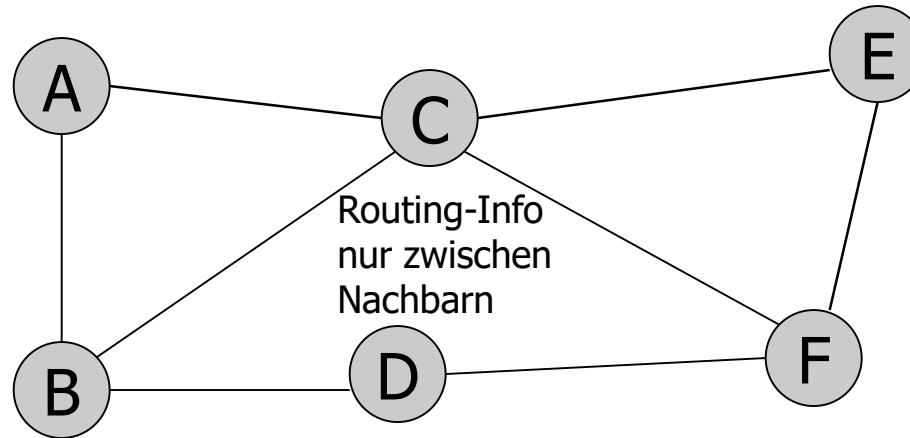
- Andere Bezeichnung: Bellman-Ford-Routing (Bellman, 1957 und Ford, 1962)
- Jeder Router führt eine **dynamisch aktualisierte Routing-Tabelle** mit allen Zielen
  - Einträge enthalten bevorzugte Ausgangsleitung zu einem Ziel
- **Metrik** kann z.B. sein:
  - Verzögerung in ms
  - Anzahl der Teilstrecken (**Hops**) zum Ziel

Ziel	Ausgangsleitung	Nächster Knoten	Metrik

## Distance-Vector-Routing (2)

---

- Verteilt – iterativ – asynchron (unabhängig voneinander)
- **Benachbarte** Router tauschen Routing-Information aus

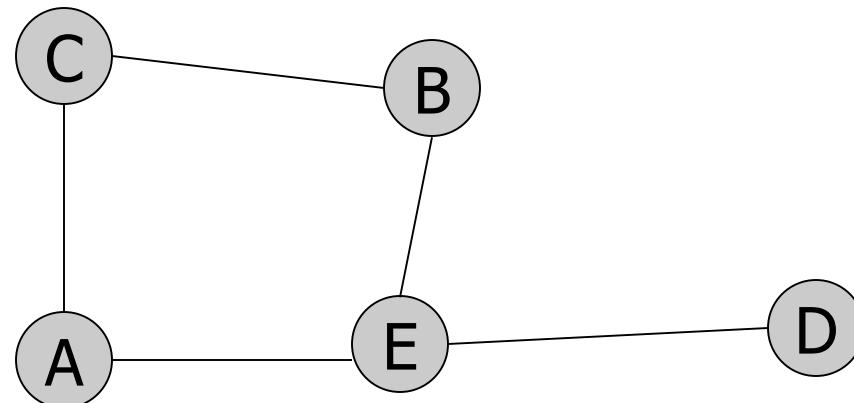


- **Schleifen** möglich „**Count-to-Infinity-Problem**“
  - Bei Ausfall eines Links evtl. keine Terminierung mehr sichergestellt
- **Gute Nachrichten** verbreiten sich schnell
- **Schlechte Konvergenz**
  - Schlechte Nachrichten verbreiten sich sehr langsam in Netz

## Distance-Vector-Routing (3) - Kommunikation

---

- Routing-Informationen werden nur zwischen Nachbarn ausgetauscht
- Es wird **nur die Sicht der Nachbarn und nicht die gesamte Topologie** kommuniziert
- Beispiel:
  - C kommuniziert nur mit A und B
  - E Kommuniziert nur mit A, B und D
  - D kommuniziert nur mit E,...
  - B teilt z. B. C mit, dass er E über einen und D über zwei Hops erreichen kann



# Distance-Vector-Routing - Beispiel

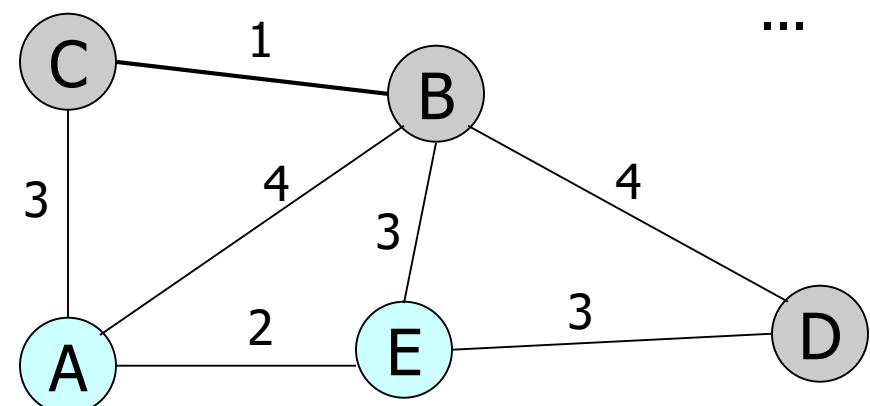
Knoten A

Ziel	Distanz	Nächster Knoten
B	4	B
C	3	C
E	2	E
D	5	E

Knoten E

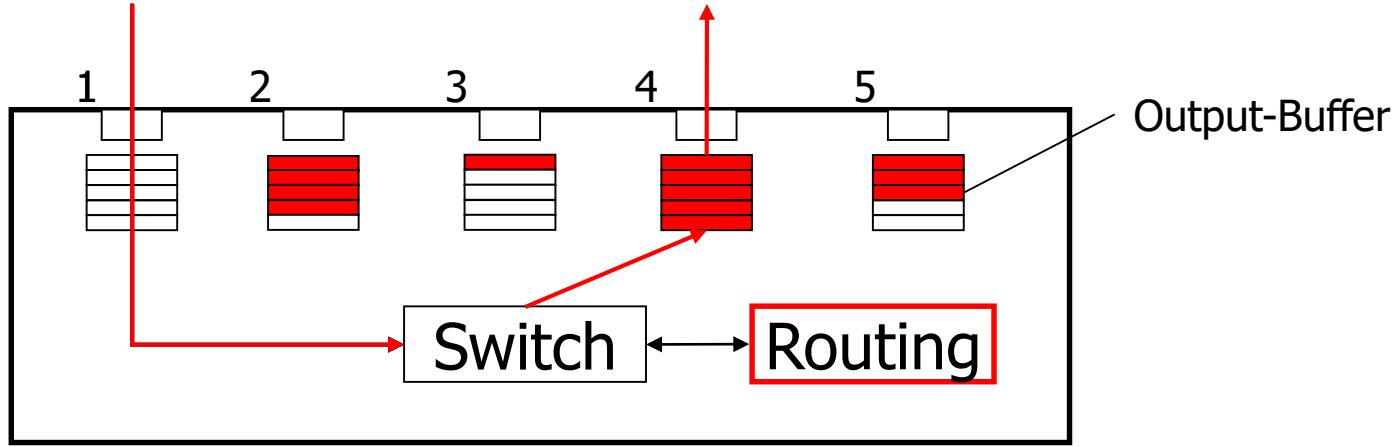
Ziel	Distanz	Nächster Knoten
A	2	A
B	3	B
C	4	B
D	3	D

Nach einiger Zeit (**Konvergenzzeit**)  
verfügen alle Router über optimale  
Routing-Tabellen



## Link-State-Routing (1)

---



- Jeder Router verwaltet eine **Kopie der gesamten Netzwerktopologie** (Link-State-Datenbank)
- Jeder Knoten muss alle Kosteninformationen kennen
- Jeder Router berechnet selbst die optimale Route für ein Paket, also den **absolut kürzesten Pfad**
- Verschiedene Optimierungskriterien sind möglich

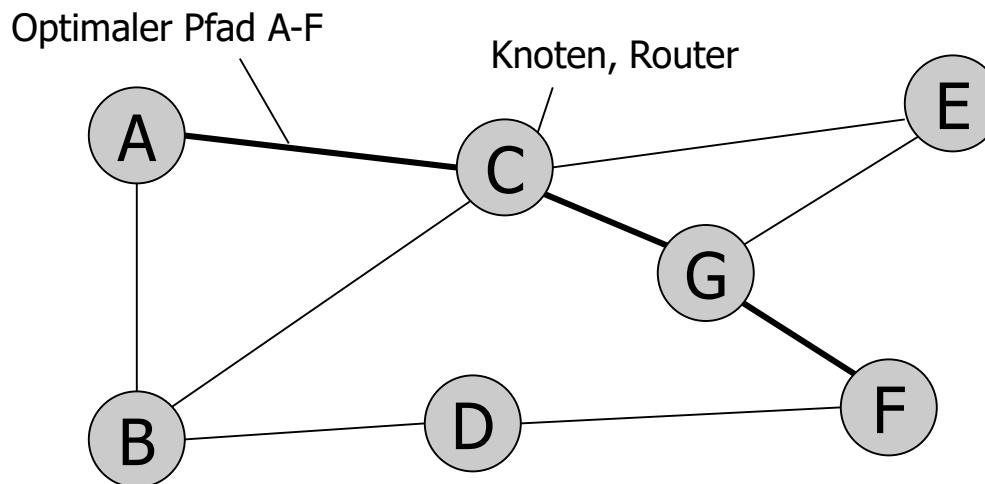
## Link-State-Routing (2)

---

- Jeder Router verteilt die lokale Information per Flooding an alle anderen Router im Netz
- Die Berechnung der Routen erfolgt **dezentral**
- **Berechnung der kürzesten Pfade** z.B. über Shortest-Path-Algorithmus (z.B. Dijkstra- oder Bellmann-Ford-Algorithmus)
- **Keine Schleifen** möglich, da jeder Knoten die gleiche Information über die Topologie besitzt
- Schnelle Reaktion auf Topologieänderungen möglich

# Routing – Optimalitätsprinzip (1)

- Das Optimalitätsprinzip nach Richard Bellmann besagt:
  - Wenn der optimale Pfad A-F zwischen A und F über die Knoten C und G führt, muss auch der Pfad zwischen C und G ein optimaler Pfad zwischen diesen Knoten sein
  - Wäre das nicht der Fall, könnte A-F nochmals verkürzt werden und dann wäre A-F kein kürzester Weg zwischen A und F  
→ Widerspruch zur Annahme



# Routing – Optimalitätsprinzip (2)

---

## ■ **Sink Tree**

- Die optimalen Routen von allen Quellen zu einem bestimmten Ziel bilden einen **Baum**, dessen Wurzel das Ziel ist
- Dieser Baum enthält keine Schleifen und heißt **Sink Tree** oder **Senke** (optimierter Spanning Tree)

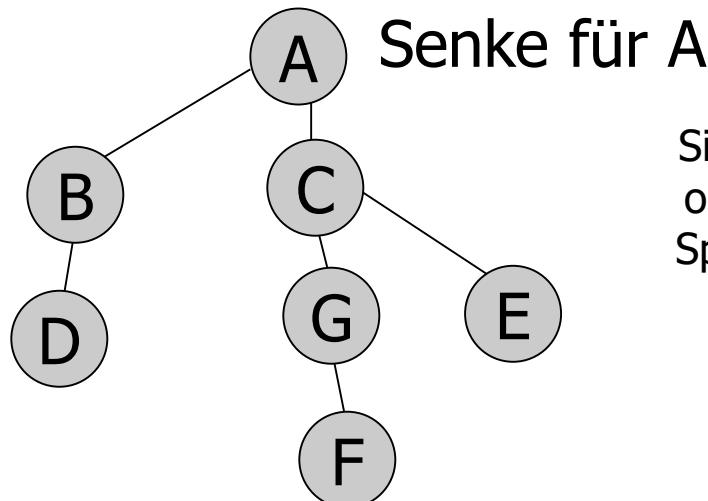
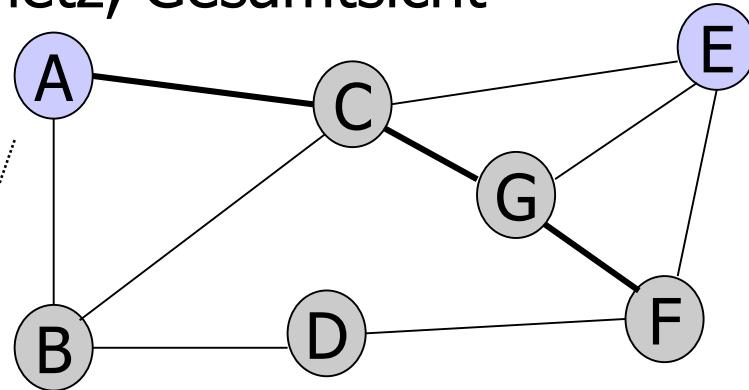
## ■ Ziel von Routing-Algorithmen

- Sink Trees für alle Router ermitteln
- Sink Trees für das Routing nutzen

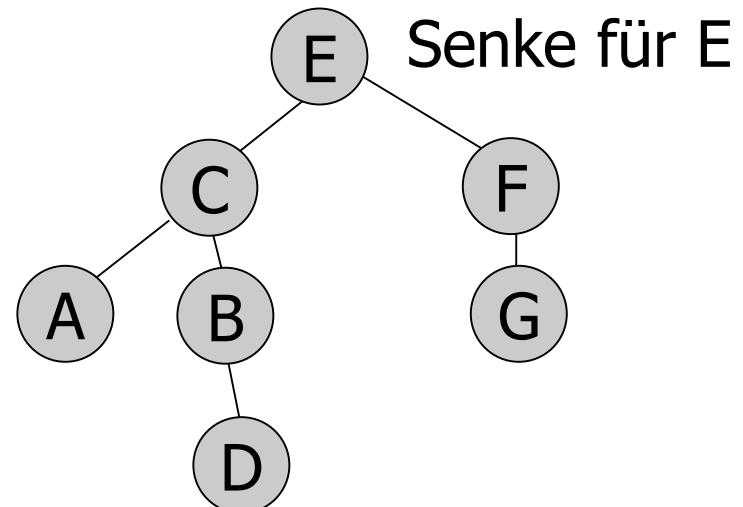
**Def. Spanning Tree:** Teilgraph eines ungerichteten Graphen, der alle Knoten des Graphen enthält. Ein minimaler „Spannbaum“ eines kantengewichteten Graphen hat die kleinste Summe aller Kantengewichte. Es gibt also keinen Spannbaum für den Graphen, der eine kleinere Summe der Kantengewichte hat.

## Routing – Optimierungsprinzip (3)

- Ursprungsnetz, Gesamtsicht



Sink Tree =  
optimierter  
Spannbaum



# Shortest-Path-Routing

---

- **Graph des Teilnetzes** wird erstellt (statisch oder dynamisch)
  - Knoten entspricht Router
  - Kante entspricht einer Leitung zwischen zwei Routern
- Kante wird **beschriftet** („Pfadlänge“), die Metrik hierfür kann berechnet werden aus
  - Entfernung
  - Bandbreite
  - Durchschnittsverkehr
  - Durchschnittliche Warteschlangenlänge in den Routern
  - Verzögerung
  - ...
- Berechnung des kürzesten Pfads z.B. über Dijkstra- oder Bellmann-Ford-Algorithmus

# Dijkstra-Algorithmus

- Erfindung von **Edsger W. Dijkstra** im Jahr 1959
- Nur für positive Kantengewichte
- Gehört zur Klasse der Greedy-Algorithmen
  - Schrittweise wird ein Folgezustand ausgewählt, der den größten Fortschritt verspricht (greedy = gierig)
- Problemstellung aus der Graphentheorie
  - Finde für einen **Startknoten s** und einen **Endknoten e** eines **gewichteten Graphen G** mit der **Knotenmenge V**, der **Kantenmenge E** und der **Kostenfunktion k** einen Weg zwischen s und e mit minimalen Kosten
  - Die Kostenfunktion k bezieht sich auf eine Kante zwischen zwei Knoten

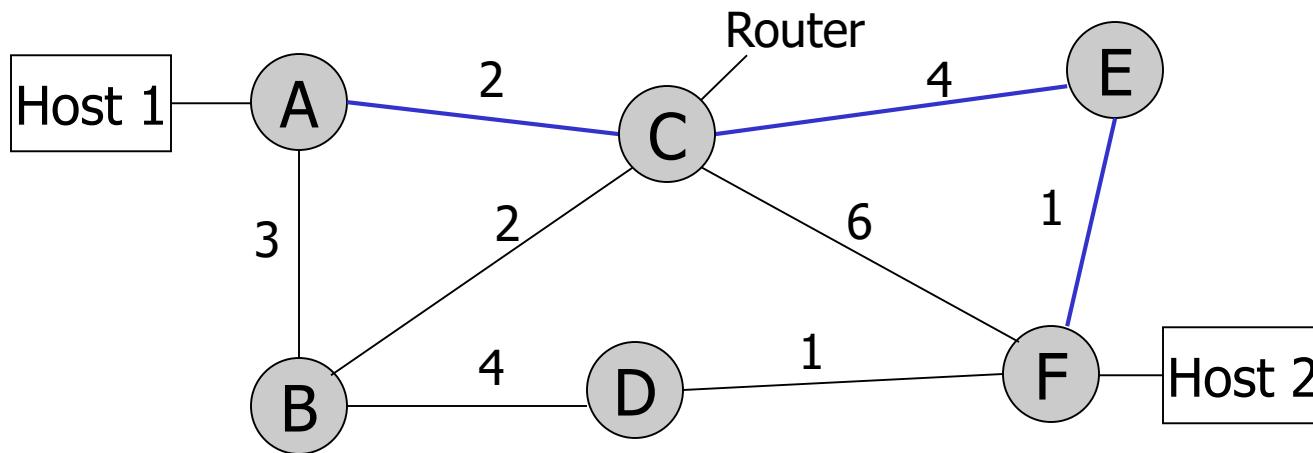


Fotoquelle: [http://de.wikipedia.org/wiki/Edsger\\_W.\\_Dijkstra](http://de.wikipedia.org/wiki/Edsger_W._Dijkstra)

# Shortest-Path-Routing

## Beispiel

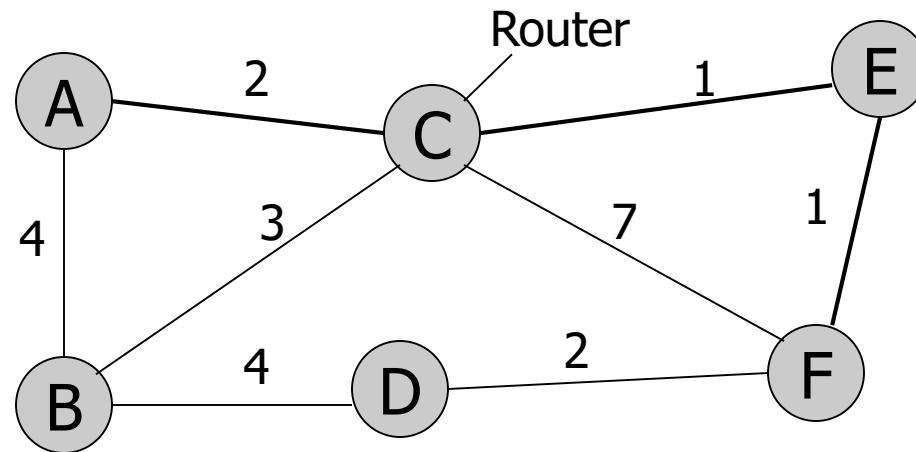
- Beispiel: Der kürzeste Pfad zwischen Host1 und Host2 geht von A nach F über A C E F
- Summierte Pfadlänge = 7



## Routing - Übung

---

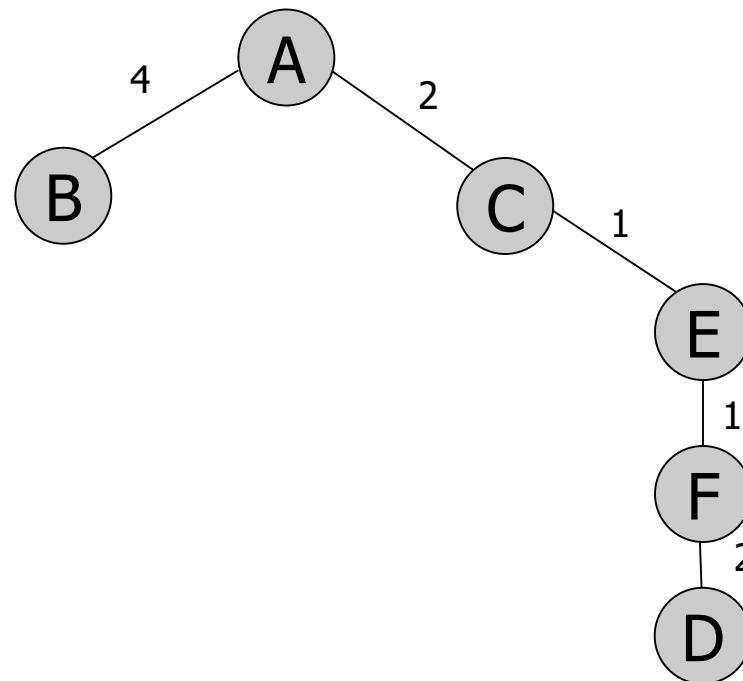
- Ermitteln Sie die Sink Trees (optimale Spannbäume) des vorliegenden kantengewichteten Graphen für die Knoten A und E und zeichnen Sie die Graphen
- Beschriften Sie die Kanten mit den Kantengewichten!



# Routing – Lösung zur Übung (1)

---

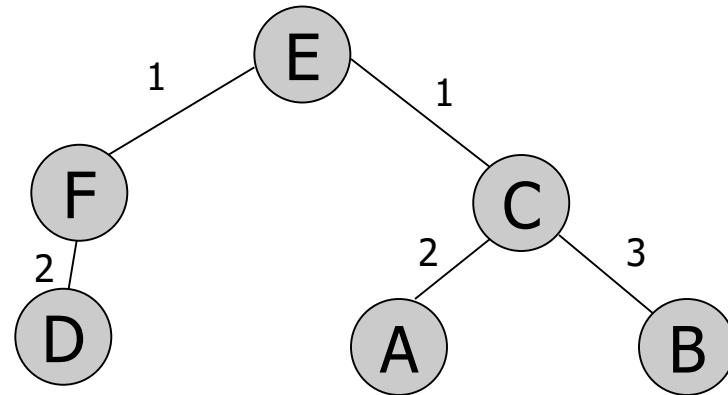
- Sink Tree (= optimaler Spannbaum) für A



## Routing – Lösung zur Übung (2)

---

- Sink Tree (= optimaler Spannbaum) für E



# Dijkstra-Algorithmus

## Pseudocode (1)

Quelle: Algorithmus aus  
<http://de.wikipedia.org/wiki/Dijkstra-Algorithmus>, didaktisch angepasst

```
01: void Dijkstra(Graph g, Startknoten s) {
02:   // Initialisierung des Graphen
03:   for all Knoten v in Graph g {
04:     g[v].abstand := unendlich;
05:     g[v].vorgänger := null;
06:   }
07:   g[s].abstand := 0;
08:   Q := Die Menge aller Knoten in Graph g;
09:
10:  // Der eigentliche Algorithmus
11:  while Q nicht leer    {
12:    u := Knoten in Q mit kleinstem Abstand;
13:    entferne u aus Q // für u ist der kürzeste Weg nun bestimmt
14:    for all Nachbarn v von u { // Vu ist die Menge der Nachbarn von u
15:      if v in Q
16:        // prüfe Abstand vom Startknoten zu v
17:        distanceUpdate(u, v, g); // Ändert auch gleich den Weg im Graphen g
18:    }
19:  }
20:  return g; // mit allen Vorgängern
21: }
```

Graph g hier als Array von Knoten mit den Attributen abstand und vorgänger modelliert:

```
class Graph {
  int vorgänger;
  int abstand;
}
```

Initialisierung:  
Graph[] g = new Graph[10];  
Alle Knoten in g eintragen;

# Dijkstra-Algorithmus

## Pseudocode (2)

Quelle: Algorithmus aus  
<http://de.wikipedia.org/wiki/Dijkstra-Algorithmus>, didaktisch angepasst

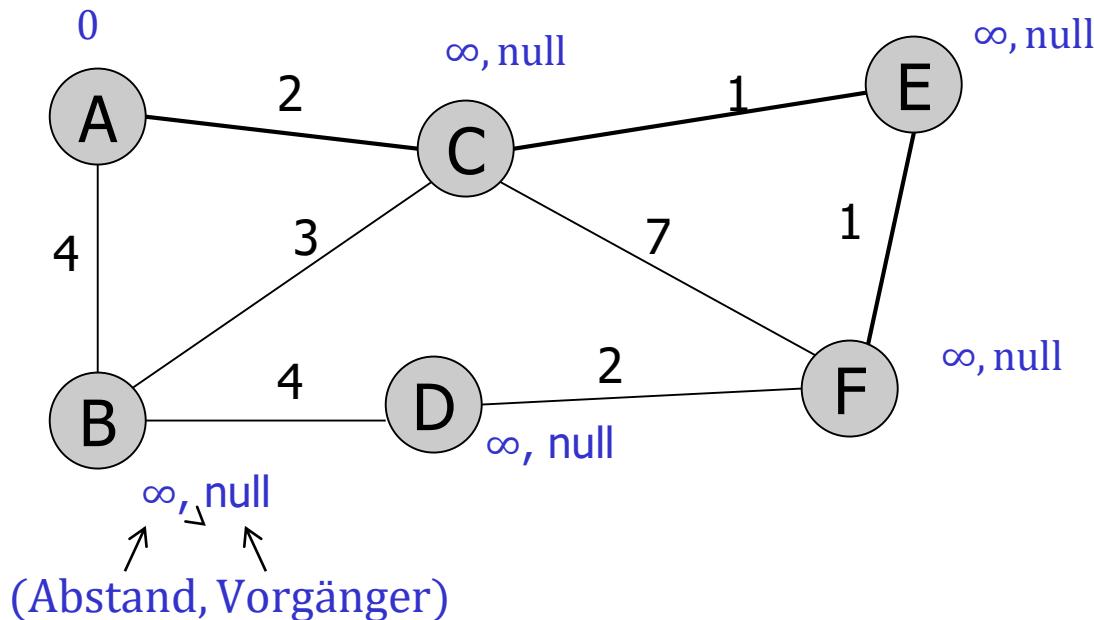
```
01: void distanceUpdate(u, v, g) {  
02:     // Weglänge vom Startknoten nach v über u  
03:     alternative := g[u].abstand + abs(g[u].abstand - g[v].abstand); // Absolutbetrag;  
04:     if alternative < g[v].abstand {  
05:         g[v].abstand := alternative;  
06:         g[v].vorgänger := u;  
07:     }  
08: }
```

- **Idee des Algorithmus:** Folge immer derjenigen Kante, die den kürzesten Streckenabschnitt vom Startknoten aus verspricht
  - Der Startknoten erhält als Distanz (Abstand) den Wert 0
  - Q ist die Menge aller Knoten, für die noch kein kürzester Weg gefunden wurde (Abstand zunächst unendlich)

# Dijkstra-Algorithmus

## Beispiel (1)

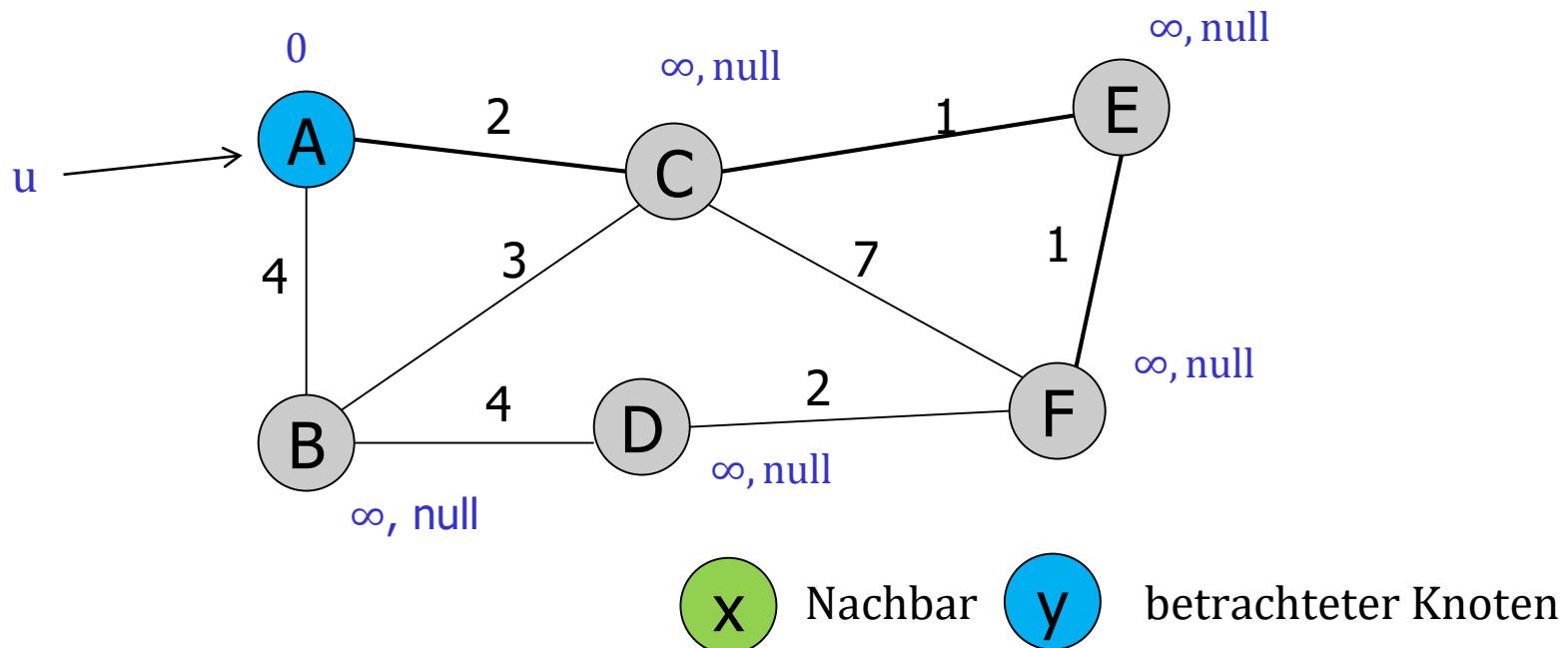
- A ist Startknoten, Distanz = 0
- Initialisierung aller Knoten im Graph mit Abstand = „unendlich“ und „kein Vorgänger im Graph“ (null)



# Dijkstra-Algorithmus

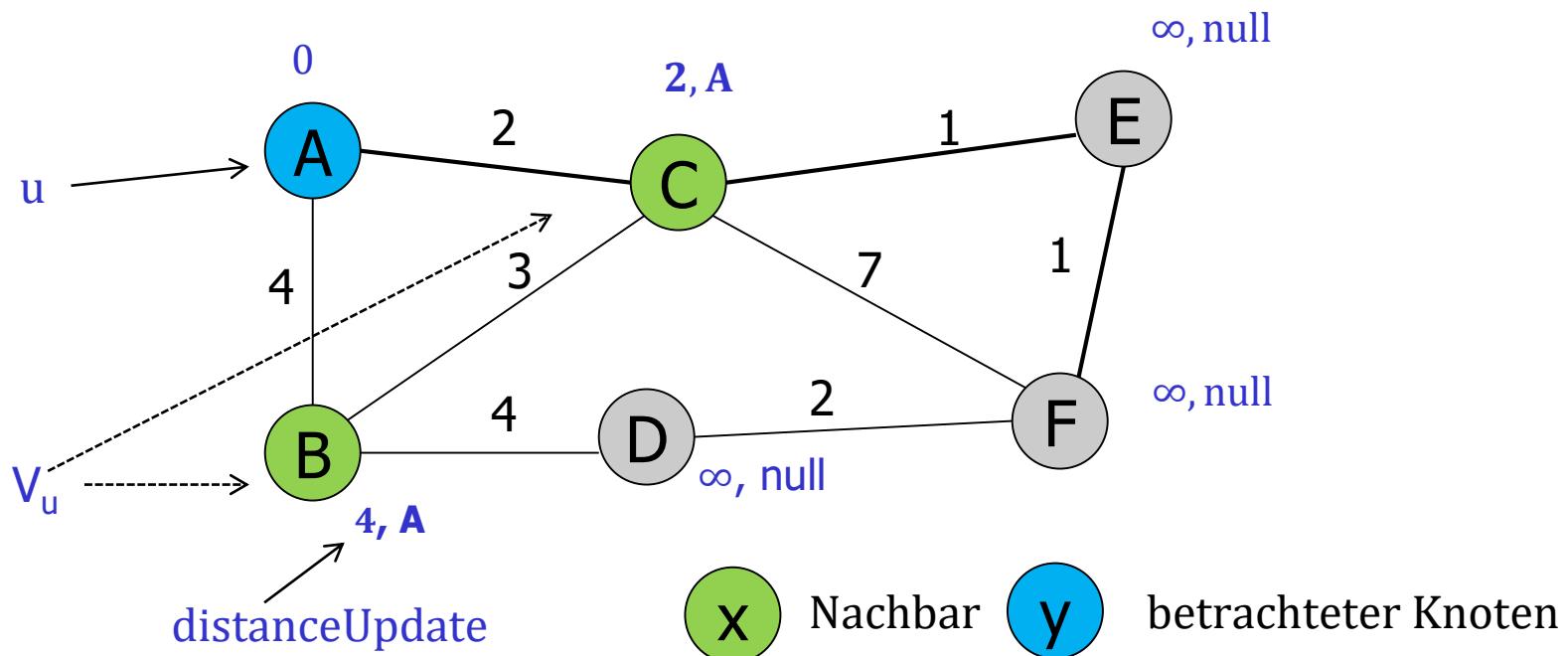
## Beispiel (2)

- while Q nicht leer ...
  - Einstiegsknoten festlegen und los geht's
  - $u := A$
  - $Q = \{B, C, D, E, F\}$ ; für alle Knoten in Q ist noch kein kürzester Weg gefunden



# Dijkstra-Algorithmus Beispiel (3)

- 1. Iteration
    - $u := A$
    - $V_u := \{B, C\}$
    - $Q = \{B, C, D, E, F\}$
    - Abstände zu B und C prüfen über `distanceUpdate(...)`

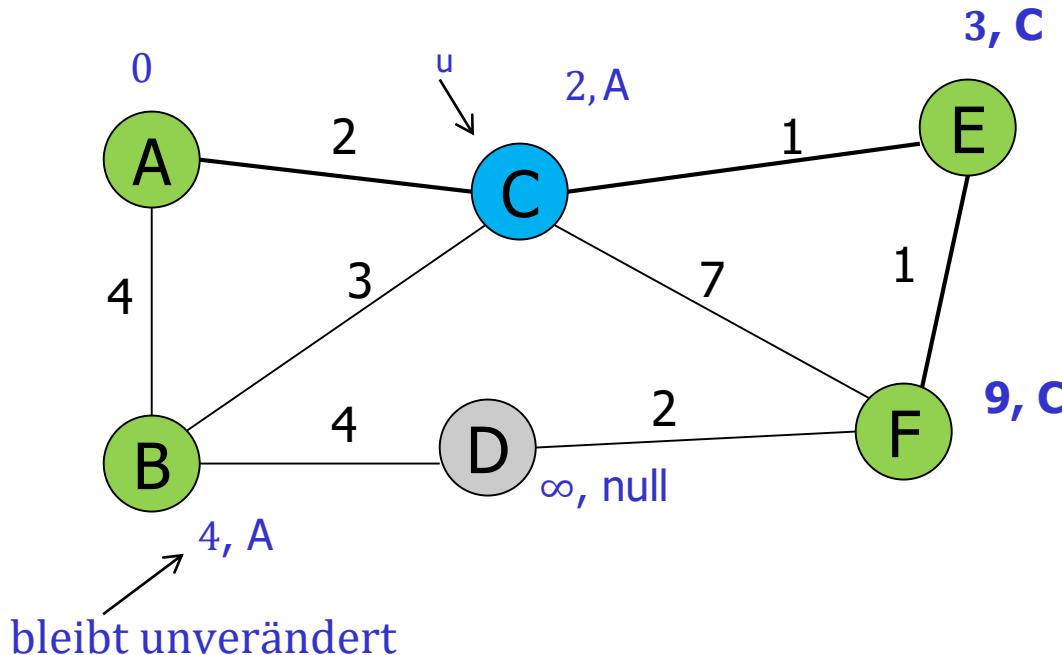


# Dijkstra-Algorithmus

## Beispiel (4)

- 2. Iteration

- $u := C$  (Knoten mit kleinstem Abstand in  $Q$ )
- $V_u := \{A, B, E, F\}$
- $Q = \{B, D, E, F\}$
- Abstände prüfen über `distanceUpdate(...)`

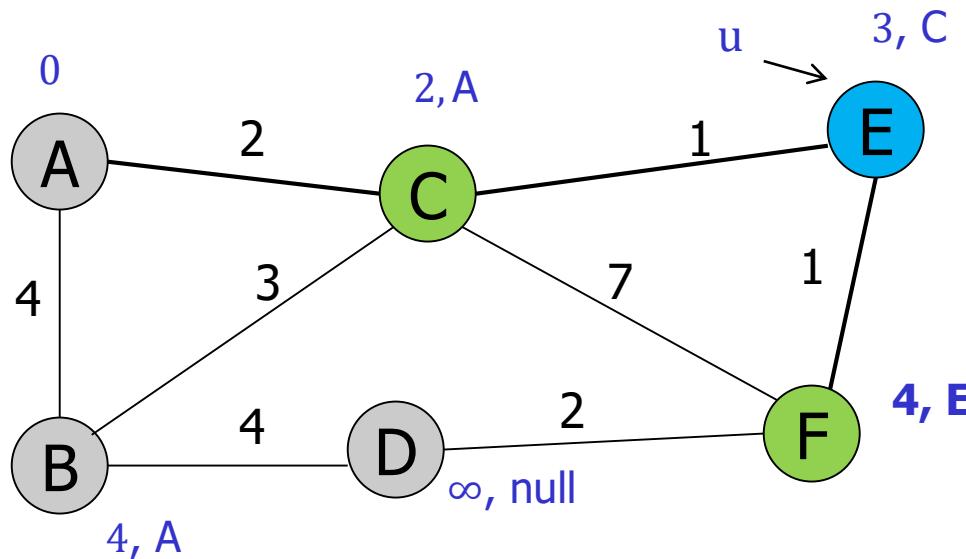


# Dijkstra-Algorithmus

## Beispiel (5)

### ■ 3. Iteration

- $u := E$  (Knoten mit kleinstem Abstand in  $Q$ )
- $V_u := \{C, F\}$
- $Q = \{B, D, F\}$
- Abstände prüfen über `distanceUpdate(...)`

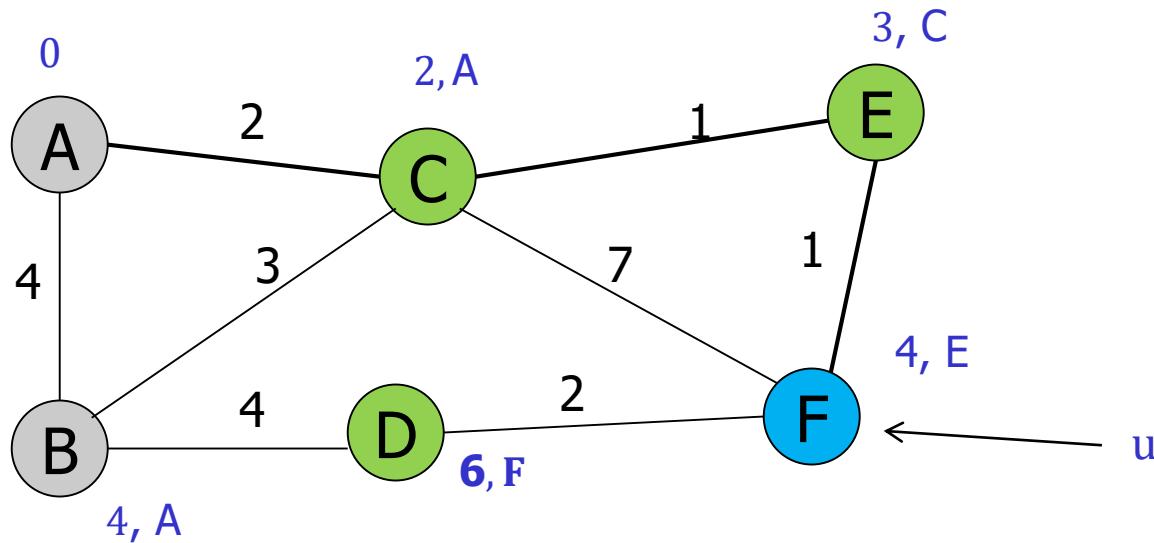


# Dijkstra-Algorithmus

## Beispiel (6)

### ■ 4. Iteration

- $u := F$  (Knoten mit kleinstem Abstand in  $Q$ , könnte auch  $B$  sein)
- $V_u := \{C, D, E\}$
- $Q = \{B, D\}$
- Abstand prüfen über `distanceUpdate(...)`

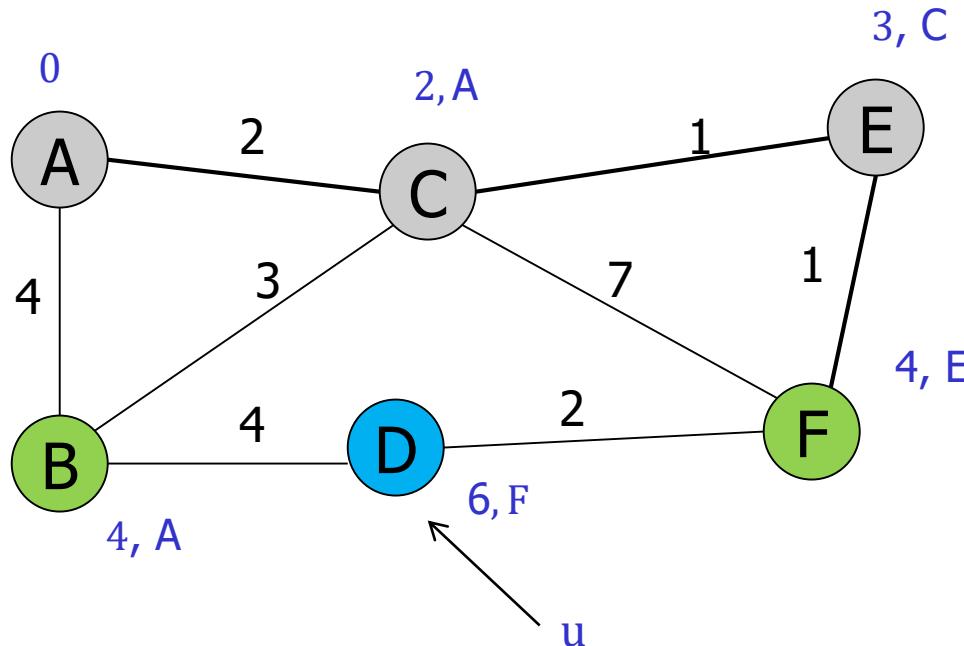


# Dijkstra-Algorithmus

## Beispiel (7)

### ■ 5. Iteration

- $u := D$  (Knoten mit kleinstem Abstand in  $Q$ )
- $V_u := \{B, F\}$  (keine Verbesserung des Abstands zu B)
- $Q = \{B\}$
- Abstand prüfen über `distanceUpdate(...)`

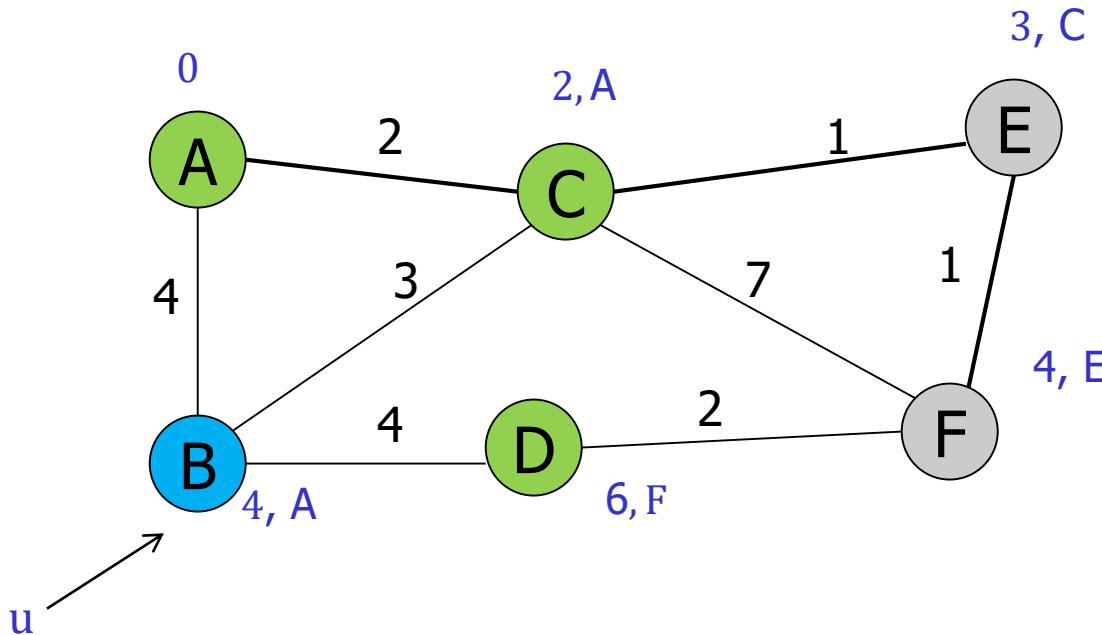


# Dijkstra-Algorithmus

## Beispiel (8)

### ■ 6. Iteration

- $u := B$
- $V_u := \{A, C, D\}$ , jedoch nicht mehr in  $Q$ , also passiert nichts
- $Q = \{\}$   $\rightarrow$  Ende der Berechnung
- Abstand prüfen über `distanceUpdate(...)`



# Dijkstra-Algorithmus

## Beispiel (9)

---

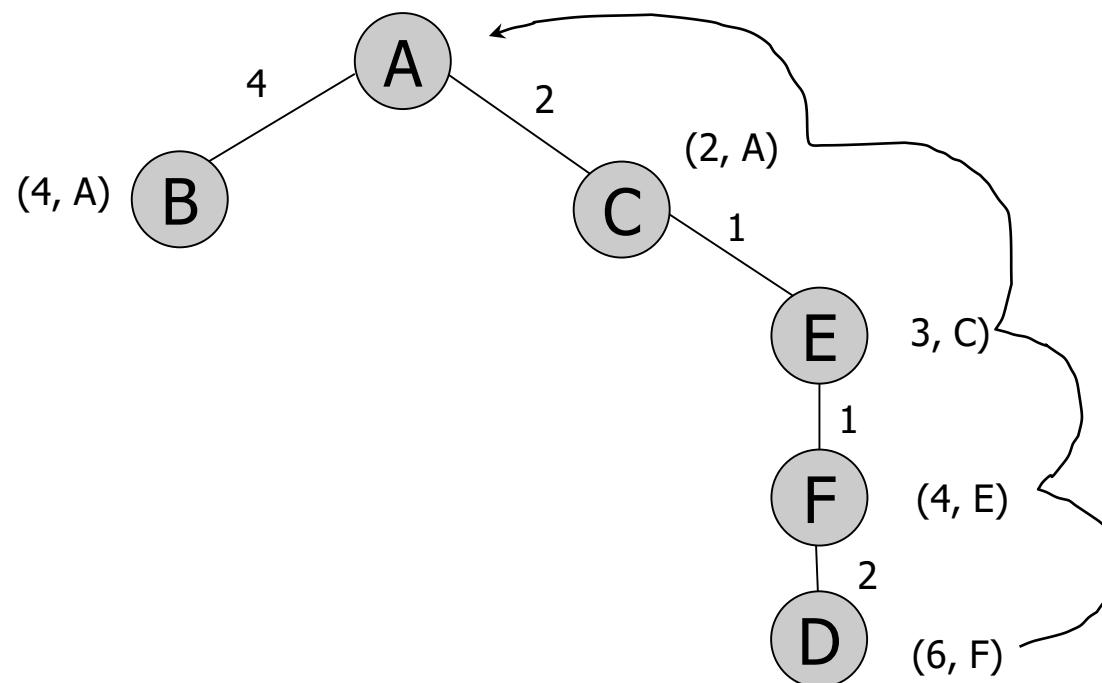
- Über die Methode *erstelleKürzestenPfad* kann dann im Graphen der kürzeste Weg von einem Knoten zu einem Zielknoten ermittelt werden
- Es wird über alle Vorgänger des Zielknotens iteriert

```
01: void erstelleKürzestenPfad(Zielknoten z, Graph g) {  
02:   Weg[] := [z];  
03:   u := z; // Zielknoten  
04:   while g[u].vorgänger nicht null { // Der Vorgänger des Startknotens ist null  
05:     u := g[u].vorgänger;  
06:     füge u am Anfang von Weg[] ein;  
07:   }  
08:   return Weg[];  
09: }
```

# Dijkstra-Algorithmus

## Beispiel (10)

- Kürzester Weg von Senke A zu Zielknoten D
  - Von D über F, über E, über C zu A
- Auch der gesamte Sink Tree zu einer Senke lässt sich ermitteln



## Rückblick

---

- ✓ Aufgaben und Dienste
- ✓ Vermittlungsverfahren
- ✓ Wegewahl, Routing

- 1 Kommunikationssysteme und verteilte Anwendungen
- 2 Grundlagen der Transportschicht
- 3 Transportprotokolle TCP und UDP
- 4 Grundlagen der Vermittlungsschicht
- 5 **Internet und Internet Protocol (IP)**
- 6 Routing-Verfahren und -Protokolle
- 7 Internet-Steuerprotokolle und DNS
- 8 Internet Protocol Version 6 (IPv6)
- 9 Netzwerkschnittstelle

## 1. Überblick

- Organisation des Internets, Autonome Systeme (AS), Internet Exchange Points
- Aufgaben des Internet Protokolls

## 2. IPv4-Adressierung

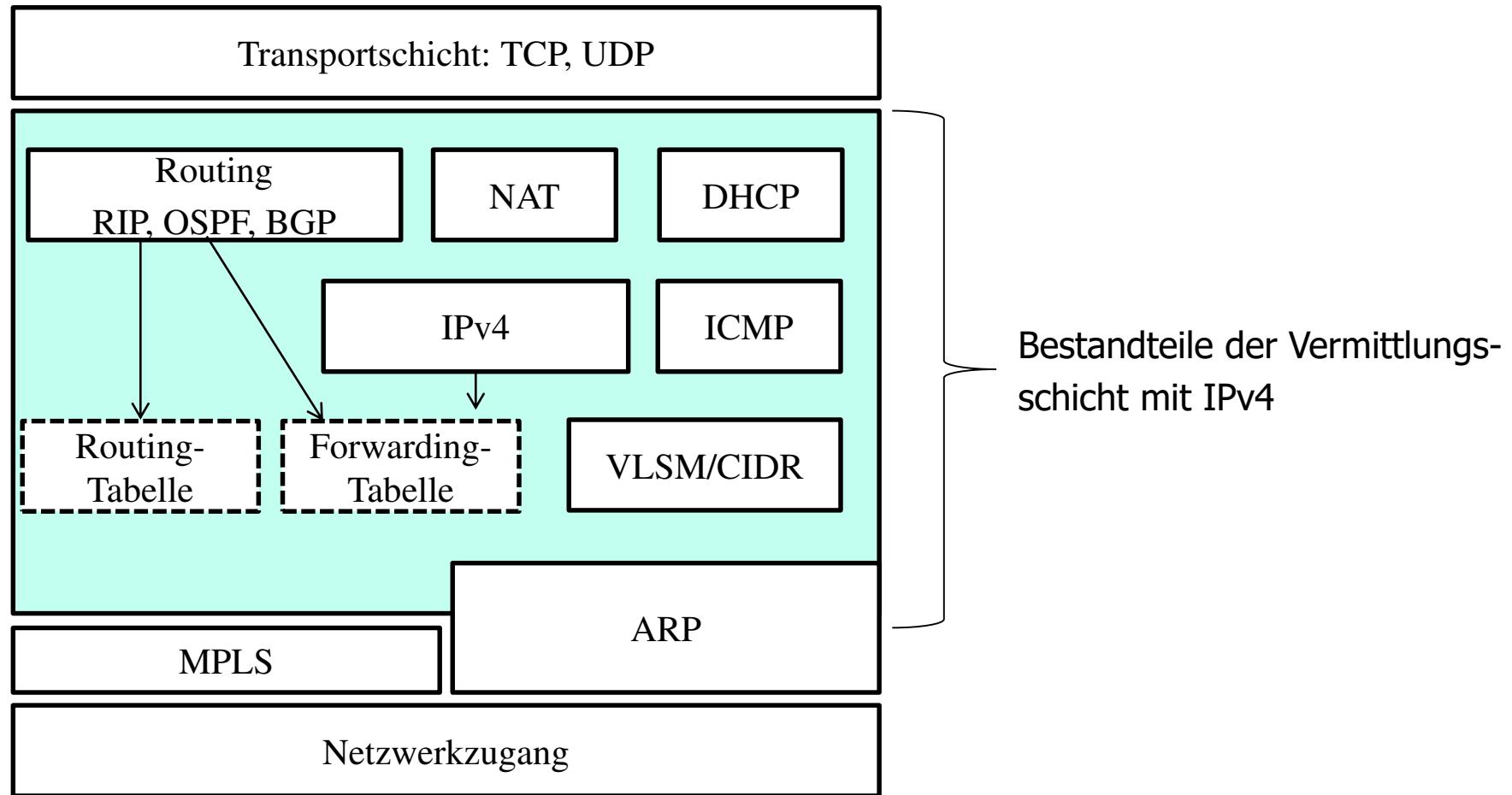
- IPv4: Adressierung
- IPv4-Subnetting
- VLSM und CIDR

## 3. IPv4-PDU

- Aufbau und Felder

## 4. Fragmentierung

# Überblick: Die Internet-Vermittlungsschicht



MPLS = Multi Protocol Label Switching:

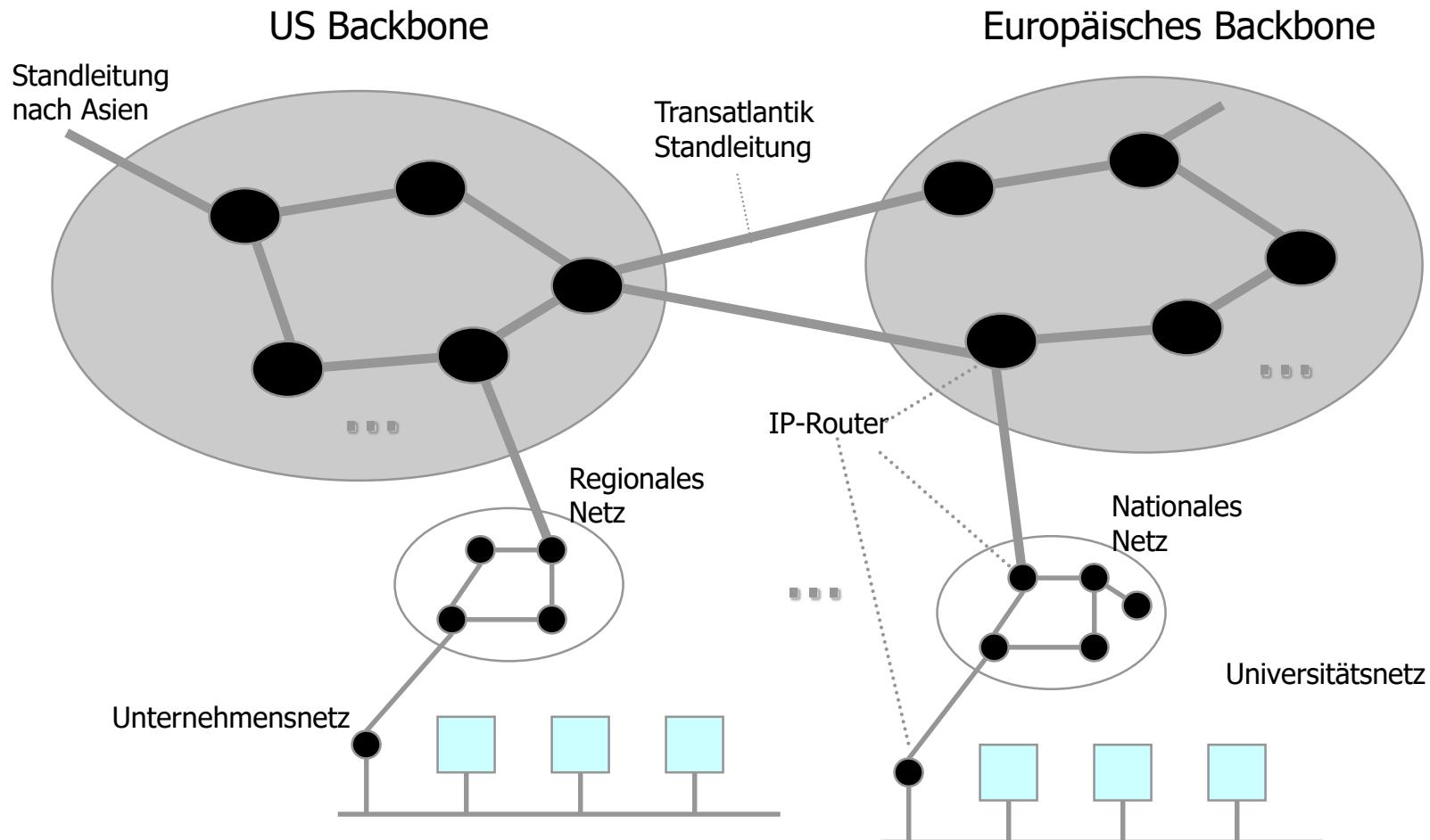
Verbindungsorientierte Übertragung in IP-Netzen über signalisierte Pfade

# Überblick über das Internet

---

- Das Internet ist **weltweiter Verbund** von Rechnernetzen (Global Area Network, GAN)
- **Große Backbones** sind über Leitungen mit hoher Bandbreite und schnellen Routern miteinander verbunden
  - Ein Backbone kann als Basisnetz eines Telekommunikationsnetzes mit sehr Datenübertragungsrate verstanden werden
- An den Backbones hängen **regionale Netze**
- An den regionalen Netzen hängen die **Netze von Unternehmen, Universitäten, Internet Service Providern (ISP),...**

# Überblick über das Internet, Backbone



Quelle: Tanenbaum, A.: et al.: Computer Networks, 5. Auflage, Pearson Studium, 2011

# Das Internet heute, Autonome Systeme (AS)

---

- Autonome Systeme sind **große autonom verwaltete Netzwerke**
- Autonome Systeme haben ein **eigenes Routing-System**, das nach außen kohärent erscheint
- Es gibt derzeit mehr als 60.000 autonome Systeme weltweit
- CIDR Report: <http://www.cidr-report.org/as2.0/>
- Jedes AS hat eine **eindeutige Nummer (ASN)**
  - 11, Harvard-University
  - 1248, Nokia
  - 2022, Siemens 3680, Novell
  - 6142, Sun
  - **12816, MWN ...**

Whois:

<http://www.cidr-report.org/as2.0/autnums.html>

# Das Internet heute: AS-Kategorien

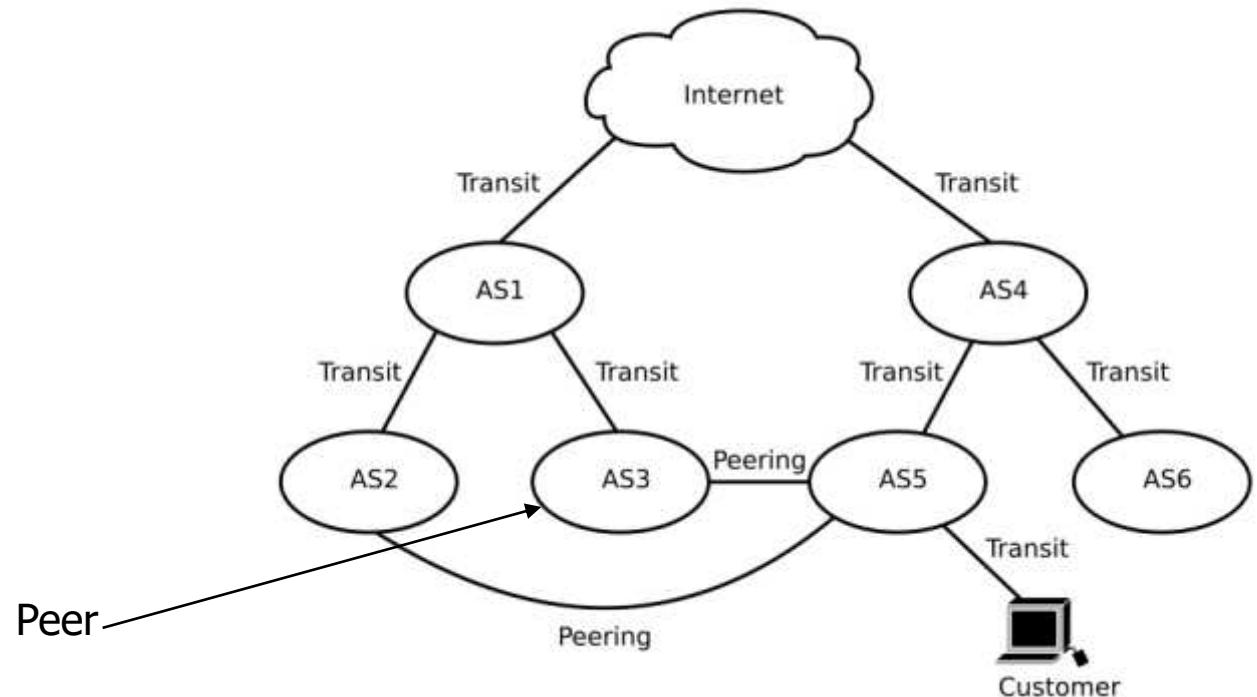
---

- **Tier-3:** Kleine, lokale Provider, Endkundengeschäft
  - M-net in Bayern (Hauptgesellschafter: Münchner Stadtwerke)
  - Hansenet (Hamburg, Tochter der Telecom Italia)
  - Versatel (Berlin)
- **Tier-2:** Betreiber großer, überregionaler Netze
  - Deutsche Telekom
  - France Telecom
  - Tiscali (Telekom-Unternehmen, Italien)
- **Tier-1:** Betreiber von globalen Internet-Backbones (nur Peering!)
  - AT&T (US-amerikanischer Telekom-Anbieter)
  - AOL (US-amerikanischer Online-Dienst)
  - NTT (Nippon Telegraph and Telephone Corporation)
  - Verizon Communications (US-amerikanischer Telekom-Anbieter)

# Das Internet heute, Kunden - Provider - Peers

---

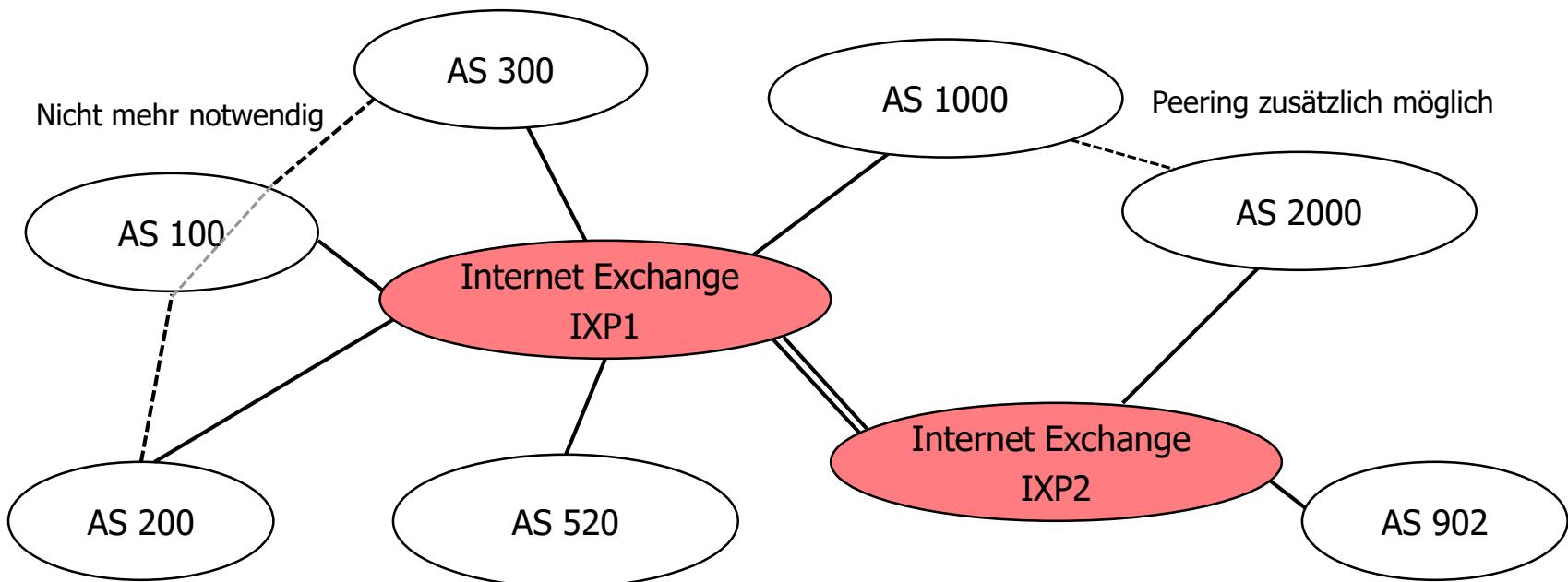
- Rollen: Kunden - Provider - Peers
- Peering-Abkommen zwischen AS (Tier-1, Tier-2, Tier-3)
- Tier-1-Provider „peeren“ kostenlos miteinander, die anderen je nach Vereinbarung



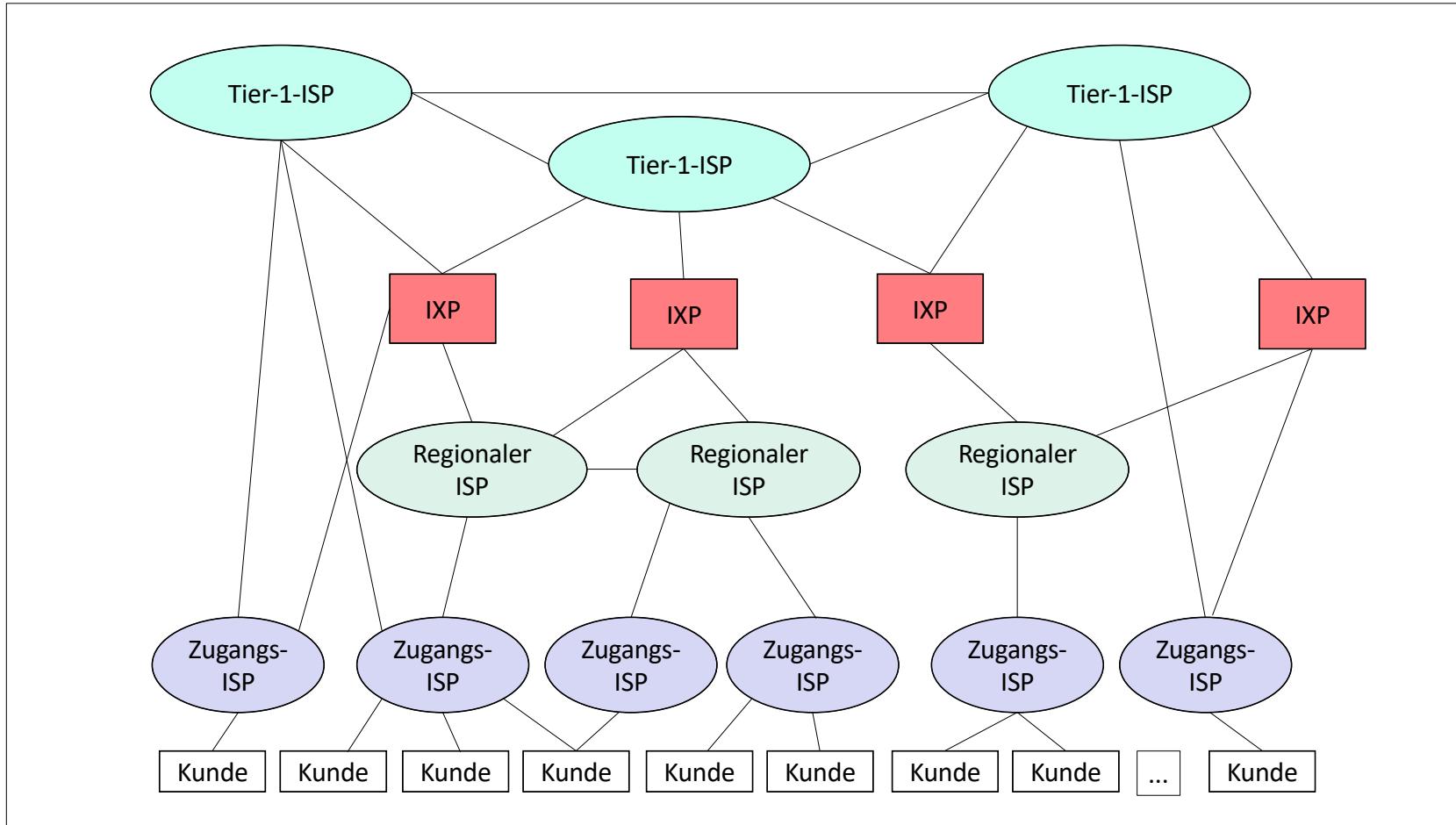
Quelle: Wikipedia

# Internet Exchange Points - IXP

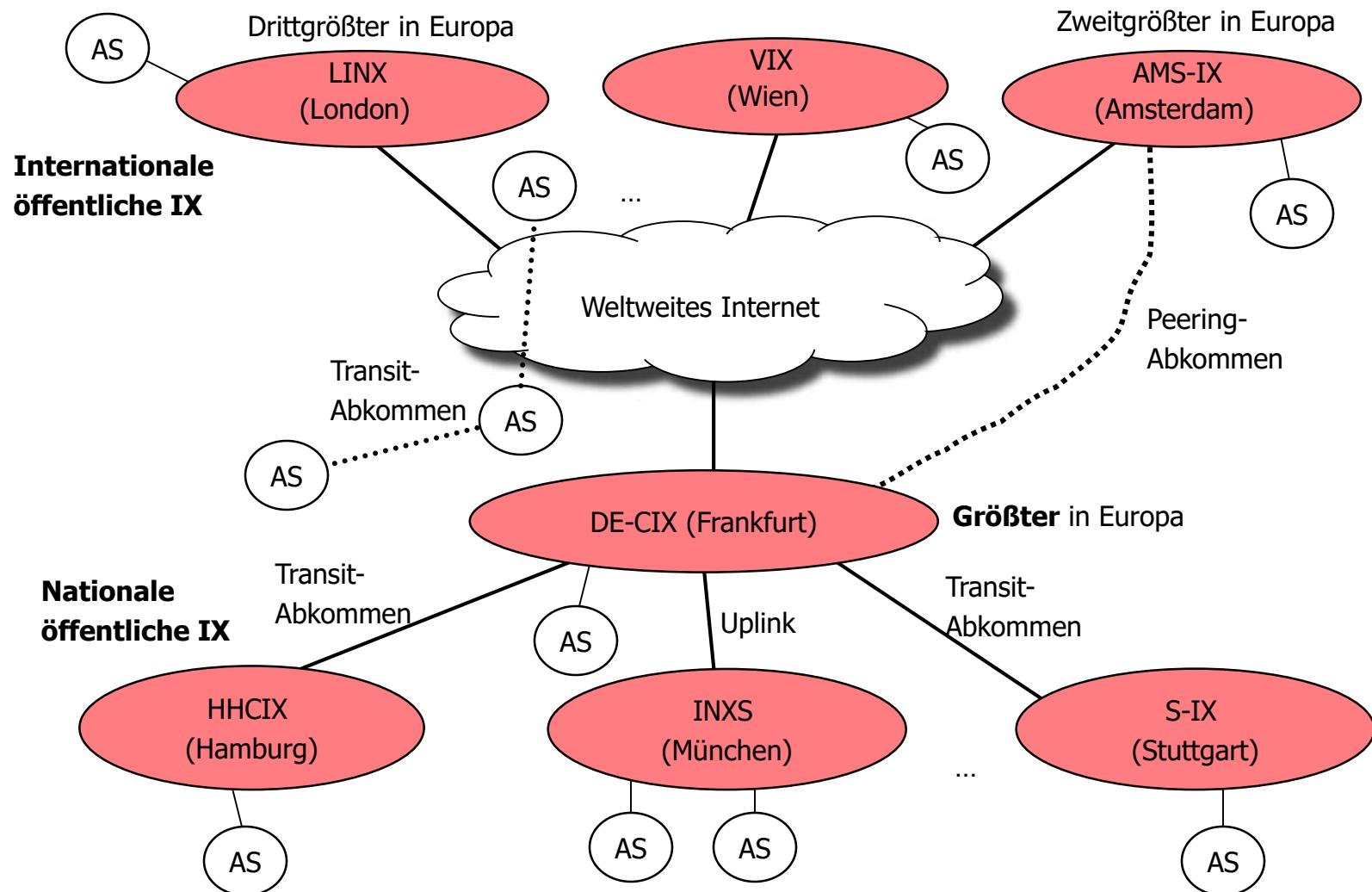
- IXPs sind keine Internet Service Provider (ISP)
- IXPs verbinden Autonome Systeme, eine AS-AS-AS-Verbindung wird damit überflüssig



# Internet Exchange Points (IXPs) im Gesamtkontext

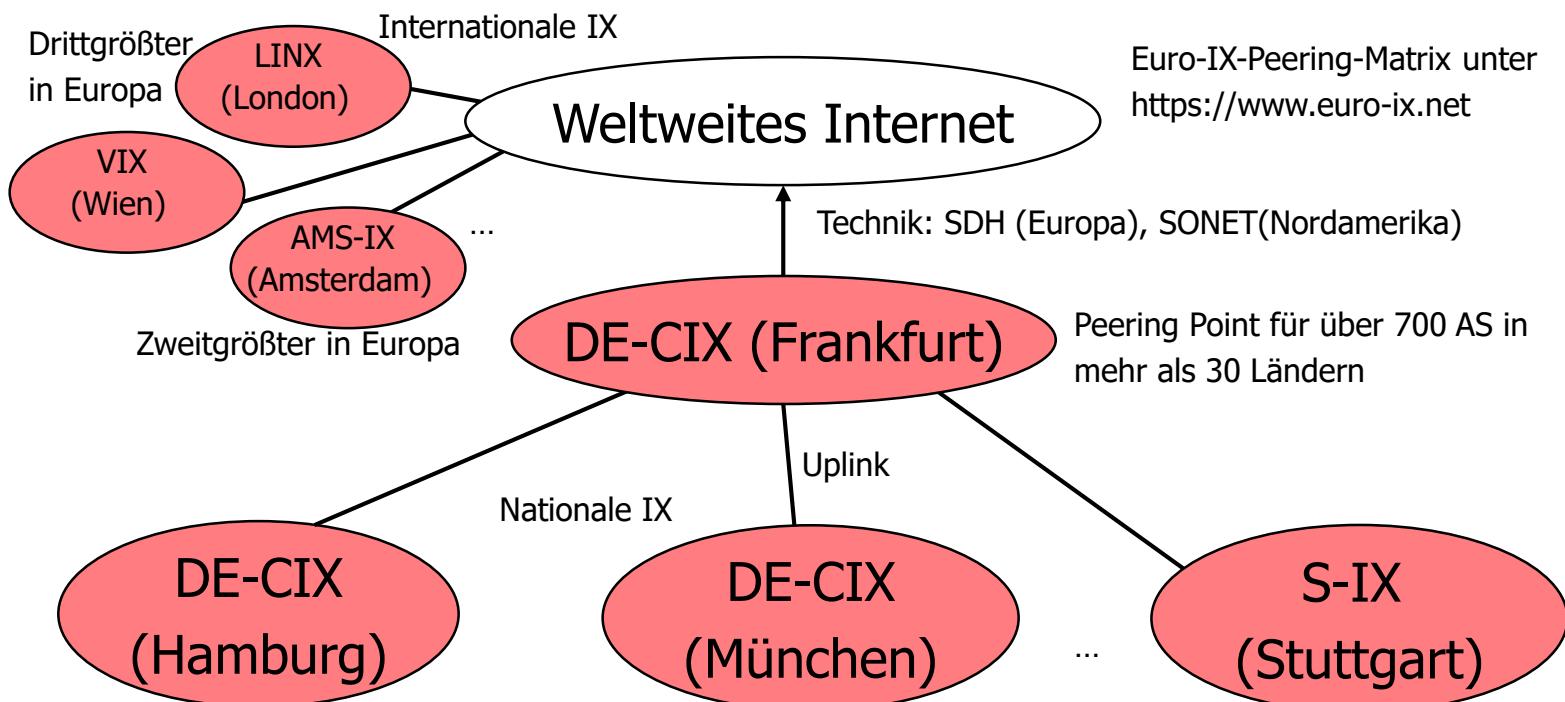


# Das Internet heute, Typische Verbindungen

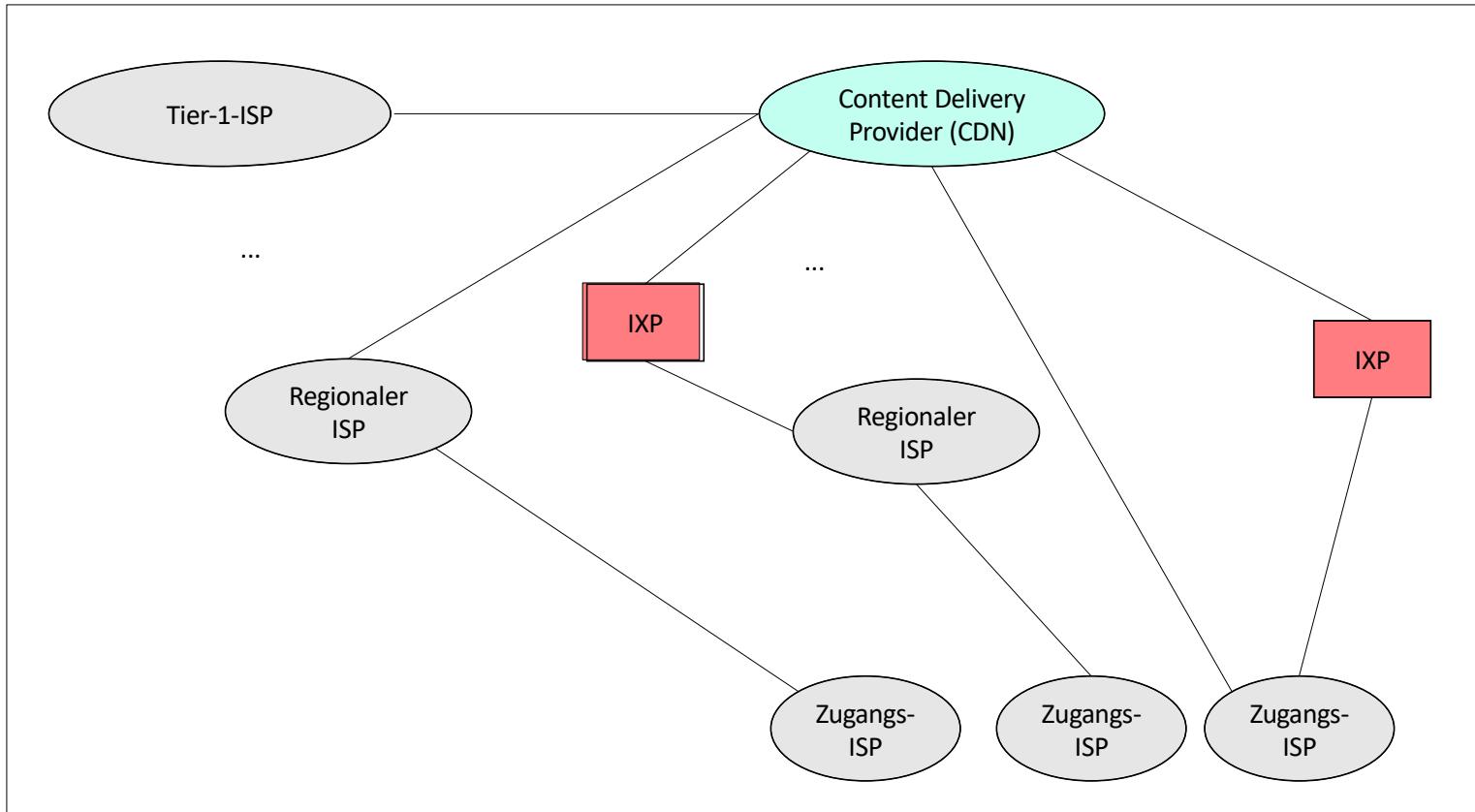


# Das Internet heute, physikalische Sicht

- DE-CIX Management GmbH: Betreibt DE-CIX international Internet Exchange („IX“) in Frankfurt und weiter Standorte
- Größter IX weltweit (<http://www.de-cix.de>)



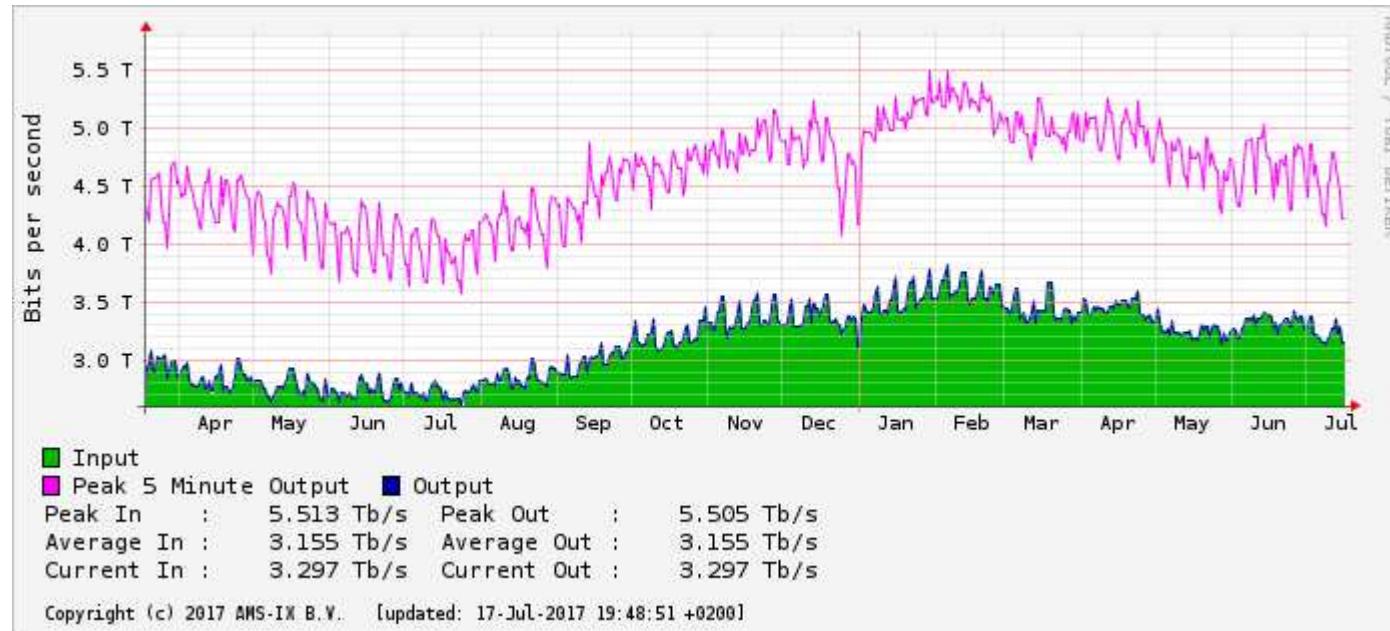
# Einschub: Das Internet heute Content Distribution Networks (CDN)



- Zusätzlich: CDNs von Unternehmen wie Akamai und Google
- Hunderttausende von Rechnern, die weltweit auf viele Rechenzentren verteilt sind
- Die Rechenzentren sind wiederum durch ein privates IP-Netz miteinander verbunden

# Das Internet heute, öffentliche Peering-Punkte (1)

- **AMS-IX** im July 2017, Quelle: <http://www.ams-ix.net/>



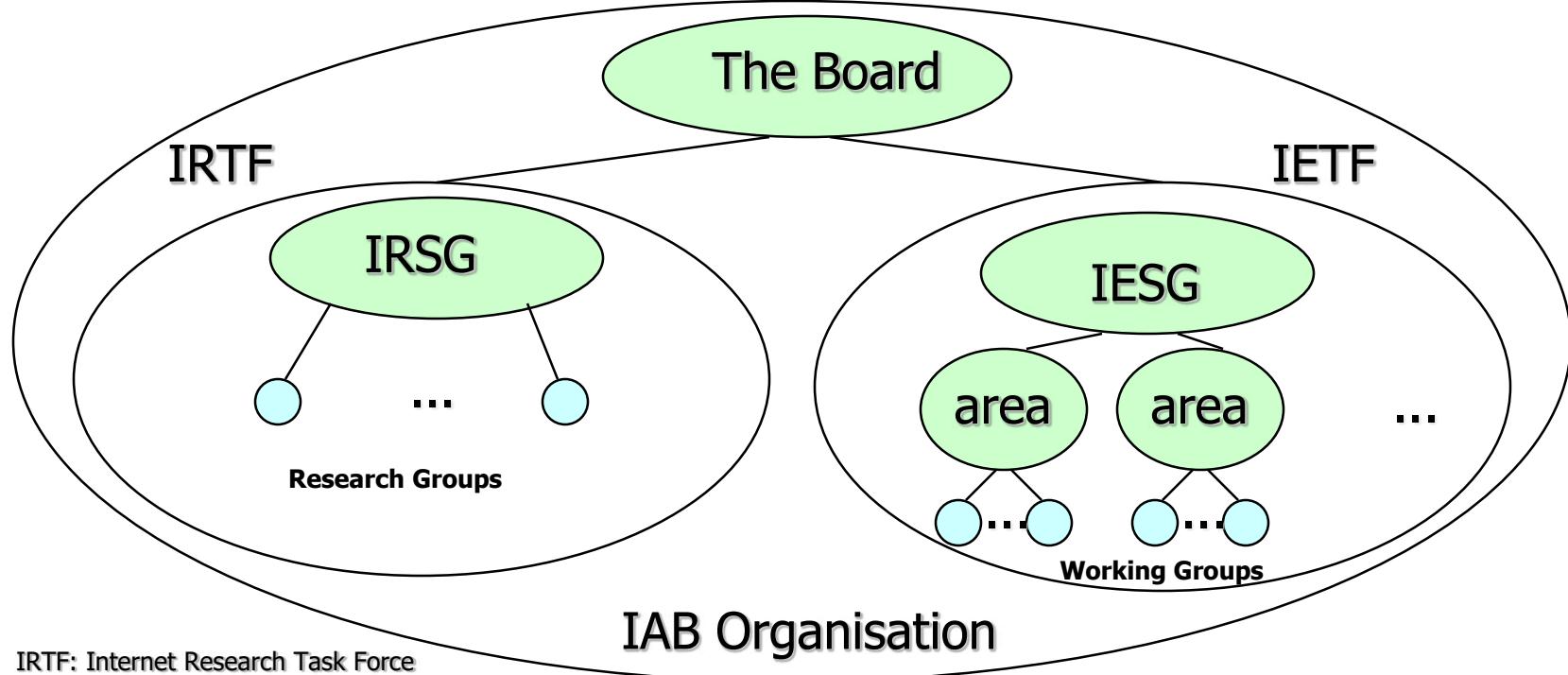
## Das Internet heute, öffentliche Peering-Punkte (2)

- **DE-CIX** im July 2017, mittlerweile der Größte
- Quelle: <http://www.de-cix.net/>, siehe auch: Statistiken zu Internet-Knoten in Wikipedia/Internet-Knoten



# Exkurs: Das Internet heute – IAB (1)

- Zuständig für die Weiterentwicklung des Internet ist das IAB (Internet Activity (heute: Architecture) Board), das bereits **1983** gegründet und **1989** umorganisiert wurde.



IRTF: Internet Research Task Force  
IETF: Internet Engineering Task Force  
IRSG: Internet Research Steering Group  
IESG: Internet Engineering Steering Group

## Exkurs:

### Das Internet heute – IAB (2)

---

- Das **IAB** (Board) bestimmt die Richtlinien der Politik
- Die **IETF** kümmert sich in verschiedenen Bereichen (areas) um kurz- und mittelfristige Probleme
- Die **IESG** koordiniert die IETF Working Groups
- Die **IRTF** ist ein Forschungsbereich, der die TCP/IP-Forschungsthemen koordiniert
- Die **IRSG** koordiniert die Forschungsaktivitäten der einzelnen Gruppen
- Die Working Groups setzen sich aus freiwilligen Mitarbeitern zusammen

## Exkurs:

### Das Internet heute, NIC und DENIC

---

- Das NIC (Network Information Center, gesprochen NICK), ist zuständig für
  - die Dokumentation und
  - die Verwaltung der umfangreichen Information über
    - Protokolle
    - Standards
    - Services, usw.
    - Siehe [www.nic.net](http://www.nic.net)
- In Deutschland ist die DENIC als nationale Vertretung eingerichtet (Frankfurt), siehe [www.denic.de](http://www.denic.de)

## Exkurs:

### Das Internet heute, Dokumentation, RFCs

---

- Die Dokumentation und die Standards werden als technische Reports gesammelt und heißen RFCs (Request for Comments)
- RFCs durchlaufen während ihrer Lebenszeit verschiedene Stati
  - Proposed Standard
  - Draft Standard
  - Internet Standard
- Manche RFCs werden auch nie zum Internet Standard
- RFCs sind frei verfügbar (z.B. unter [www.rfc-editor.org](http://www.rfc-editor.org))

# Internet Protocol: Hauptaufgaben (1)

---

- **Paketvermitteltes** (datagramm-orientiertes) und verbindungsloses Protokoll der Vermittlungsschicht
- IP dient der **Beförderung von Datagrammen** von einer Quelle zu einem Ziel evtl. über verschiedene Zwischenknoten (IP-Router)
- **Routing-Unterstützung** ist eine zentrale Aufgabe von IP (Routingprotokoll basiert auf diversen Protokollen)
- Datagramme werden während des Transports zerlegt und am Ziel wieder zusammengeführt, bevor sie der Schicht 4 übergeben werden = **Fragmentierung**

# Internet Protocol: Hauptaufgaben (2)

---

- IP stellt einen **ungesicherten verbindungslosen** Dienst zur Verfügung
  - Es existiert **keine Garantie der Paketauslieferung**
  - Die Übertragung erfolgt nach dem **Best-Effort-Prinzip**
    - „Auslieferung nach bestem Bemühen“
  - Jedes Paket des Datenstroms wird **isoliert** behandelt
  - Das IP-Paket wird in einem Rahmen der zugrundeliegenden Schicht 2 transportiert, für den Längenrestriktionen bestehen:
    - bei Ethernet ist eine Länge von 1500 Bytes üblich
    - Siehe hierzu auch: MTU = Maximum Transfer Unit

## 1. Überblick

- Organisation des Internets, Autonome Systeme (AS), Internet Exchange Points
- Aufgaben des Internet Protokolls

## 2. IPv4-Adressierung

- IPv4: Adressierung
- IPv4-Subnetting
- VLSM und CIDR

## 3. IPv4-PDU

- Aufbau und Felder

## 4. Fragmentierung

# Adressierung im Internet, IP-Adressen

---

- IP-Adressen sind **32 Bit** lange Adressen
  - **Tupel** aus (Netzwerknummer, Hostnummer)
- IP-Adressen von Quelle und Ziel werden **in allen** IP-Paketen mit übertragen
- Es gibt verschiedene Adressformate
  - man unterscheidet die Klassen A, B, C, D und E
- Schreibweise für IP-Adressen in gepunkteten Dezimalzahlen (dotted decimal), jeweils ein Byte als Dezimalzahl zwischen 0 und 255
  - Beispiel:
    - Hexformat: 0xC0290614
    - Dezimalzahl: 192.41.6.20

# Adressierung im Internet, IP-Adressformate (classful)

4 Byte, 32 Bit					
					
Klasse A	0	Netz		Host	1.0.0.0 bis 127.255.255.255
Klasse B	10	Netz		Host	128.0.0.0 bis 191.255.255.255
Klasse C	110	Netz		Host	192.0.0.0 bis 223.255.255.255
Klasse D	1110	Multicast-Adresse			224.0.0.0 bis 239.255.255.255
Klasse E	11110	Reservierte Adresse			240.0.0.0 bis 247.255.255.255

Alle Adressen der Form 127.xx.yz.z sind Loopback-Adressen!

## Adressierung im Internet, IP-Adressen

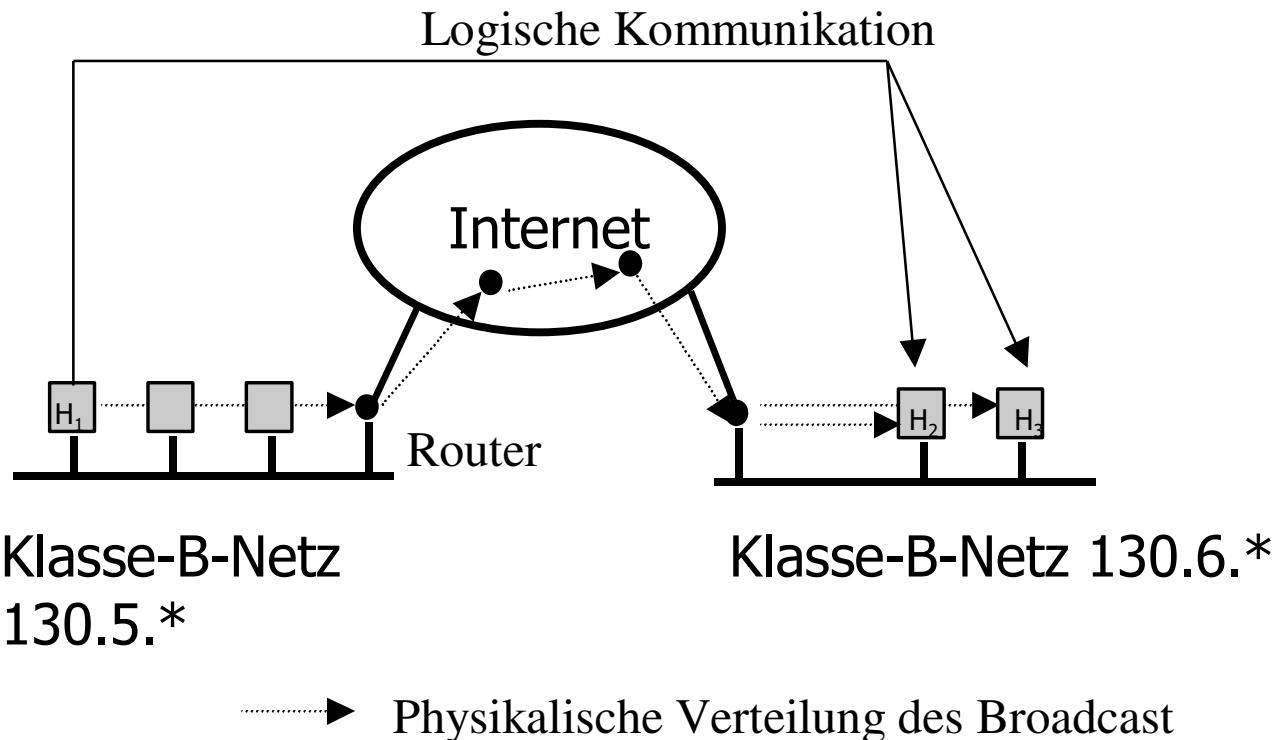
---

- Die niedrigste IP-Adresse ist **0.0.0.0**
  - Die Adresse **0.0.0.0** hat die besondere Bedeutung „ein bestimmtes Netz“ oder „ein bestimmter Host“
- Die höchste IP-Adresse ist **255.255.255.255**
  - Die Adresse **255.255.255.255** (-1) wird als Broadcast-Adresse verwendet
  - Limited Broadcast → im lokalen Netz, wird nicht geroutet
- Weiterer Broadcast-Typ:
  - Directed Broadcast → Broadcast an anderes Netzwerk

# Direkter Broadcast

---

Ziel-IP-Adresse:  
**130.6.255.255**



# Adressierung im Internet

---

## ■ IPv4-Adressen

- Klasse-A-Netze sind im ersten Byte nummeriert von 1 - 126
  - Netzwerk 0.\* und 127.\* haben besondere Bedeutung
- Klasse-B-Netze sind im ersten Byte nummeriert von 128 - 191
  - Der Adressbereich 169.254.0.0 - 169.254.255.255 ist reserviert für APIPA-Betrieb (Automatic Private IP-Addressing), ein Mechanismus ohne DHCP-Server (*kommt später*)
- Klasse-C-Netze sind im ersten Byte nummeriert von 192 - 223
- Private Adressen
  - Klasse A: 10.0.0.0 - 10.255.255.255, Klasse B: 172.16.0.0 - 172.31.255.255, Klasse C: 192.168.0.0 - 192.168.255.25

Klasse	Anzahl Netze	Max. Anzahl Hosts je Netz	Anteil am IP-Adressraum	Adressen insgesamt
A (/8)	126 ( $2^7 - 2$ )	16.777.214 ( $2^{24} - 2$ )	50 %	2.147.483.638 ( $2^{31}$ )
B (/16)	16.384 ( $2^{14}$ )	65.534 ( $2^{16} - 2$ )	25 %	1.073.741.824 ( $2^{30}$ )
C (/24)	2.097.152 ( $2^{21}$ )	254 ( $2^8 - 2$ )	12,5 %	536.870.912 ( $2^{29}$ )

## Adressierung im Internet, Subnetting

---

- Die Hostadresse kann zu besseren organisatorischen Gliederung für eine **Subnetz**-Bildung in zwei Teile zerlegt werden:
  - Teilnetznummer (Subnetz)
  - Hostnummer

Beispiel mit  
Klasse B

10	Netz	Subnetz	Host
----	------	---------	------

- Außerhalb des Teilnetzes ist die Aufgliederung nicht sichtbar
- IP-Router in einem Teilnetz berücksichtigen die Subnetzadresse
- **Netzwerkmaske** wird als Bitmaske verwendet, um die Bits der Subnetzwerksnummer zu identifizieren

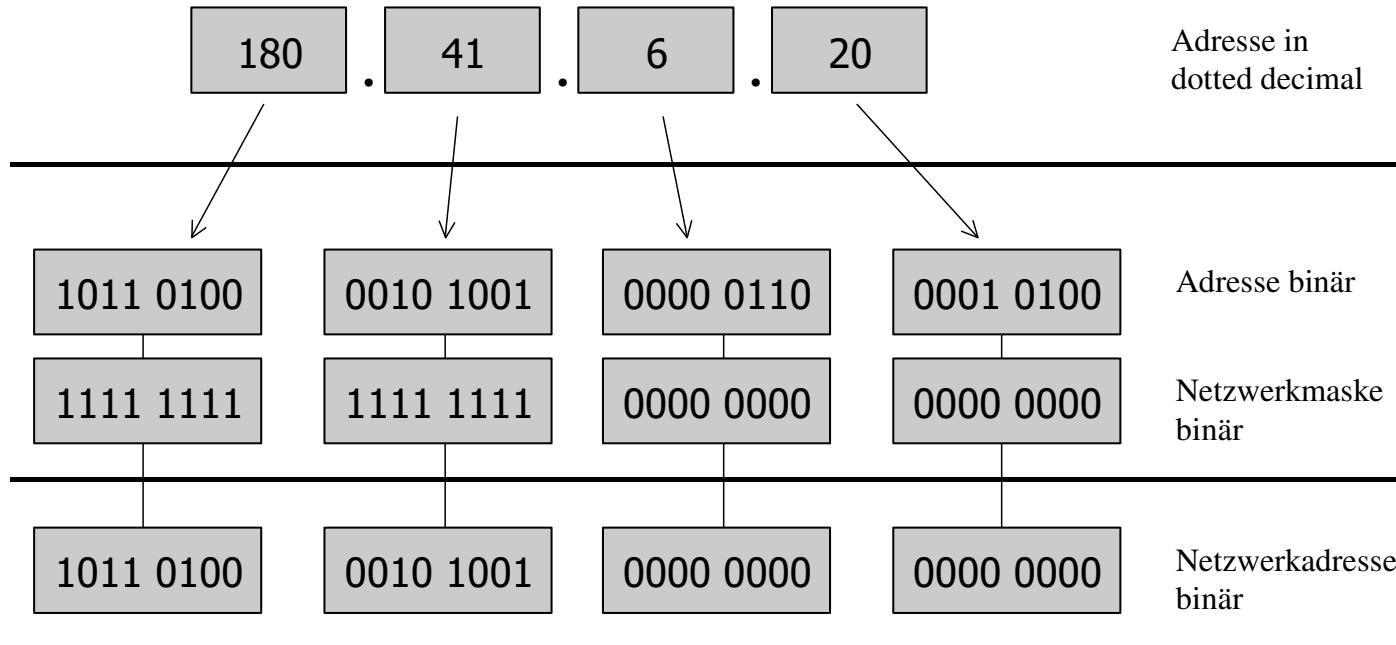
## Adressierung im Internet, Subnetting

---

- Der lokale Administrator besitzt alle Freiheiten zur Bildung von Subnetzen, ohne die Komplexität auf den Internet-Router zu übertragen
- Bei einer Klasse B-Adresse wäre folgende Struktur denkbar:
  - 3. Byte gibt die Organisationseinheit im Unternehmen an (Subnetz-Adresse)
  - 4. Byte erlaubt die Nummerierung der Geräte: (Stationsadresse)
    - Netzkomponenten (Switch, Hub, etc.): 1 - 9
    - Arbeitsplätze: 10 - 249
    - Server: 250 - 254

# Adressierung, Beispiel

- Hier handelt es sich um eine Klasse B-Adresse, warum?



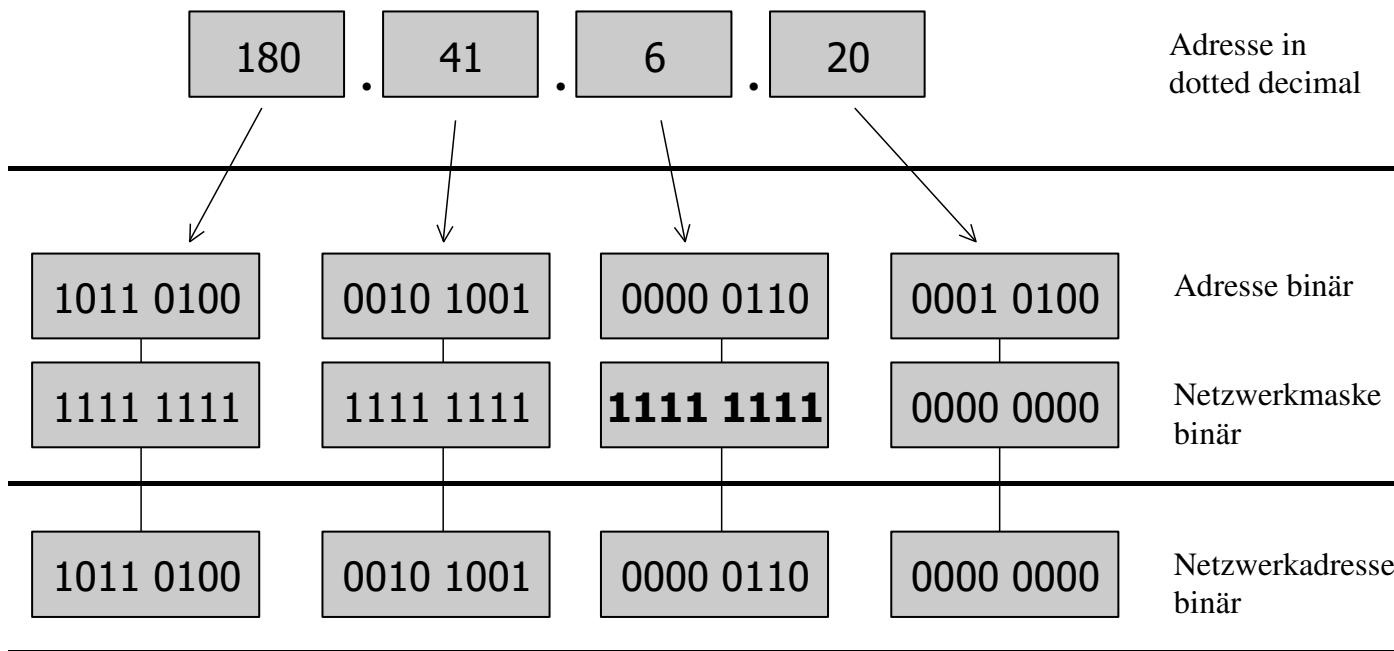
Logische Und-Verknüpfung der IP-Adresse mit der Netzmaske

Die Netzwerkadresse ist demnach: 180.41.0.0

Netzwerkmaske in dotted decimal: 255.255.0.0

# Adressierung, Beispiel mit Subnetzadressierung

---



Subnetz mit zwei Byte für Netzwerknummer und ein Byte für Subnetzwerknummer  
Die Netzmaske ist hier: 255.255.255.0

# CIDR und VLSM

---

- Problem:
  - Vergeudung von vielen IP-Adressen durch die Aufteilung des Adressraums in Klassen
  - Bei Klasse-A/B-Netzen werden viele Adressen gar nicht genutzt
- VLSM und CIDR helfen, die Adressproblematik etwas abzumildern
  - Beschreibt ein Verfahren zur effizienteren Nutzung des bestehenden 32-Bit-IPv4-Adressraumes, soweit noch verfügbar
  - **VLSM** = Variable Length Subnet Mask
  - **CIDR** = Classless InterDomain Routing (RFC 4632)
  - CIDR ist VLSM im öffentlichen Internet!

# CIDR und VLSM

---

- Konzept von VLSM/CIDR (Einführung ab 1993)
  - Restliche Klasse-C-Netze (ca. 2 Millionen) wurden in Blöcken variabler Länge vergeben → Vergabe durch ISP
  - Beispiel: Brauchte ein Standort 2000 Adressen wurden ihm 8 aufeinanderfolgende Klasse-C-Netze zugewiesen, man konnte also auf eine B-Adresse verzichten
- Weitere Verbesserung für das Routing durch Zuordnung der Klasse-C-Adressenbereiche zu Zonen, z. B.
  - Europa: 194.0.0.0 bis 195.255.255.255
  - Nordamerika: 198.0.0.0 bis 199.255.255.255
  - Europäischer Router erkennt anhand der Zieladresse, ob ein Paket in Europa bleibt oder weitergeleitet werden soll

## CIDR und VLSM: Netzwerkpräfix-Notation

---

- Netzwerkpräfix-Notation (NP-Notation) ermöglicht die Angabe der Netz-Id-Bits in der IP-Adresse
- Notationsbeispiel: 194.24.19.25/25
  - IP-Adresse binär:
    - **11000010.00011000.00010011.00011001** -> Klasse C
  - Die Präfixlänge (hier 25) gibt die Anzahl der fortlaufenden Einsen in der Netzwerkmaske an:
  - Netzwerkmaske der IP-Adresse:
    - **11111111. 11111111. 11111111. 10000000** = 255.255.255.128
  - Klasse A = /8
  - Klasse B = /16
  - Klasse C = /24

- Cambridge University benötigt 2000 (fast  $2^{11}$ ) **öffentliche** IP-Adressen
  - Klasse-C-Netz mit max. 254 ( $2^8 - 2$ ) Adressen reicht nicht aus
  - Alternative ist ein Klasse-B Netz mit 65534 ( $2^{16} - 2$ ) Adressen  
-> 63488 ( $2^{16} - 2^{11}$ ) öffentliche IP-Adressen werden nicht benötigt und somit verschwendet! (über 95% der bereitgestellten Adressen!!!)
  - Besser: Nutzung mehrerer zusammenhängender Klasse-C Netze
    - Für den Hostanteil der Adresse werden **zusätzlich** 3 Bit benötigt ( $2^{11}$  Adressen)
    - Cambridge University wird folgender Adressbereich zugewiesen:  
 $194.24.0.0/21 \rightarrow 194.24.0.0 \text{ bis } 194.24.7.255$  (Netzwerkmaske 255.255.248.0)  
oder  
**11000010.00011000.00000000.00000000**      bis  
**11000010.00011000.00000111.11111111**      mit  
**11111111.11111111.11111000.00000000**      als Netzwerkmaske

- Nach dem gleichen Verfahren werden auch den Universitäten Oxford und Edinburgh mehrere Klasse-C Netze zugewiesen
  - Oxford benötigt 4000 (fast  $2^{12}$ ) öffentliche IP-Adressen
  - Edinburgh benötigt 1000 (fast  $2^{10}$ ) öffentliche IP-Adressen
- Folgende Adressbereiche werden zugewiesen:
  - Cambridge: 194.24.0.0/21                  194.24.0.0                  bis          194.24.7.255
  - Edinburgh: 194.24.8.0/22                  194.24.8.0                  bis          194.24.11.255
  - Verfügbar: 194.24.12.0/22                  194.24.12.0                  bis          194.24.12.255
  - Oxford: 194.24.16.0/20                  194.24.16.0                  bis          194.24.31.255

- Ein Standard IPv4-Router kann die mittels VLSM zusammengefassten Klasse-C-Netze nicht erkennen
- Das Routing muss also um CIDR erweitert werden

### Routingtabelle ohne CIDR

194.24.0.0	-> Cambridge
194.24.1.0	-> Cambridge
...	
194.24.7.255	-> Cambridge
194.24.8.0	-> Edinburgh
...	
194.24.11.255	-> Edinburgh
194.24.16.0	-> Oxford
...	
194.24.31.255	-> Oxford

### Routingtabelle mit CIDR

194.24.0.0/21	-> Cambridge
194.24.8.0/22	-> Edinburgh
194.24.16.0/20	-> Oxford

Nur 3 statt 28 Einträge!!!

# CIDR und VLSM: Beispiel LRZ - Hochschule München - Fakultät 07 (1)

---

- Das Leibnitz-Rechenzentrum (LRZ) organisiert den **privaten** IP-Adressbereich 10.0.0.0/8 und ordnet der Informatik an der Hochschule München den Adressbereich 10.28.0.0/16 zu

10.20.0.0      -> Lothstraße 34

10.21.0.0      -> Lothstraße 21

10.22.0.0      -> Karlstraße 6

10.23.0.0      -> Infanteriestraße 13/14

...

10.26.0.0      -> Pasing

...

10.28.0.0      -> Hessstraße (Informatik)

...

## CIDR und VLSM:

### Beispiel LRZ - Hochschule München - Fakultät 07 (2)

---

- Die Fakultät untergliedert den Adressbereich derzeit weiter in vier /18-Subnetze
- Beispiel: Netz 0
  - 10.28.0.0/18 mit 18 Bit Netzwerkanteil und  $2^{14} - 2$  Hostadressen
  - Binär: 0000 1010 0001 1100 **00**xx xxxx xxxx xxxx
- Beispiel: Netz 1
  - 10.28.64.0/18 mit 18 Bit Netzwerkanteil und  $2^{14} - 2$  Hostadressen
  - Binär: 0000 1010 0001 1100 **01**xx xxxx xxxx xxxx
- Subnetze der Fakultät:
  - 10.28.0.0/18 → Netz 0: Wird im Münchener Hochschulnetz geroutet
  - 10.28.64.0/18 → Netz 1: Wird innerhalb der Hochschule München geroutet
  - 10.28.128.0/18 → Netz 2: wie Netz 1, aber DHCP-Adressvergabe
  - 10.28.192.0/18 → Netz 3: kein Routing

## CIDR und VLSM, Übung 1

---

- Welche IP-Adressen repräsentiert die CIDR-Adresse 180.41.214.192/28 (Netzwerkpräfix-Notation)?
- Wie lautet die directed Broadcast-Adresse?
- Lösung:
  - Insgesamt 4 Bit für Hosts
  - Adressbereich von 180.41.214.192 bis 180.41.214.207
  - Also von 1011 0100 . 0010 1001 . 1101 0110 . 1100 **0000**  
bis 1011 0100 . 0010 1001 . 1101 0110 . 1100 **1111**
  - Das sind 16 IP-Adressen, 14 davon sind nutzbar
  - Directed Broadcast-Adresse = 180.41.214.207
    - 1011 0100 . 0010 1001 . 1101 0110 . 1100 **1111**

## CIDR und VLSM, Übung 2 (1)

---

- Ein Unternehmen besitzt die öffentliche Klasse-C-Adresse 193.1.1.0 und hat 8 Abteilungen
- Jede Abteilung hat 25 Rechner
- Jede Abteilung soll einen eigenen Adressbereich erhalten
- Teilen Sie das Klasse-C-Netzwerk in 8 Subnetze auf und verwenden Sie dabei CIDR-Adressen!
  - Hinweis: Für 8 Subnetze braucht man 3 Bit

## CIDR und VLSM, Übung 2 (2)

---

- Ergänzen Sie dazu die CIDR-Formate (Werte x1 – x8 und y1 – y8)!

Netz	Adresse in Bitdarstellung	CIDR-Format	Anzahl Rechner
Basisnetz	11000001.00000001.00000001.00000000	193.1.1.0/24	254
Subnetz 0	11000001.00000001.00000001. <b>00000000</b>	193.1.1.x1/y1	30
Subnetz 1	11000001.00000001.00000001. <b>00100000</b>	193.1.1.x2/y2	30
Subnetz 2	11000001.00000001.00000001. <b>01000000</b>	193.1.1.x3/y3	30
Subnetz 3	11000001.00000001.00000001. <b>01100000</b>	193.1.1.x4/y4	30
Subnetz 4	11000001.00000001.00000001. <b>10000000</b>	193.1.1.x5/y5	30
Subnetz 5	11000001.00000001.00000001. <b>10100000</b>	193.1.1.x6/y6	30
Subnetz 6	11000001.00000001.00000001. <b>11000000</b>	193.1.1.x7/y7	30
Subnetz 7	11000001.00000001.00000001. <b>11100000</b>	193.1.1.x8/y8	30

# CIDR und VLSM, Übung 2 - Lösung

---

- Kalkulationshilfe: <http://www.vlsm-calc.net/>, letzter Zugriff am 10.01.2018
- Netzwerkmaske: 255.255.255.224

Netz	Adresse in Bitdarstellung	CIDR-Format	Anzahl Rechner
Basisnetz	11000001.00000001.00000001.00000000	193.1.1.0/24	254
Subnetz 0	11000001.00000001.00000001. <b>00000000</b>	193.1.1.0/27	30
Subnetz 1	11000001.00000001.00000001. <b>00100000</b>	193.1.1.32/27	30
Subnetz 2	11000001.00000001.00000001. <b>01000000</b>	193.1.1.64/27	30
Subnetz 3	11000001.00000001.00000001. <b>01100000</b>	193.1.1.96/27	30
Subnetz 4	11000001.00000001.00000001. <b>10000000</b>	193.1.1.128/27	30
Subnetz 5	11000001.00000001.00000001. <b>10100000</b>	193.1.1.160/27	30
Subnetz 6	11000001.00000001.00000001. <b>11000000</b>	193.1.1.192/27	30
Subnetz 7	11000001.00000001.00000001. <b>11100000</b>	193.1.1.224/27	30

## Einschub:

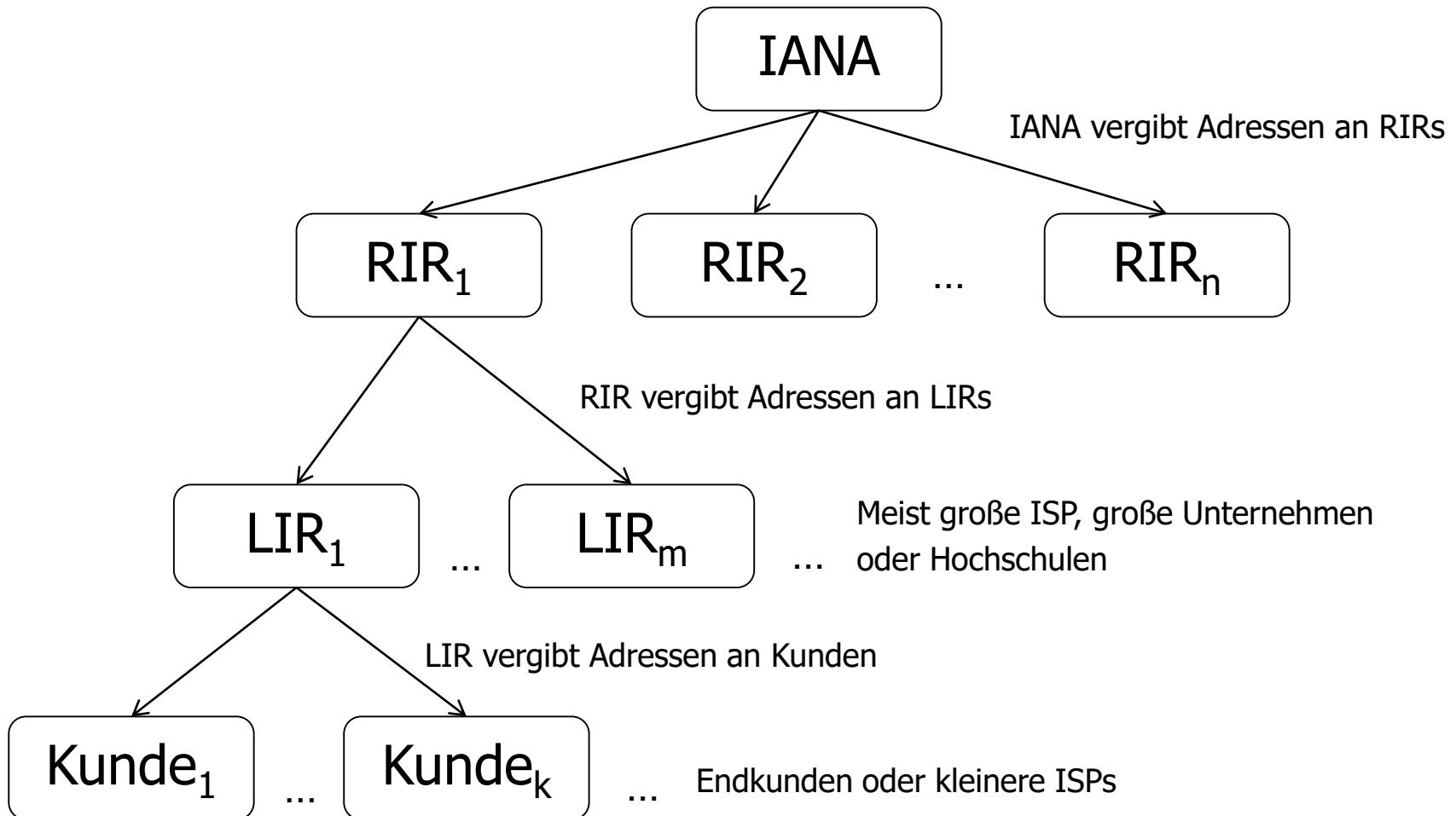
### IPv4-Adressvergabe: Registries (1)

---

- **Internet Registries** (IR) sind verantwortlich für die Vergabe des IP-Adressraums
- **IANA** (Internet Assigned Number Authority) ist die zentrale Organisation
- Regional Internet Registries (**RIR**) bekommen Adressraum von IANA und bedienen große geographische Regionen
- Local Internet Registries (**LIR**) bekommen Adressraum von den RIRs und vergeben ihn an die Kunden
  - LIRs sind meist ISPs

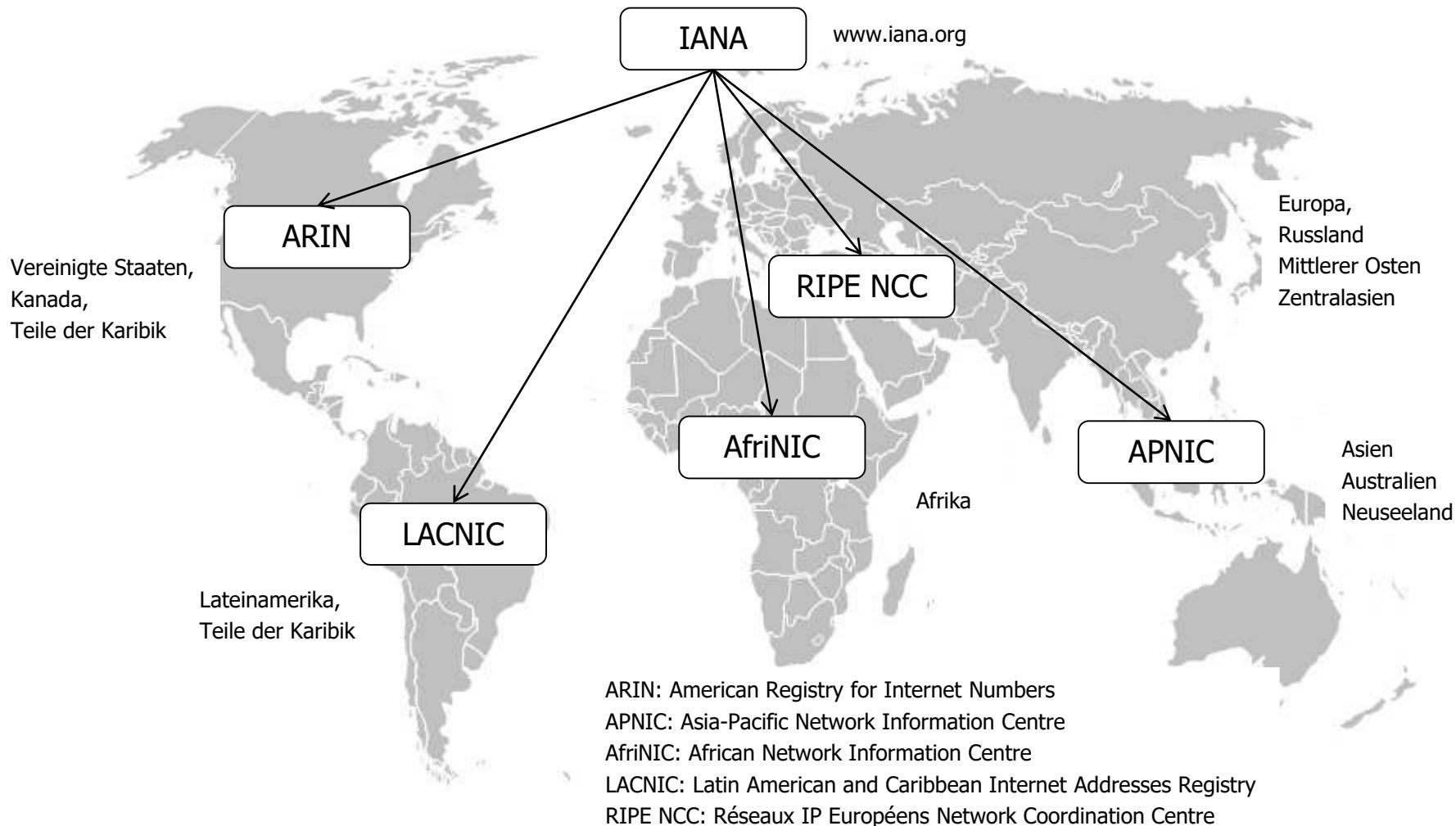
## Einschub:

### IPv4-Adressvergabe: Registries (2)



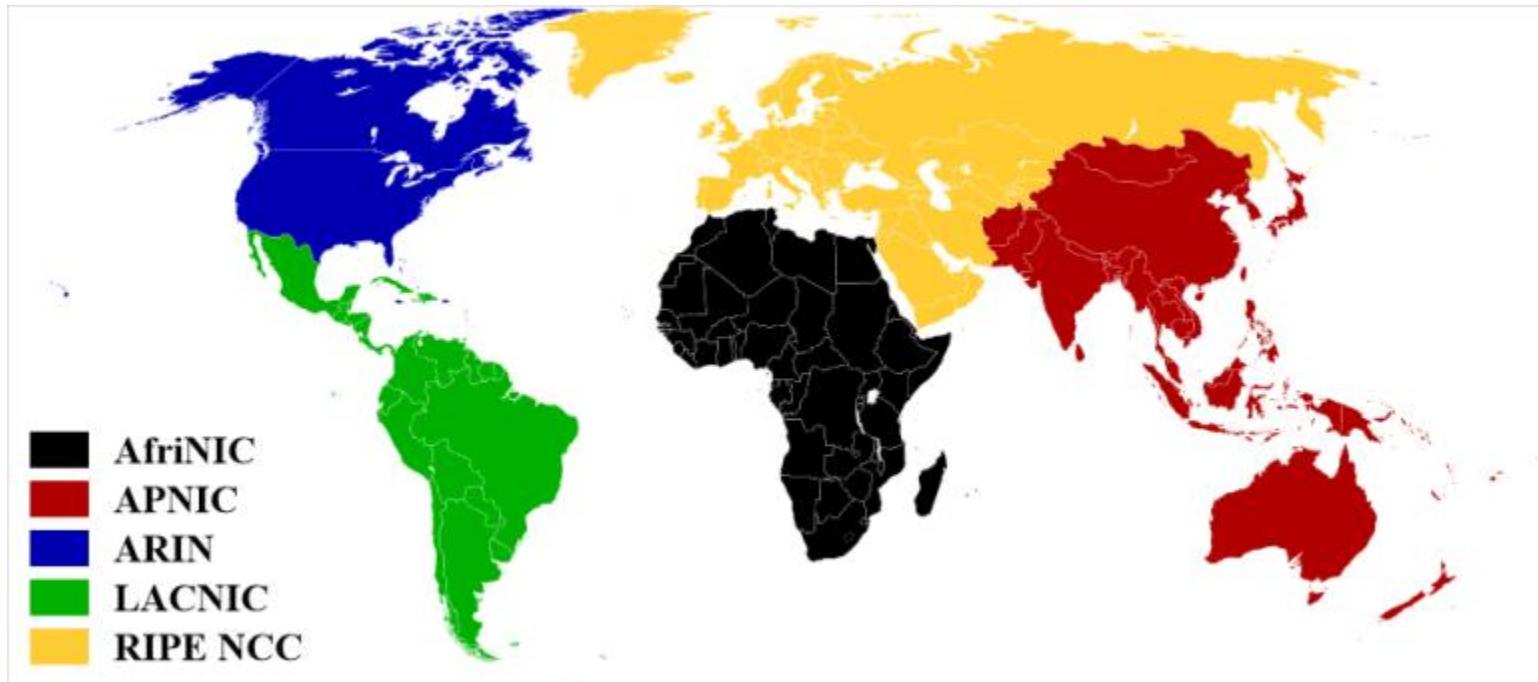
# Einschub: IPv4-Adressvergabe: Registries (3)

---



# Einschub: IPv4-Adressvergabe: Registries (4)

---



ARIN: American Registry for Internet Numbers

APNIC: Asia-Pacific Network Information Centre

AfriNIC: African Network Information Centre

LACNIC: Latin American and Caribbean Internet Addresses Registry

RIPE NCC: Réseaux IP Européens Network Coordination Centre

---

Quelle: Wikipedia

## Einschub:

### IPv4-Adressvergabe: Registries (5)

---

- Weitere Aufgaben von IANA und RIRs
  - Betrieb von DNS-Root-Servern
  - Vergabe von AS-Nummern
  - Verwaltung der Domain-Namen
  - Protokollregistrierung (well-known Ports)
  - ...

## Einschub:

### IPv4-Adressvergabe: Aktueller Stand

---

- Januar 2011: APNIC erhält von IANA die letzten zwei freien IPv4-Adressräume
- April 2011: APNIC teilt die letzten freien Adressen im Raum Südostasien zu
- September 2012: RIPE NCC teilt die letzten Adressen LIRs zu
  
- Heute gibt es fast nur noch Adress-Zuteilungen an die LIRs mit minimaler Zuteilungsgröße (/24-Netze), solange sie verfügbar sind

## 1. Überblick

- Organisation des Internets, Autonome Systeme (AS), Internet Exchange Points
- Aufgaben des Internet Protokolls

## 2. IPv4-Adressierung

- IPv4: Adressierung
- IPv4-Subnetting
- VLSM und CIDR

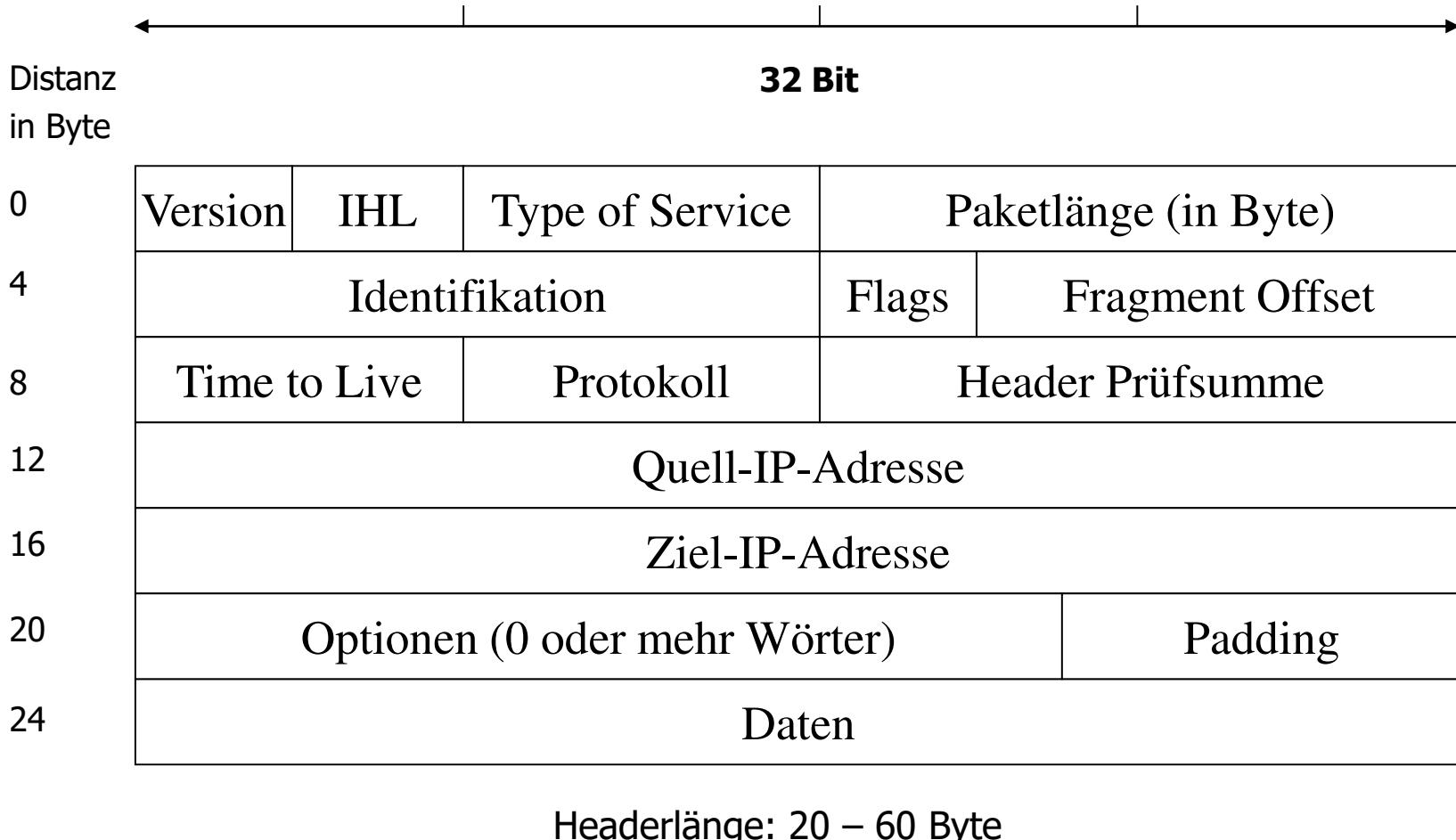
## 3. IPv4-PDU

- Aufbau und Felder

## 4. Fragmentierung

# Protokollheader IPv4 (1)

---



## Protokollheader IPv4 (2)

---

Version	IHL	Type of Service	Paketlänge (in Byte)
---------	-----	-----------------	----------------------

- **Version:** Spezifiziert die genutzte IP-Version; z.Zt. Wechsel von IPv4 auf IP Next Generation (IPv6)
- **IHL:** Gibt die Länge des Paket-Headers an, gemessen in 32-Bit-Worten; ist aufgrund der variablen Länge des Optionsfeldes nötig (mind. 5 → keine Option, max. 15 Worte → 60 Byte)
- **Type of Service:** Dieses 8-Bit-Feld ist wiederum seit 2001 wie folgt aufgeteilt:
  - Bits 0-5: DSCP (Differentiated Services Code Point)
  - Bits 6-7: ECN (Explicit Congestion Notification – IP-Flusskontrolle)
  - Wurde früher in IPv4 anders genutzt: ToS zur Vereinbarung von QoS-Eigenschaften, selten genutzt
- **Paketlänge:** Gesamtlänge des Datenpakets inkl. Header; gemessen in Byte, max. 65.535 Byte

# Einschub: DSCP

Siehe auch: <http://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-packet-marking/10103-dscpvalues.html>, letzter Zugriff am 10.01.18

Version	IHL	Type of Service	Paketlänge (in Byte)
---------	-----	-----------------	----------------------

- **DSCP** (Differentiated Services Code Point) wird heute anstelle von Type of Service genutzt
  - 6 Bits kodieren DCSPs zwischen 0 und 63
  - Damit ist eine Klassifizierung der IP-Pakete möglich
  - Man spricht auch von DiffServe (Differenciated Services)
  - Nutzung durch Internet Service Provider (ISP), um ihren Kunden bestimmte Service-Qualitätsmerkmale bereitzustellen:
    - Assured Forwarding von IP-Paketen
    - Express Forwarding von IP-Paketen
    - Best Effort (klassisch)
    - ...
  - DSCP-Codes sind in RFC genormt: RFC 2474, 2475, ...
  - ISP-Router müssen das Feld unterstützen

<b>DS5</b>	<b>DS4</b>	<b>DS3</b>	<b>DS2</b>	<b>DS1</b>	<b>DS0</b>	ECN	ECN
------------	------------	------------	------------	------------	------------	-----	-----

## Einschub: ECN

---

Version	IHL	Type of Service	Paketlänge (in Byte)
---------	-----	-----------------	----------------------

- **ECN** (Explicit Congestion Notification), RFC 3168
  - Bits 6-7 des alten ToS-Feldes sind heute 2 ECN-Bits
    - Bei Bitkombinationen 01 und 10 → ECN ist im Router eingeschaltet
  - Dient der IP-Staukontrolle, Router kann damit einen drohenden Stau frühzeitiger anzeigen als bei bisherigen, rein TCP-basierten Verfahren
  - Wenn die Routerwarteschlange einen bestimmten Schwellwert erreicht hat, kann dies durch Setzen der ECN-Bits über die Router zum Endsystem gemeldet werden
  - Algorithmus im Router wird auch als *Weighted Random Early Detection (WRED)* bezeichnet

DS5	DS4	DS3	DS2	DS1	DS0	<b>ECN</b>	<b>ECN</b>
-----	-----	-----	-----	-----	-----	------------	------------

## Einschub: ECN: Zusammenspiel mit TCP(1)

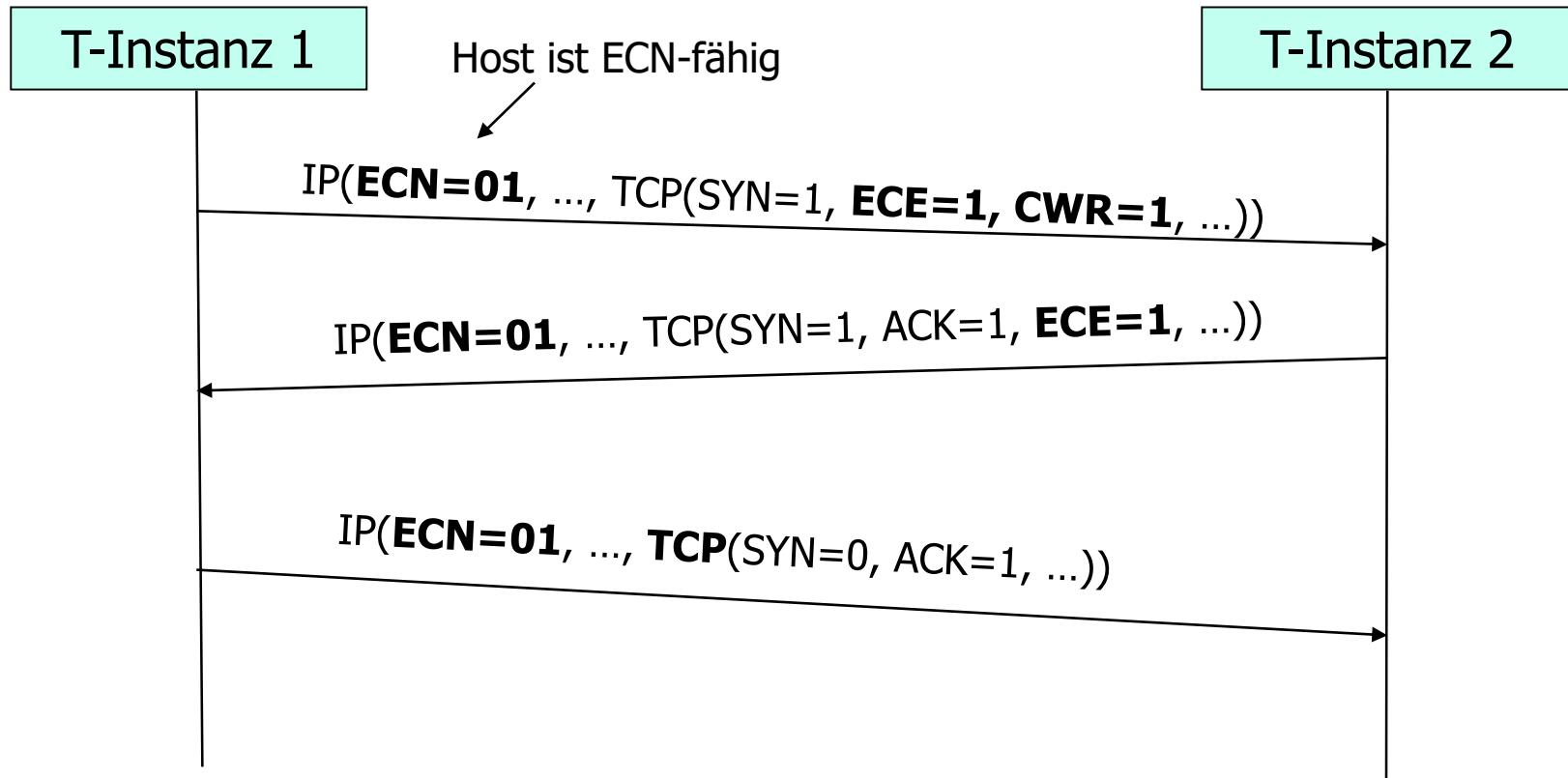
---

- ECN ist ein Lösungsansatz für die Staukontrolle, der die Schichten 3 und 4 mit einbezieht
- Zusätzliche TCP-Flags
  - ECE-Flag (Explicit Congestion Notification)
  - CWR-Flag (Congestion Window Reduced)
- Zu Auffrischung: Flags im TCP-Header

URG	ACK	PSH	RST	SYN	FIN	<b>ECE</b>	<b>CWR</b>
-----	-----	-----	-----	-----	-----	------------	------------

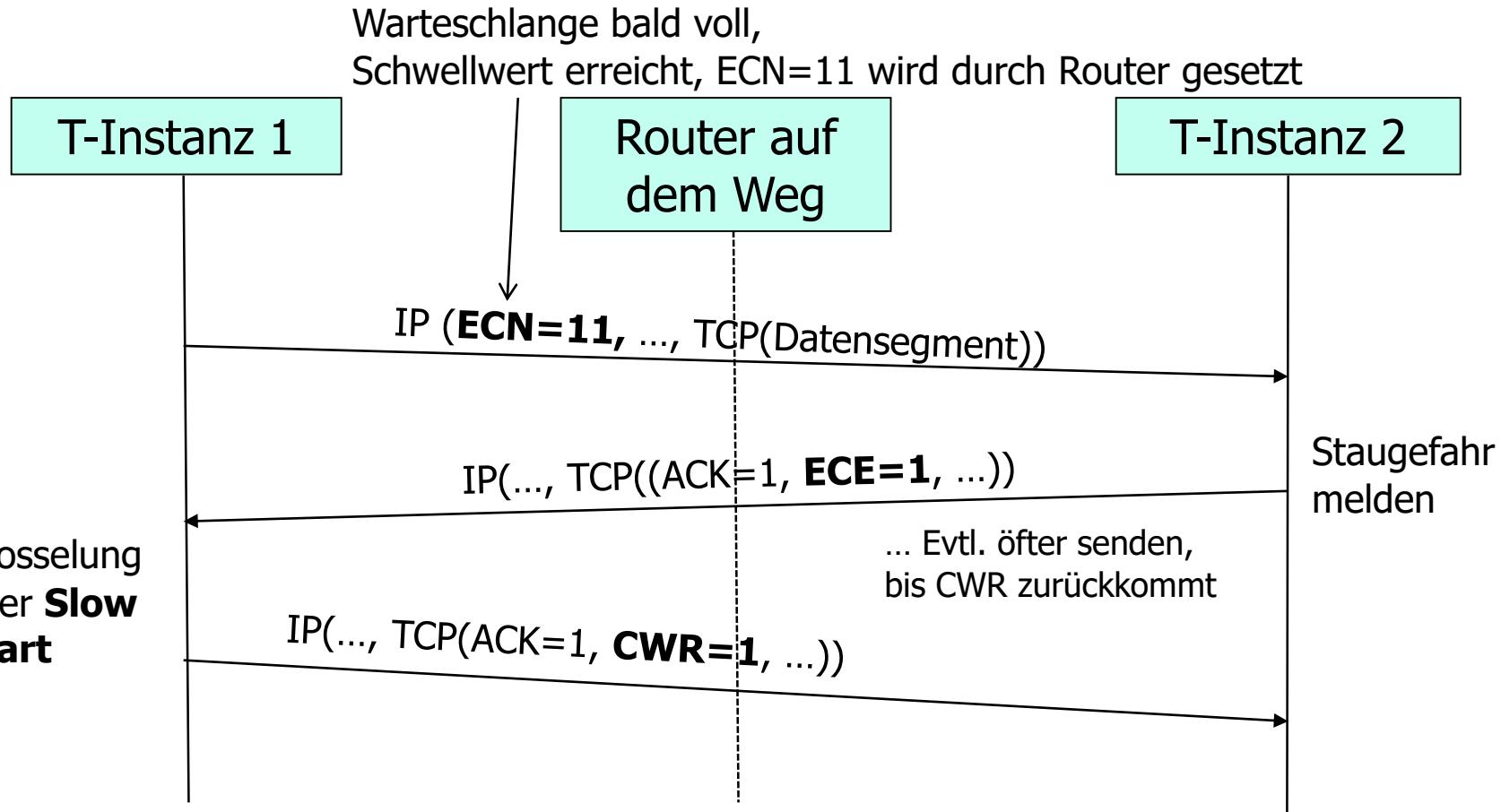
## Einschub: ECN: Zusammenspiel mit TCP(2)

- Verbindungsauftbau mit Signalisierung der ECN-Fähigkeit zwischen den Endsystemen



## Einschub: ECN: Zusammenspiel mit TCP(3)

- Stausignalisierung durch einen Router im IP-Header



## Protokollheader IPv4 (3)

---

Identifikation	Flags	Fragment Offset
----------------	-------	-----------------

- **Identifikation:** Alle Fragmente eines Datagramms erhalten hier den gleichen Wert
- **Flags:** (3 Bit) Dient der Kontrolle der Fragmentierung
  - Drei Flags, erstes unbenutzt: 0|DF|MF
  - DF=1 → Fragmentierung ist nicht erlaubt
  - More Fragments=0 → letztes Fragment des Ursprungspakets; MF=1 weitere Fragmente des Ursprungspakets folgen
- **Fragment Offset:** Dient der korrekten Herstellung der Ursprungssequenz, da Pakete das Ziel in unterschiedlicher Reihenfolge erreichen können (gemessen in 8-Byte-Worten)
  - Dient der Ermittlung der relativen Lage des Fragments im Datagram
  - 13 Bit

## Protokollheader IPv4 (4)

---

Time to Live	Protokoll	Header Prüfsumme
--------------	-----------	------------------

- **Time to live (TTL):**
  - Es dient dazu, alte Pakete irgendwann vom Netz zu nehmen
  - War früher gedacht als Zeitwert in Sekunden (max. 255 s)
  - Wird aber heute als Hop-Count genutzt, jeder Router subtrahiert vor dem WeiterSenden 1 vom TTL-Wert
  - Wenn TTL=0, dann verwerfen und ICMP-Nachricht zum Quellrechner senden
- **Protokoll:** Definiert das darüber liegende Protokoll, an das IPv4 die Daten des Pakets weiterreicht (6=TCP, 17=UDP, 89=OSPF,...)
- **Header-Prüfsumme:** Header-Absicherung
  - 16-Bit-Wörter addieren und Einerkomplement der Summe bilden
  - Prüft also **nur** den Header, Daten müssen in höheren Protokollen geprüft werden!
  - Muss für jede Teilstrecke neu berechnet werden, warum?

## Protokollheader IPv4 (5)

---

Quell-IP-Adresse
Ziel-IP-Adresse

- **Quell-IP-Adresse und Ziel-IP-Adresse:**
  - Jeweils 32 Bits
  - Identifikation der einzelnen Endsysteme (Hosts)
  - Hier stehen die IP-Adressen von Sender- und Empfängerhost

## Protokollheader IPv4 (6)

---



- **Optionen:** Zusätzliche, optionale Angaben:
  - *Loose Source Routing* → Möglichkeit, den Weg eines Pakets durch das Internet partiell vorzugeben; max. 9 Router; RFC 791 (\*)
  - *Strict Source Routing* → die Pakete müssen die Pfadvorgabe einhalten (max. 9 Knoten in der Route); max. 9 Router **fix**; RFC 791 (\*)
  - *Record Routing* → Jeder durchlaufene Router trägt seine Adresse ein
  - ...
- **Padding:** Wenn eine Option genutzt wird, ist das Datagramm bis zur nächsten 32-Bit-Grenze mit Nullen aufzufüllen
- **Daten:** Die Nutzdaten der höheren Schicht

(\*) Sicherheitsproblem, viele Router verwerfen diese Option

# Überblick

---

## 1. Überblick

- Organisation des Internets, Autonome Systeme (AS), Internet Exchange Points
- Aufgaben des Internet Protokolls

## 2. IPv4-Adressierung

- IPv4: Adressierung
- IPv4-Subnetting
- VLSM und CIDR

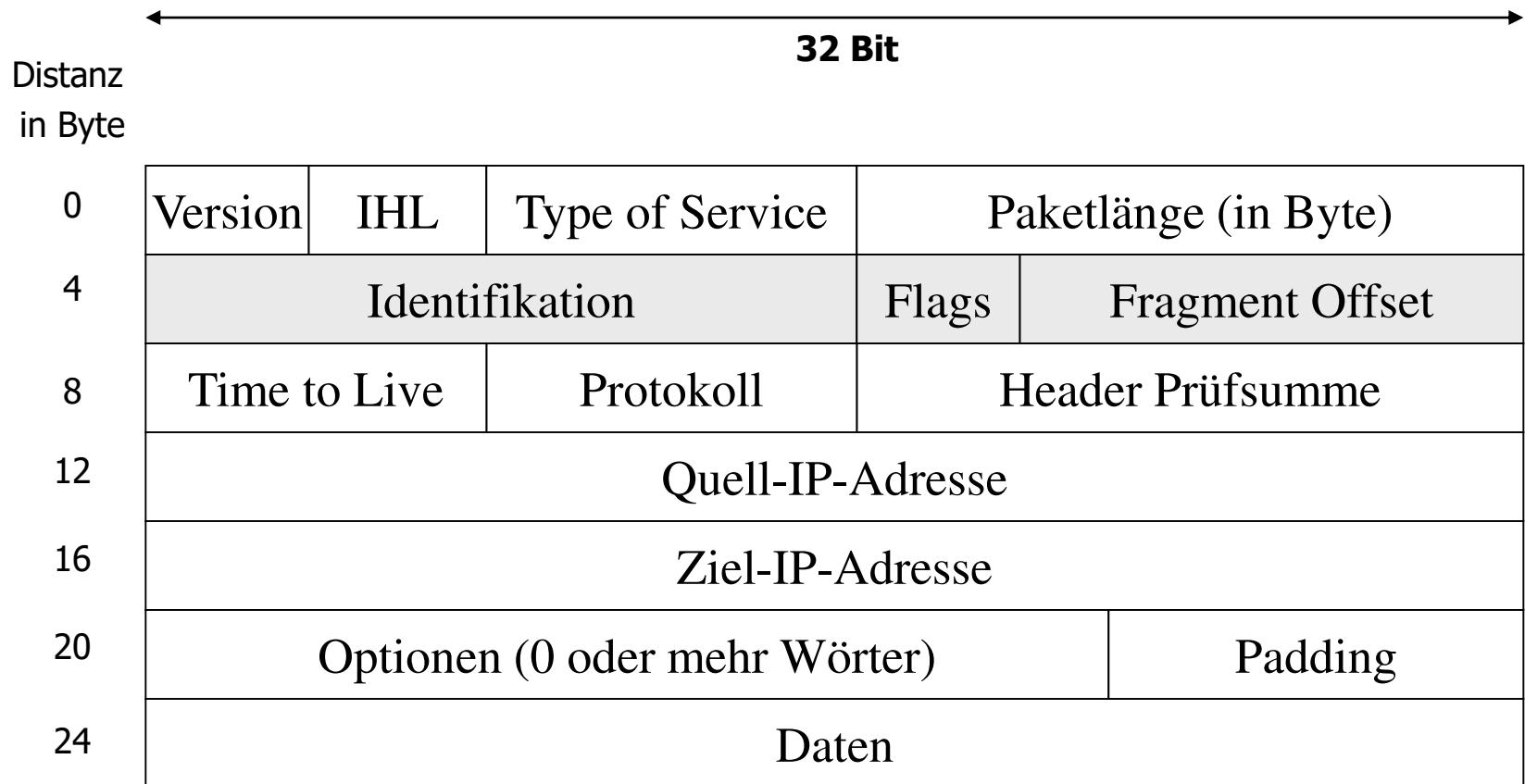
## 3. IPv4-PDU

- Aufbau und Felder

## 4. Fragmentierung

# Protokollheader

---



## Fragmentierung, Ablauf - grob

---

- Wenn ein IP-Paket von einem Netzknoten zum anderen weitergeleitet wird, muss es
    - evtl. verschiedene physikalische Netze durchqueren
    - in unterschiedlich zulässige Paketgrößen aufgeteilt werden
  - Daher besteht die Notwendigkeit, IP-Datagramme zu zerlegen und am Ziel wieder zusammenzusetzen
    - Fragmentierung und Defragmentierung
    - Alle Router müssen Fragmente der Größe **576 Byte** oder kleiner akzeptieren (TCP-Segmentgröße bei Standard-Headerlängen 536 Byte)
  - Sobald eine Fragmentierung einsetzt, laufen in einem Knoten mehrere Schritte ab
-

## Fragmentierung, Ablauf im IPv4-Router (1)

---

- Das **DF**-Flag wird überprüft, um festzustellen, ob eine Fragmentierung erlaubt ist. Ist das Bit auf „1“ gesetzt und es wäre eine Fragmentierung notwendig, wird das Paket verworfen
- Ansonsten wird entspr. der zulässigen Paketgröße das Datenfeld des Ur-Pakets in mehrere Teil zerlegt
- Alle neu entstandenen Pakete weisen - mit Ausnahme des letzten Pakets - eine Länge mit einem **Vielfachen von 8 Byte** auf
- Alle Datenteile werden in neu erzeugte IP-Pakete eingebettet. Die Header dieser Pakete sind Kopien des Ursprungskopfes mit einigen Modifikationen

## Fragmentierung, Ablauf im IPv4-Router (2)

---

- Header-Modifikation bei der Fragmentierung:
  - Das **MF**-Flag wird in allen Fragmenten mit Ausnahme des letzten auf „1“ gesetzt
  - Das **Fragment-Offset**-Feld enthält Angaben darüber, wo das Datenfeld in Relation zum Beginn des nicht fragmentierten Ur-Pakets platziert ist
  - Enthält das Ur-Paket Optionen, wird abhängig vom Type-Byte entschieden, ob die Option in jedes Paketfragment aufgenommen wird (z.B. Protokollierung der Route)
  - Die **Headerlänge** (IHL) und die Paketlänge sind jeweils **neu** zu bestimmen
  - Die **Header-Prüfsumme** wird neu berechnet

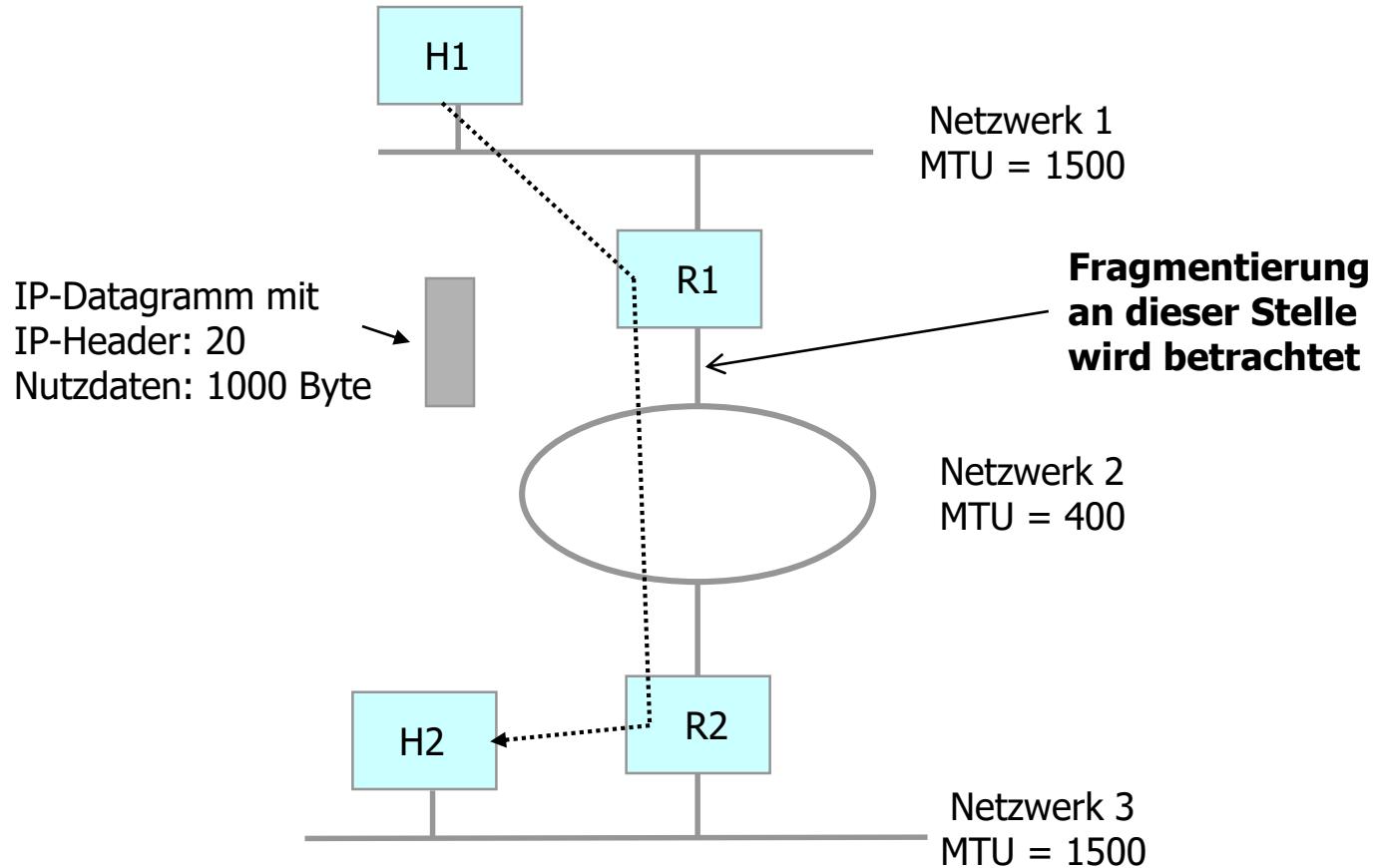
# Fragmentierung, Ablauf im Zielsystem

---

- Ablauf der Defragmentierung:
  - Die Zielstation setzt die Fragmente eines Datagramms wieder zusammen
  - Die **Zusammengehörigkeit** entnimmt sie dem **Identifikationsfeld**
  - Die ankommenden Fragmente werden zunächst gepuffert
  - Bei Eintreffen des ersten Fragments wird ein **Timer** gestartet
  - Ist der **Timer** abgelaufen bevor alle Fragmente eingetroffen sind, wird alles **verworfen**
  - Im anderen Fall wird das Datagramm am N-SAP **zur Transportschicht hochgereicht**

# Fragmentierung, Übung (1)

---



MTU = Maximum Transfer Unit

---

## Fragmentierung, Übung (2)

---

### Fragment 1

Rest des Headers		
Identifikation=120	Flags:MF = __	FO = _____
<b>Datenbyte 0 ... _____</b>		

### Fragment 2

Rest des Headers		
Identifikation=__	Flags:MF = __	FO = _____
<b>Datenbyte ____ ... _____</b>		

### Fragment 3

Rest des Headers		
Identifikation=__	Flags:MF = __	FO = _____
<b>Datenbyte ____ ... _____</b>		

# Fragmentierung, Übung (3)

---

## Fragment 1

Rest des Headers		
Identifikation=120	Flags:MF = 1	<b>FO = 0</b>
<b>Datenbyte 0 ... 375</b>		

## Fragmentierung, Übung (4)

---

### Fragment 1

Rest des Headers		
Identifikation=120	Flags:MF = 1	<b>FO = 0</b>
<b>Datenbyte 0 ... 375</b>		

### Fragment 2

$$376 / 8 = 47$$

Rest des Headers		
Identifikation=120	Flags:MF = 1	<b>FO = 47</b>
<b>Datenbyte 376 ... 751</b>		

## Fragmentierung, Übung (5)

---

### Fragment 1

Rest des Headers		
Identifikation=120	Flags:MF = 1	<b>FO = 0</b>
<b>Datenbyte 0 ... 375</b>		

### Fragment 2

$$376 / 8 = 47$$

Rest des Headers		
Identifikation=120	Flags:MF = 1	<b>FO = 47</b>
<b>Datenbyte 376 ... 751</b>		

### Fragment 3

$$752 / 8 = 94$$

Rest des Headers		
Identifikation=120	Flags:MF = 0	<b>FO = 94</b>
<b>Datenbyte 752 ... 999</b>		

# Rückblick

---

- ✓ Überblick
- ✓ IPv4-Adressierung
- ✓ IPv4-PDU
- ✓ Fragmentierung

- 1 Kommunikationssysteme und verteilte Anwendungen
- 2 Grundlagen der Transportschicht
- 3 Transportprotokolle TCP und UDP
- 4 Grundlagen der Vermittlungsschicht
- 5 Internet und Internet Protocol (IP)
- 6 Routing-Verfahren und -Protokolle**
- 7 Internet-Steuerprotokolle und DNS
- 8 Internet Protocol Version 6 (IPv6)
- 9 Netzwerkschnittstelle

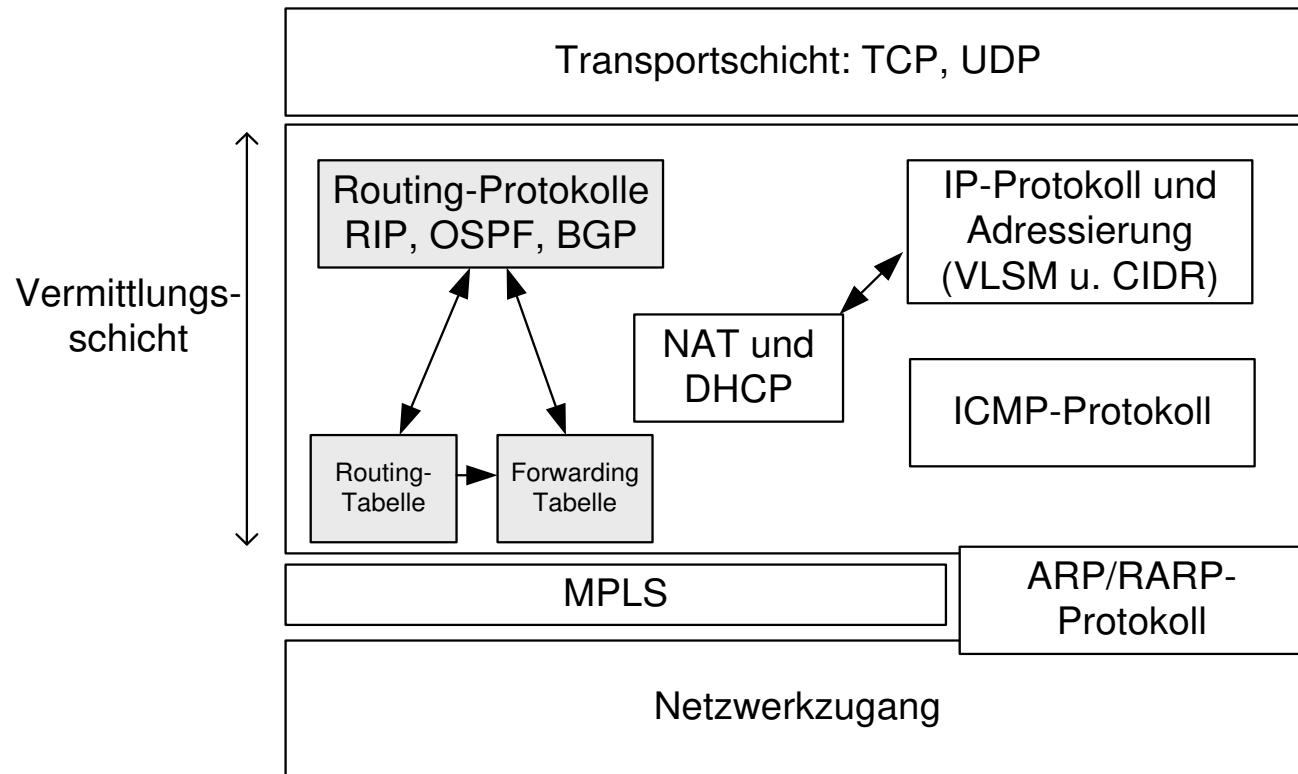
- 1. Überblick, Routing-Tabellen**
2. Routing in Endsystemen
3. IGP und EGP: Überblick
4. Routing Information Protocol (RIP)
5. Open Shortest Path First (OSPF)
6. Border Gateway Protocol (BGP)
7. Multiprotocol Label Switching (MPLS)
8. Multicast-Routing

ARPANET-IMP



# Die Internet-Vermittlungsschicht

---



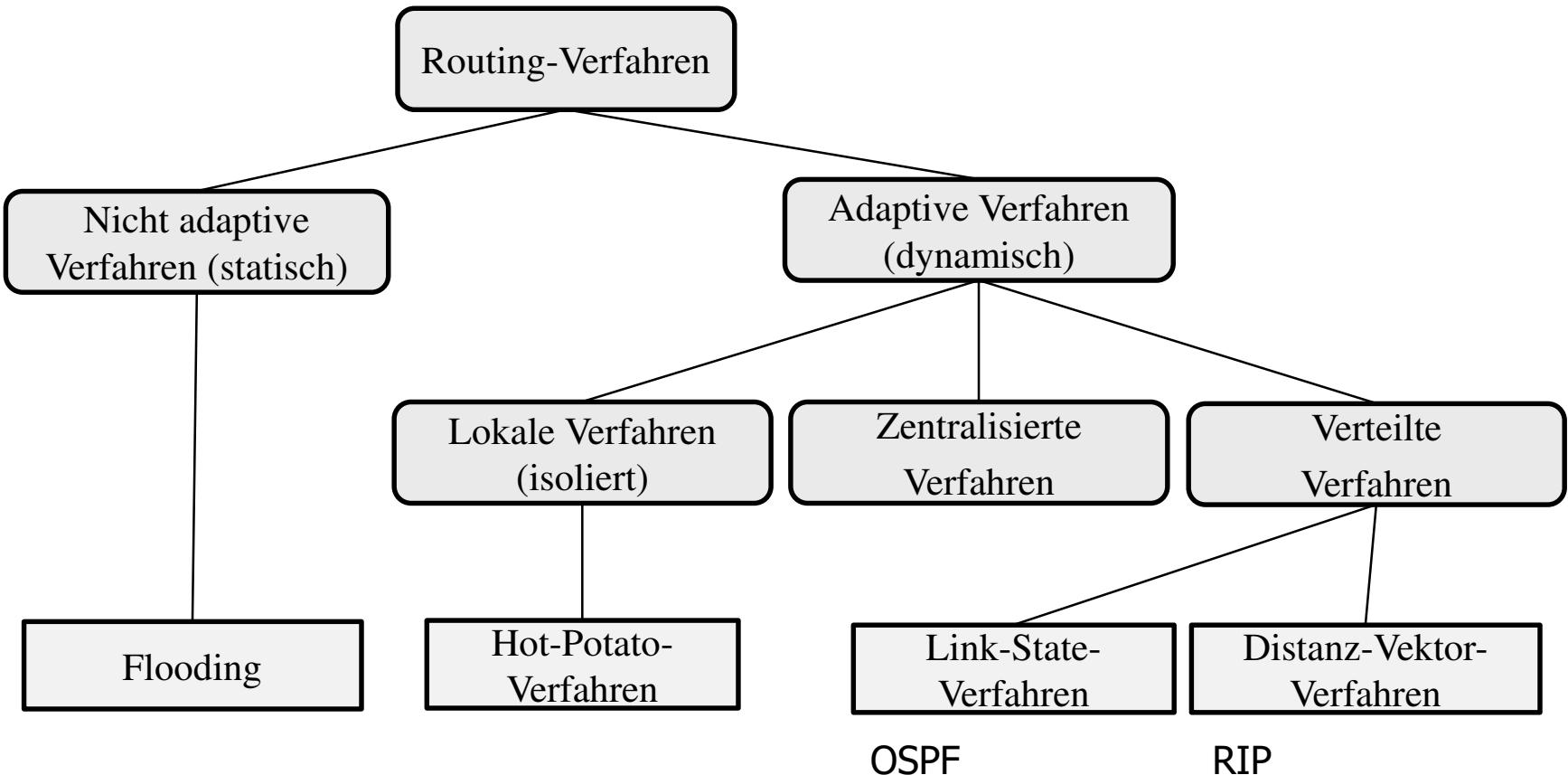
# Unterscheidung: Routing und Forwarding

---

- Router unterscheiden zwischen den Prozessen Routing und Forwarding
  - **Routing** bestimmt den gesamten Weg eines Pakets durch das Netzwerk (netzwerkweiter Prozess)
  - **Forwarding** (Weiterleitung) ist ein lokaler Entscheidungsprozess eines einzelnen Netzknotens, um zu ermitteln, zu welchem Nachbarn (Ausgangsschnittstelle) ein Paket weiterleitet
- Aus der Routing-Tabelle ergibt sich die **Forwarding-Tabelle** (= Weiterleitungstabelle) mit Zieladressen und Ausgabeschnittstellen

# Überblick: Routing – Einordnung der Verfahren

---



## Forwarding-Tabellen

---

- Jeder IP-Router verwaltet eine Forwarding-Tabelle (Weiterleitungsstabelle)

Netzwerkziel	Netzwerkmaske	Nächster Router	Ausgangsport	Metrik
...	...	...	...	...

- Ausgangsport = die dem Interface zugeordnete IP-Adresse
- Metrik = (meist) Anzahl der Hops zum Ziel, entspricht der Anzahl zu passierender Router zum Ziel
- Hinweis:
  - Die Netzwerkmaske ist erst seit der Einführung von CIDR notwendig, vorher hat man aus den ersten drei Bits der Zieladresse die Netzwerkklasse ermittelt

1. Überblick, Routing-Tabellen
- 2. Routing in Endsystemen und Routern**
3. IGP und EGP: Überblick
4. Routing Information Protocol (RIP)
5. Open Shortest Path First (OSPF)
6. Border Gateway Protocol (BGP)
7. Multiprotocol Label Switching (MPLS)
8. Multicast-Routing

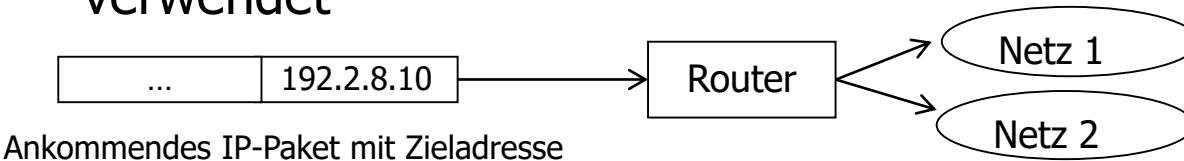
ARPANET-IMP



# Routenbestimmung: Regelwerk

---

- IPv4-Paket kommt am Router an. Was passiert?
  - Zieladresse des Pakets wird mit Einträgen in der Forwarding-Tabelle verglichen → **Longest Prefix Matching**
    - Bitweise Und-Verknüpfung zwischen Zieladresse aus IP-Paket und Netzwerkmaske aus Routeneintrag (für alle Einträge)
    - Vergleich des Ergebnisses mit Netzwerkziel aus Routeneintrag
    - Übereinstimmung → potenzielle Route gefunden!
  - Die Route mit der größten Übereinstimmung (Bits von links nach rechts) wird ausgewählt
  - Bei gleichwertigen Einträgen: Beste Metrik entscheidet!
  - Keine Übereinstimmung → Es wird die sog. Standardroute verwendet



# Forwarding-Tabelle im Endsystem, Beispiel

> netstat -r (route print unter Windows, Netzwerkdienstprogramm unter Mac OS)

## Aktive Routen:

Netzwerkziel	Netzwerkmaske	Gateway	Schnittstelle	Metrik
0.0.0.0	0.0.0.0	10.28.1.253	10.28.16.21	20
127.0.0.0	255.0.0.0	127.0.0.1	127.0.0.1	1
10.28.16.21	255.255.255.255	127.0.0.1	127.0.0.1	20
224.0.0.0	240.0.0.0	10.28.16.21	10.28.16.21	20
255.255.255.255	255.255.255.255	10.28.16.21	10.28.16.21	1

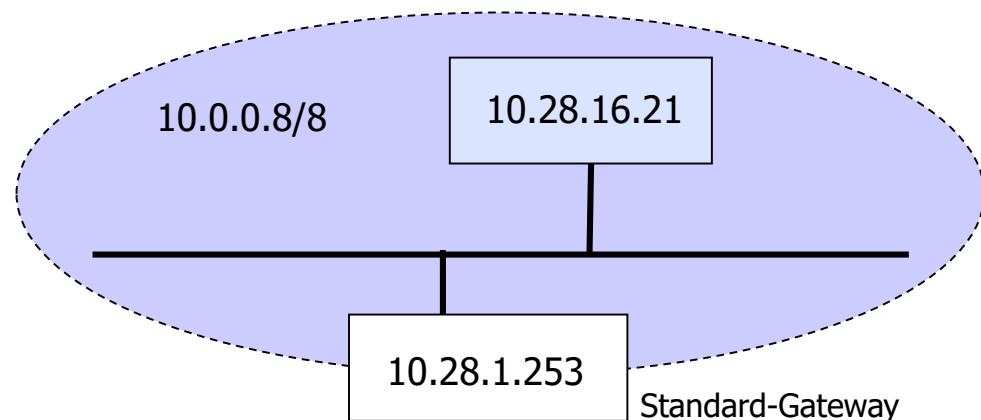
**Standardgateway:** 10.28.1.253

## Annahmen:

- Eigene IP-Adresse: 10.28.16.21
- Nur eine Ethernet-Karte im Rechner  
Standard-Gateway: 10.28.1.253

## Anmerkungen:

- „Gateway“ entspricht „Nächster Router“
- „Schnittstelle“ entspricht „Ausgangsport“



# Forwarding-Tabelle im Endsystem, Beispiel Standardroute

---

> netstat -r

## Aktive Routen:

Netzwerkziel	Netzwerkmaske	Gateway	Schnittstelle	Metrik
<b>0.0.0.0</b>	<b>0.0.0.0</b>	<b>10.28.1.253</b>	<b>10.28.16.21</b>	<b>20</b>

- Dies ist die **Standardroute**: Immer Netzwerkziel 0.0.0.0 und Netzwerkmaske 0.0.0.0 (/0)
- Jede IPv4-Zieladresse, für die eine bitweise logische UND-Operation mit 0.0.0.0 ausgeführt wird, führt zu dem Ergebnis 0.0.0.0
- Die Standardroute führt daher zu einer Übereinstimmung mit jeder IPv4-Zieladresse
- Wenn die Standardroute die längste übereinstimmende Route ist, lautet die Adresse des nächsten Knotens 10.28.1.253 (Standard-Gateway) und die Schnittstelle für den nächsten Knoten ist der Netzwerkadapter mit der IPv4-Adresse 10.28.16.21 (einiger LAN-Adapter)

# Forwarding-Tabelle im Endsystem, Beispiel Loopback-Route

---

> netstat -r

## Aktive Routen:

Netzwerkziel	Netzwerkmaske	Gateway	Schnittstelle	Metrik
0.0.0.0	0.0.0.0	10.28.1.253	10.28.16.21	20
<b>127.0.0.0</b>	<b>255.0.0.0</b>	<b>127.0.0.1</b>	<b>127.0.0.1</b>	<b>1</b>

- **Loopback-Route:** Netzwerkziel 127.0.0.0 und der Netzwerkmaske 255.0.0.0 (/8)
- Für alle Pakete, die an Adressen in der Form 127.x.y.z gesendet werden, wird die Adresse des nächsten Knotens auf 127.0.0.1 (die Loopback-Adresse) gesetzt
- Die Schnittstelle für den nächsten Knoten ist die Schnittstelle mit der Adresse 127.0.0.1 (die IP-Loopback-Schnittstelle)
- Das Paket wird nicht in das Netzwerk gesendet

# Forwarding-Tabelle im Endsystem, Beispiel Hostroute

---

> netstat -r

## Aktive Routen:

Netzwerkziel	Netzwerkmaske	Gateway	Schnittstelle	Metrik
0.0.0.0	0.0.0.0	10.28.1.253	10.28.16.21	20
127.0.0.0	255.0.0.0	127.0.0.1	127.0.0.1	1
<b>10.28.16.21</b>	<b>255.255.255.255</b>	<b>127.0.0.1</b>	<b>127.0.0.1</b>	<b>20</b>

- **Hostroute:** Netzwerkziel 10.28.16.21 und Netzwerkmaske 255.255.255.255 (/32) für die IPv4-Adresse des Hosts
- Für alle (von einer lokalen Anwendung) an die Adresse 10.28.16.21 (eigene IP-Adresse) gesendeten IPv4-Pakete wird die Adresse des nächsten Knotens auf 127.0.0.1 gesetzt
- Die Schnittstelle für den nächsten Knoten ist also die Loopback-Schnittstelle
- Das Paket wird nicht in das Netzwerk gesendet

# Forwarding-Tabelle im Endsystem, Beispiel Multicast/Broadcast-Routen

---

> netstat -r

## Aktive Routen:

Netzwerkziel	Netzwerkmaske	Gateway	Schnittstelle	Metrik
...				
224.0.0.0	240.0.0.0	10.28.16.21	10.28.16.21	20
255.255.255.255	255.255.255.255	10.28.16.21	10.28.16.21	1

- Der Eintrag mit dem Netzwerkziel 224.0.0.0 und der Netzwerkmaske 240.0.0.0 (/4) ist eine Route für **Multicast-Verkehr**, der von diesem Host gesendet wird
- Für alle Multicast-Pakete wird die Adresse des nächsten Knotens auf 10.28.16.21 (hier das Standard-Gateway) gesetzt
- Der Eintrag mit dem Netzwerkziel 255.255.255.255 und der Netzwerkmaske 255.255.255.255 (/32) ist eine Hostroute, die der **limited Broadcast-Adresse** entspricht
- Für alle an 255.255.255.255 gesendeten IPv4-Pakete wird die Adresse des nächsten Knotens auf 10.28.16.21 gesetzt und die Schnittstelle des nächsten Knotens ist der LAN-Adapter

---

Multicast-Adressbereich: 224.0.0.0 bis 239.255.255.255

# Forwarding-Tabelle im Endsystem

## Beispiel 2

---

**route print**

=====

**Schnittstellenliste**

**0x1 .....MS TCP Loopback interface**

**0x2 ...00 15 f2 16 ee 5a ..... VIA-kompatibler Fast Ethernet-Adapter - Paketplaner-Miniport**

=====

=====

**Aktive Routen:**

<b>Netzwerkziel</b>	<b>Netzwerkmaske</b>	<b>Gateway</b>	<b>Schnittstelle</b>	<b>Anzahl</b>
0.0.0.0	0.0.0.0	192.168.2.1	192.168.2.116	20
127.0.0.0	255.0.0.0	127.0.0.1	127.0.0.1	1
192.168.2.0	255.255.255.0	192.168.2.116	192.168.2.116	20
192.168.2.116	255.255.255.255	127.0.0.1	127.0.0.1	20
192.168.2.255	255.255.255.255	192.168.2.116	192.168.2.116	20
224.0.0.0	240.0.0.0	192.168.2.116	192.168.2.116	20
255.255.255.255	255.255.255.255	192.168.2.116	192.168.2.116	1

**Standardgateway:** **192.168.2.1**

=====

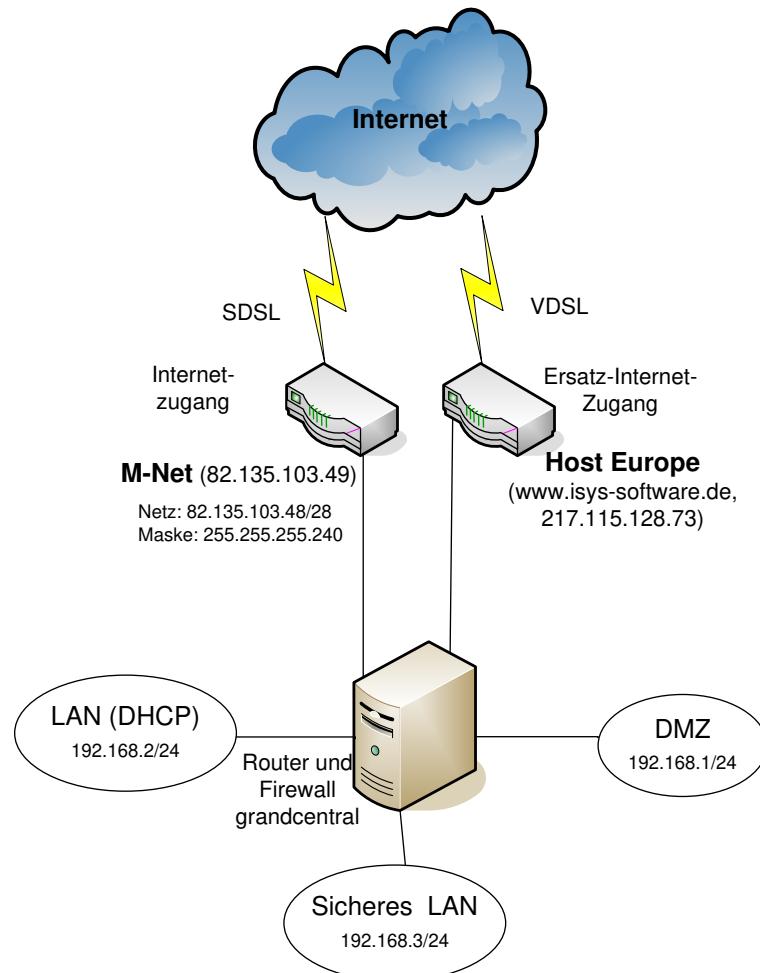
**Ständige Routen:**

**Keine**

- Eigener Host, an dem Kommando abgesetzt wurde: 192.168.2.116
- Subnetzmaske: 255.255.255.0
- Eine Ethernet-LAN-Karte

# Beispielnetz 1: Unternehmensnetz (1)

---



- Provider für IP-Adresse ermitteln: [www.utrace.de](http://www.utrace.de)
-

## Beispielnetz 1: Unternehmensnetz (2)

---

- Kommandos tracert oder pathping unter Windows aus dem Unternehmensnetz heraus:

**C:\tracert -w 30 www.hm.edu**

Routenverfolgung zu www.hm.edu [129.187.244.229] über maximal 30 Abschnitte:

```
1  <1 ms  <1 ms  <1 ms  grandcentral.isys-software.de [192.168.2.1]
 2  1 ms   <1 ms   1 ms  192.168.250.1
 3  32 ms   31 ms   46 ms  217.5.98.13
 4  35 ms   35 ms   35 ms  217.237.152.62
 5  40 ms   41 ms   41 ms  l-eb2-i.L.DE.NET.DTAG.DE [62.154.89.110]
 6  44 ms   44 ms   *     80.156.160.142
 7  49 ms   *     49 ms  xr-aug1-te2-1.x-win.dfn.de [188.1.144.150]
 8  51 ms   51 ms   53 ms  xr-gar1-te2-1.x-win.dfn.de [188.1.144.109]
 9  48 ms   48 ms   51 ms  kr-lrz-muenchen2.x-win.dfn.de [188.1.37.90]
10  48 ms   49 ms   49 ms  vl-3003.csr3-kb1.lrz.de [129.187.0.138]
11  50 ms   49 ms   50 ms  vl-3003.csr3-kb1.lrz.de [129.187.0.138]
12  51 ms   *     50 ms  hm.edu [129.187.244.229]
```

Ablaufverfolgung beendet.

---

## Beispielnetz 1: Unternehmensnetz (3)

---

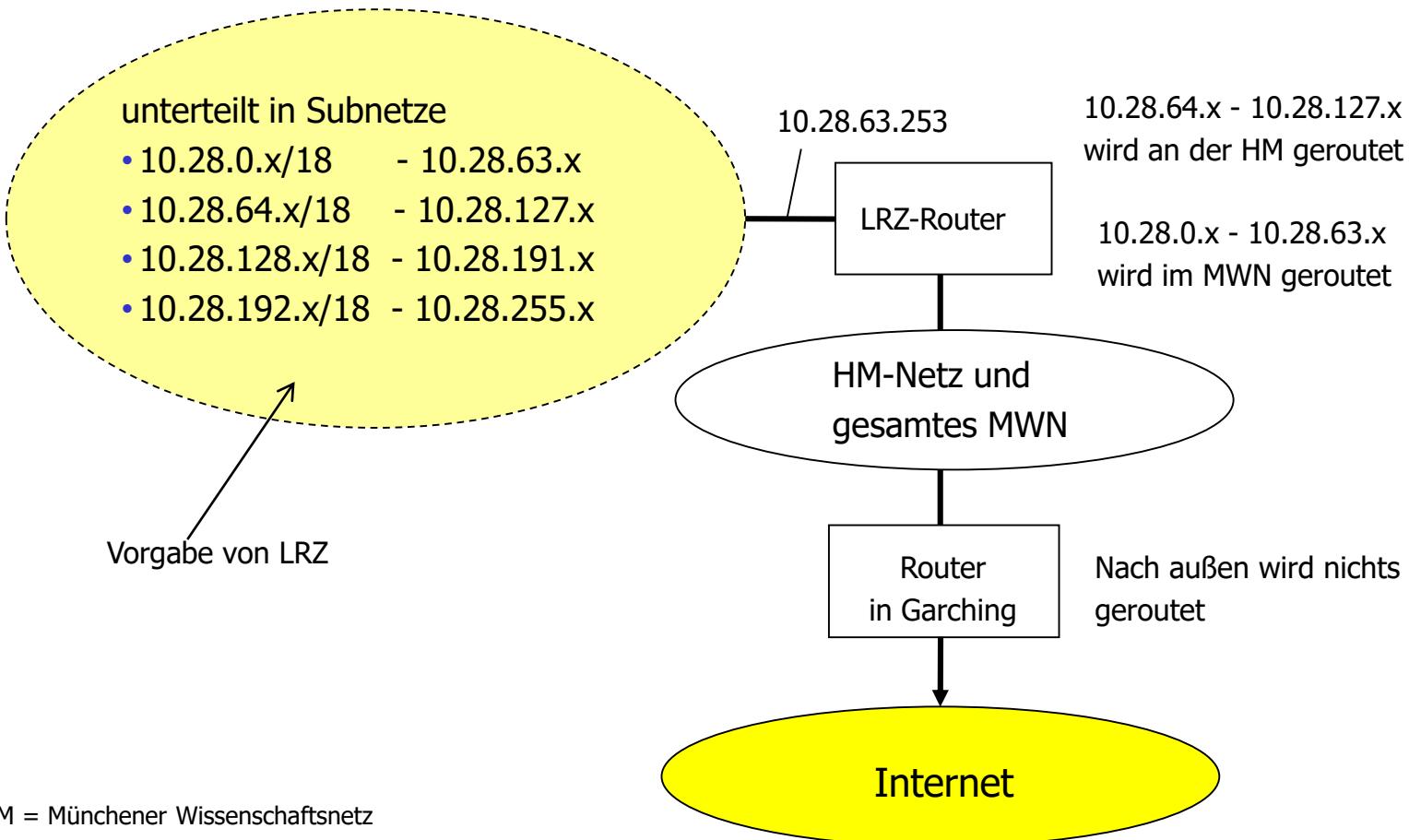
- Globale IP-Adresse von iSYS in Präfixnotation:  
**82.135.103.48/28**
- Netzwerkmaske: 255.255.255.240

82.135.103.48 = .... 0011 0000 (nicht nutzbar)  
82.135.103.49 = .... 0011 0001 Internet Router  
82.135.103.50 = .... 0011 0010  
82.135.103.51 = .... 0011 0011 IPSEC VPN Gateway  
82.135.103.52 = .... 0011 0100 iSYS Websites ([www.isys.de](http://www.isys.de))  
82.135.103.53 = .... 0011 0101 Mailserver ([mail.isys.de](mailto:mail.isys.de))  
82.135.103.54 = .... 0011 0110 OpenVPN Gateway  
82.135.103.55 = .... 0011 0111 ...  
82.135.103.56 = .... 0011 1000  
82.135.103.57 = .... 0011 1001  
82.135.103.58 = .... 0011 1010 SVN Server, VPN-Gateway  
82.135.103.59 = .... 0011 1011  
82.135.103.60 = .... 0011 1100  
82.135.103.61 = .... 0011 1101 iCAL Server  
82.135.103.62 = .... 0011 1110 IPSEC VPN Gateway  
82.135.103.63 = .... 0011 1111 (nicht nutzbar)

- **14 verfügbare Adressen!!**

# Beispielnetz 2: HM - Fakultät-07-Netz

## Fakultätsnetz mit IP-Adressraum 10.28.0.0/16



MWN = Münchener Wissenschaftsnetz

LRZ = Leibnitz Rechenzentrum

# Routenbestimmung: Regelwerk – Übung 1 (1)

---

- Bei IP-Router R1 kommt von Router R5 aus dem Netzwerk 128.10.0.0/16 ein IP-Paket mit der **Zieladresse 193.1.1.200** an
- Welche Route wählt R1 mit folgender Forwarding-Tabelle?

## Forwarding-Tabelle R1:

Netzwerkziel	Netzwerkmaske	Gateway	Schnittstelle	Anzahl
0.0.0.0	0.0.0.0	193.1.1.3	193.1.1.1	1
128.10.0.0	255.255.0.0	128.10.1.2	128.10.1.1	1
193.1.1.192	255.255.255.192	193.1.1.3	193.1.1.1	1
196.1.1.0	255.255.255.0	194.1.1.2	194.1.1.1	2
197.1.1.0	255.255.255.0	195.1.1.2	195.1.1.1	2

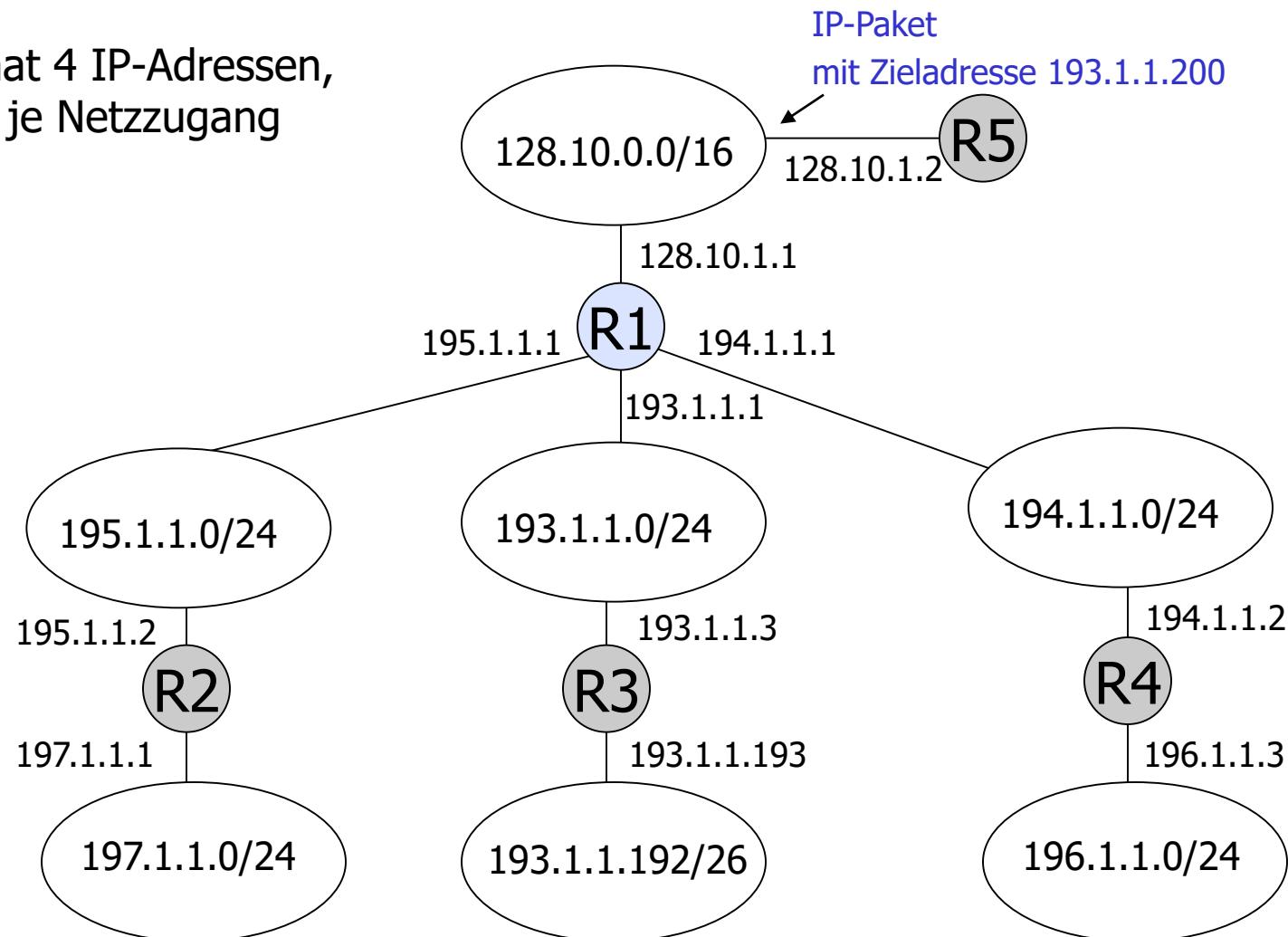
...

**Standardgateway:** 193.1.1.3

- Hinweis: Route mit größter Übereinstimmung finden!

## Routenbestimmung: Regelwerk – Übung 1 (2)

R1 hat 4 IP-Adressen,  
eine je Netzzugang



# Routenbestimmung: Regelwerk – Übung 1 (3)

---

## ■ Lösung:

- Zieladresse  $193.1.1.200 = 1100\ 0001 \cdot 0000\ 0001 \cdot 0000\ 0001 \cdot 1100\ 1000$
- Stimmt am besten überein mit Routingtabellen-Eintrag

193.1.1.192	255.255.255.192	193.1.1.3	193.1.1.1	1
-------------	-----------------	-----------	-----------	---

- Nachweis:

$1100\ 0001 \cdot 0000\ 0001 \cdot 0000\ 0001 \cdot 1100\ 1000$  (Zieladresse 193.1.1.200)

$1111\ 1111 \cdot 1111\ 1111 \cdot 1111\ 1111 \cdot 1100\ 0000$  (Netzwerkmaske 255.255.255.192) **AND**

---

$1100\ 0001 \cdot 0000\ 0001 \cdot 0000\ 0001 \cdot 1100\ 0000$  (Ergebnis der log. Und-Operation)

$1100\ 0001 \cdot 0000\ 0001 \cdot 0000\ 0001 \cdot 1100\ 0000$  (Netzwerkziel 193.1.1.192 in Routing-Tabelle)

---

Übereinstimmung mit Zieladresse 193.1.1.192 aus der Routing-Tabelle an  
26 Stellen des Netzwerkanteils → beste Route!

→ Ausgewählte Route: 193.1.1.3, Schnittstelle 193.1.1.1

## Routenbestimmung: Regelwerk – Übung 2

---

- Ein Router verfügt über die folgende Forwarding-Tabelle

Netzwerkziel	Netzwerkmaske	Schnittstelle
193.1.100.0	255.255.255.0	i1
193.1.100.64	255.255.255.192	i2
193.1.100.128	255.255.255.192	i3
198.1.100.64	255.255.255.192	i4
198.1.100.128	255.255.255.192	i5

- Es kommt ein Paket mit der Zieladresse 193.01.100.72 an
- Über welche Schnittstelle wird das Paket weitergeleitet?

Lösung:

$$\begin{aligned}11000001.00000001.01100110.01000110 &= 103.51.100.72 \text{ (Zieladresse)} \\ \text{AND } 11111111.11111111.11111111.11000000 &= 103.51.100.64 \text{ (Netzwerkmaske)} \\ = 11000001.00000001.01100110.01000000 &= 193.1.100.64 \rightarrow \textbf{Schnittstelle i2} \\ 26 \text{ Bit Übereinstimmung, } 193.1.100.128 \text{ hätte nur 24 Bit Übereinstimmung}\end{aligned}$$

---

## Statisches Routing

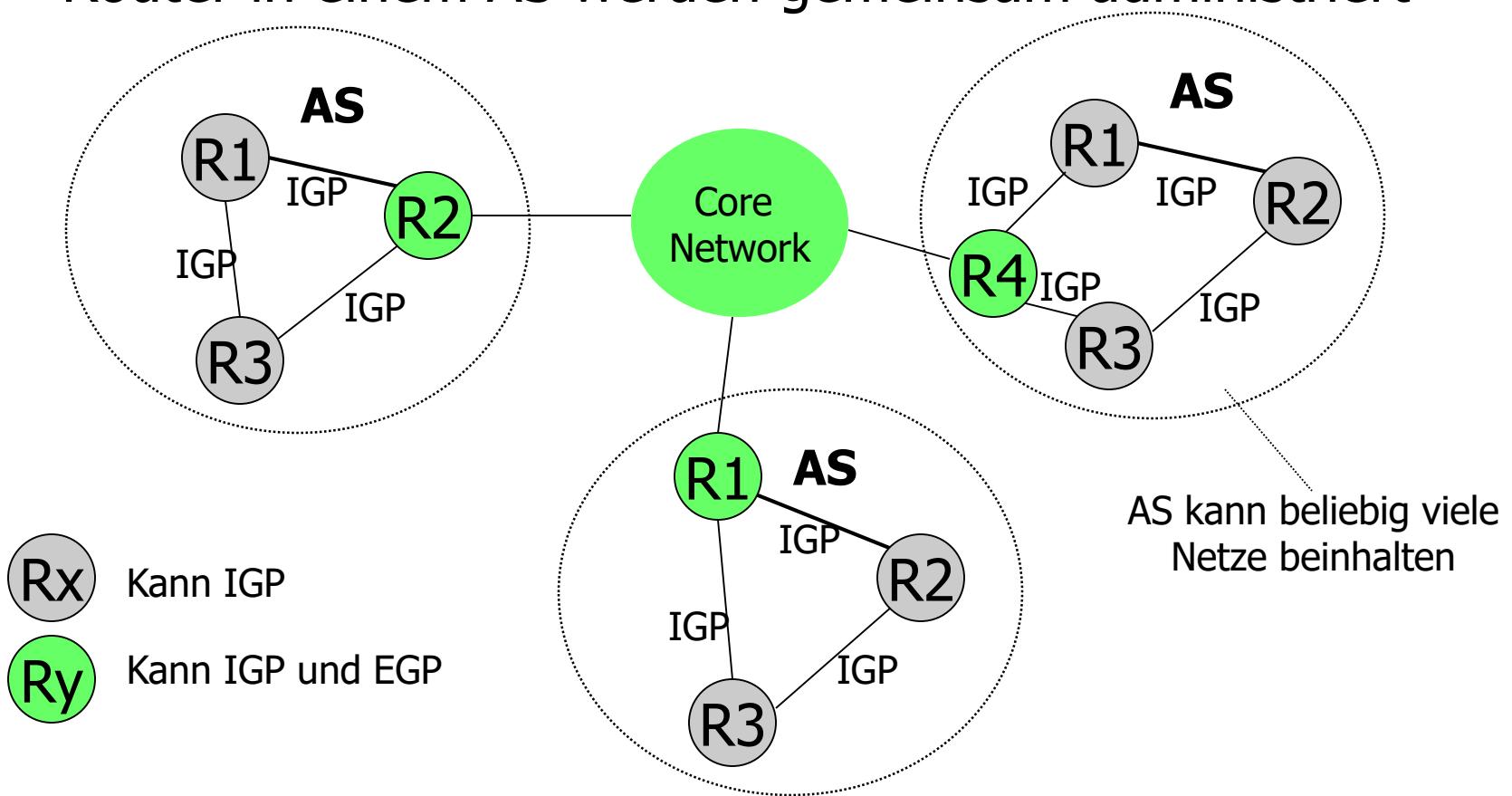
---

- In kleinen Unternehmensnetzen (bis zu ca. 1000 Rechner) sind dynamische Routing-Protokolle oft nicht erforderlich
- Die Forwarding-Tabellen können statisch konfiguriert werden

1. Überblick, Routing-Tabellen
2. Routing in Endsystemen und Routern
- 3. IGP und EGP: Überblick**
4. Routing Information Protocol (RIP)
5. Open Shortest Path First (OSPF)
6. Border Gateway Protocol (BGP)
7. Multiprotocol Label Switching (MPLS)
8. Multicast-Routing

# Routing in autonomen Systemen

- IGP innerhalb eines AS muss gleich sein
- Router in einem AS werden gemeinsam administriert



# Routing in autonomen Systemen

---

- **IGP:** Routing-Protokolle für autonome Systeme werden als Interior Gateway Protokolle (IGP) bezeichnet
  - Jedes autonome System kann intern eigene Routing-Algorithmen verwenden
  - Beispiele: RIP und OSPF
- **RIP:** Älteres Verfahren für kleinere Netze
  - Distance-Vector-Protocol, aber: Count-to-Infinity-Problem
- **OSPF:** Seit 1990 ist (Open Shortest Path First) der empfohlene Standard, RFC 1247
  - Wird von der Internet-Gemeinde empfohlen
  - Netz wird als gerichteter Graph abstrahiert
  - Kanten zwischen den Knoten werden mit Kosten gewichtet (Bandbreite, Entfernung, Verzögerung,...)
  - Entscheidung über das Routing anhand der Kosten

# Routing zwischen autonomen Systemen

---

- Zwischen autonomen Systemen werden andere Routing-Protokolle benötigt (Exterior Gateway Protocol, **EGP**)
  - Andere Ziele werden verfolgt, Beispiele:
    - Nicht alle Pakete sollen befördert werden
    - Gehe nicht über ein bestimmtes AS (z.B. wegen Konkurrenz)
    - Für Transitverkehr muss bezahlt werden, Kosten niedrig halten
    - Wichtige Informationen nicht durch unsichere autonome Systeme senden
    - Sende nie über ein bestimmtes Land
    - ...
  - Routing-Regeln sind erforderlich, die vom Routing-Protokoll unterstützt werden müssen
  - Im Internet wird das Border Gateway Protokoll (**BGP**) empfohlen
    - Pfadvektorprotokoll, verwandt zu Distance-Vector-Protokollen
    - Im Gegensatz zum Distance-Vector-Protokoll werden ganze Pfade ausgetauscht und damit können Routing-Schleifen vermieden werden

1. Überblick, Routing-Tabellen
2. Routing in Endsystemen und Routern
3. IGP und EGP: Überblick

## **4. Routing Information Protocol (RIP)**

5. Open Shortest Path First (OSPF)
6. Border Gateway Protocol (BGP)
7. Multiprotocol Label Switching (MPLS)
8. Multicast-Routing

## RIP (1)

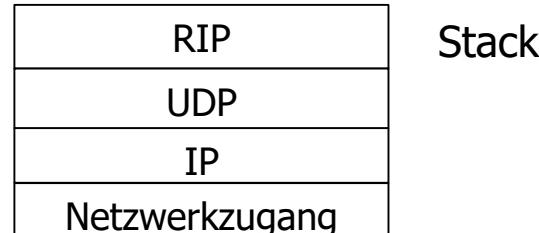
---

- RIP (Routing Information Protocol) wurde ursprünglich von XEROX entwickelt und wird in kleinen AS immer noch stark verwendet
- Einfach und leicht zu implementierendes **Distance-Vector-Protocol**
- Als Metrik wird „**Hop-Count**“ verwendet
- RIPv1 ist **klassenorientiert** und ermittelt das Zielnetzwerk anhand der ersten Bits der Ziel-IP-Adresse
  - 0 = Klasse A, 10 = Klasse B, 110 = Klasse C, ...
- Implementierung unter Unix durch **routed**-Prozess
- Forwarding-Tabelle anschauen: **netstat -r**

## RIP (2)

---

- RIP versendet die Routing-Einträge alle 30 s (**Update-Timer**) in sog. Advertisement-PDUs an die Nachbar-Router
  - Max. 25 Routeneinträge pro PDU, evtl. mehrere PDUs
  - RIPv1 über MAC-Broadcast, RIPv2 nutzt Multicast auf Subnetzebene (nicht über IGMP)
  - RIP nutzt UDP und ist **nicht** geeignet für WAN-Routing, eher im LAN wegen Broadcast/Multicast
- Timermechanismen
  - **Timeout-Timer:** Wenn eine Route 180 s nicht mehr gesehen wird, wird sie auch nicht mehr propagiert
  - **Flush-Timer:** Nach weiteren 120 s wird die Route vollständig entfernt



## RIP (3)

---

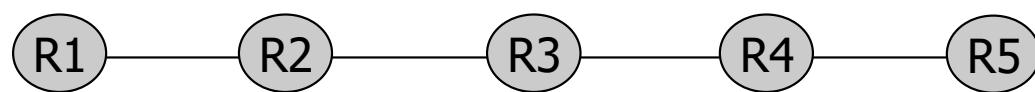
- **Konvergenzzeit:** Zeit, die erforderlich ist, bis alle Router die aktuelle Struktur eines Netzes kennen gelernt haben
  - Danach liegt ein konsistentes Netzwerk vor
    - Ist bei RIP relativ lange, da RIP-Router Änderungen erst verarbeiten, also in ihre Routing-Tabellen eintragen, und dann an die Nachbarn propagieren
    - Max. 15 Hops wurden gewählt, um die Konvergenzzeit zu beschränken → nach 15 WeiterSendungen wird ein Paket durch den letzten Router entfernt
- **Split Horizon** ist eine Methode, um die Konvergenzzeit kürzer zu halten und um (direkte) Routing-Schleifen zu vermeiden
  - Es wird festgehalten, woher eine Routing-Information kommt

## RIP (4)

- Verbreitung von Routing-Tabellen-Einträgen in mehreren Takten zu je 30 s bestimmt die Konvergenzzeit
- Bis zum nächsten Senden einer Advertisement-PDU dauert es im Mittel 15 s, zufallsabhängig
- Beispiel: Neue Route wird propagiert, Zeitangaben ohne Verarbeitungszeit in Routern

Wartezeit bis zum Senden der

nächsten Advertisement-PDU: → 15 s



$t_0$

+15 s

+30 s

+45 s

+60 s

+75 s

$t$

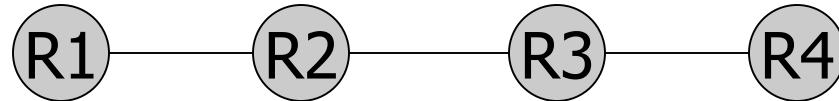
Neue Route wird  
bekannt

R2 hat neue Route 30 s nach Bekanntwerden  
(unter Vernachlässigung der Netzwerklaufzeit)

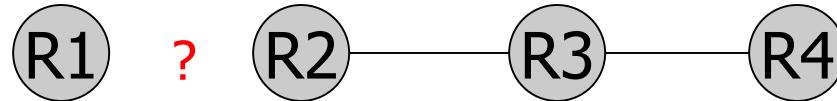
## RIP-Beispiel (1)

---

- **Routing-Schleifen** („Count-to-Infinity-Problem“) möglich
- Vermeidung:
  - Lösung 1: **Split-Horizon-Technik** → Routing-Tabellen enthalten **zusätzlich** die Info, woher die Routing-Info kommt
  - Lösung 2: **Poison-Reverse** (vergifteter Rückweg) → Alle Routen werden propagiert, aber zum Ursprungsnetz hin als „nicht erreichbar markiert durch Hop-Count = 16 → „vergiftet“
- Beispiel: a) Alle Verbindungen R1-R2, R2-R3 und R3-R4 intakt



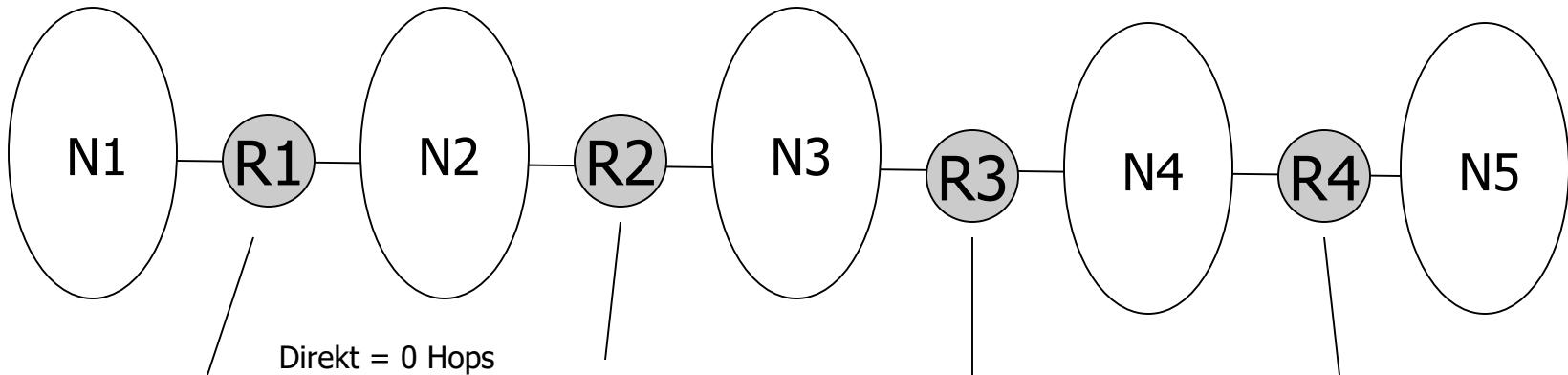
- b) Verbindung R1-R2 fällt aus



- Was passiert mit und ohne Split-Horizon?

## RIP-Beispiel (2)

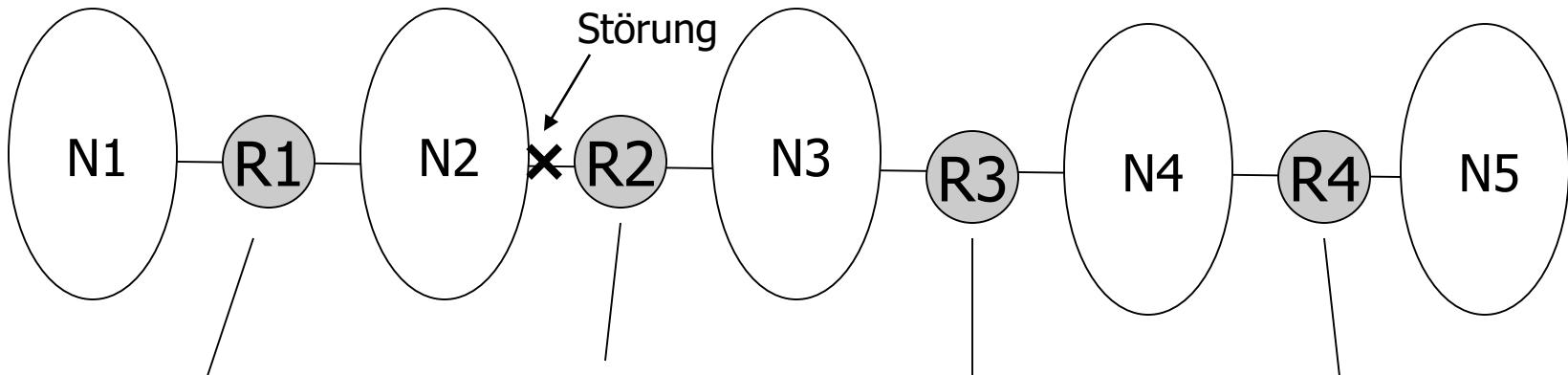
- Routing-Tabellen im eingeschwungenen Zustand (nach einer angemessenen Konvergenzzeit)



Router R1			Router R2			Router R3			Router R4		
Ziel-netz	Anzahl Hops	Über Router									
N1	0	direkt	N1	1	R1	N1	2	R2	N1	3	R3
N2	0	direkt	N2	0	direkt	N2	1	R2	N2	2	R3
N3	1	R2	N3	0	direkt	N3	0	direkt	N3	1	R3
N4	2	R2	N4	1	R3	N4	0	direkt	N4	0	direkt
N5	3	R2	N5	2	R3	N5	1	R4	N5	0	direkt

## RIP-Beispiel (3)

- Störung im Netzwerkzugang von R2 zu N2: R2 korrigiert sofort



Router R1			Router R2			Router R3			Router R4		
Ziel-netz	Anzahl Hops	Über Router									
N1	0	direkt	N1	<b>16</b>	---	N1	2	R2	N1	2	R3
N2	0	direkt	N2	<b>16</b>	---	N2	1	R2	N2	2	R3
N3	1	R2	N3	0	direkt	N3	0	direkt	N3	1	R3
N4	2	R2	N4	1	R3	N4	0	direkt	N4	0	direkt
N5	3	R2	N5	2	R3	N5	1	R4	N5	0	direkt

## RIP-Beispiel (4)

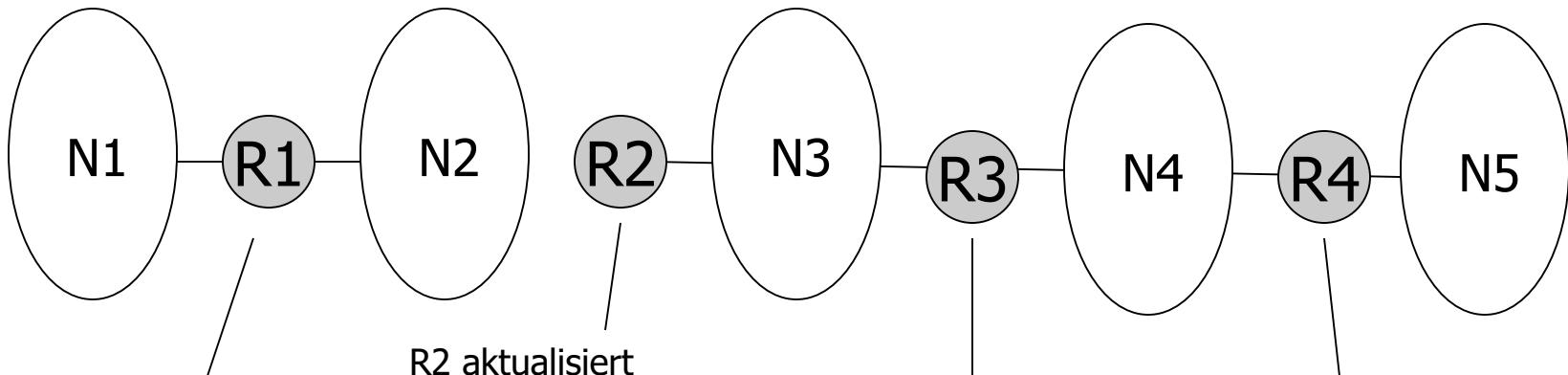
---

### ■ Ohne Split Horizon:

- R3 hat noch die Routing-Info, **dass N1 über zwei Hops und N2 über einen Hop** erreichbar ist
- R3 propagiert diese Info an R2, also an den Router, über den R1 erreicht wurde
- R2 glaubt dies und sendet Pakete zu R1 nun über R3
- Ping-Pong-Effekt, Routing-Schleife bis Hop-Count = 16, dann erst wird R1 als nicht erreichbar markiert
  
- Im Folgenden aus Sicht von R2 und R3 skizziert!

## RIP-Beispiel (5)

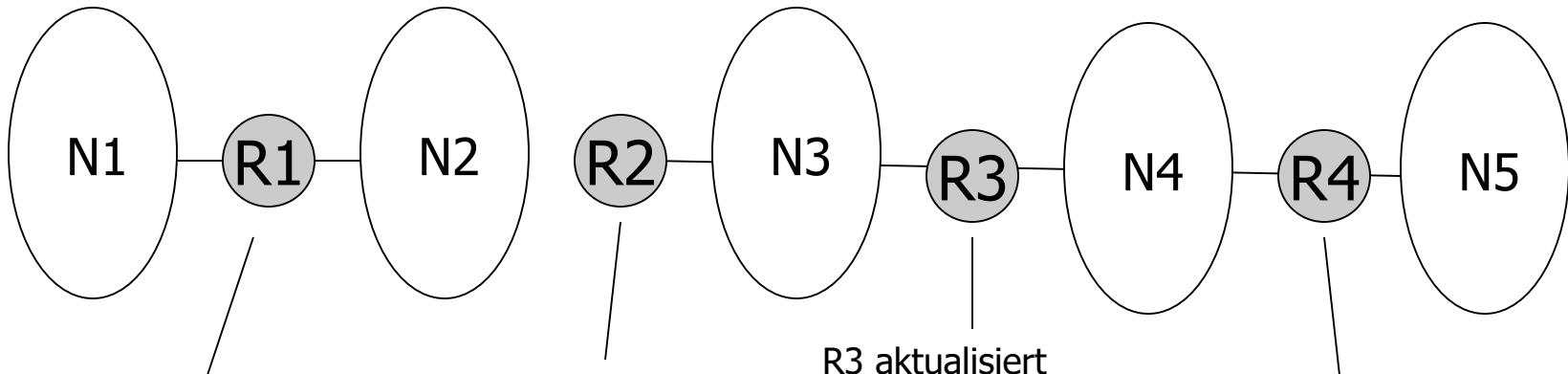
- Nun sendet R3 eine RIPv1-Advertisement-PDU an seinen Nachbarn R2 →  $\{(N1, 2), (N2, 1), (N3, 0), (N4, 0), (N5, 1)\}$



Router R1			Router R2			Router R3			Router R4		
Ziel-netz	Anzahl Hops	Über Router									
N1	0	direkt	N1	3	R3	N1	2	R2	N1	3	R3
N2	0	direkt	N2	2	R3	N2	1	R2	N2	2	R3
N3	1	R2	N3	0	direkt	N3	0	direkt	N3	1	R3
N4	2	R2	N4	1	R3	N4	0	direkt	N4	0	direkt
N5	3	R2	N5	2	R3	N5	1	R4	N5	0	direkt

## RIP-Beispiel (6)

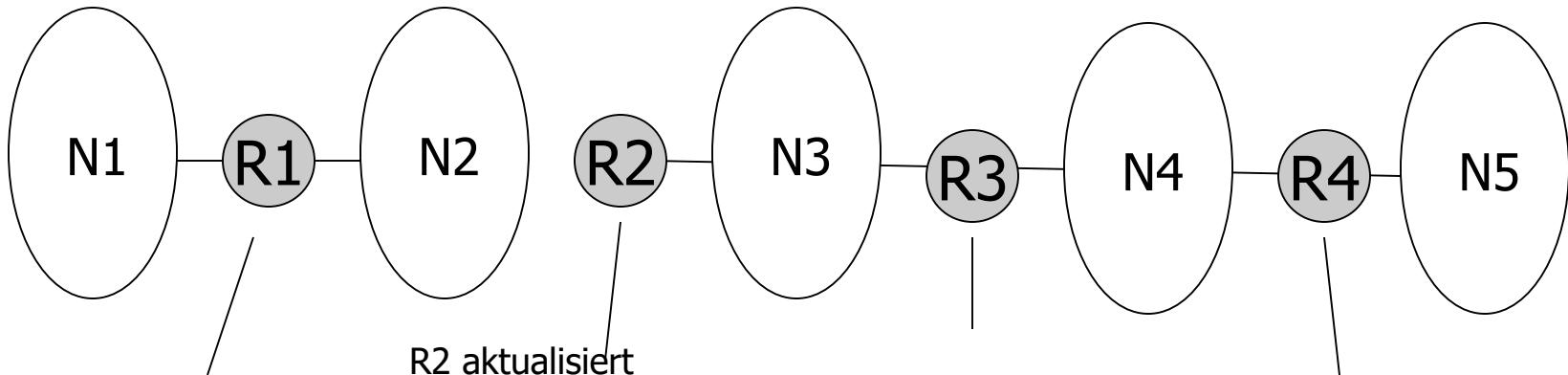
- Als nächstes sendet R2 eine RIPv1-Advertisement-PDU an seinen Nachbarn R3 →  $\{(N1, 3), (N2, 2), (N3, 0), (N4, 1), (N5, 2)\}$



Router R1			Router R2			Router R3			Router R4		
Ziel-netz	Anzahl Hops	Über Router									
N1	0	direkt	N1	3	R3	N1	<b>4</b>	R2	N1	3	R3
N2	1	direkt	N2	2	R3	N2	<b>3</b>	R2	N2	2	R3
N3	1	R2	N3	0	direkt	N3	0	direkt	N3	1	R3
N4	2	R2	N4	1	R3	N4	0	direkt	N4	0	direkt
N5	3	R2	N5	2	R3	N5	1	R4	N5	0	direkt

## RIP-Beispiel (7)

- Nun sendet R3 wieder eine RIPv1-Advertisement-PDU an seinen Nachbarn R2 →  $\{(N1, 4), (N2, 3), (N3, 0), (N4, 0), (N5, 1)\}$



Router R1			Router R2			Router R3			Router R4		
Ziel-netz	Anzahl Hops	Über Router									
N1	0	direkt	N1	5	R3	N1	4	R2	N1	3	R3
N2	0	direkt	N2	4	R3	N2	3	R2	N2	2	R3
N3	1	R2	N3	0	direkt	N3	0	direkt	N3	1	R3
N4	2	R2	N4	1	R3	N4	0	direkt	N4	0	direkt
N5	3	R2	N5	2	R3	N5	1	R4	N5	0	direkt

usw.

## RIP-Beispiel (8)

---

- **Mit Split Horizon:**

- R3 weiß, woher die Routing-Info für **N1** und **N2** kommt (nämlich von R2)
- Route mit höheren Kosten wird nicht zurückpropagiert
- Keine Routing-Schleife (in diesem Fall), da es sich um ein einfaches Single-Path-Netzwerk handelt, d.h. die Routing-Informationen können nur linear verbreitet werden

# RIPv1-PDU

---

- Metrik = 16 → Netzwerkziel nicht erreichbar
- AFI = Adressierungsart, bei IP-Adressen immer 0x02



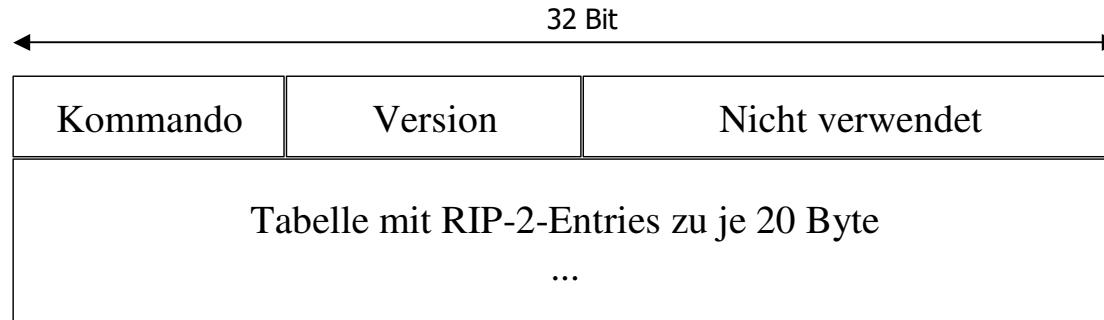
RIP-1-Entry:

Address-Family-Identifier	Nicht verwendet
IPv4-Adresse	
Nicht verwendet	
Nicht verwendet	
Metrik	

# RIPv2-PDU

---

- Next-Hop: Direkte Angabe eines Zielhosts möglich



RIP-2-Entry:

Address-Family-Identifier	Route-Tag
IPv4-Adresse	
Subnet-Mask	
Next-Hop	
Metrik	

## Sonstiges zu RIPv1 und RIPv2

---

- **Sonstiges zu RIP**
  - RIPv1 **unterstützt** CIDR/VLSM **nicht**
    - Subnetzmaske wird in RIPv1 **nicht** übermittelt
    - Klasse wird aus den ersten Bits der Ziel-IP-Adresse ermittelt
  - RIPv2 **unterstützt** CIDR/VLSM
  - RIPv2 kann **Split-Horizon** mit und ohne **Poison-Reverse**
  - RIPv2 kann selbst ausgelöste Router-Aktualisierungen (**Triggered Updates**) bei Ankunft einer Advertisement-PDU
    - Höhere, aber immer noch nicht perfekte Konvergenz, mehr Netzwerklast
  - RIPv1 kommuniziert über **Broadcast**, RIPv2 über **Multicast** (Adresse: 224.0.0.9)

## Anmerkung zu Split Horizon mit Poison Reverse

---

- Kein Vorteil bei sog. **Single-Path**-Netzen (linear)
  - Hier schafft man es mit Split Horizon schon, Routing-Schleifen zu vermeiden
- Vorteil bei **Multi-Path**-Netzen
  - Hier können Routing-Schleifen stark reduziert, aber auch nicht ganz vermieden werden, da Routen von verschiedenen Quellen gelernt werden

## Einschub:

### Ein weiteres Distanzvektorprotokoll: EIGRP

---

- Enhanced Interior Gateway Routing Protocol (EIGRP)
- Erweiterung von IGRP (auch Cisco)
- Von Cisco im Jahre 1992 veröffentlicht
- Seit 2013 offenes Protokoll
  
- Weitere Infos siehe Quelle

Quelle: Odom, W.; Sequeira, A.: Cisco CCNA Routing und Switching ICND2 200-101. Das offizielle Handbuch zur erfolgreichen Zertifizierung. Ciscopress.com, 2013

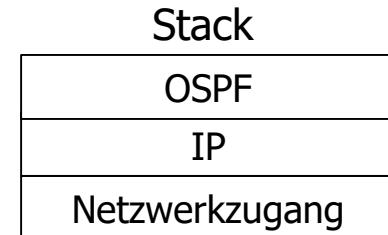
---

1. Überblick, Routing-Tabellen
2. Routing in Endsystemen und Routern
3. IGP und EGP: Überblick
4. Routing Information Protocol (RIP)
- 5. Open Shortest Path First (OSPF)**
6. Border Gateway Protocol (BGP)
7. Multiprotocol Label Switching (MPLS)
8. Multicast-Routing

# OSPF, Einführung

---

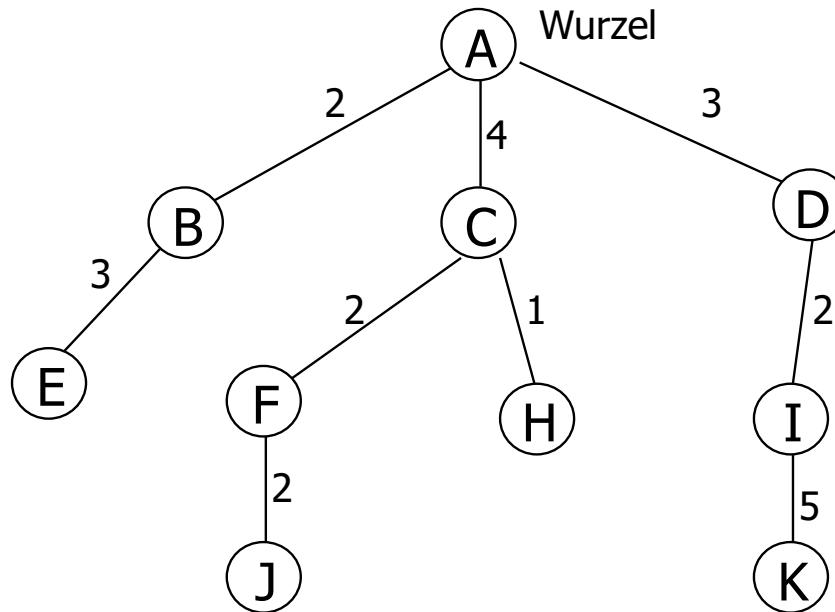
- OSPF ist **für große Unternehmensnetze** gedacht
- **Offener** Standard (Open SPF), RFC 1247 u. 2328
- OSPF ist ein **Link-State-Protocol**
  - „Link State“ ist der Zustand einer Verbindung zweier Router
  - zustandsorientiert statt entfernungsorientiert (RIP)
- Jeder Router führt **eigene Datenbasis** (Link-State-Datenbank) mit allen Routing-Einträgen des Netzes
- OSPF nutzt **IP direkt**, (siehe IP-Header, Protokoll = 89)



## OSPF, SPF-Baum

---

- Jeder IP-Router erzeugt aus seiner Sicht einen Spanning Tree (SPF-Baum) für das ganze Netzwerk
- Wurzel ist der Router selbst
- Verzweigung = günstigste Route



## OSPF, Funktionalität

---

- Alle Router suchen beim Start ihre Nachbarn mit **Hello-PDUs**, aber nicht alle angrenzenden Router werden auch zu Nachbarn (sog. **adjacents**)
- **Zyklischer Abgleich** der Link-State-Datenbanken mit den Nachbarn
- **Lebendüberwachung** periodisch unter den Nachbarn
- **Link State Updates** (Konsistente Datenhaltung in allen Routern) periodisch **und** bei Topologieänderungen
- **Refreshing** spätestens alle 30 Minuten

## OSPF, Sonstige Features

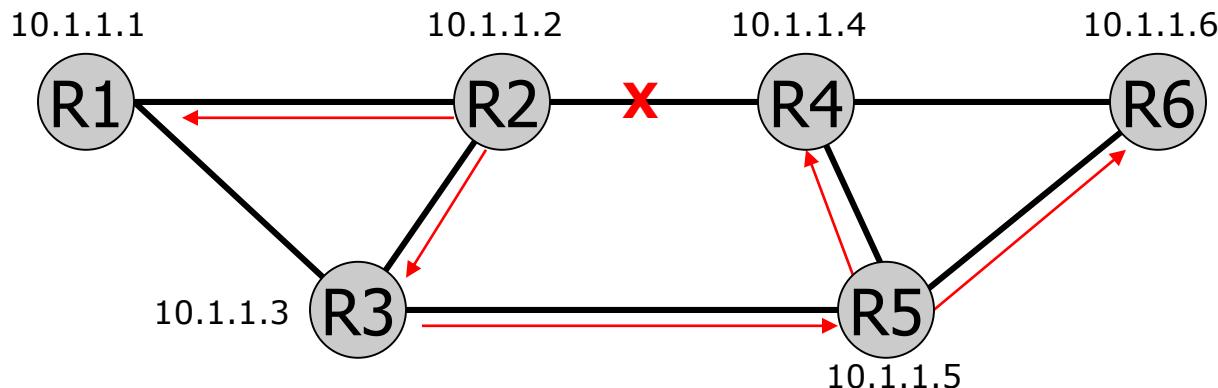
---

- **Load Balancing** bei Pfaden mit gleichen Kosten
  - gleichmäßige Verteilung, besser als bei RIP
- Nutzung spezieller **Multicast-Adressen** zur Kommunikation aller OSPF-Router eines Netzwerksegments
  - 224.0.0.5 für die gegenseitige Überwachung
  - 224.0.0.6 für den Austausch von Routing-Information
- Unterstützung der Router-**Authentifizierung** zur Vermeidung von Angriffen (mehr Sicherheit)
- Metriken (Kosten) in RFCs nicht festgelegt
  - Cisco IOS (Internet Operating System) verwendet z.B. als Metrik die Gesamtbandbreite aller Ausgangsschnittstellen vom aktuellen Router bis zum Zielnetzwerk

# OSPF-Konvergenz

---

- Verteilung einer Veränderung im Netz geht schnell und Information wird vor der Verarbeitung weiterkommuniziert
  - Hohe Konvergenz
- Beispiel für Routen-Austausch:
  - Verbindung zwischen 10.1.1.2 und 10.1.1.4 fällt aus
  - Link-State Updates werden über das ganze Netz verteilt
  - Nachdem DB synchronisiert ist, gibt es nur noch jeweils eine kürzeste Route zu von allen Routern zu allen anderen Routern



Vgl. Dirk Jakob, Folenvortrag

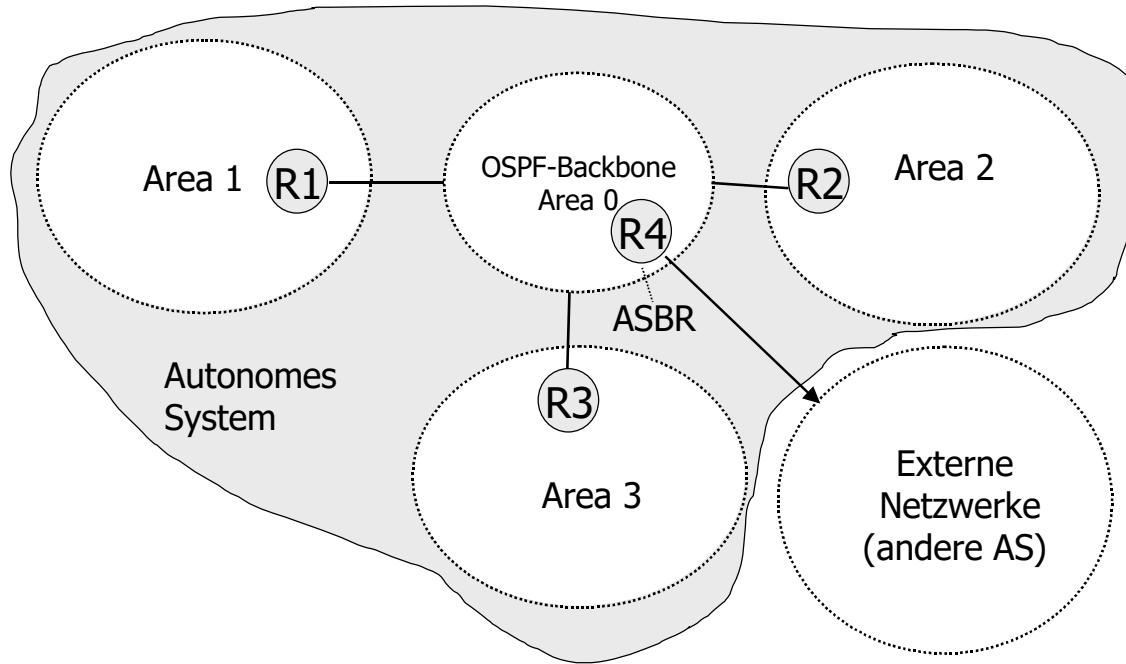
# OSPF-Routertypen

---

- Bei OSPF gibt es vier Router-Klassen
  - Interne Router der Area
  - Router an Bereichsgrenzen (Area-Grenzen)
    - Verbinden zwei oder mehrere Areas
  - Backbone-Router
    - Befinden sich im OSPF-Backbone
  - AS-Grenz-Router (ASBR)
    - Vermitteln zwischen autonomen Systemen
- Bei Einsatz in Broadcast-orientierten LANs:  
Verwendung von sog. designierten Routern
  - Alle Router bauen eine Nachbarschaft (Adjacencies) zu diesem Router auf → Reduzierung der Kommunikation

# OSPF-Backbone

---



- R1, R2, R3 = Router an Bereichsgrenzen
- R4 = AS-Grenz-Router (ASBR), vermittelt zwischen autonomen Systemen

## OSPF-Nachbarschaften

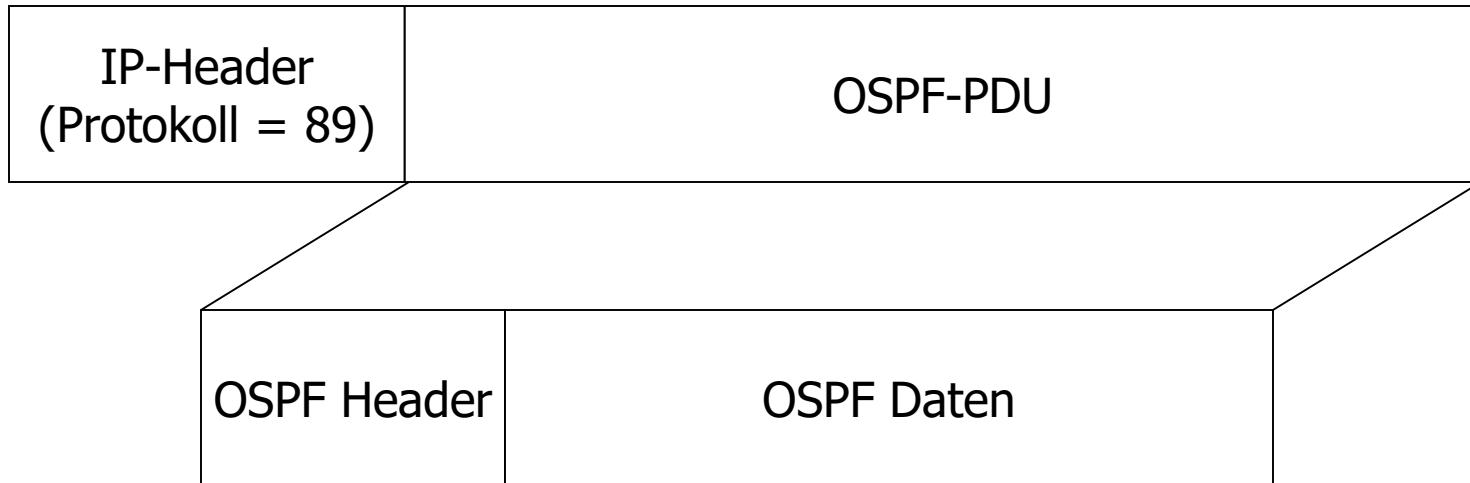
---

- Nachbarschaften werden bei der Initialisierung aufgebaut
  - Senden von Hello-PDUs alle 10 s
  - Hello-PDUs enthalten bekannte Nachbar-Router
  - Generell: OSPF-PDUs werden **nur zwischen Nachbar-Routern** ausgetauscht
- Wenn Nachbarschaft aufgebaut ist, wird alle 40 s eine Hello-PDU als Heartbeat gesendet
  - Bleibt diese aus, wird Nachbar für ausgefallen erklärt
- Die Verteilung von Änderungen der Routing-Tabellen erfolgt immer zu allen Nachbarn

## OSPF-PDUs (1)

---

- Es gibt fünf OSPF-PDU-Typen:
  - **Hello**: Feststellung der Nachbarn, Aufbau von Nachbarschaften
  - **Database Description**: Bekanntgabe der neuesten Daten
  - **Link State Request**: Informationen vom Partner anfordern
  - **Link State Update**: Informationen an Nachbarn verteilen
  - **Link State Acknowledgement**: Bestätigung eines Updates



# Übungsfragen (1)

---

- Welche Routing-Informationen verwaltet RIP und welche OSPF?
  - RIP verwaltet als Distanz-Vektor-Protokoll die Entfernung (Anzahl Hops) und Richtung zum Ziel (Vektor)
  - OSPF verwaltet sog. Links in einer sog. Link-State-Datenbasis. In dieser wird die gesamte Topologie des Netzes verwaltet
  - Bei OSPF sind also alle Routen im Netz genau bekannt, bei RIP nicht
- Warum konvergiert OSPF im Vergleich zu RIPv2 besser?
  - Beide senden Veränderungen sofort (Triggered Update)
  - RIPv2 verarbeitet die Routing-Updates zuerst und dann werden sie an die Nachbarn weitergereicht, bei OSPF ist dies genau umgekehrt

## Übungsfragen (2)

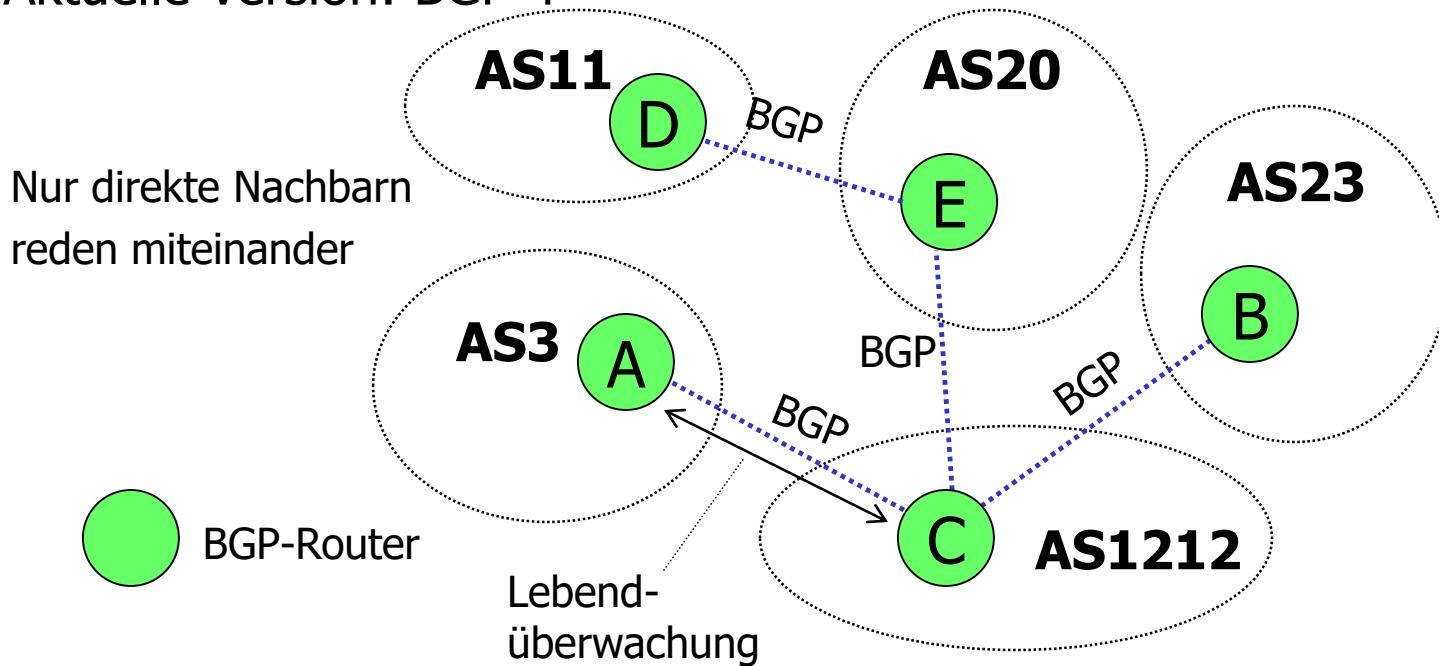
---

- Wie werden bei RIP und bei OSPF die Kosten (Metrik) berechnet?
  - Bei RIP: Anzahl Hops ist die verwendete Metrik
  - Bei OSPF: Nicht festgelegt, Cisco nutzt z.B. die Bandbreite des Gesamtpfades als Metrik, je größer, umso besser. Dies geht, da die Gesamttopologie bekannt ist.

1. Überblick, Routing-Tabellen
2. Routing in Endsystemen und Routern
3. IGP und EGP: Überblick
4. Routing Information Protocol (RIP)
5. Open Shortest Path First (OSPF)
- 6. Border Gateway Protocol (BGP)**
7. Multiprotocol Label Switching (MPLS)
8. Multicast-Routing

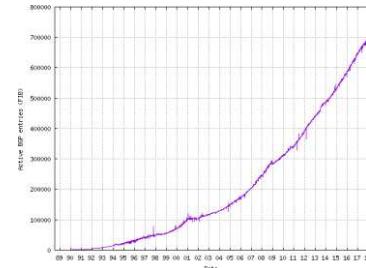
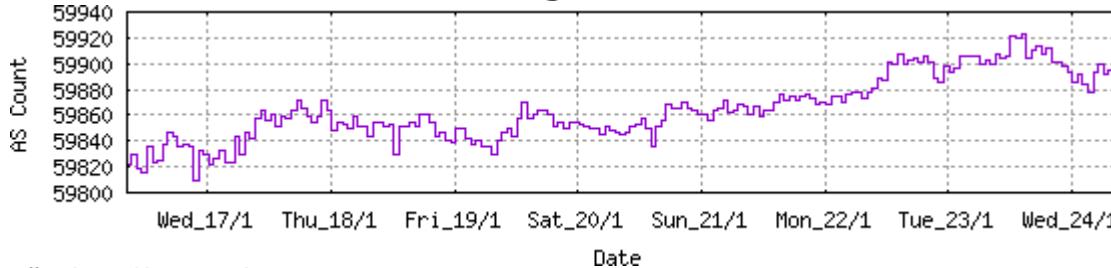
# BGP, Überblick (1)

- Das Border Gateway Protocol (BGP) ist ein EGP (RFC 4271)
  - Derzeit ist im Internet nur BGP im Einsatz
- BGP ermöglicht das Routing zwischen verschiedenen autonomen Systemen (AS)
- Aktuelle Version: BGP 4



## BGP, Überblick (2)

- BGP ist ein **Pfadvektorprotokoll** (Path Vector Protocol), ähnelt dem Distance-Vector-Protokoll
- BGP-Router kennen die besten Route zu anderen AS als **vollständigen Pfad (auf AS-Ebene)**
- Jeder BGP-Router führt eine **Datenbank** mit Routen zu allen erreichbaren autonomen Systemen
- BGP-Routing-Tabellengröße:
  - > 700.000 Einträge und mehr bei mehr als 58.000 AS

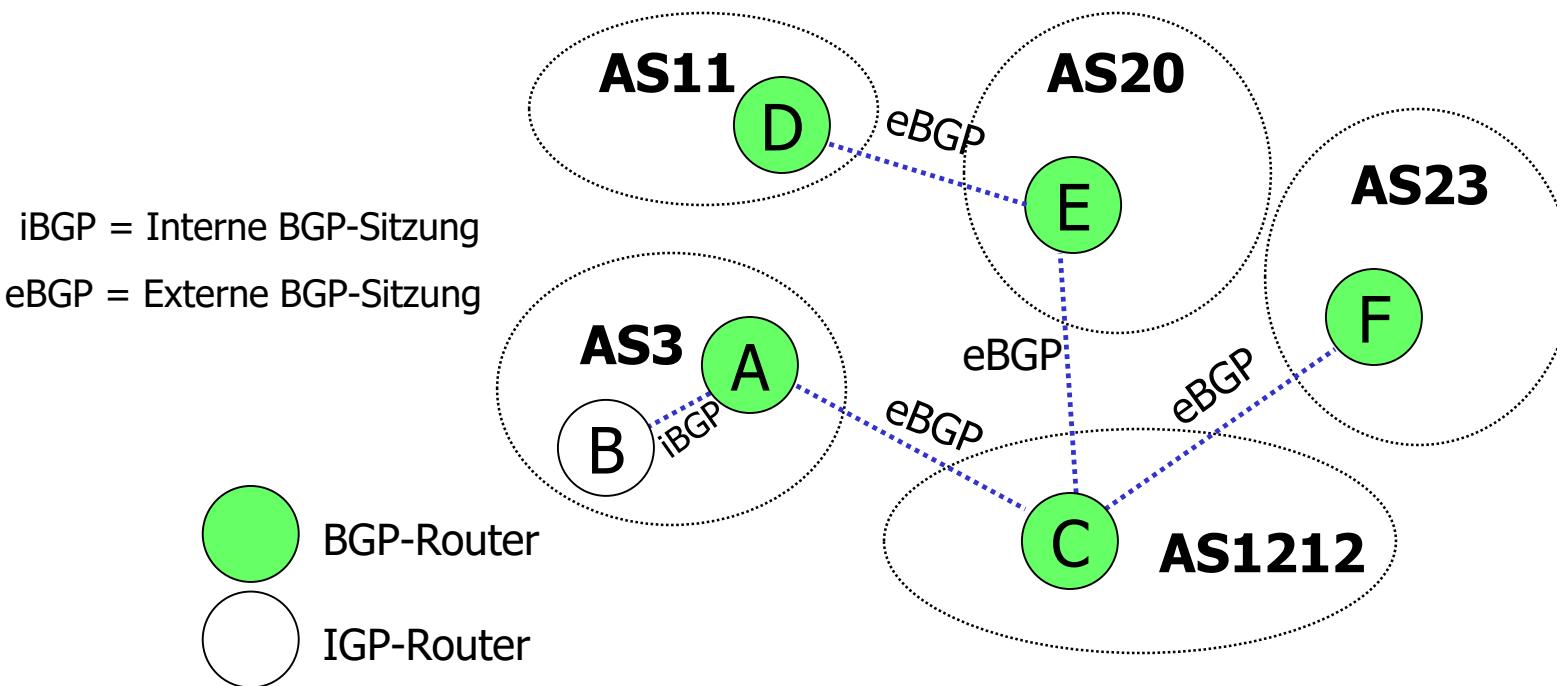


Quelle: <http://www.cidr-report.org/as2.0/#Aggs>,

letzter Zugriff am 24.01.2018

# BGP, Routing-Tabellen

- Routing-Tabelle von D enthält folgende Routen:
  - AS11 – AS20
  - AS11 – AS20 – AS1212
  - AS11 – AS20 – AS1212 – AS23
  - AS11 – AS20 – AS1212 – AS3



## BGP-Router: Zusammenspiel (1)

---

- BGP-Sitzung zwischen Routern zweier AS wird als **externe BGP-Sitzung (eBGP)** bezeichnet
- Ein BGP-Router informiert **periodisch** alle **Nachbar-** BGP-Router über die zu nutzenden Routen
  - UPDATE-PDUs werden versendet, sog. Advertisements
  - Es werden CIDR-Präfixe mit benachbarten BGP-Routern ausgetauscht werden
- Interne BGP-Sitzungen (iBGP) werden zwischen IGP-Routern innerhalb autonomer Systeme aufgebaut
  - Übertragung spezieller BGP-Attribute wie „Next-Hop“
  - Weiterleitung der CIDR-Präfixe, sodass auch interne Router eine Weiterleitung von Paketen in andere AS durchführen können

## BGP-Router: Zusammenspiel (2)

---

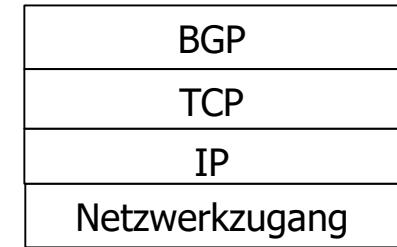
- Zyklen in Routen (Routing-Schleifen) werden bei Übernahme der Information geprüft
  - Es wird auf Routing-Schleifen geprüft, Count-to-Infinity-Problem tritt nicht auf
  - Eigene AS-Nummer darf mehrmals hintereinander in einer Route sein
    - Route kann dadurch künstlich verlängert werden, um Sie unattraktiv zu machen (AS Path Prepending)
- BGP-Router überwachen sich gegenseitig (Heartbeat-Protokoll → KEEPALIVE-PDU)

# BGP, Routing-Policy

---

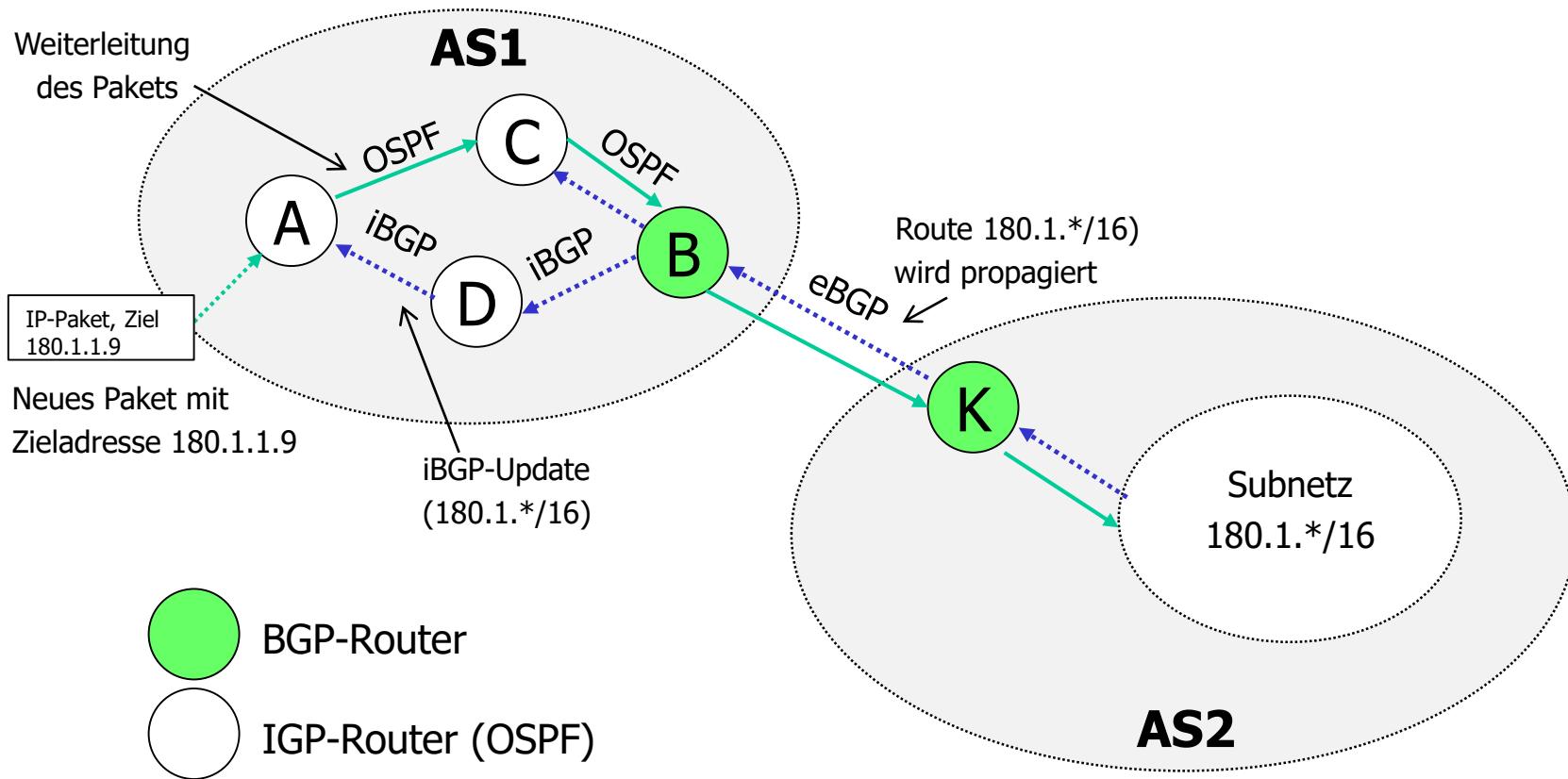
- BGP-Router verwendet zur Auswahl der besten Routen eine **Routing-Policy**
- Routing-Policy (Regeln), Beispiele:
  - Kein Verkehr über einen bestimmten Knoten
  - Sicherheitsaspekte, Kostenaspekte, ...
- Eine Route, die die Regeln verletzt, wird auf „unendlich“ gesetzt
- BGP nutzt TCP (Port 179) als Transportprotokoll für seine Nachrichten (verbindungsorientiert!)

Stack



# Zusammenspiel zwischen IGP- und BGP-Routern

- Next Hop Feld im iBGP-Paket dient der Weiterleitung einer Route zu Subnetz 180.1.\*/16 in AS2
- OSPF ermittelt Weg von A über C zu B (BGP-Gateway) hin zu AS2



1. Überblick, Routing-Tabellen
2. Routing in Endsystemen und Routern
3. IGP und EGP: Überblick
4. Routing Information Protocol (RIP)
5. Open Shortest Path First (OSPF)
6. Border Gateway Protocol (BGP)
- 7. Multiprotocol Label Switching (MPLS)**
8. Multicast-Routing

# MPLS – Multiprotocol Label Switching (1)

---

- „Verbindungsorientierte“ Übertragung von IP-Paketen entlang eines zuvor vereinbarten (signalisierten) Pfads
- Entlastung der Routing-Tabellen durch Nutzung der Vorteile verbindungsorientierter Vermittlung
  - Vorsignalisierte Datenpfade
  - Router muss nicht bei jedem Paket ermitteln, wie es weitergeroutet wird
- Nutzung in erster Linie bei Netzwerkbetreibern bzw. innerhalb von autonomen Systemen für das interne Routing

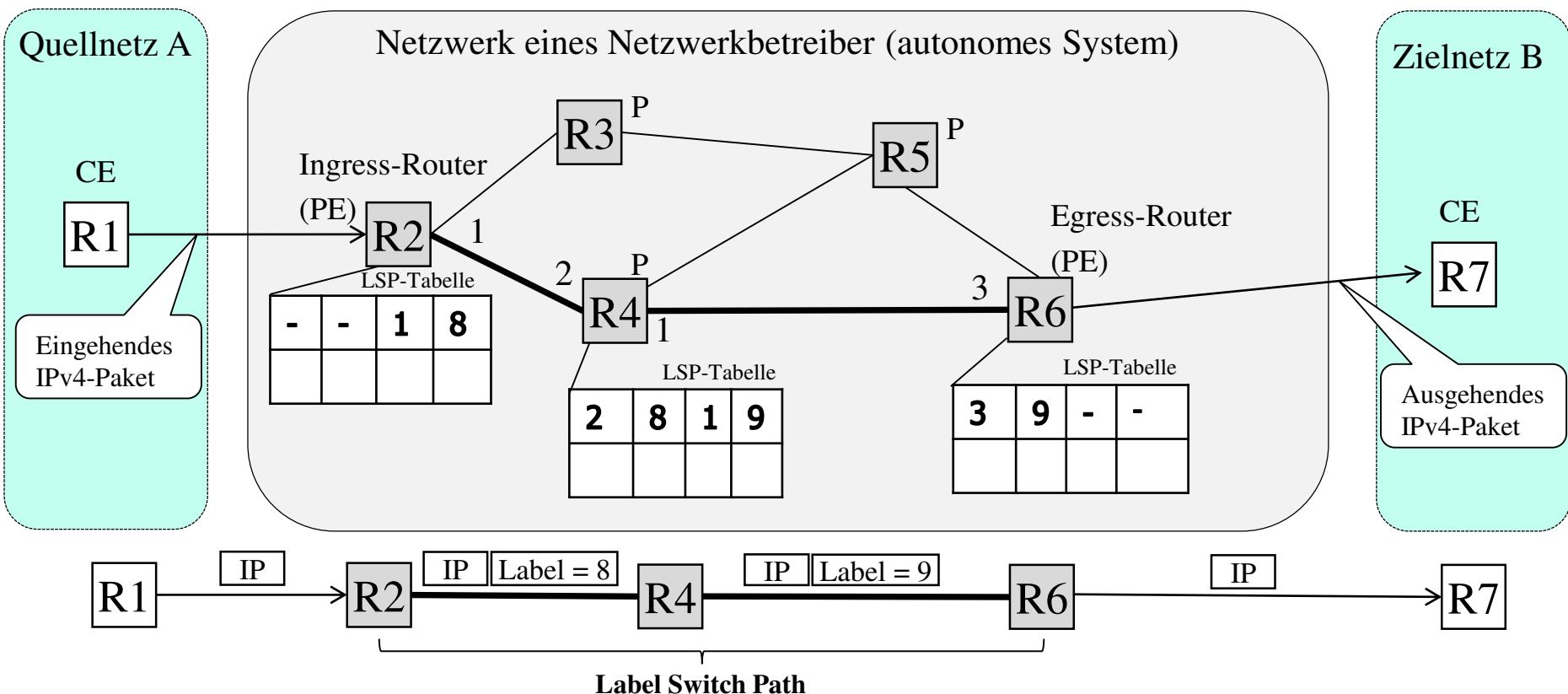
Schicht 2	MPLS Header	IP Header	TCP Header	Nutzlast
-----------	-------------	-----------	------------	----------

# MPLS – Multiprotocol Label Switching (2)

CE = Customer Edge Router  
 PE = Provider Edge Router  
 P = Provider Router

LSP-Tabelle = MPLS-Forwarding-Tabelle (Aufbau)

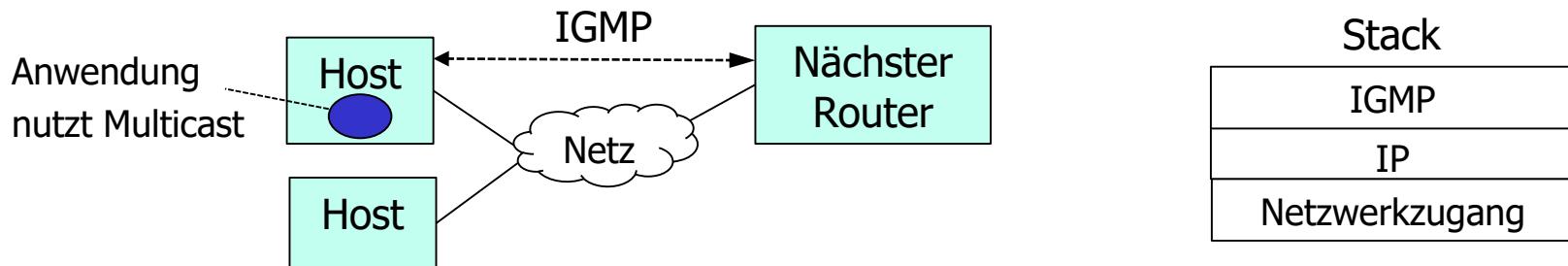
In-Interface	In- Label	Out-Interface	Out-Label



1. Überblick, Routing-Tabellen
2. Routing in Endsystemen und Routern
3. IGP und EGP: Überblick
4. Routing Information Protocol (RIP)
5. Open Shortest Path First (OSPF)
6. Border Gateway Protocol (BGP)
7. Multiprotocol Label Switching (MPLS)
- 8. Multicast-Routing**

# Multicasting und IGMP

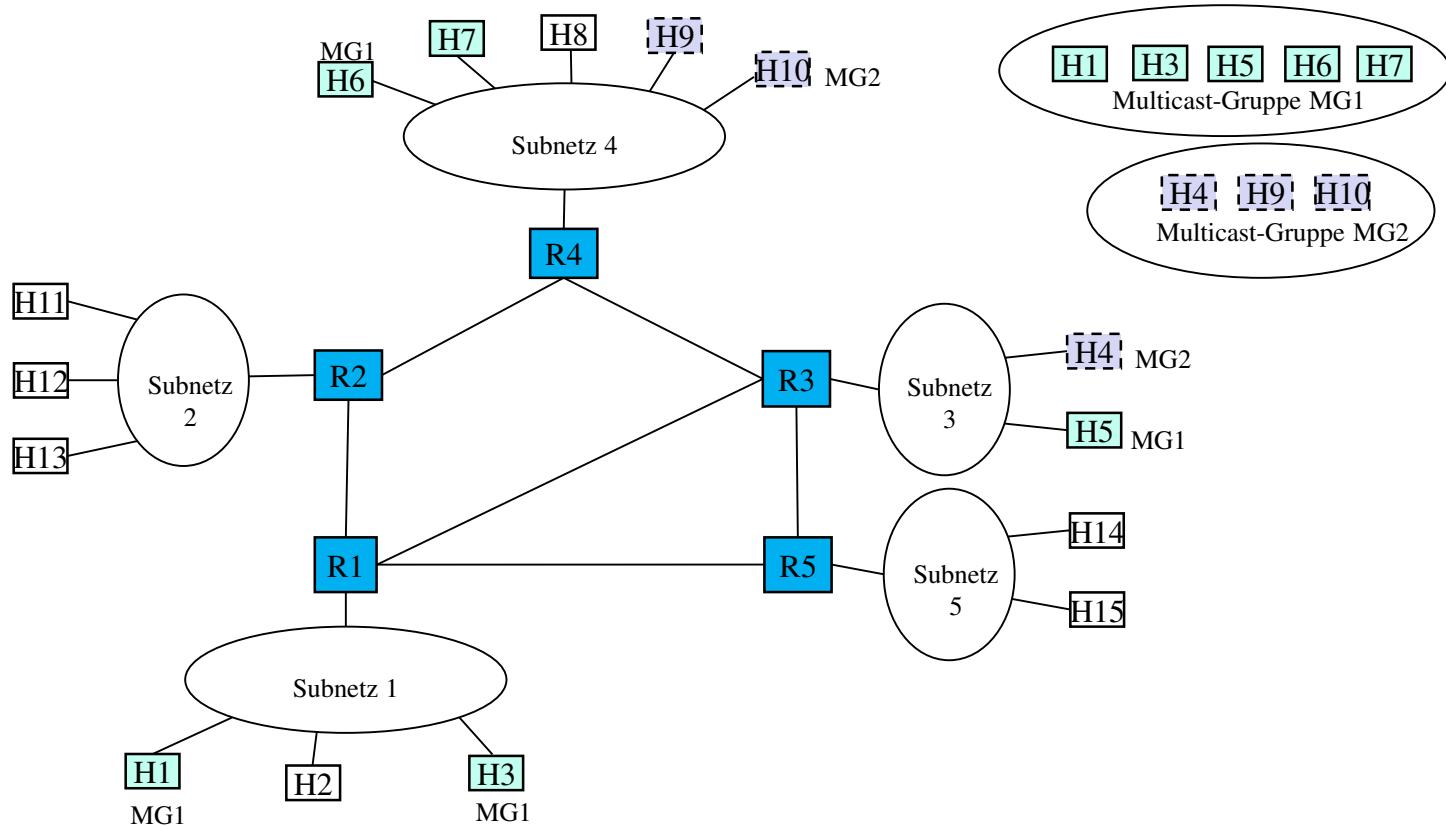
- Das IGMP-Protokoll verwaltet Gruppenzugehörigkeiten für Multicastgruppen zwischen Hosts und dem nächstgelegenen Router
- In IGMP sind drei Nachrichtentypen definiert:
  - *membership\_query* dient einem Router dazu, in einem Subnetz zu erfragen, welche Hosts in welchen Multicast-Gruppen sind
  - Hosts antworten auf eine Anfrage mit einer *membership\_report*
  - Mit *leave\_group* können Hosts den Austritt aus einer Gruppe bekanntgeben



# Multicast Routing

## Problemstellung

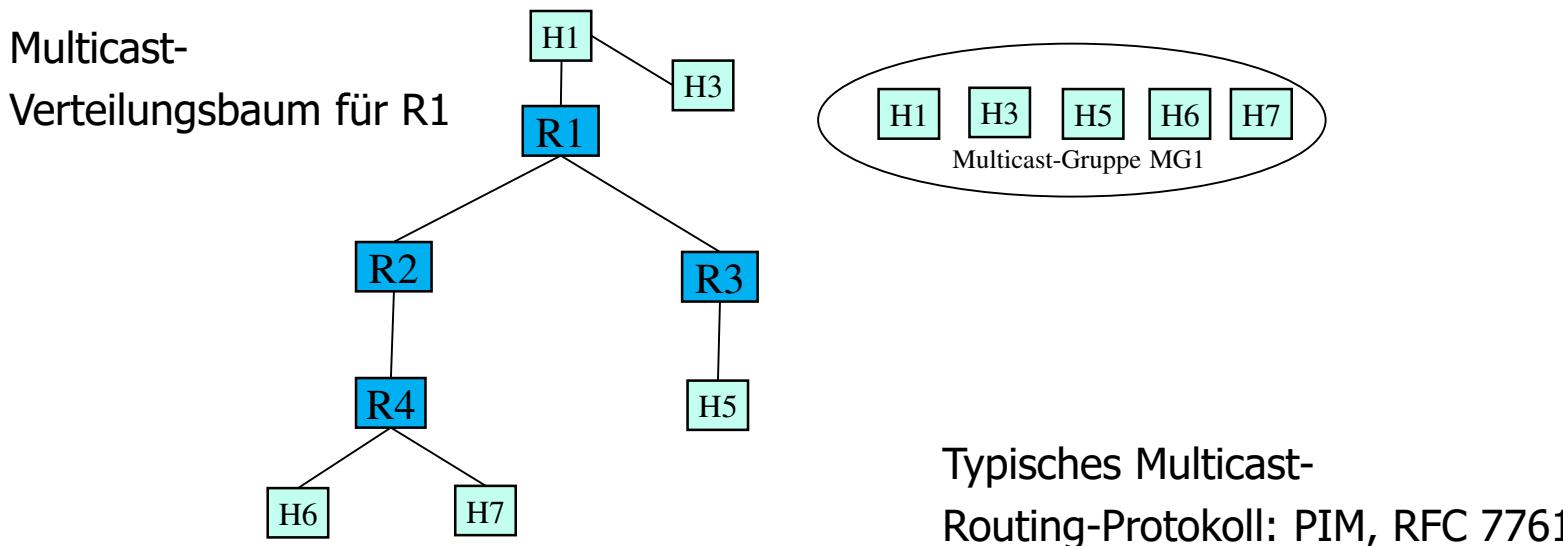
- Multicast-Nachrichten müssen im Internet geroutet werden
- Austausch von Routing-Informationen (Gruppenzugehörigkeiten) erfolgt über Multicast-Routing-Protokolle



# Multicast Routing

## Reverse Path Forwarding (RPF)

- Routing-Schleifen und unkontrolliertes Flooding von Multicasts sind zu vermeiden → Nutzung von **Reverse Path Forwarding**
- Jeder Router muss einen **Multicast-Verteilungsbaum** ermitteln
- Wenn das Paket von einer anderen Netzwerkschnittstelle empfangen wird, als von der in der Forwarding-Tabelle eingetragenen, wird das Paket verworfen



# Rückblick

---

- ✓ Überblick, Routing-Tabellen
- ✓ Routing in Endsystemen und Routern
- ✓ IGP und EGP: Überblick
- ✓ Routing Information Protocol (RIP)
- ✓ Open Shortest Path First (OSPF)
- ✓ Border Gateway Protocol (BGP)
- ✓ Multiprotocol Label Switching (MPLS)
- ✓ Multicast-Routing

- 1 Kommunikationssysteme und verteilte Anwendungen
- 2 Grundlagen der Transportschicht
- 3 Transportprotokolle TCP und UDP
- 4 Grundlagen der Vermittlungsschicht
- 5 Internet und Internet Protocol (IP)
- 6 Routing-Verfahren und -Protokolle
- 7 **Internet-Steuerprotokolle und DNS**
- 8 Internet Protocol Version 6 (IPv6)
- 9 Netzwerkschnittstelle

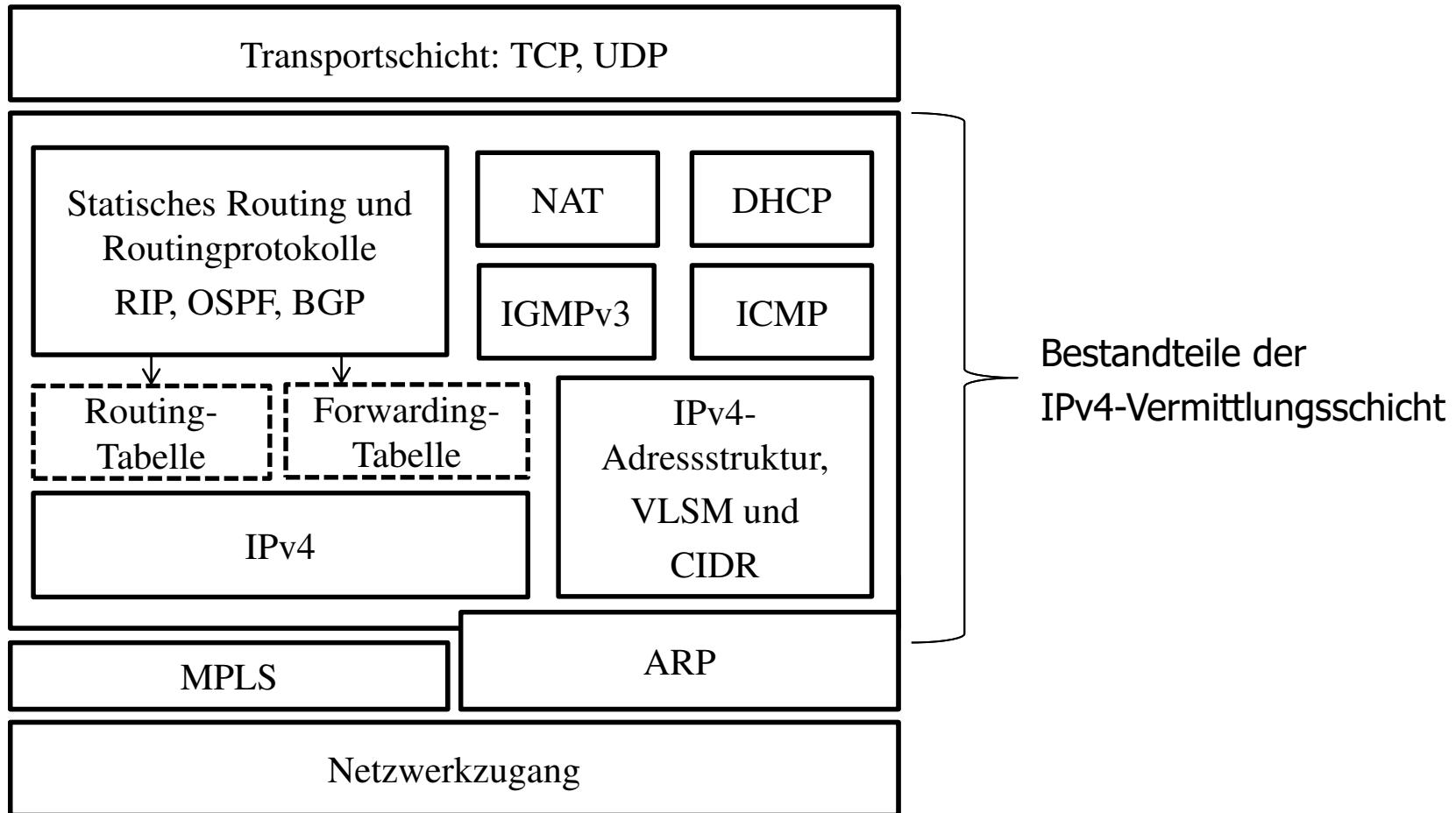
## 1. Steuerprotokolle

- ICMP
- ARP und RARP
- NAT
- DHCP

## 2. Domain Name System

- Namensraum und Organisation
- Root-Name-Server
- Adressauflösung
- DNS-PDU

# Überblick: Die Internet-Vermittlungsschicht



# ICMP: Einführung

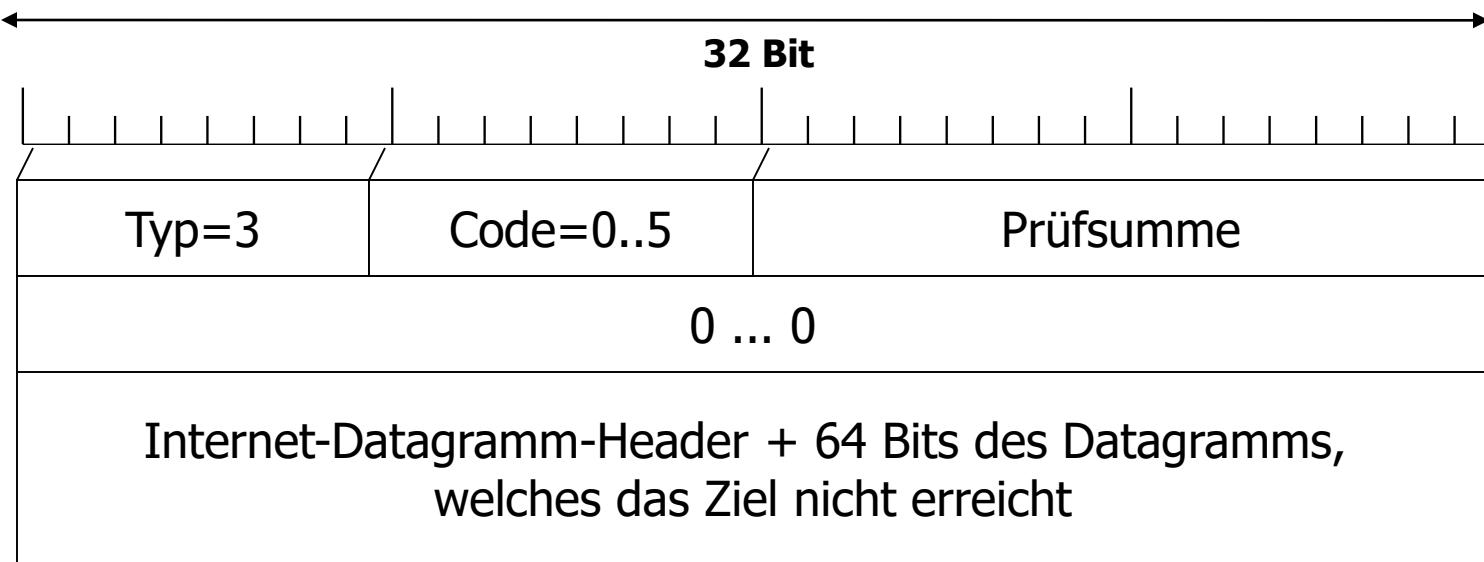
---

- ICMP (Internet Control Message Protocol, RFC 792)
    - Dient der Übertragung von unerwarteten Ereignissen und für Testzwecke
    - Beispiel 1: Ein Netzwerk ist nicht erreichbar: Ein IP-Router sendet in diesem Fall die ICMP-PDU „Network unreachable“ an den Absender des IP-Pakets
    - Beispiel 2: Das **ping**-Kommando verwendet z.B. ICMP-PDUs Pakettypen 8 und 0 (Echo Request, Echo Reply)
    - Beispiel 3: Das Kommando **traceroute** (tracert) nutzt ICMP Echo-Requests, Echo-Responses und Pakettyp=11 (Time-to-live exceeded)
  - ICMP-Nachrichten werden in IP-Datagrammen versendet
-

## ICMP: PDU

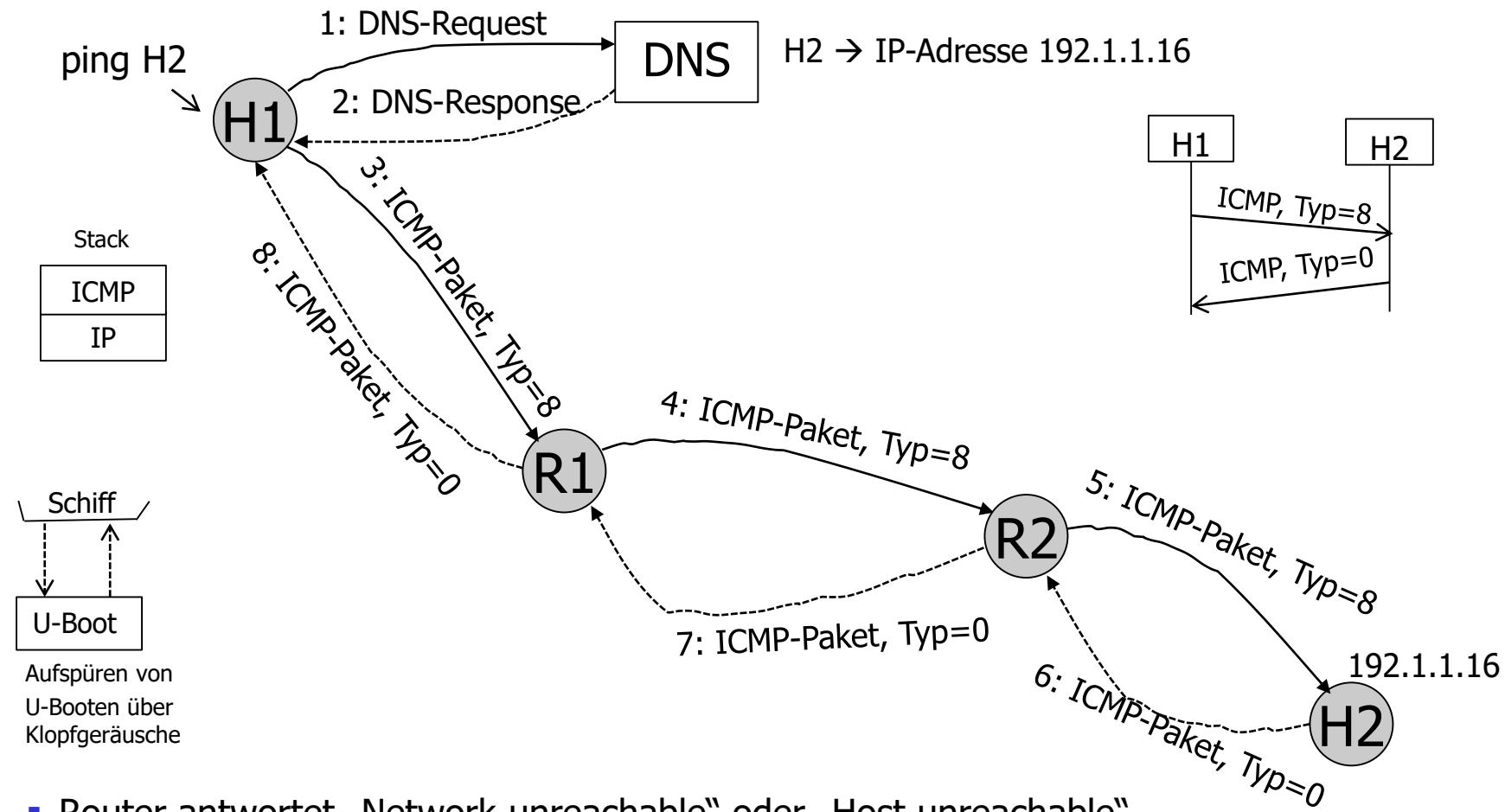
---

- ICMP-Beispiel: **Destination unreachable** → Ein Router kann ein Datagramm nicht ausliefern



- Router sendet ICMP-Nachricht an den Absender
- Code: 0 = Netzwerk nicht erreichbar, 1 = Rechner nicht erreichbar, ...

# Beispielablauf von ping



- Router antwortet „Network unreachable“ oder „Host unreachable“ falls H2 nicht erreichbar ist

## Einschub: ping deaktivieren

---

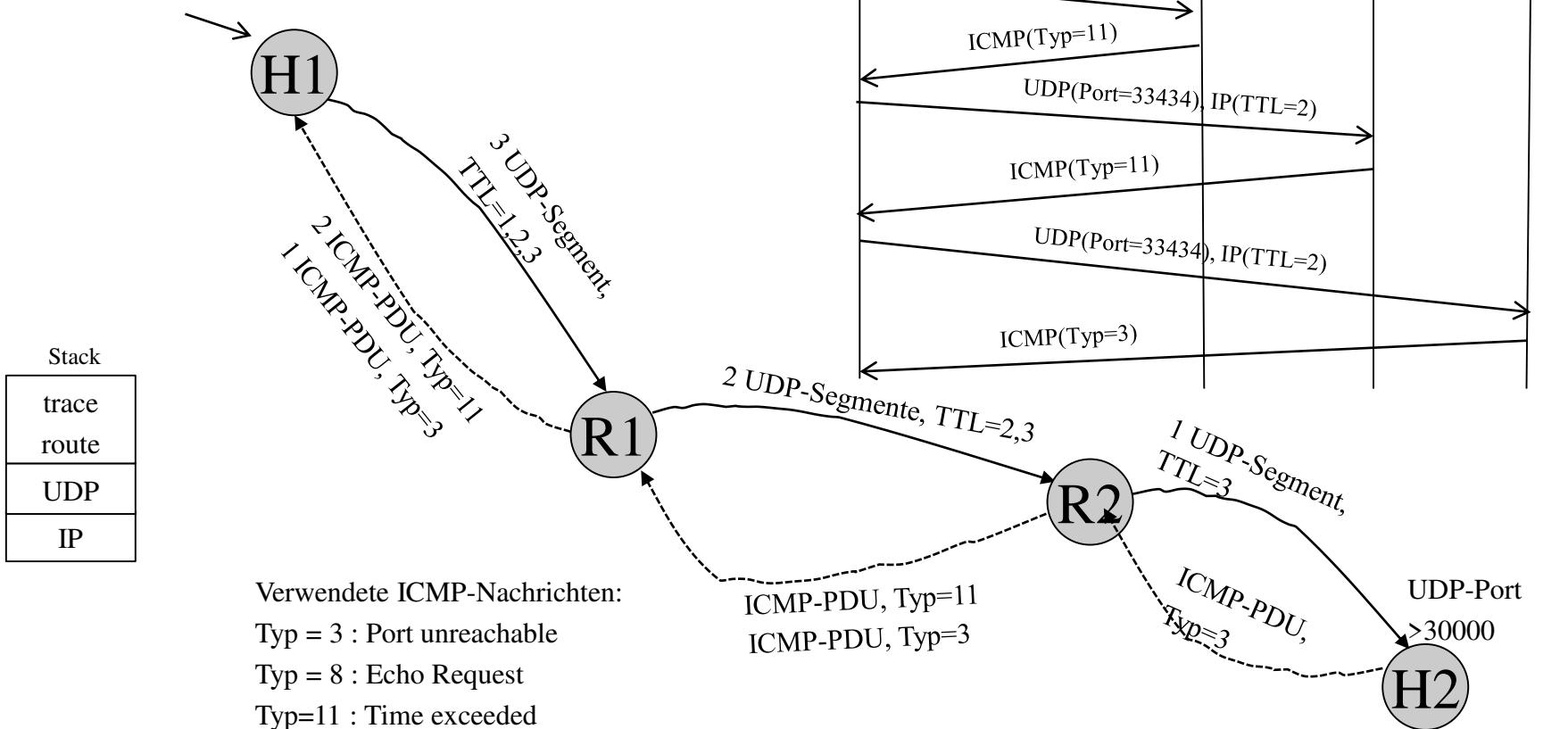
- Manchmal ist es aus Sicherheitsgründen hilfreich, ping (Echo-Requests) auf einem Rechner zu deaktivieren
- Ausschalten unter Linux:
  - echo 1 > /proc/sys/net/ipv4/icmp\_echo\_ignore\_all
- Einschalten unter Linux:
  - echo 0 > /proc/sys/net/ipv4/icmp\_echo\_ignore\_all
- Unter Windows:
  - Firewall-Einstellung → Erweiterte Einstellungen → Eingehende Regeln
    - Datei- und Druckerfreigabe (Echoanforderung - ICMPv4/6 eingehend) deaktivieren

# Beispielablauf von traceroute

- **traceroute** (Unix) bzw. **tracert** oder **pathping** (Windows)

traceroute -p 33434 H2

Default-Port: 33434



# Ausgabe von traceroute (tracert)

---

C:\Users\mandl>**tracert www.hm.edu** (von einem anderen Netzwerk aus)

Routenverfolgung zu www.hm.edu [129.187.244.229] über maximal 30 Abschnitte:

```
1  <1 ms  <1 ms  <1 ms  grandcentral.isys-software.de [192.168.2.1]
2  <1 ms  <1 ms  <1 ms  192.168.250.1
3  36 ms  36 ms  35 ms  217.5.98.12
4  36 ms  35 ms  35 ms  217.237.152.58
5  52 ms  54 ms  52 ms  l-ea4-i.L.DE.NET.DTAG.DE [62.154.89.230]
6  48 ms  48 ms  47 ms  80.156.160.142
7  52 ms  52 ms  52 ms  cr-gar1-hundredgige0-5-0-0.x-win.dfn.de [188.1.144.250]
8  52 ms  52 ms  52 ms  kr-gar33-10.x-win.dfn.de [188.1.37.90]
9  53 ms  53 ms  53 ms  vl-3002.csr4-kb1.lrz.de [129.187.0.134]
10 52 ms  52 ms  52 ms  vl-3003.csr3-kb1.lrz.de [129.187.0.138]
11 53 ms  52 ms  55 ms  vl-3013.csr1-kra.lrz.de [129.187.0.162]
12 53 ms  53 ms  53 ms  hm.edu [129.187.244.229]
```

Ablaufverfolgung beendet.

- Je drei Versuche werden je TTL gemacht → 3 Zeitangaben für die RTTs
- Ausgabe von „\*“ bedeutet eine Zeitüberschreitung (5 Sekunden)

# Path MTU Discovery

## Aufgabe und Zielsetzung

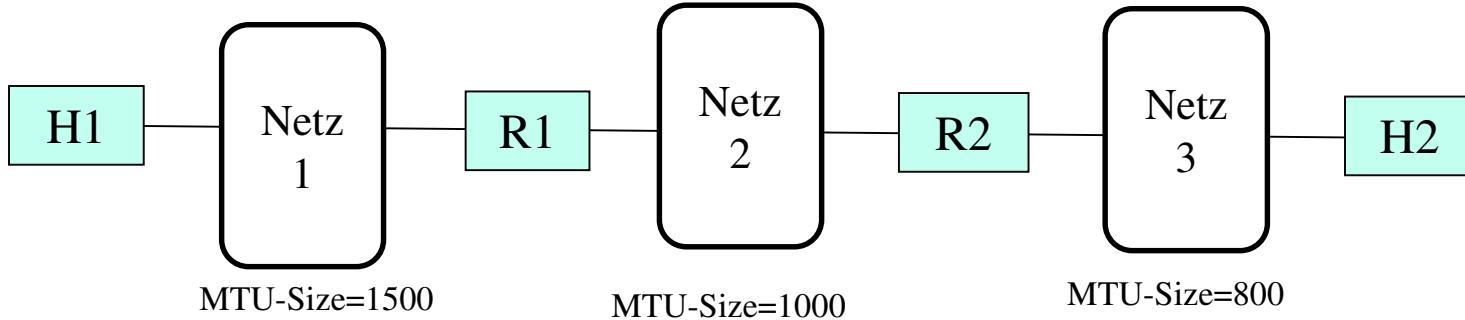
---

- In jedem Teilnetz, das ein IPv4-Paket von der Quelle zum Ziel passiert, gibt die jeweilige Netzwerkzugriffsschicht eine Obergrenze für die Frame-Länge vor → **MTU-Size**
- Aufgabe der Path MTU Discovery ist es, eine MTU-Size herausfinden, die für den ganzen Pfad passt
- Ziel ist die **Vermeidung** von IPv4-Fragmentierung
- Geregelt in RFC 1191 für IPv4, RFC 1981 für IPv6

# Path MTU Discovery

## Beispielnetz

---

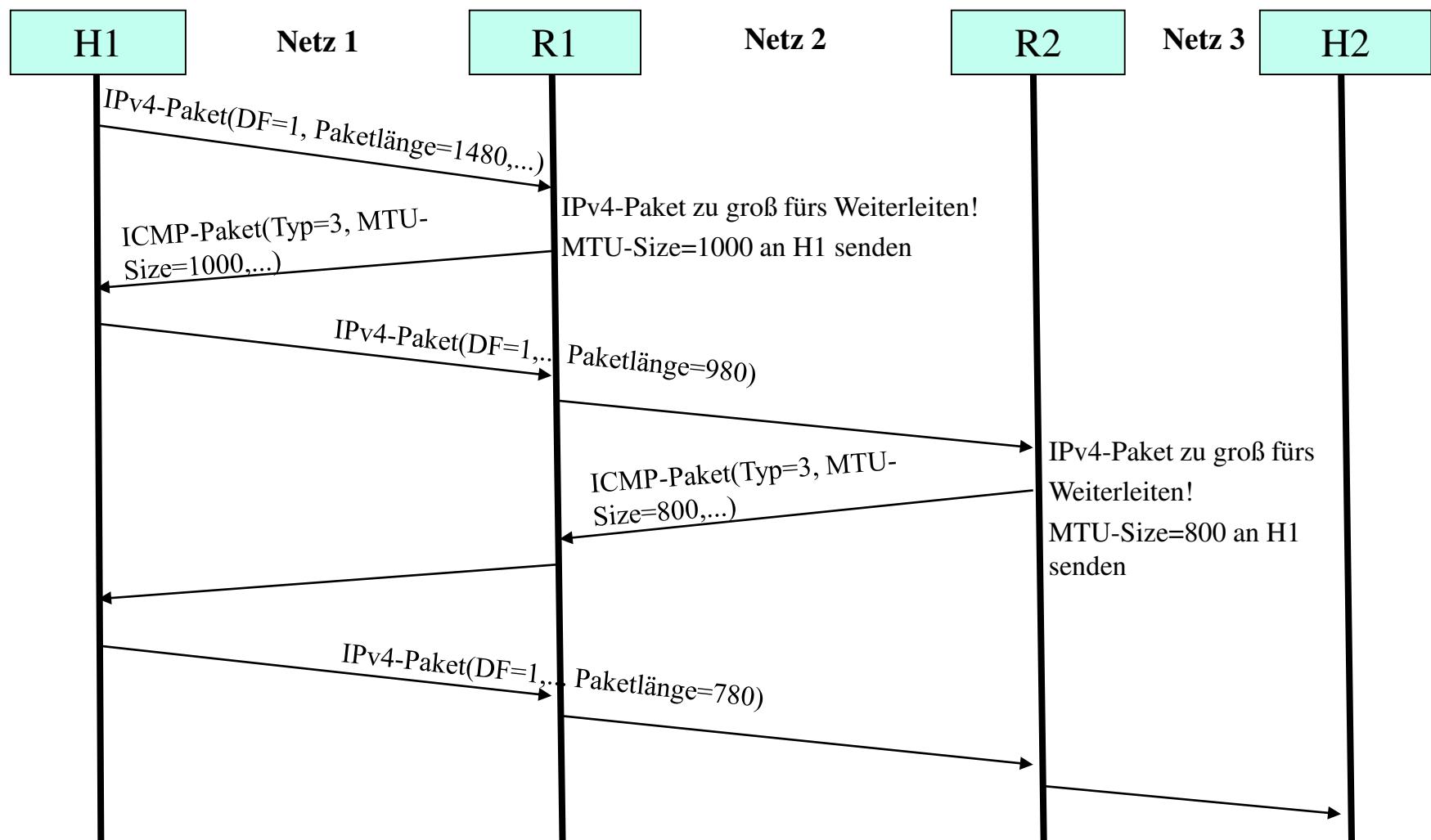


### ■ Ablauf

- Host sendet ein größeres IP-Paket
- Im IP-Header DF=1 (Don't fragment) setzen
- Prüfung für jede Verbindung auf der Strecke zum Zielrechner mit lokaler MTU-Size beginnen
- MTU-Size wird von Routern über ICMP-Paket (Typ=3, „Fragmentation needed“) zurückgesendet, wenn Fragmentierung erforderlich wäre

# Path MTU Discovery

## Ablauf im Beispielnetz



# ARP

---

- ARP (Address Resolution Protocol), RFC 826
  - ARP dient dem **dynamischen Mapping** von IP-Adressen auf Schicht-2-Adressen (MAC-Adressen)
  - Jeder Host kennt seine eigene Schicht-2-Adresse, nicht aber die Adressen der anderen Hosts
  - Jeder Host führt einen **ARP-Cache** und merkt sich darin Schicht-2-Adressen, die über ARP im **IP-Broadcasting** erfragt werden können → **Periodisches Löschen vermeidet Inkonsistenz!**
  - Ist der Zielhost nicht gespeichert, wird ein **ARP-Broadcast** mit der IP-Zieladresse als Parameter versendet
  - Der Zielrechner antwortet mit einem **ARP-Reply** (MAC-Adresse)
  - IP-Router übernehmen Rolle des **ARP-Proxy**

# ARP-Cache

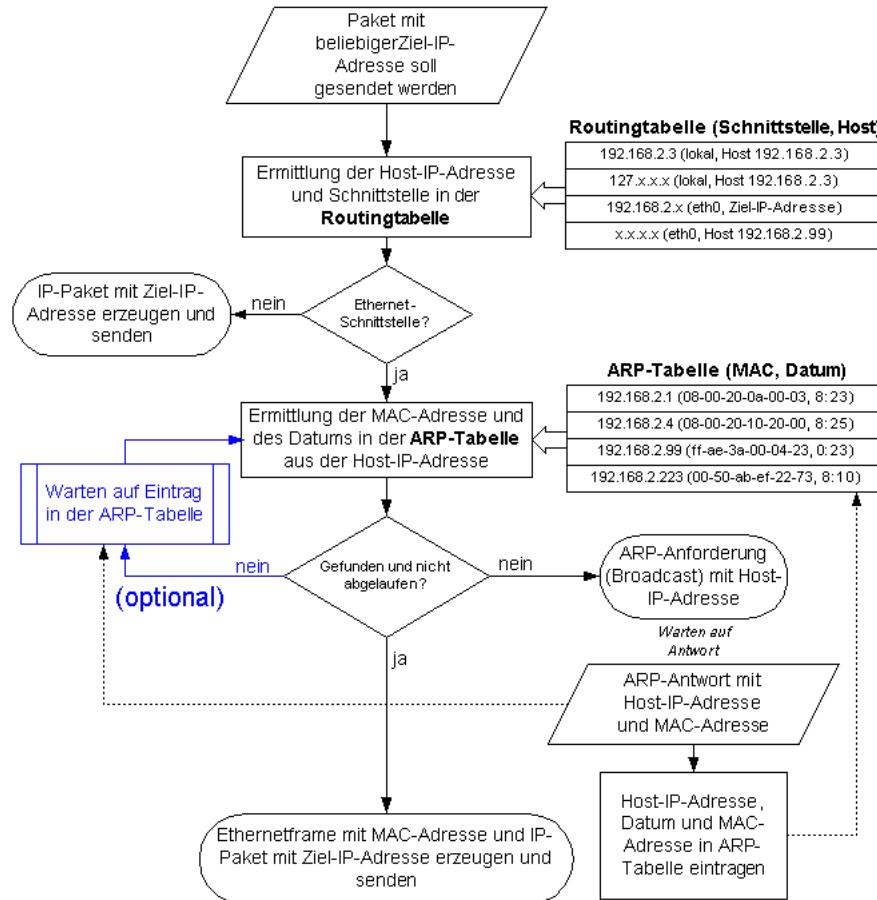
---

- arp-Kommando zum Anzeigen des aktuellen ARP-Cache: *arp -a (Windows)*
- arp-Kommando zum Löschen eines Eintrags aus dem ARP-Cache: *sudo arp -d 192.168.178.1 (MAC OS)*
- Man kann in den Cache auch statische Adressen eintragen: *arp -s <ip-Adresse> <MAC-Adresse>*

C:\>**arp -a**

Schnittstelle: 192.168.2.116 --- 0x2		
Internetadresse	Physikal. Adresse	Typ
192.168.2.1	00-50-fc-cb-7e-da	dynamisch
192.168.2.14	00-e0-4c-10-17-32	dynamisch
192.168.2.250	08-00-37-31-de-ae	dynamisch

# ARP-Funktionsweise



Quelle: [http://de.wikipedia.org/wiki/Address\\_Resolution\\_Protocol](http://de.wikipedia.org/wiki/Address_Resolution_Protocol) (Zugriff am 20.01.2018)

# RARP

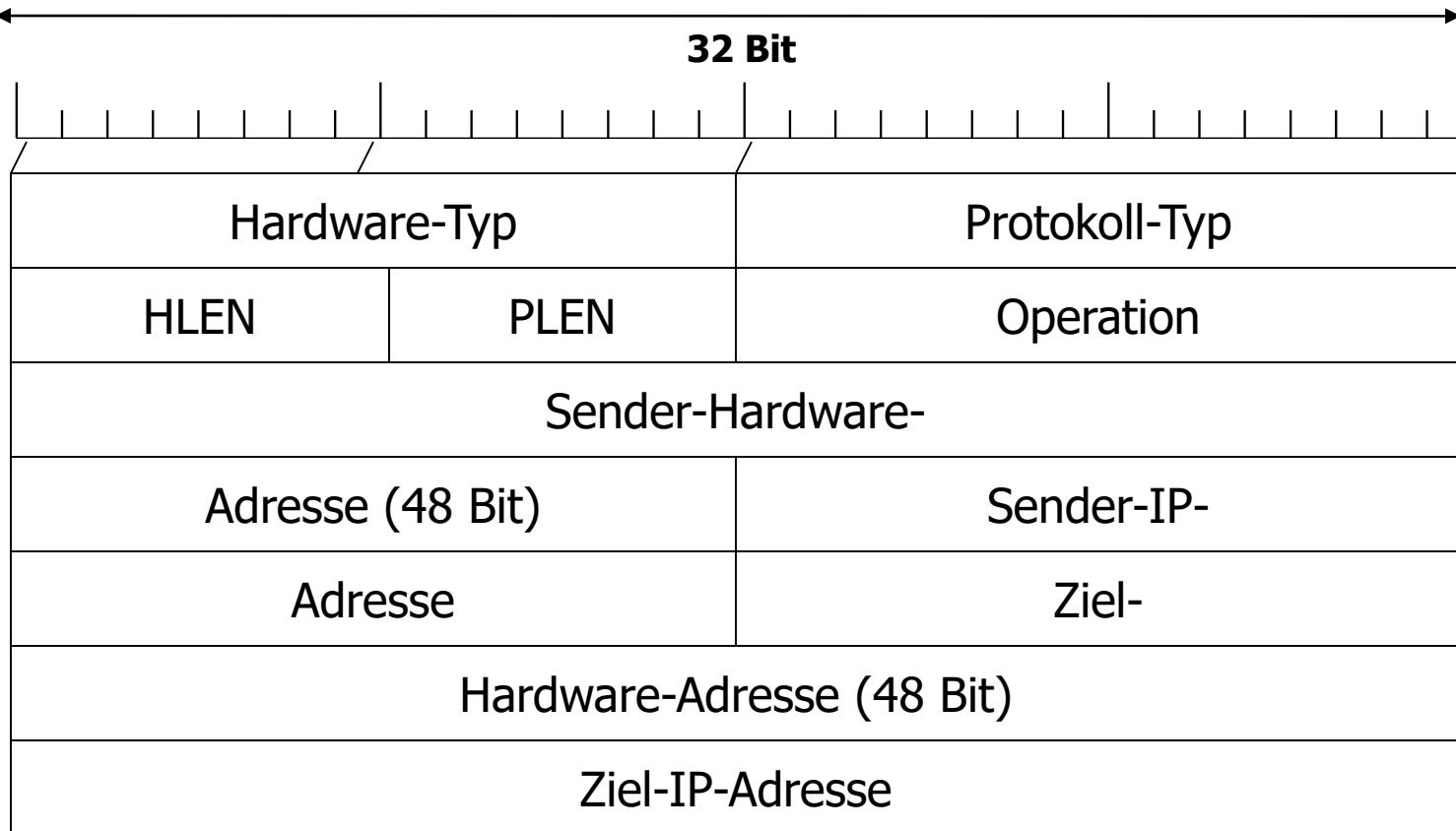
---

- RARP = Reverse ARP
  - RARP wird verwendet, wenn die eigene IP-Adresse eines Hosts nicht bekannt ist, aber benötigt wird
  - RARP sendet RARP-Request mit der eigenen MAC-Adresse als Broadcast
  - Anwendungsfall:
    - Plattenlose Desktop-Arbeitsplätze, die beim Booten Ihre IP-Adresse ermitteln wollen

# ARP-PDU

---

- Nachrichtenformat eines ARP/RARP-Pakets



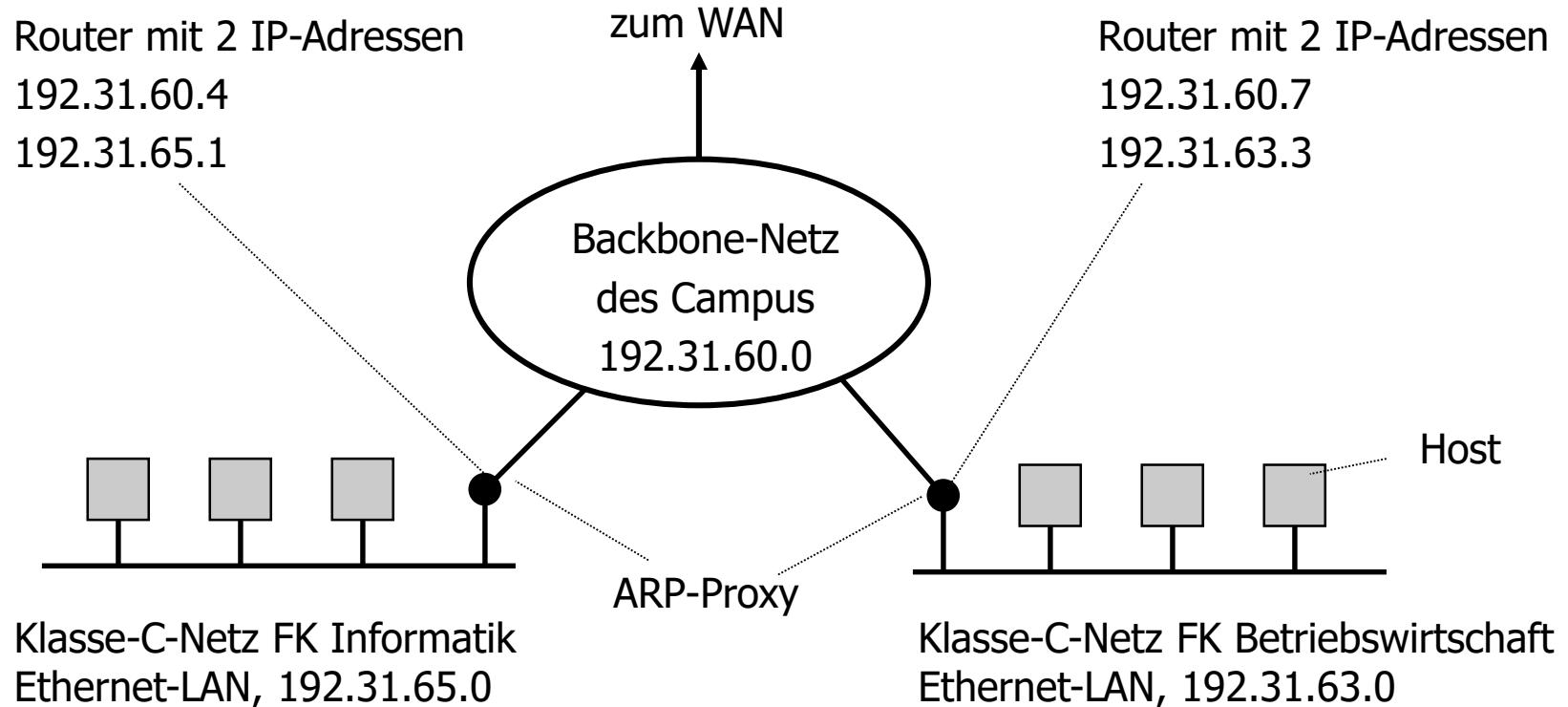
# ARP-PDU

---

- **Hardware-Typ:**
  - Hardware-Typ, 1 = Ethernet
- **Protokoll-Typ:**
  - Typ des High-Level-Protokolls, X`0800` = IP
- **HLEN:**
  - Hardware-Adressenlänge
- **PLEN:**
  - IP-Adressenlänge
- **Operation:**
  - 1 = ARP-Request
  - 2 = ARP-Response
  - 3 = RARP-Request
  - 4 = RARP-Response
- ...

# ARP: Beispiel

- Typisches Netzbeispiel: Campusnetz (nach Tanenbaum)
- ARP-Proxy im Zielhost ermittelt MAC-Adresse des Zielhosts
- Lokaler ARP-Proxy ist Ansprechpartner für entfernte Hosts



# Network Address Translation (NAT)

## Einführung

---

- NAT dient dazu, Netzwerkadressen einzusparen
  - NAT ist auch eine **Sicherheitsmaßnahme** (aber nicht alleine für sich ausreichend und auch nicht primär dafür gedacht)
  - Für ein Netz (Unternehmensnetz) benötigt man **nur noch wenige offizielle IP-Adressen**
  - Intern kann dann eine beliebige, nach außen nicht sichtbare, Netzwerksnummer verwendet werden
- Private IP-Adressen!

# Network Address Translation (NAT)

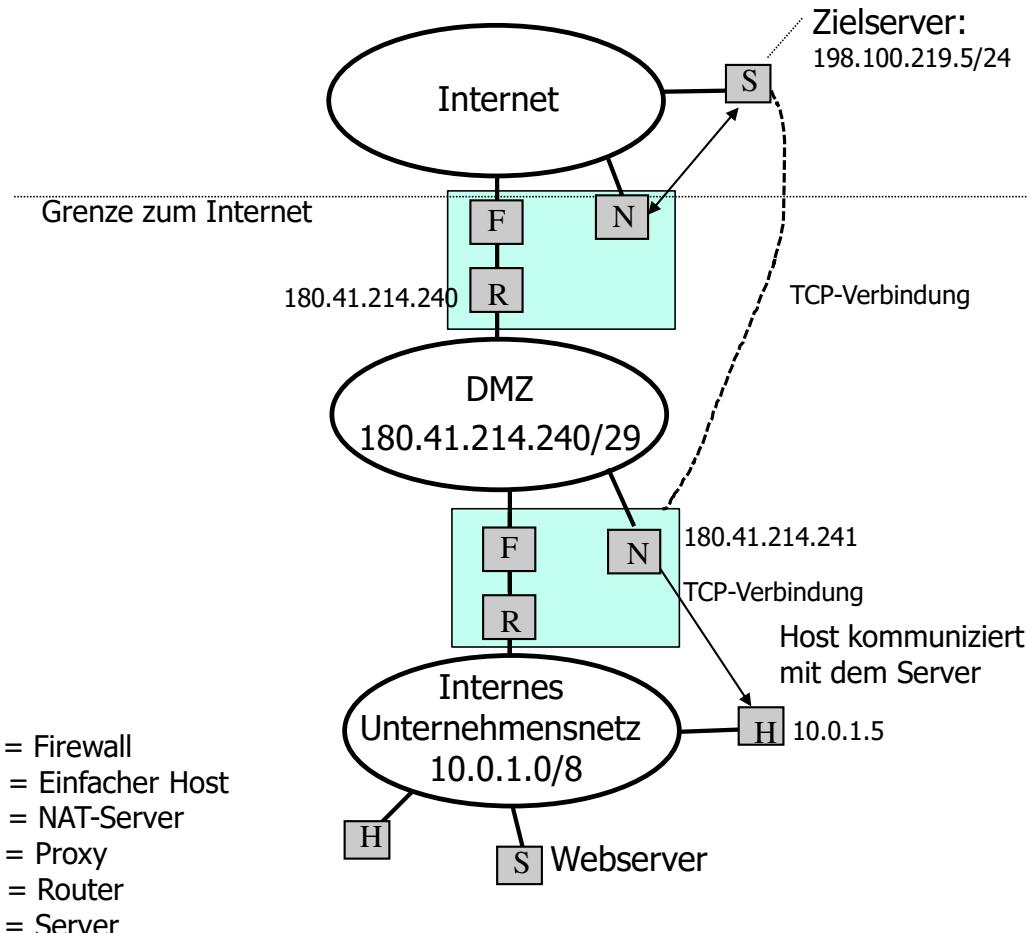
## Funktionsweise

---

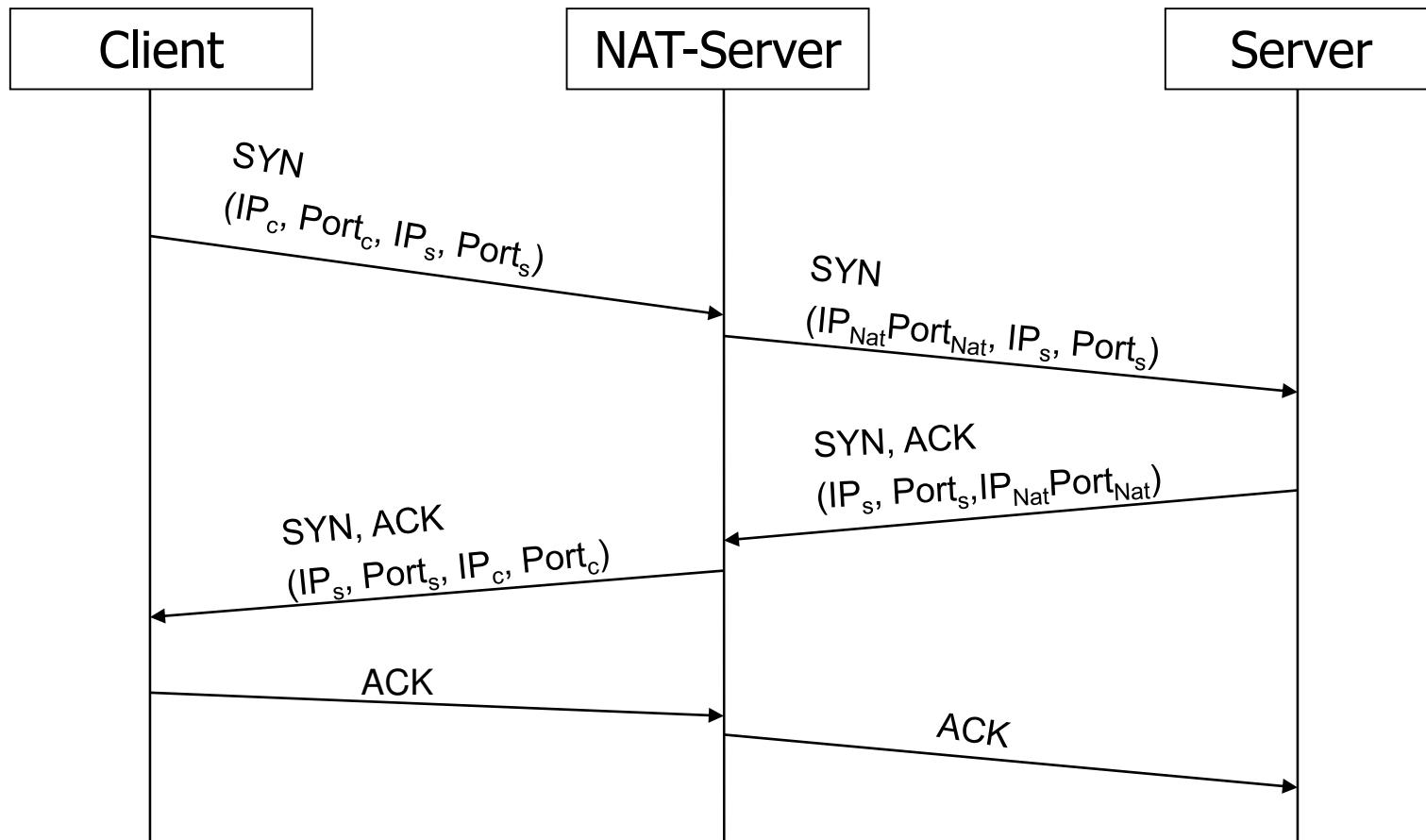
- IP-Router bzw. NAT-Server „**mappen**“ bei NAT ankommende Pakete auf interne Hostadressen und auch die Ports und umgekehrt
- IP-Router arbeiten nach außen als **Stellvertreter** (Proxies) für alle internen Hosts
  - Source-NAT (SNAT): Adresse des lokalen Hosts wird verändert
  - Destination-NAT (DNAT): Adresse des Zielhosts wird verändert
- Verschiedene Varianten von NAT verfügbar

# Network Address Translation Beispiel

---



# Network Address Translation: Nachrichtenfluss beim Verbindungsaufbau von innen nach außen



- Diskussion: End-to-End-Beziehung

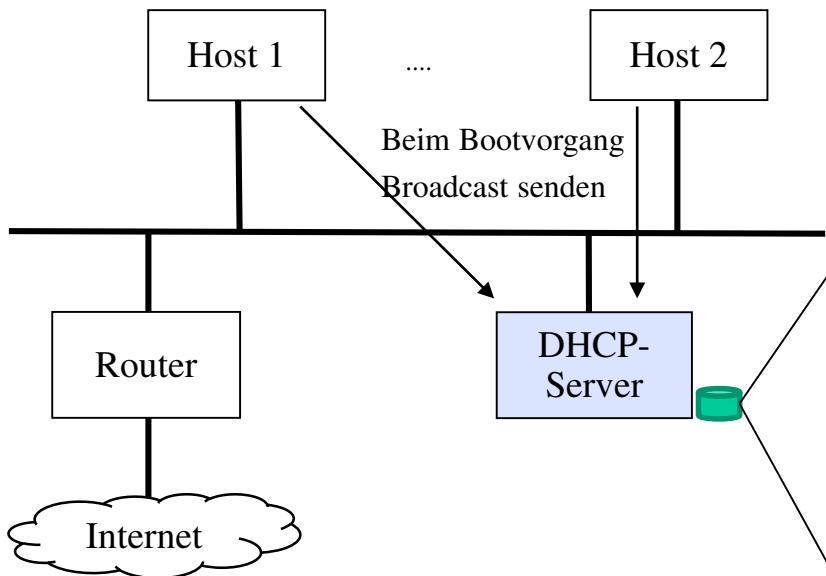
# DHCP: Einführung

---

- Manuelle Netzwerkkonfiguration ist schon bei kleinen Netzen ein Problem
- Dynamic Host Configuration Protocol schafft Abhilfe
  - Hosts müssen Adressen nicht mehr kennen, sie werden dynamisch beim Booten besorgt
- Dynamische Vergabe von IP-Adressen und weiteren Netzwerk-Parametern über einen DHCP-Server:
  - Subnetzmaske
  - DNS-Server-Adresse
  - IP-Router-Adresse
  - ...

# DHCP: Beispielnetz

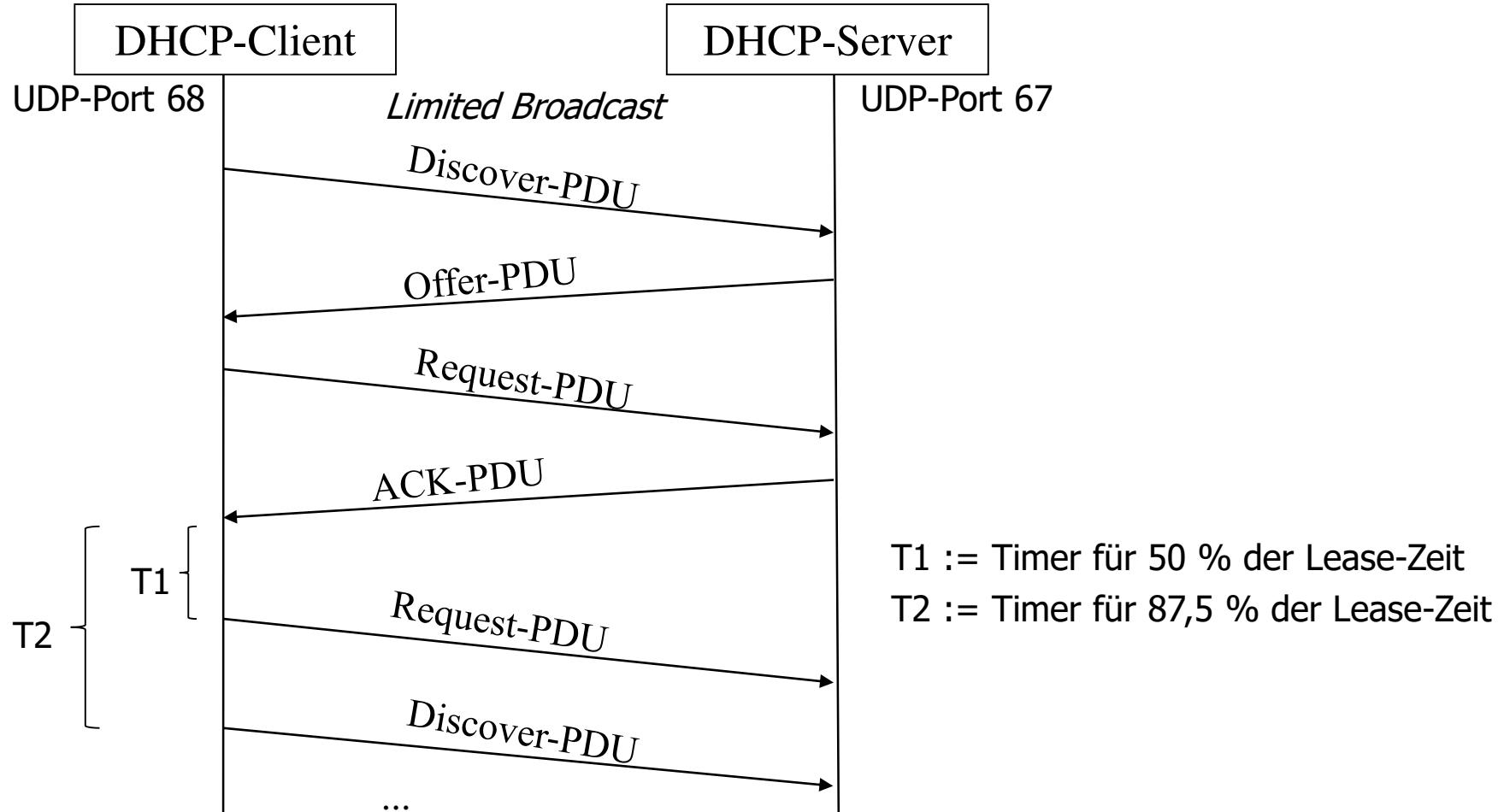
- Meist beziehen nur Arbeitsplatzrechner ihre IP-Adresse vom DHCP-Server



## Konfigurationsdatei z. B.: /etc/dhcpd.conf

```
default-lease-time 600; # 10 Minuten  
max-lease-time 7200; # 2 Stunden  
  
option domain-name "isys.com";  
option domain-name-servers 192.168.1.1 192.168.1.2;  
option broadcast-address 192.168.1.255;  
option routers 192.168.1.254;  
option subnet-mask 255.255.255.0;  
  
subnet 192.168.1.0 netmask 255.255.255.0  
{  
    range 192.168.1.10 192.168.1.20;  
    range 192.168.1.100 192.168.1.200;  
}  
  
host mandl  
    hardware ethernet 00:00:45:12:EE:E4;  
    fixed-address 192.168.1.21;
```

# DHCP: Kommunikation beim Bootvorgang



# DHCP: Leases

---

- Lease-Zeit = Nutzungszeit für IP-Adresse
  - Parameter (Timer) werden beim dyn. Konfigurieren an den Client gesendet
  - Parameter T1 gibt standardmäßig 50 % der Lease-Zeit an
    - Client sendet erneut einen DHCP-Request
  - Parameter T2 87,5 % der Lease-Zeit
    - Wenn kein ACK vom Server kommt, dann erneutes DHCP-Discovery durch Client

## Ergänzung:

### Wichtige Administrations-Kommandos

---

- Diagnose- und Konfigurationskommandos im TCP/IP-Umfeld, die man öfter mal braucht:
  - ping
  - hostname
  - netstat
  - nslookup (kommt später bei DNS)
  - arp
  - traceroute (Windows: tracert), pathping
  - ifconfig
  - route
  - ipconfig (Windows)
  - nbtstat (Windows, NetBIOS-Info, usw.)

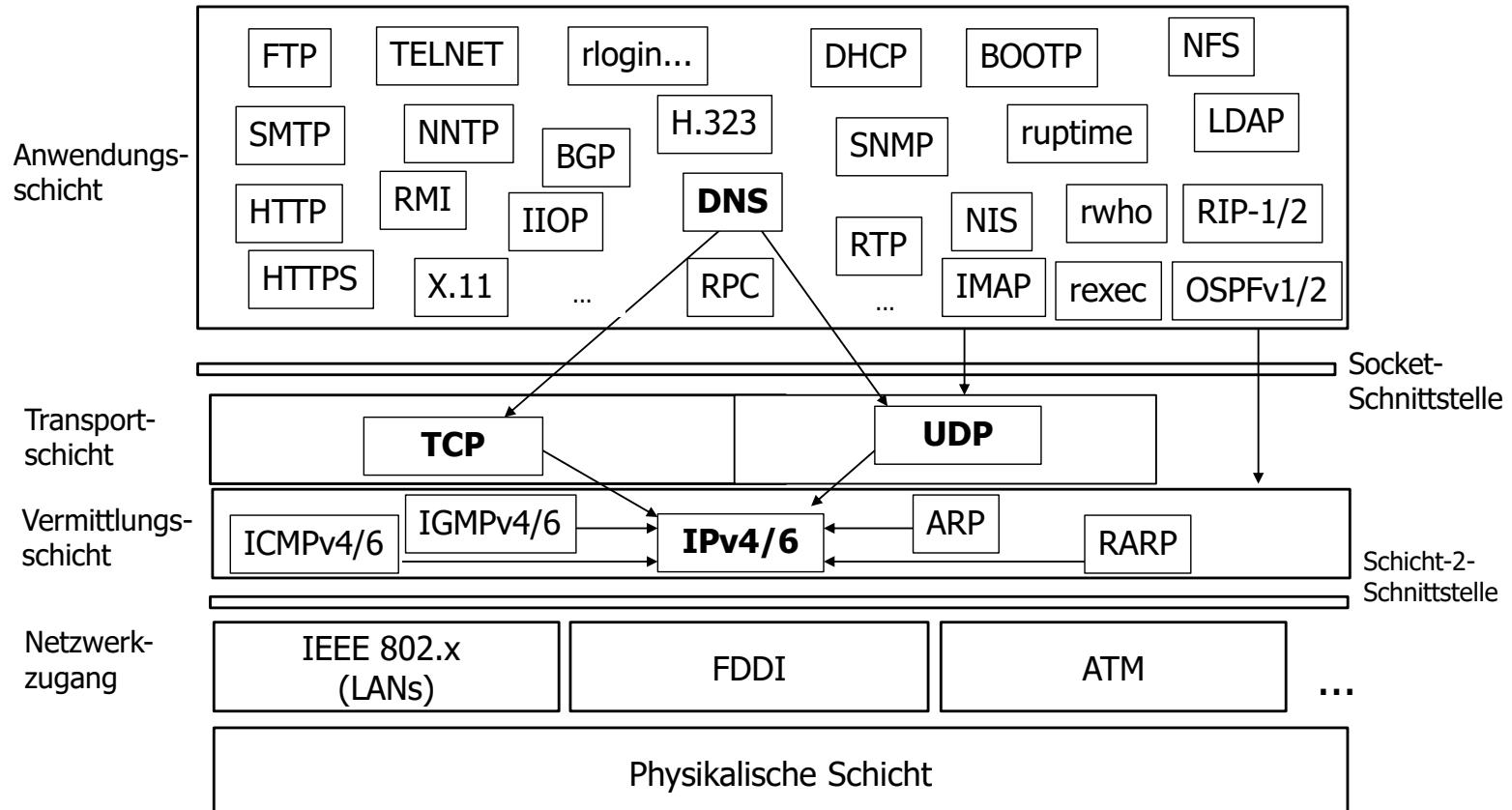
## 1. Steuerprotokolle

- ICMP
- ARP und RARP
- NAT
- DHCP

## 2. Domain Name System

- Namensraum und Organisation
- Root-Name-Server
- Adressauflösung
- DNS-PDU

# Übersicht Einordnung in die TCP/IP-Protokollfamilie



DNS-Port: 53 TCP/UDP

# Domain Name System (DNS): Hintergrund

---

- Im ARPANET mit einigen hundert Hosts war die Verwaltung der Adressen noch einfach
  - Es gab eine Datei **hosts.txt** auf einem Verwaltungsrechner, die alle IP-Adressen enthielt (flacher Namensraum)
  - Die Datei wurde nachts auf die anderen Hosts kopiert
- Als das Netz immer größer wurde, führte man DNS ein
- DNS dient prinzipiell der **Abbildung von Hostnamen auf eMail- und IP-Adressen**
- DNS ist ein Internet **Directory Service**

Achtung: DNS nicht mit Routing verwechseln!

---

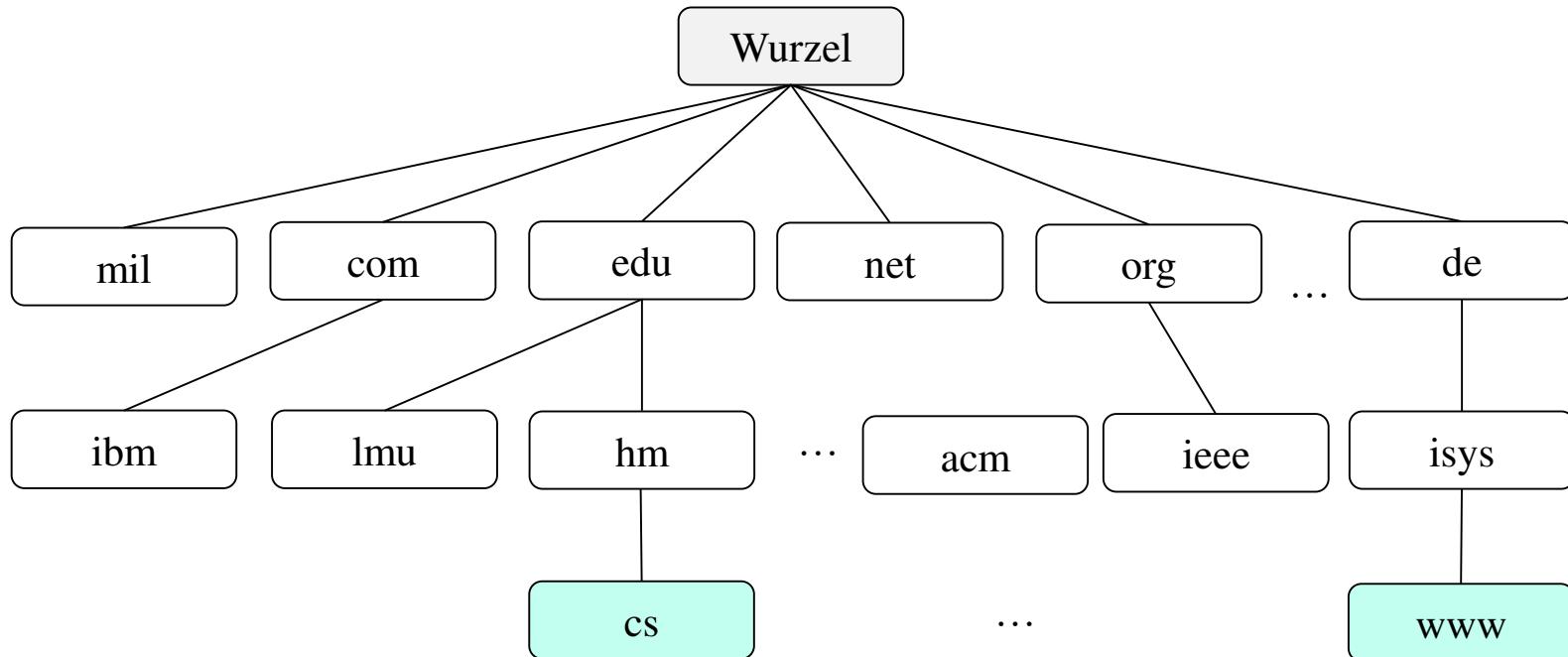
## Domänen und Subdomänen

---

- DNS ist in den RFCs 1034 und 1035 definiert
- DNS ist ein hierarchischer Namensverzeichnis für IP-Adressen (Adressbuch des Internets)
- DNS verwaltet eine **Datenbank**, die sich über zahlreiche Internet-Hosts erstreckt
- Konzeptionell ist das Internet in **mehrere hundert Domänen** aufgeteilt
- Die Domänen sind wiederum in **Teildomänen (Subdomains)** untergliedert, usw.

# DNS-Namensraum

- Weltweit verteilter Namensraum
- **Baum**, an dessen Blättern dann die Hosts hängen (**rein organisatorisch**, nicht physikalisch)



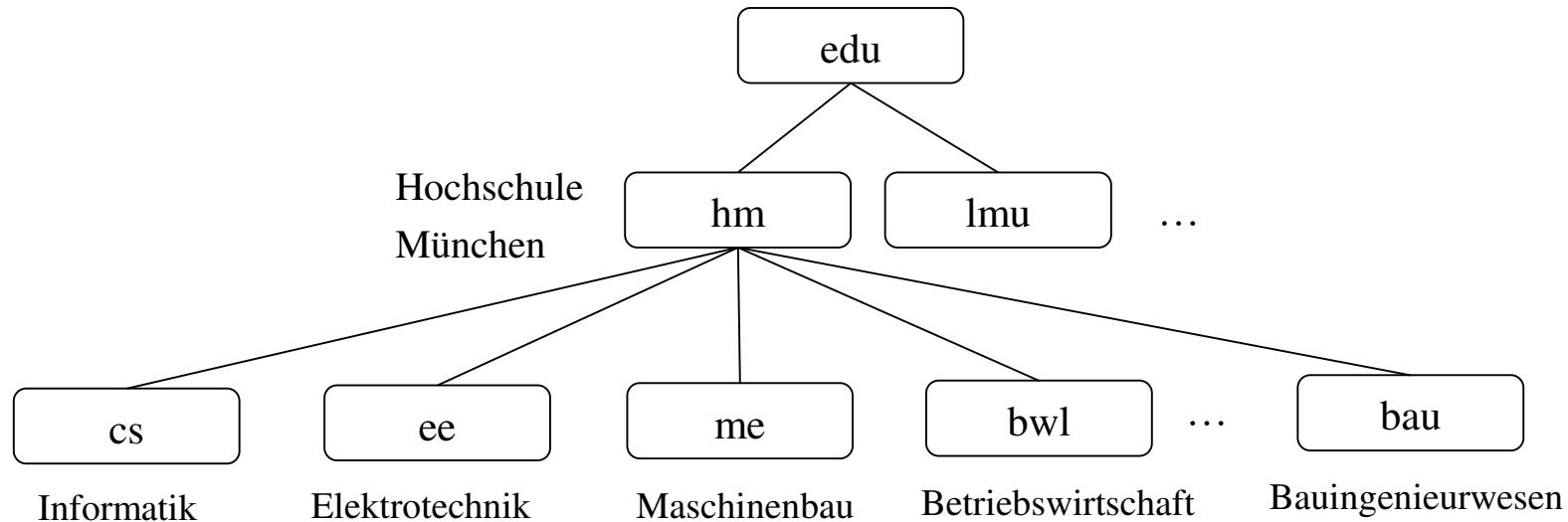
Namensbildung durch Konkatenation der Knoten entlang des Baumes vom Blatt zur Wurzel.

Beispiel: **cs.hm.edu**

# DNS-Namensraum

---

- **Beispiele** für Domänen mit untergeordneten Subdomänen



# Nicht gesponserte Domains

---

- **ICANN** (Internet Corporation for Assigned Names and Numbers) pflegt diese „nicht-gesponserten“ Domains
- Man unterscheidet:
  - Geographische oder länderspezifische (**Country-Code, ccTLDs**) Domains wie de, at, us, uk, gb, usw.
    - Für jedes Land ist nach ISO 3166 ein Code mit zwei Buchstaben vorgesehen
    - Es gibt derzeit über 200 ccTLDs
    - Für die Europäische Union wurde .eu als Gemeinschaft ebenfalls dieser Art von Domains zugeordnet, obwohl .eu als eine Ausnahme behandelt wird (Liste der Ausnahmen zu ISO 3166)
  - Allgemeine Domains für Organisationen (**generic oder gTLDs**) und auch neue Namen (**new gTLDs**)
  - **Infrastruktur-Domains** als Sonderfall (spezielle Domain .arpa)
- Es gibt darüber hinaus „gesponserte“ Domains

# Generic TLDs

---

Domain	Beschreibung
com	Kommerzielle Organisationen (sun.com, ibm.com, ...)
edu	Bildungseinrichtungen (hm.edu)
gov	Amerikanische Regierungsstellen (nfs.gov)
mil	Militärische Einrichtung in den USA (navy.mil)
net	Netzwerkorganisationen (nsf.net)
org	Nichtkommerzielle Organisationen
int	Internationale Organisationen (nato.int)
biz	Business, für Unternehmen
info	Informationsanbieter
arpa	TLD des ursprünglichen Arpanets, die heute als sog. <i>Address and Routing Parameter Area</i> verwendet wird (auch als "Infrastruktur-Domain" bezeichnet).
pro	Professions, Berufsgruppen der USA, Deutschland und des Vereinigten Königreichs

## Gesponserte Domains

---

- Werden von unabhängigen Organisationen in Eigenregie kontrolliert und finanziert
- Eigene Richtlinien für die Vergabe von Domainnamen
- Zu den „gesponserten“ TLDs gehören:
  - **.aero:** Aeronautics, für in der Luftfahrt tätige Organisationen, weltweiter Einsatz
  - **.coop:** Steht für *cooperatives* (Genossenschaften), weltweiter Einsatz
  - **.info:** Informationsanbieter, weltweiter Einsatz
  - **.int:** Internationale Regierungsorganisationen (Beispiel [www.nato.int](http://www.nato.int) oder [www.eu.int](http://www.eu.int))
  - **.mobi:** Darstellung von Webseiten speziell für mobile Endgeräte, weltweiter Einsatz ...

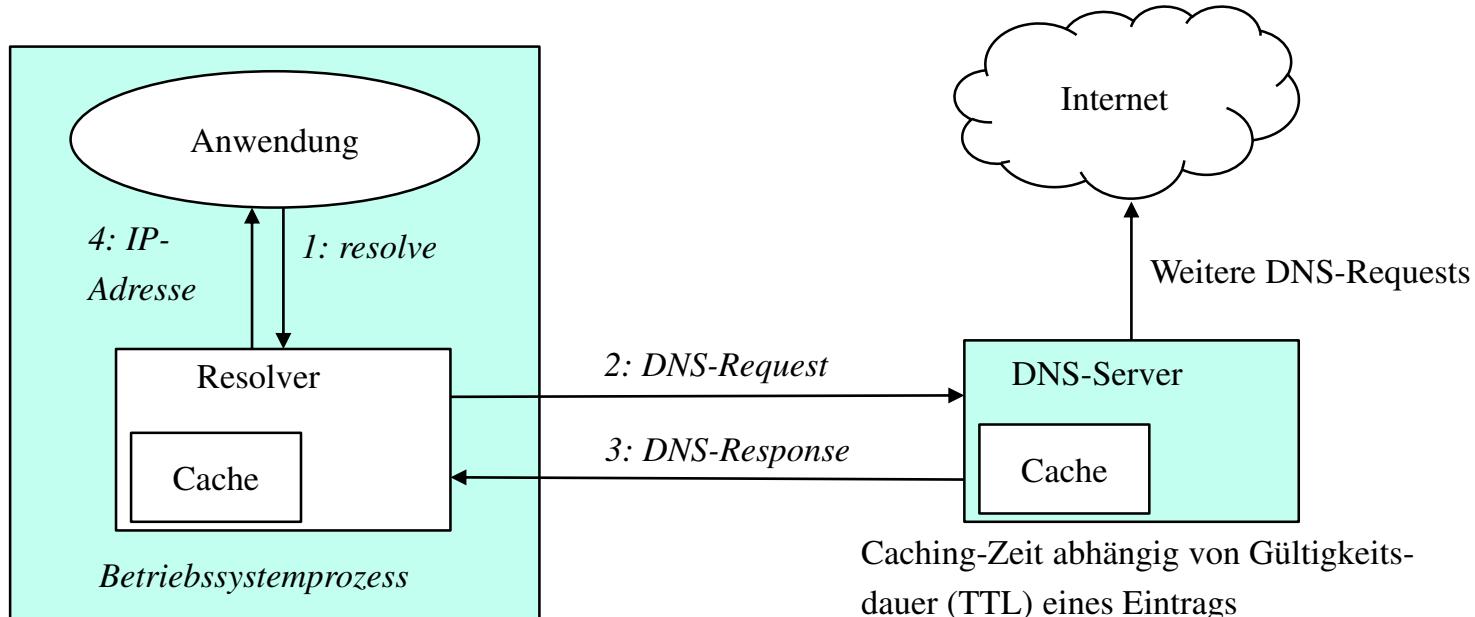
## DNS-Namensraum und DNS-Datenbank

---

- DNS ist eine baumförmige **weltweite Vernetzung** von Name-Servern
- DNS bildet eine weltweit verteilte Datenbank (**DNS-Datenbank**)
- Jeder Knoten im DNS-Baum stellt eine Domäne dar und kann mit einem Verzeichnis in einem Dateisystem verglichen werden
- Jeder Knoten hat einen Namen

# Adressauflösung

- Eine Anwendung, die eine Adresse benötigt, wendet sich lokal an einen **Resolver** (Library), der eine Anfrage an einen DNS-Server (unter Unix z.B. **named**) richtet.



# Name Server und Zonen

---

- DNS-Name-Server verwaltet jeweils **Zonen** des DNS-Baums, wobei eine Zone an einem Baumknoten beginnt und die darunter liegenden Zweige beinhaltet
- Ein Name-Server (bzw. die entspr. Organisation) kann die Verantwortung für Subzonen an einen weiteren Name Server **delegieren**
- Die Name-Server kennen jeweils ihre Nachbarn in der darunter- oder darüber liegenden Zone
- Informationen des DNS werden in sog. **Resource Records** verwaltet → in Konfigurationsdateien der DNS-Server

# Autoritative Name Server und andere...

---

- **Autoritativer** Name Server:
  - Verantwortlich für eine Zone
  - Mind. einer muss in Zone sein (Primary Nameserver)
  - Kennt alle Adressen der Zone genau
  - Siehe Konfigurationsdatei (**SOA-Resource-Record**, = „*Start of Authority*“, Angaben zur Verwaltung einer Zone)
- **Nicht-autoritativer** Name Server:
  - Bezieht Informationen von anderen Servern
  - Verfalldatum über TTL-Parameter geregelt
- **Zonentransfer** zwischen Primary und Secondary Server

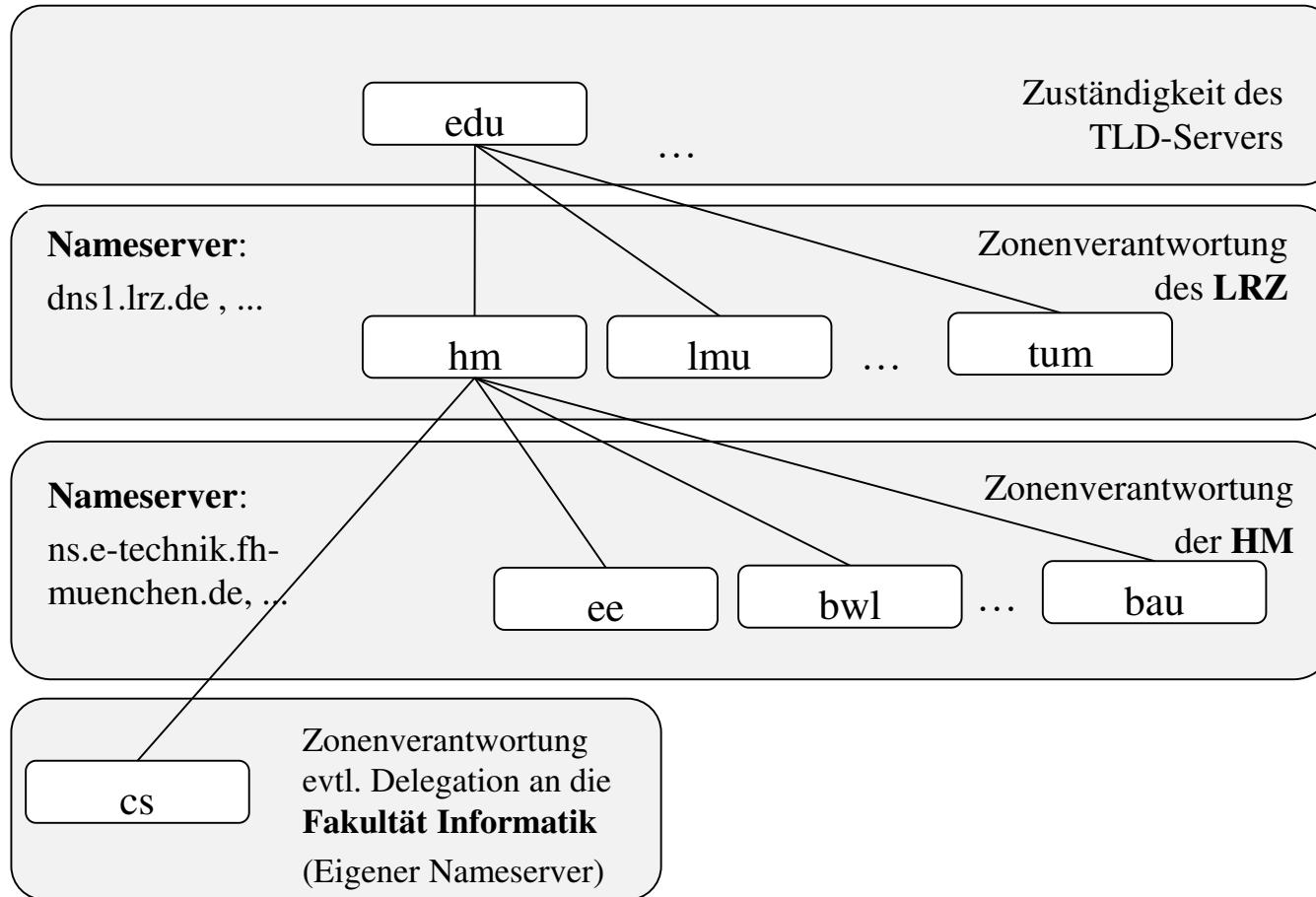
## Root-Name-Server

---

- Es gibt derzeit **weltweit 13 sog. Root-Name-Server (A...M)**
  - 10 in Nordamerika, 1 in Stockholm, 1 in London, 1 in Tokio
- Root-Name-Server geben Informationen über die weitere Suche
- Ein Root-Name-Server
  - verfügt über eine Referenz-Datenbank aller von der ICANN freigegebenen Top-Level-Domains (TLD) und
  - die wichtigsten Referenzen auf die sog. Top-Level-Domain-Server
- Ein Root-Name-Server kennt somit immer einen DNS-Server, der eine Anfrage beantworten kann

# Name Server und Zonen

## ■ Beispiel für die Zonen-Delegation



# Name Server

---

- In jeder Zone muss es mindestens zwei Name Server geben (**primary und secondary**), die auch miteinander kommunizieren
- Für normale Anfragen wird ein **UDP-basiertes** Protokoll verwendet, größere Abfragen über TCP
- Primary und Secondary kommunizieren über **TCP**
- Weitere Name Server werden auch von Internet Service Providern (ISP) und Netzbetreibern betrieben
- Die Domain **.de** verwaltet das Deutsche Network Information Center (**DENIC**)
  - betrieben in Frankfurt (früher TU Dortmund, dann TU Karlsruhe)

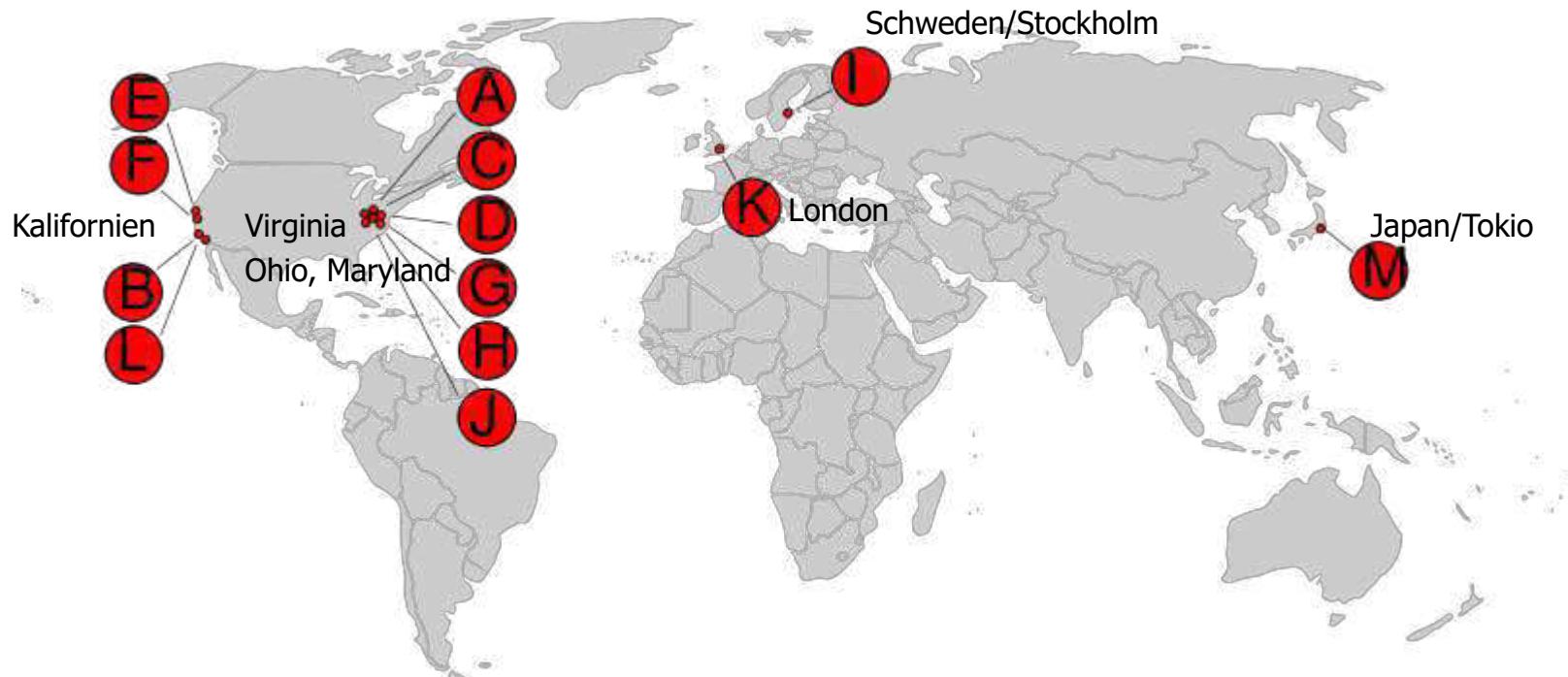
# Domainnamen

---

- DNS-Namen bestehen aus zwei Teilen, einem Hostnamen und einem Domainnamen, die zusammen den „**Fully Qualified Domain Name**“ (FQDN) bilden
  - Max. 255 Zeichen, keine Sonderzeichen, Umlaute oder Leerzeichen
  - Max. 63 Zeichen pro Teildomain oder Hostname
  - Beispieladresse: [www.isys.de](http://www.isys.de) (isys.de ist der Domainname)
- Zu jeder Domain gehört eine Körperschaft, die diese verwaltet und für die Namensvergabe zuständig ist, oberste Körperschaft ist ein NIC (Network Information Center = Domain Name Registry, z.B. DENIC)

# Root-Name-Server (früher)

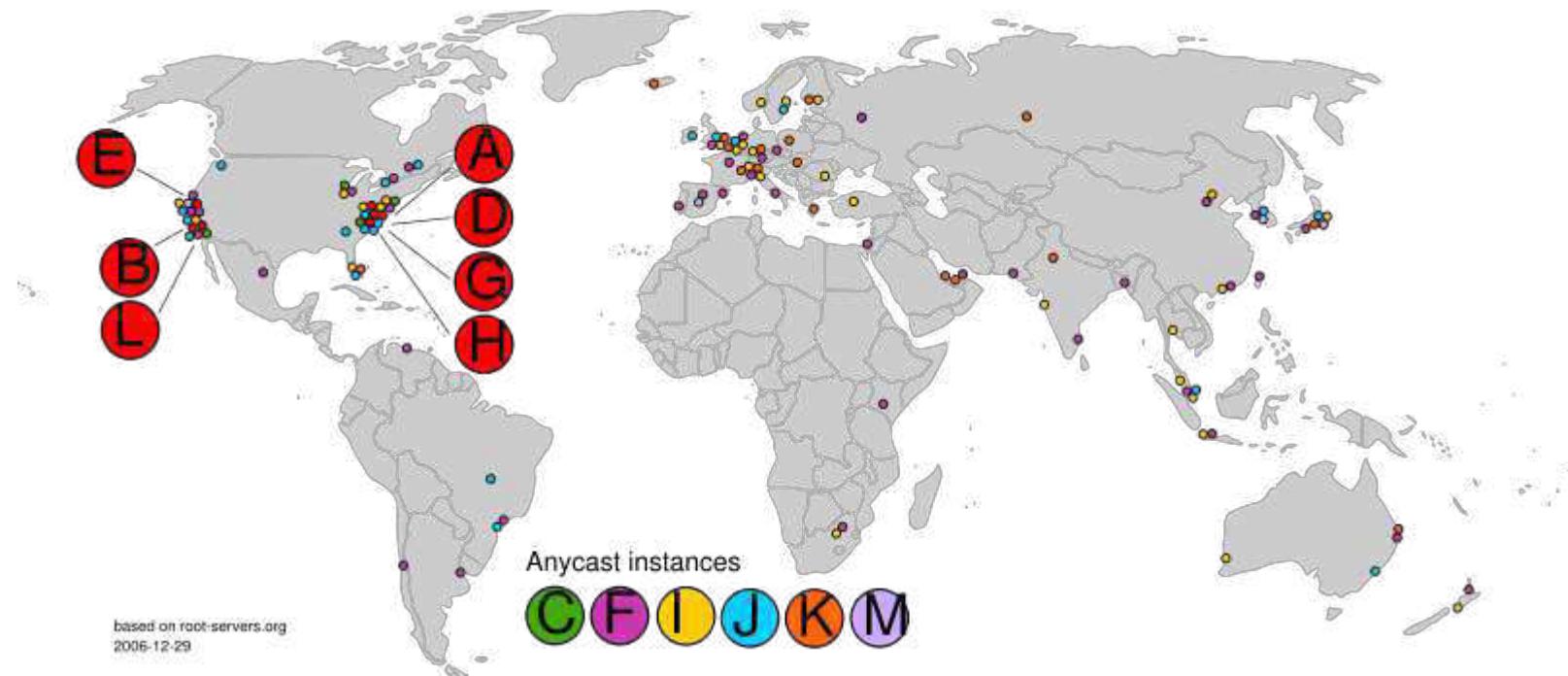
- Single Point of Failure: Synchronisation erfolgte über Root-Name-Server A (VeriSign = Amerikanisches Unternehmen und Zertifizierungsstelle für digitale Zertifikate)
- Viele DoS-Angriffe



Quelle: Wikipedia

# Root-Name-Server (heute, 1)

- Mehr Ausfallsicherheit, in 2009 mehr als 120 Root-Name-Server verfügbar
- Keine Synchronisation mit A → Kein Single Point of Failure



Quelle: Wikipedia

Anycast nutzt Routing  
→ nächster DNS-Server erhält den DNS-Request

## Root-Name-Server (heute, 2)

---

- A und J haben **zentrale** Bedeutung. Sie verteilen die Datenbasis für alle anderen Root-Name-Server zweimal täglich
- Root-Name-Server bestehen aus z.T. **vielen Einzelservern**:
  - Root-Name-Server F besteht aus 33 Serverrechnern (2009)
  - Root-Name-Server K besteht aus 16 Serverrechnern (2009)
- Diese Root-Name-Server sind auf der ganzen Welt verteilt und einige nutzen heute einen **Anycast-Mechanismus** (eine IP-Adresse) zur Datenverteilung

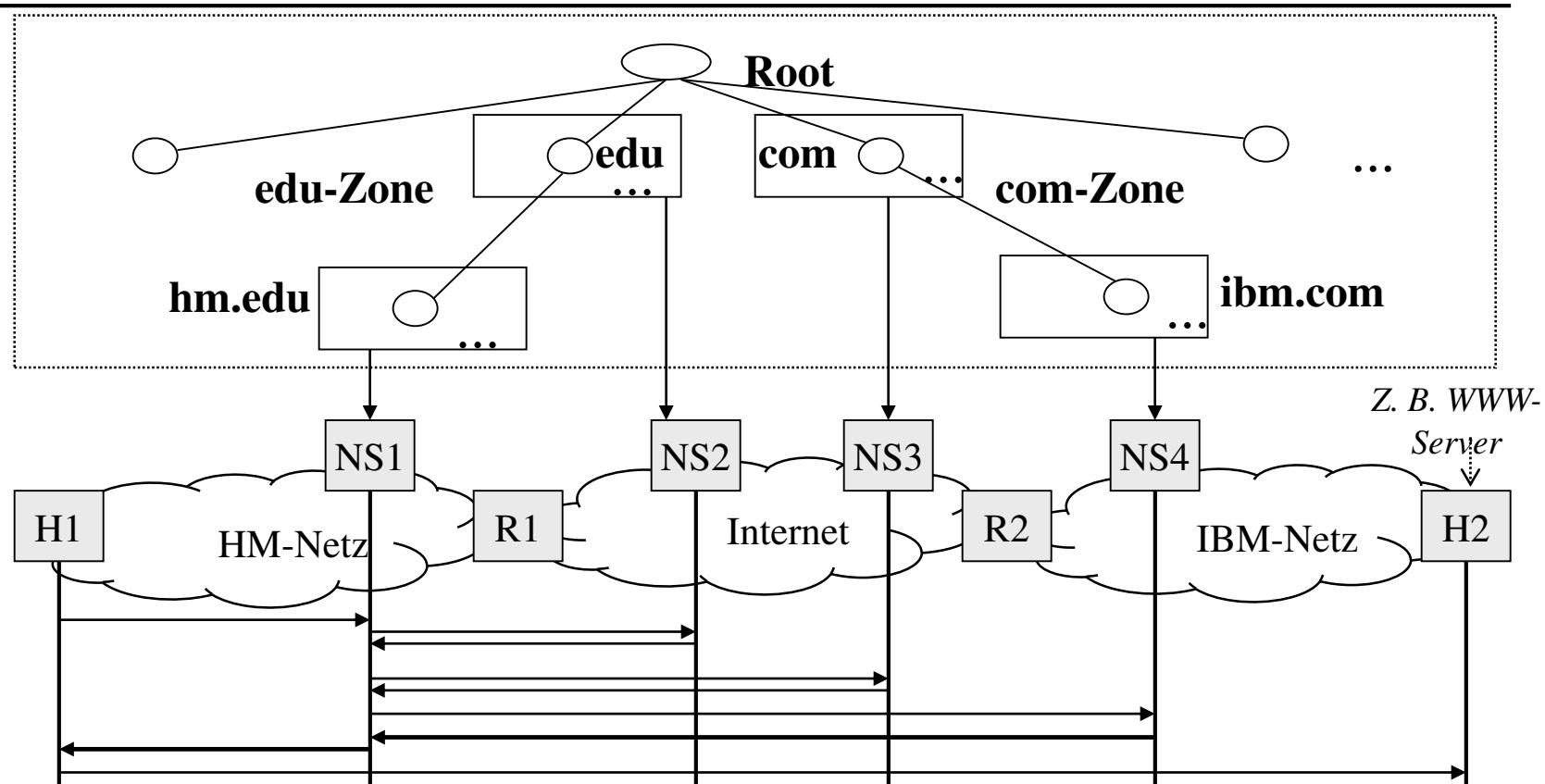
# Root-Name-Server

---

<b>Root-Server</b>	<b>Alter Name</b>	<b>Betreiber</b>	<b>Ort</b>
A	ns.internic.net	VeriSign	verteilt
B	ns1.isi.edu	ISI	Marina Del Rey, Kalifornien, USA
C	c.psi.net	Cogent Communications	verteilt
D	terp.umd.edu	University of Maryland	verteilt
E	ns.nasa.gov	NASA	Mountain View, Kalifornien, USA
F	ns.isc.org	ISC	verteilt
G	ns.nic.ddn.mil	U.S. DoD NIC	verteilt
H	aos.arl.army.mil	U.S. Army Research Lab	verteilt
...			

Quelle: Wikipedia

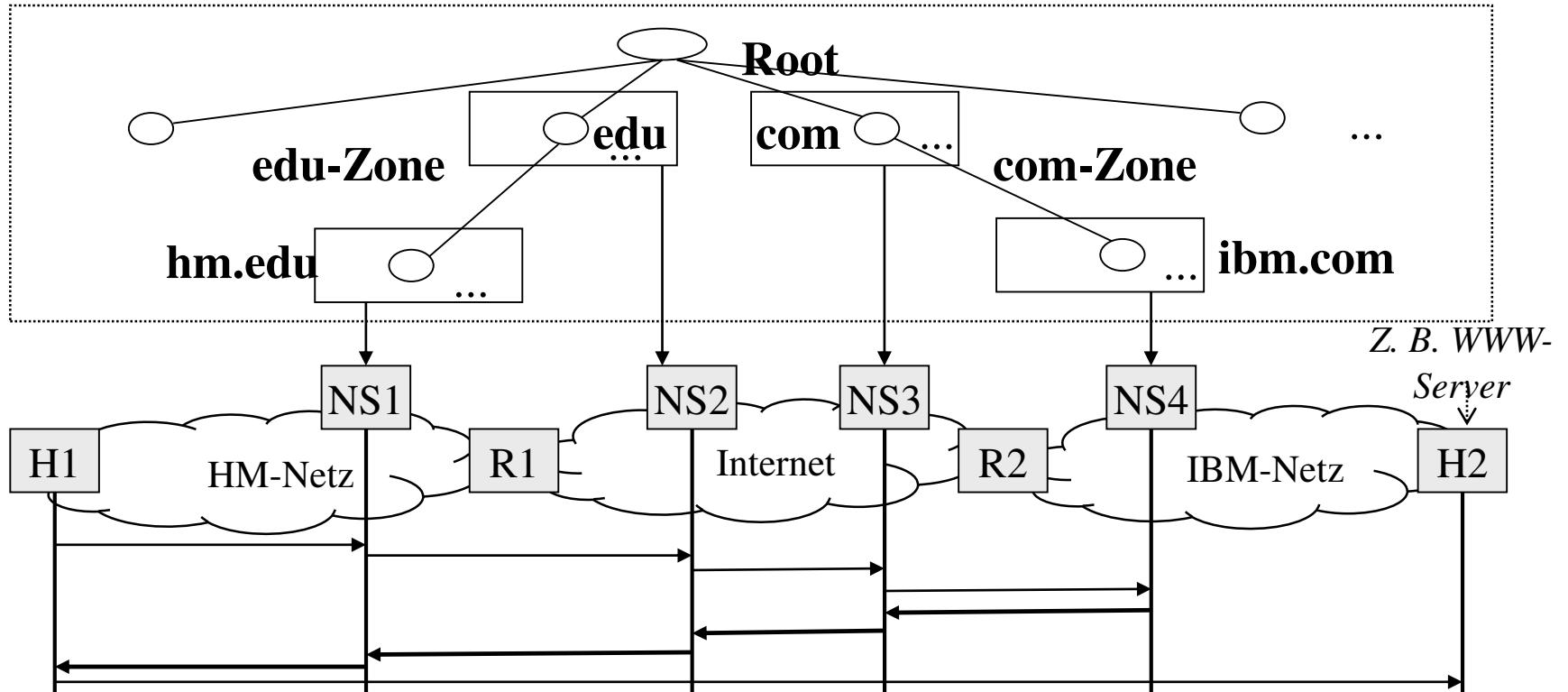
# Beispiel für eine iterative Auflösung einer IP-Adresse



$NS_x = \text{Nameserver}$ ,  $R_y = \text{Router}$

- Nameserver verfügen über eigene Resolver, die üblicherweise iterativ arbeiten

# Beispiel für eine rekursive Auflösung einer IP-Adresse



$NS_x = \text{Nameserver}$ ,  $R_y = \text{Router}$

- Resolver von Clients arbeiten rekursiv

# DNS Records

---

- Allgemeiner Aufbau eines Resource Records als Quintupel folgender Form:

(Domainname, Time-to-live, Type, Class, Value)

Domainname: Name der Domäne

Time-to-live: Stabilität (Gültigkeitsdauer) des Werts als Integerzahl (je höher desto stabiler), für das Caching wichtig (kurz: TTL)

Type: Typ des Records (A = IP-Adresse, NS = Name Server Record, MX = Mail-Record, PTR = Reverse Record, SOA = Start of Authority)

Class: immer IN (Internet)

Value: Wert des Records je nach Typ: z.B. bei Typ = A → IP-Adresse

# Einschub: NS-Report für hm.edu (Auszug)

## NS-Record(s) for domain hm.edu:

ns.fh-muenchen.de. 129.187.244.11  
ns.e-technik.fh-muenchen.de. 129.187.206.129

```
nslookup  
>set type=MX  
>isys.de  
→ MX-Records werden ausgegeben  
(dig-Kommando unter Linux)
```

## MX-Servers for hm.edu:

150 mailhost.fh-muenchen.de. 129.187.244.204 // Mail Exchange Resource Records  
140 mailrelay3.rz.fh-muenchen.de. 129.187.244.101  
120 vulcan.rz.fh-muenchen.de. 129.187.244.102

## SOA-Record for hm.edu: (*Start of Authority*)

ns.hm.edu. hostmaster.hm.edu. 2009121462 10800 1800 3600000 86400

Serial: 2009121462 (Versionsnummer)

Refresh: 10800

Retry: 1800

Expire: 3600000 (1000 hours or 42 days)

TTL: 86400

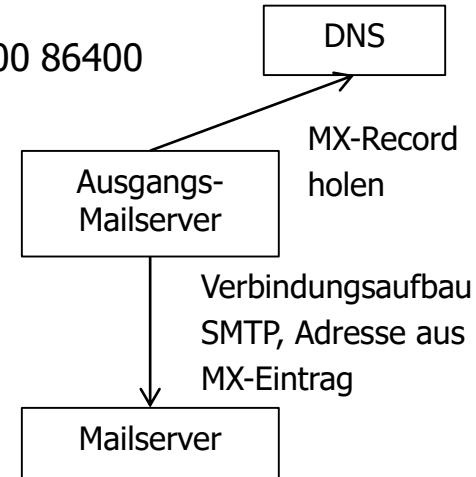
## Recursive-Queries: (Rekursive Auflösung)

ns.e-technik.fh-muenchen.de. YES - recursive queries allowed!

ns.hm.edu. YES - recursive queries allowed!

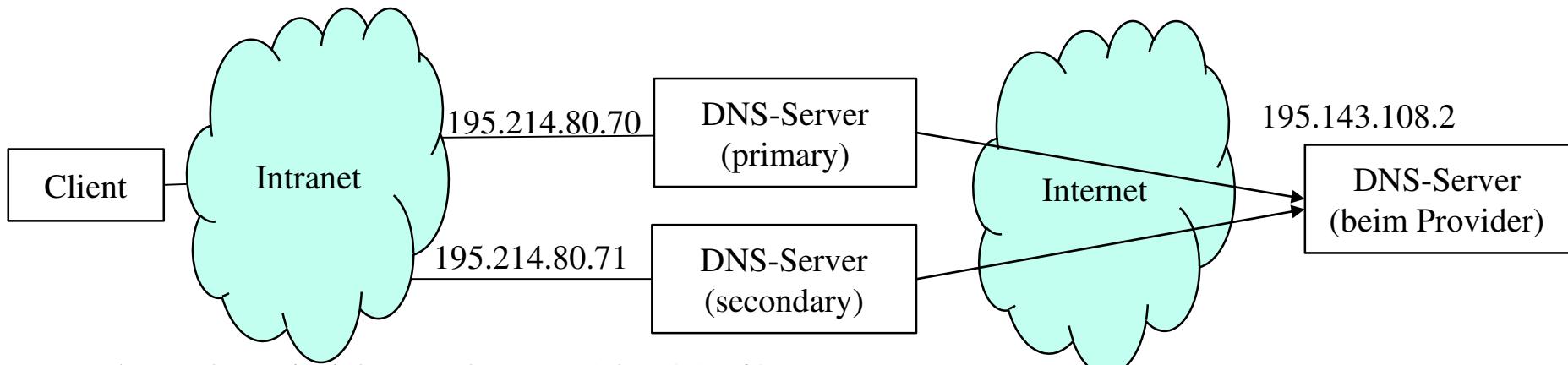
...

Hinweis: Refresh, Retry und Expire sind Zeitabstand in s zur Synchronisation zwischen Sekundär- und Masterserver, hostmaster.hm.edu ist die Mailadresse des Admins



# Unternehmensnetz anbinden

- Beispiel eines Unternehmensnetzes
- DNS-Server-Eintrag
  - Es gibt zwei DNS-Server mit den Adressen 195.214.80.70 und 195.214.80.71
  - Nächster DNS-Server beim ISP: 195.143.108.2



IP-Adresse des Beispielunternehmens: 195.214.80.64

Netzwerkmaske: 255.255.255.192

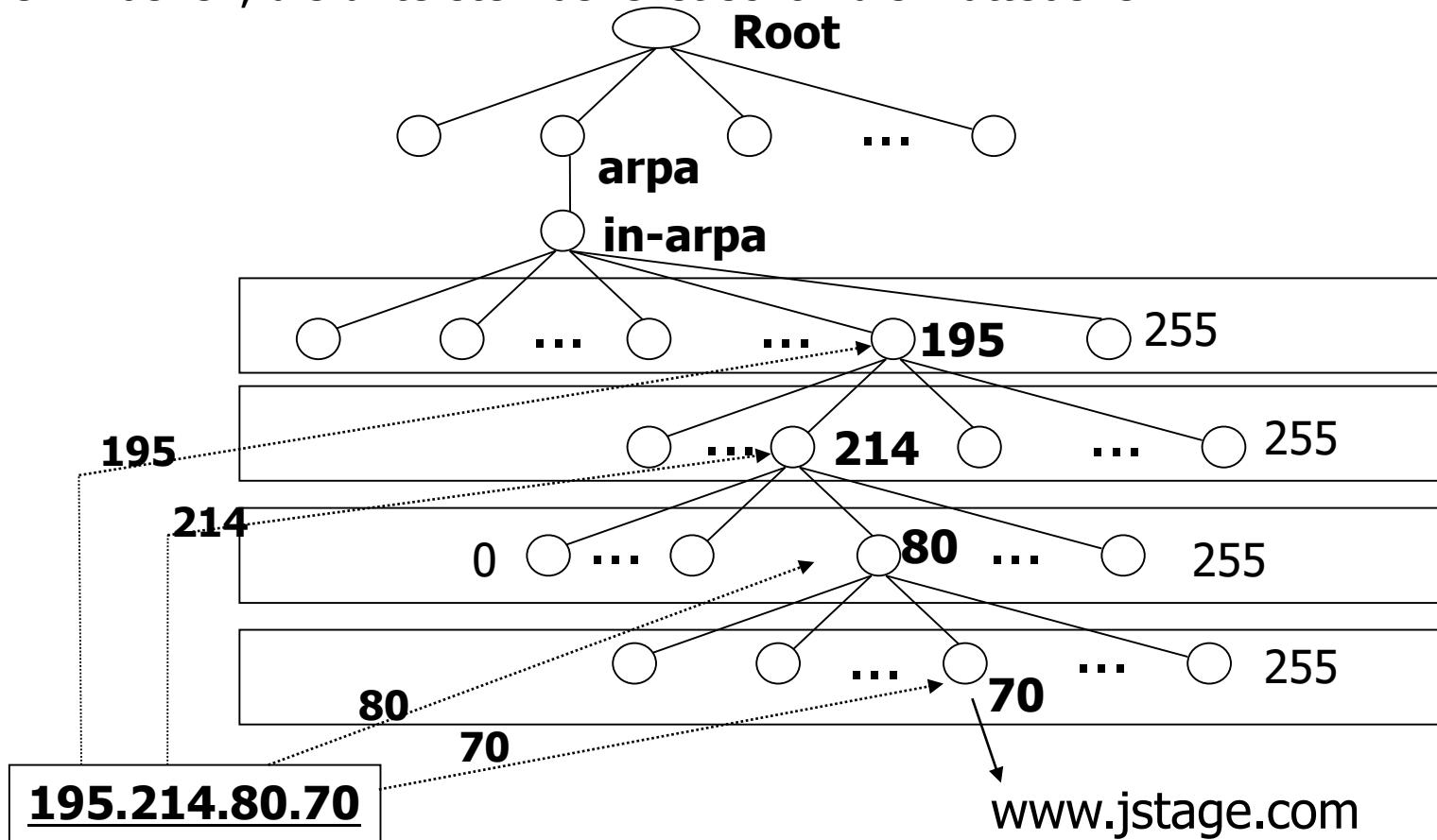
# Auflösung von IP-Adressen auf Hostnamen

---

- Über DNS lassen sich auch IP-Adressen auf Hostnamen auflösen (**inverse Abfrage**)
- Benötigt man z.B. bei der Fehlerbehebung in TCP/IP-Netzen und für Logdateien
- Diese Zuordnungen werden in der besonderen Domäne **in-addr.arpa** durch InterNIC verwaltet
- Die Knoten der Domain sind nach Zahlen in der für IP-Adressen üblichen Repräsentation benannt
  - in-addr.arpa hat 256 Subdomains
  - Die Subdomains haben jeweils wieder 256 subdomains
  - In der untersten (vierten) Stufe werden die Resource Records mit vollem Hostnamen eingetragen

# Auflösung von IP-Adressen auf Hostnamen: Reverse DNS, Beispiel einer Adressauflösung

- **in-addr.arpa-Domäne:** Unterhalb dieser Domäne existieren drei Subdomänen-Ebenen bei IPv4, bei IPv6 → **ipv6.arpa**, entsprechend mehr Ebenen, die unterste Ebene ist schon die Blattebene



# Reverse DNS: RR-Record-Beispiel

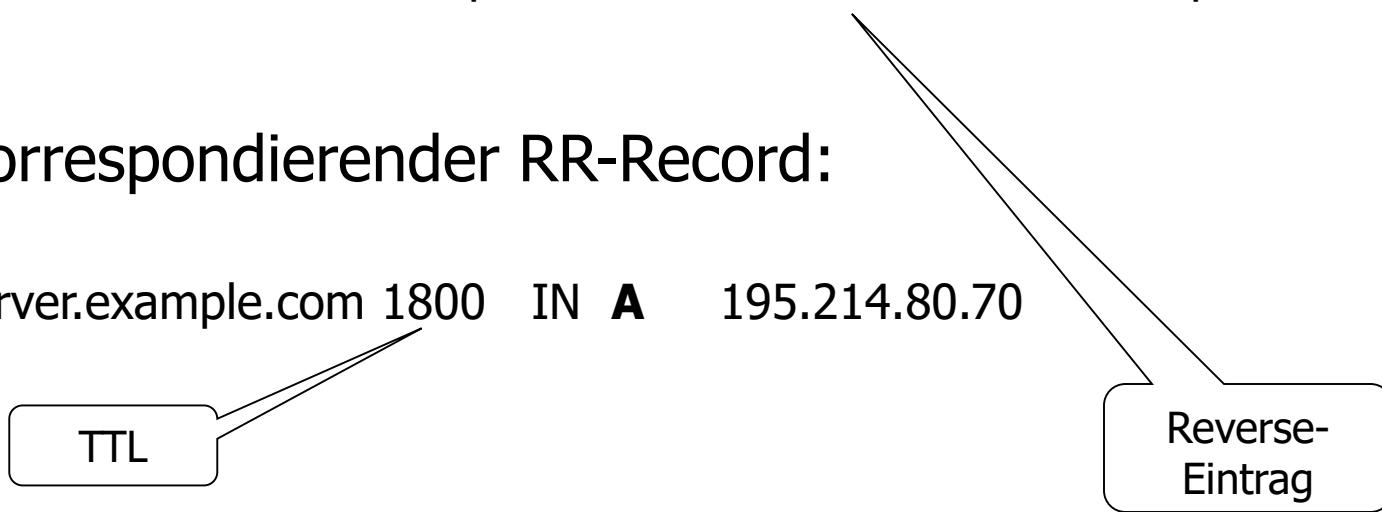
---

- Eintrag für Reverse DNS:

70.80.214.195.in-addr.arpa. 1285 IN **PTR** server.example.com

- Korrespondierender RR-Record:

server.example.com 1800 IN **A** 195.214.80.70

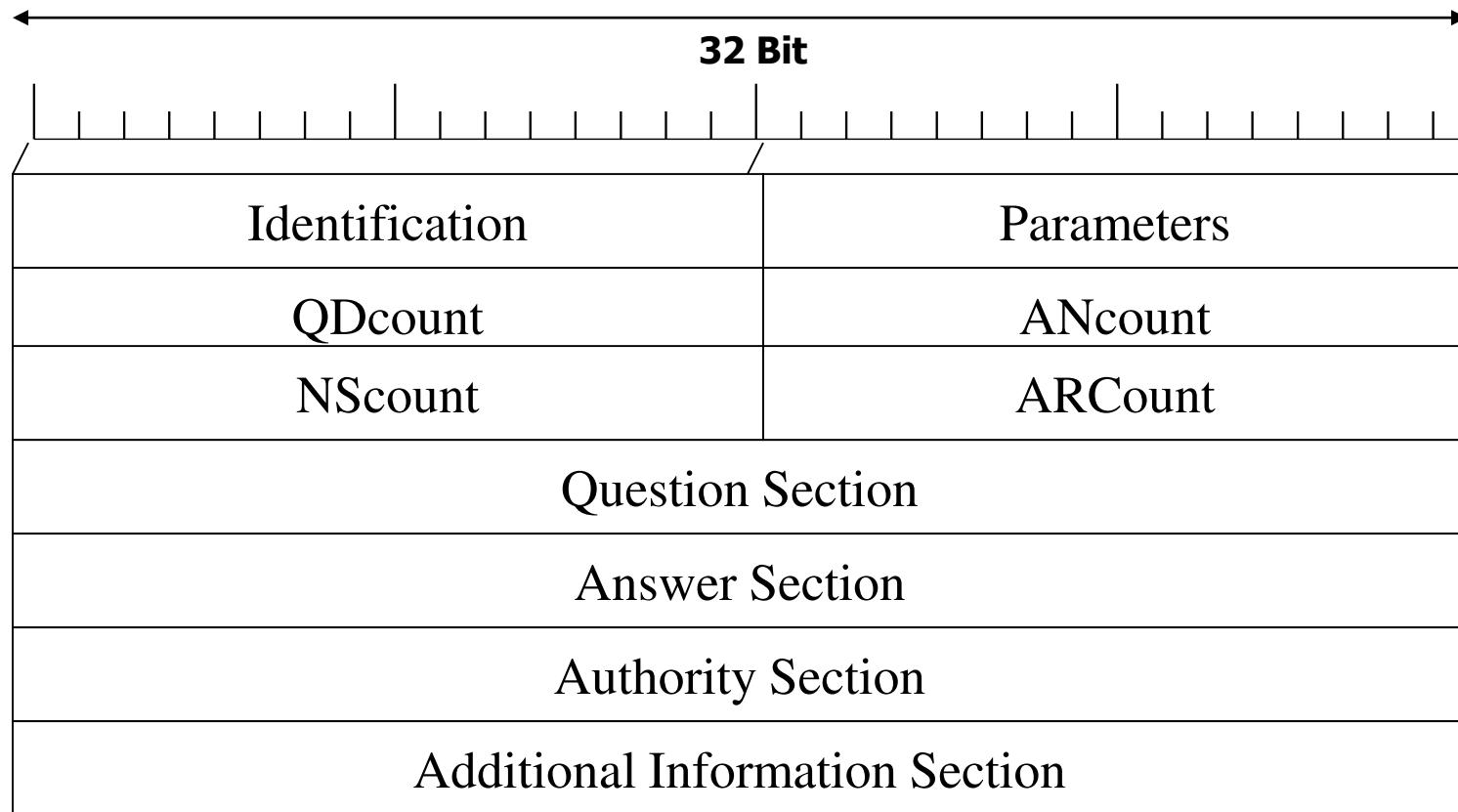


*Hinweise:*

- *reverse* → umgekehrte Reihenfolge in der in-addr.arpa-Domäne bzw. in der ip6.arpa-Domäne
- *inverse* → umgekehrte Informationsanfrage, also ein Name und nicht eine IP-Adresse wird gesucht

# DNS-PDU

- DNS erlaubt DNS-PDUs (UDP) bis zu einer Größe von 512 Byte
- EDNS (Extended DNS) als DNS-Erweiterung ermöglicht die zehnfache Größe (RFC 2671)



## DNS-PDUs

---

- DNS-Nachrichten setzen sich aus Sektionen (Sections) zusammen:
  - Der **Header** besteht aus den ersten sechs Feldern (Header Section)
  - **Question Section:** Enhält Felder zur Spezifikation der Anfrage
  - **Answer Section:** Enthält die Antwort eines Name-Servers in Form von Resource Records (RRs)
  - **Authority Section:** Enthält RRs eines autorisierten Servers
  - **Additional Information Section:** Enthält zusätzliche Informationen zur Anfrage oder zur Antwort

# DNS-Header

---

- **Identification (2 Bytes)**
  - Id der Anwendung, die die Abfrage abgesetzt hat
- **Parameters**
  - 0 := Anfrage, 1 := Response
  - Zusätzliche Information: Hinweis, ob es eine normale oder eine inverse Abfrage handelt
  - Inverse Abfrage: Für IP-Adresse einen Hostnamen suchen
- **QDcount**
  - Anzahl der Einträge in der Question Section
- **ANCount**
  - Anzahl an Resource Records (RR) in der Authority Section
- **ARcount**
  - Anzahl an RRs in der Additional Information Section

## Exkurs:

# Domain Name System Security Extension (DNSSEC)

---

- Erweiterung von DNS mit dem Ziel, dass ein DNS-Benutzer überprüfen kann, ob die empfangenen DNS-Zonen auch vom Ersteller sind
  - Vermeidung von "Cache Poisoning"
  - Seit 2010 ist DNSSEC auf allen Root-Servern installiert
- Nutzung digitaler Signaturen für Resource Records
  - Ersteller unterzeichnet mit geheimen Schlüssel, den die Nutzer mit seinem öffentlichen Schlüssel validieren können
  - Neue RR-Typen dafür eingeführt: RRSIG, DNSKEY, NSEC
  - Der öffentliche Teil des Zonenschlüssels ist im RR DNSKEY
  - Mit dem privaten Schlüssel wird jeder einzelne RR dieser Zone digital unterschrieben
  - RRSIG enthält die Signatur zum zugehörenden DNS-Eintrag
  - Mehr als 2/3 der Top-Level-Domains sind bereits signiert (2017)

## Für Interessierte: DNS-Paketlänge unter Windows

---

- Ändern der UDP-Paketlänge für DNS z.B. unter Windows 2003:
  - HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\DNS\Parameters → **MaximumUdpPacketSize**
  - Standardwert: 1280 Byte
  - Werte zwischen 512 and 16384 sind möglich

# Überblick

---

- ✓ Steuerprotokolle
  - ✓ ICMP
  - ✓ ARP und RARP
  - ✓ NAT
  - ✓ DHCP
- ✓ Domain Name System
  - ✓ Namensraum und Organisation
  - ✓ Root-Name-Server
  - ✓ Adressauflösung
  - ✓ DNS-PDU

- 1 Kommunikationssysteme und verteilte Anwendungen
- 2 Grundlagen der Transportschicht
- 3 Transportprotokolle TCP und UDP
- 4 Grundlagen der Vermittlungsschicht
- 5 Internet und Internet Protocol (IP)
- 6 Routing-Verfahren und -Protokolle
- 7 Internet-Steuerprotokolle und DNS
- 8 **Internet Protocol Version 6 (IPv6)**
- 9 Netzwerkschnittstelle

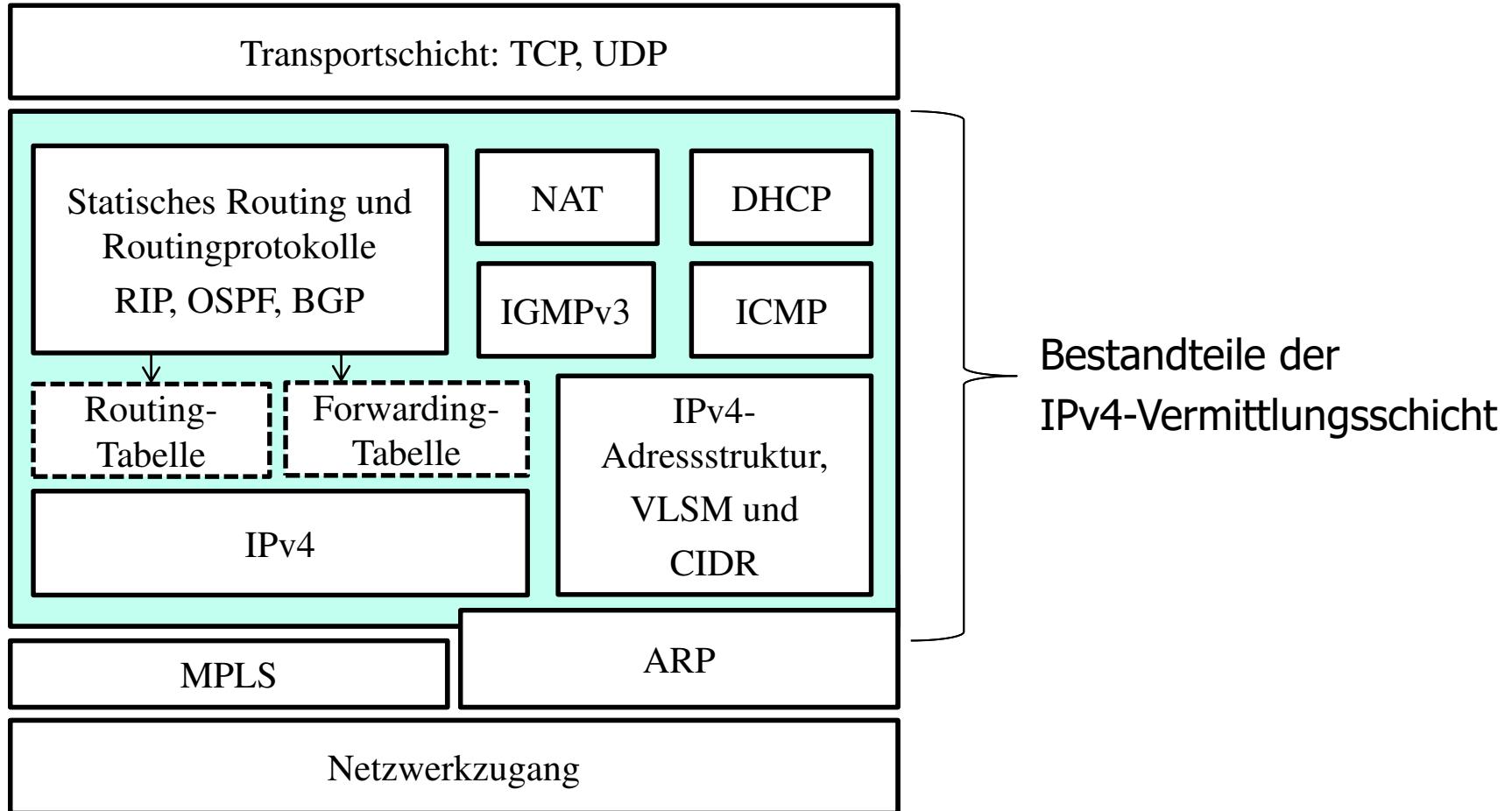
# Überblick über das Kapitel

---

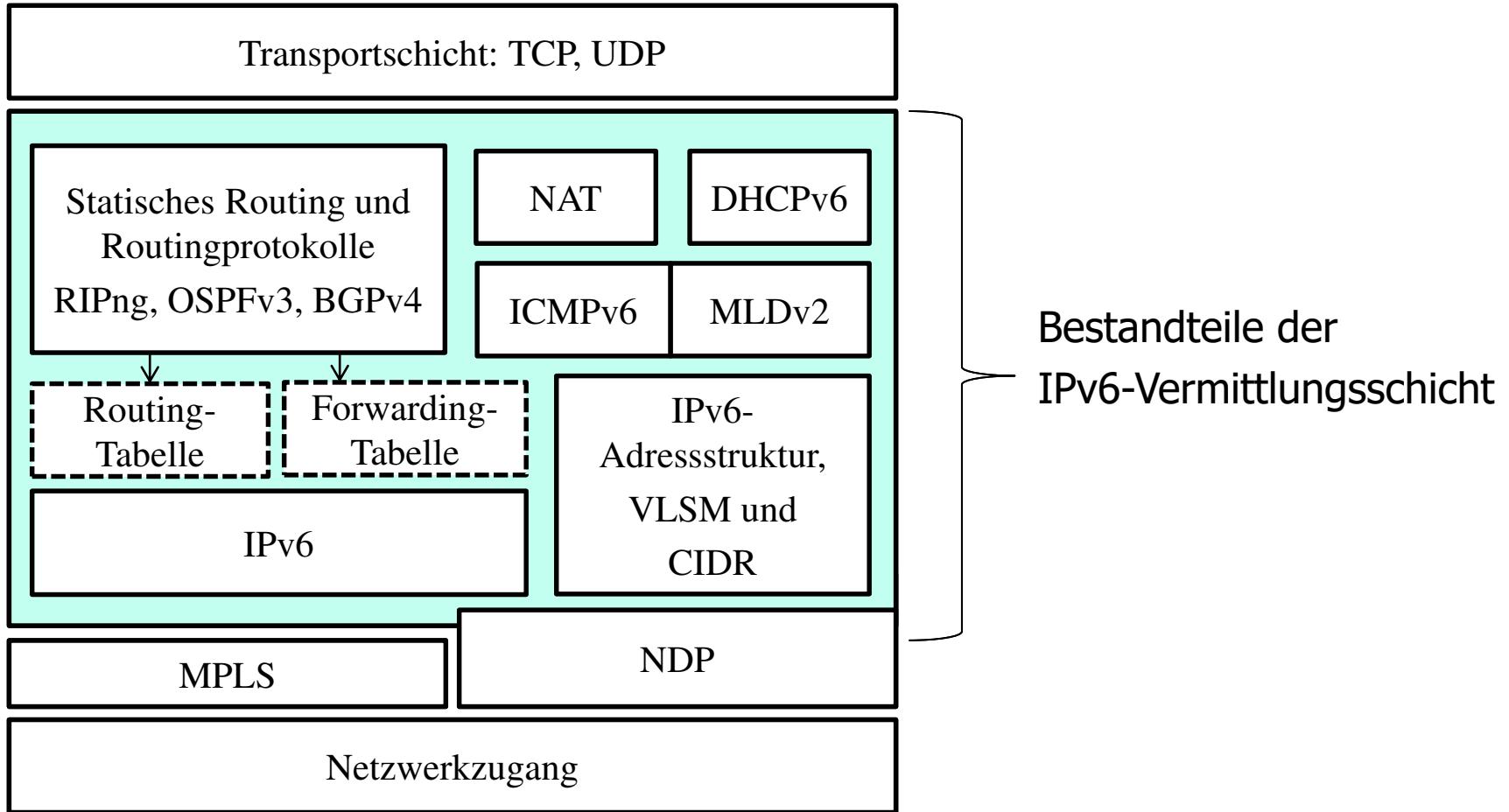
- 1. IPv6-Grundlagen und Adressierung**
2. IPv6-PDU
3. Discovery-Automatismen

Ergänzende Quellenempfehlung:  
Lüdtke, D.: IPv6-Workshop, 2. Auflage, 2013

# Überblick: Die IPv4-Vermittlungsschicht



# Überblick: Die IPv6-Vermittlungsschicht



# Einführung

---

- Die heutige IP-Adressierung ist an ihren Grenzen angelangt
- Hauptziel von IPv6 ist es, die **Adressproblematik** umfassend und langfristig zu lösen
- IPv6 ist bereits seit 1998 standardisiert
- Viele andere Verbesserungen wurden ebenfalls umgesetzt
- Koexistenz mit IPv4 ist gegeben

## Weitere Ziele und Verbesserungen zu IPv4

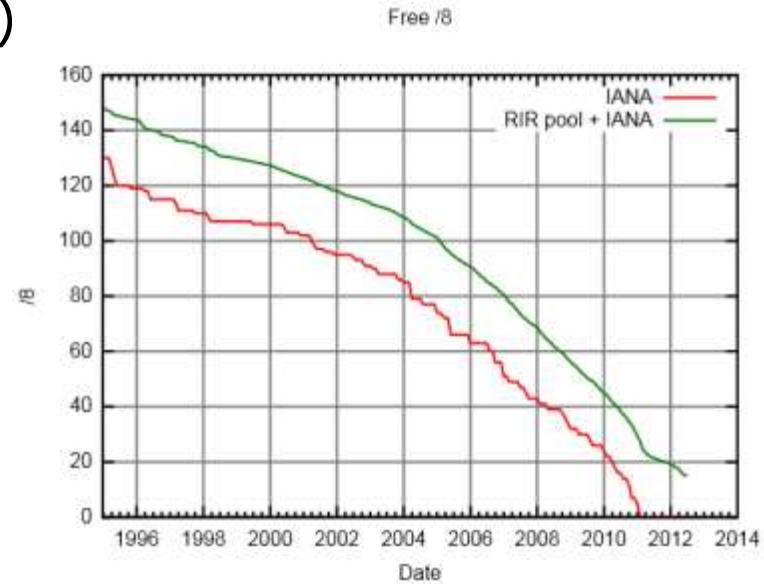
---

- **Vereinfachung** des Protokolls zur schnelleren Bearbeitung von Paketen in Routern
- Umfang der **Routing-Tabellen reduzieren**
- **Anwendungstypen** wie Multimedia-Anwendungen (Echtzeitanwendungen) unterstützen
  - Unterstützung von **Flussmarken**
- Höhere **Sicherheit** (Datenschutz, Authentifikation)
- **Multicasting** besser unterstützen
- **Mobile IP-Adressen**: Möglichkeit schaffen, dass Hosts ihr Heimatnetz verlassen können

# Verfügbare IPv4-Adressen

---

- 74 % der IPv4-Adressen sind nordamerikanischen RIRs zugeordnet
- China hat so viel Adressen wie die Universität von Kalifornien (University of California, Berkeley)
- IANA hat bereits alle /8-IPv4-Adressbereiche an die RIRs vergeben (die anderen waren ohnehin schon weg, es gab nur noch /8-Bereiche)



Quelle: wikipedia

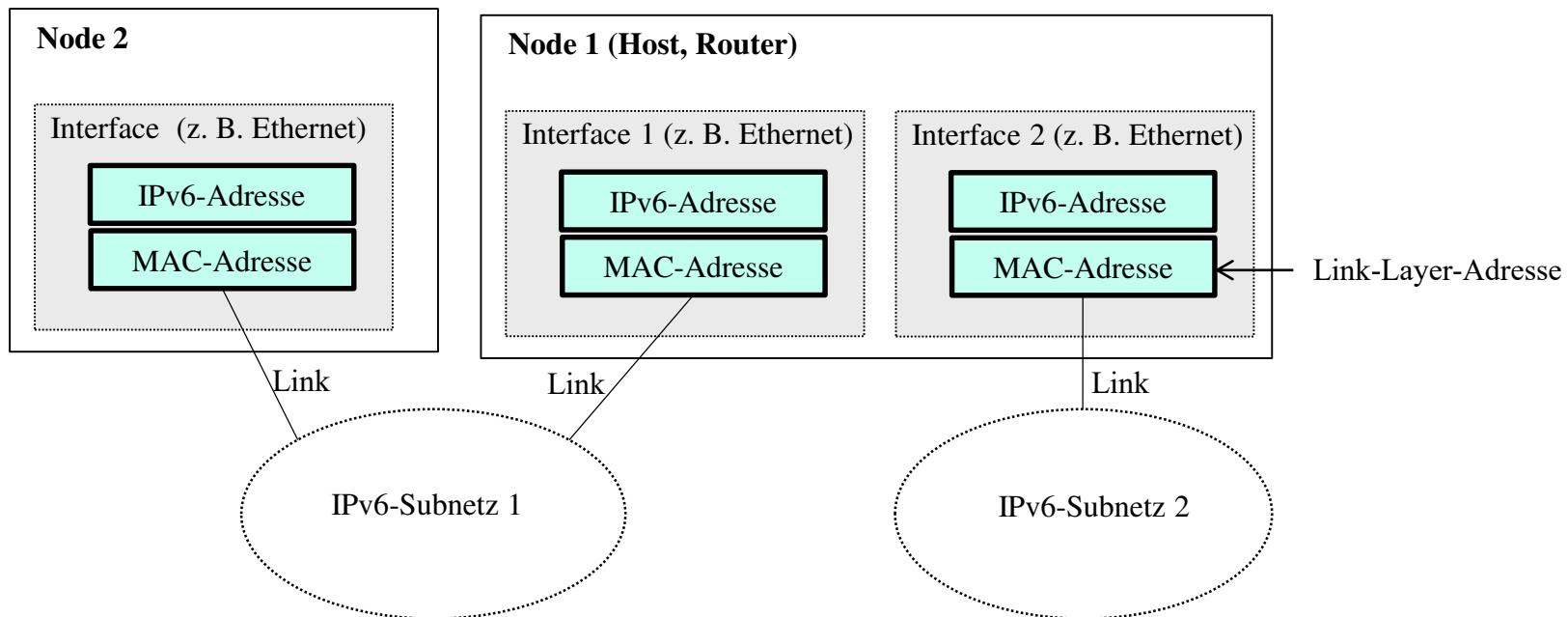
## Aktuelle IPv6-Nutzung

---

- In 2011 gab es mehr als 6000 IPv6-Präfixe in der globalen Routing-Tabelle, in 2016 waren es schon 33000
- In 2011 beteiligten sich 11 % der autonomen Systeme an IPv6, in 2016 waren es schon 26 %
- In 2018 verwendeten bereits über 22 % der Google-Nutzer weltweit IPv6 (siehe Google IPv6 Statistics 2018)
- Immer noch mehr Nutzung zur Kommunikation zwischen den Autonomen Systemen / ISPs

# IPv6-Terminologie

- Node
- Interface
- Link



# IPv6-Adressen

---

- **16-Byte-Adressen** (128 Bits) mit neuer Notation
- Analogie zur Anzahl der vorhandenen Adressen ( $2^{128}$ ):
  - Wäre die ganze Welt mit Computern bedeckt, könnte man mit IPv6  $7 * 10^{23}$  IP-Adressen pro m<sup>2</sup> ermöglichen
- Verschiedene Klassen von Adressen
  - **Unicast**-Adressen
    - Der traditionelle Adresstyp
    - Adressieren einen Netzanschluss eines Hosts oder Routers
  - **Anycast**-Adressen
    - Adressierung einer Gruppe von Interfaces
    - Aber nur ein Mitglied der Gruppe bekommt das Paket
    - Auswahl übernimmt der zuständige Router
    - Nutzung innerhalb von Teilnetzen, kein Routing außerhalb
  - **Multicast**-Adressen
    - Adressierung einer Gruppe von Interfaces
  - **Keine** Broadcast-Adresse mehr in IPv6!!
- Aufteilung des IPv6-Adressraums in **RFC 4291** geregelt

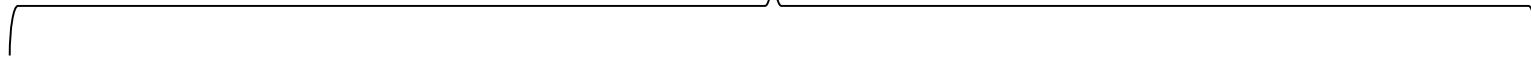
# IPv6-Adressaufbau: Struktur

## ■ Strukturierte IPv6-Adresse

- Netzwerkanteil über CIDR-Notation ( $x/y$ ) kennzeichnen, ISP erhält /32-Adressen von NIC oder DENIC usw. und gibt meist /64-Adressen weiter

Unstrukturierte IPv6-Adresse

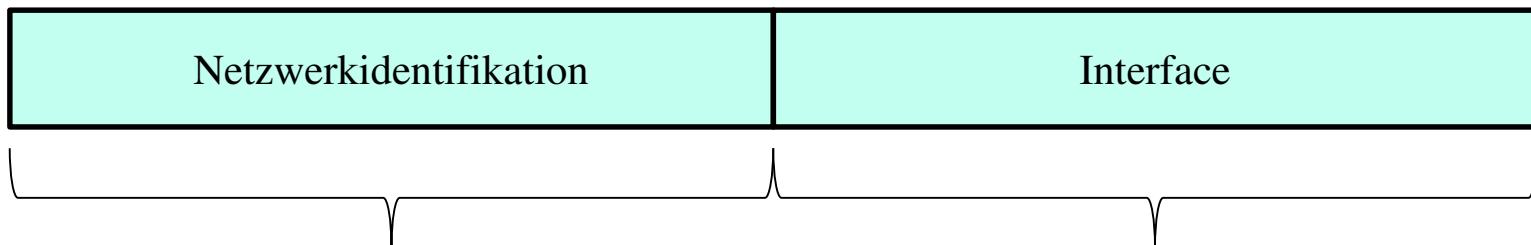
128 Bits



IPv6-Adresse

Netzwerkidentifikation

Interface



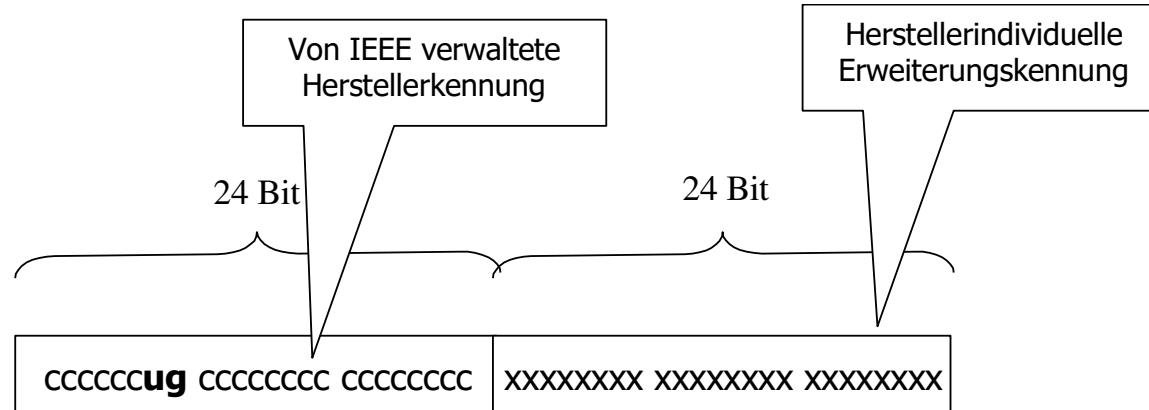
n Bits

128-n Bits

siehe RFC 4291

# Für Interessierte: IEEE 802.3-Adresse (Schicht 2) (1)

- 48-Bit-MAC-Adresse, als IEEE 802.3-Adresse bezeichnet
  - 24-Bit-Firmenkennung
  - 24-Bit-Erweiterungskennung (Platinenkennung)
  - Wird bei der Herstellung zugewiesen und ist global eindeutig
  - Dies ist die bekannte MAC-Adresse (Media Access Control)



c = Company-Bit

x = Bit vom Interface-Hersteller vergeben

u = universal oder local

g = group oder individual

siehe RFC 4291

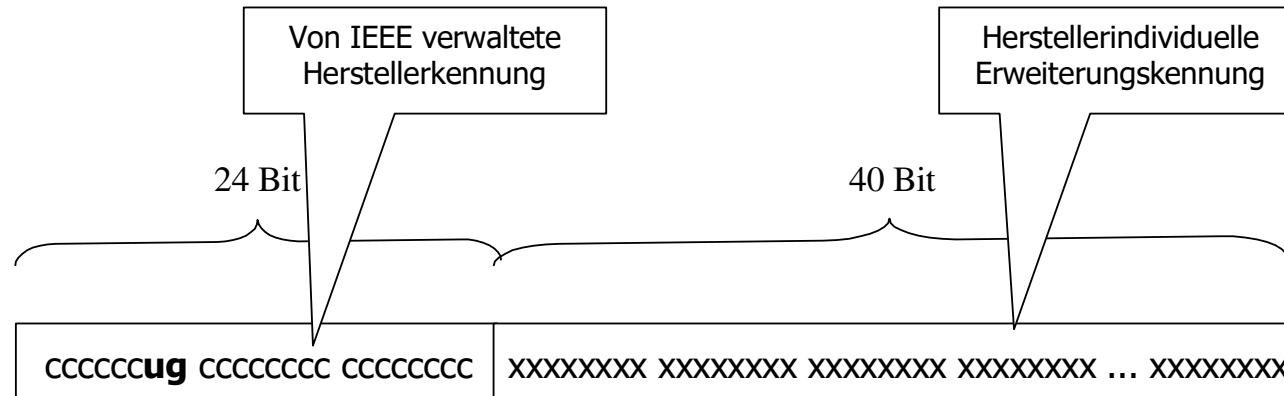
# Für Interessierte: IEEE 802.3-Adresse (Schicht 2) (2)

---

- U/L (Universal/Local) → u-Bit
  - Das U/L-Bit ist das siebte Bit des ersten Byte und wird zur Bestimmung, ob es sich um eine universell oder eine lokal verwaltete Adresse handelt, verwendet.
    - U/L-Bit = 0 → Adresse von IEEE verwaltet
    - U/L-Bit = 1 → Netzwerkadministrator verwaltet Adresse lokal
- I/G (Individual/Group) → g-Bit
  - Das I/G-Bit ist das Bit mit niedrigster Priorität des ersten Byte
  - Dient zur Festlegung, ob es sich um eine individuelle Adresse (Unicast) oder eine Gruppenadresse (Multicast) ist
    - I/G-Bit = 0 → Unicastadresse
    - I/G-Bit = 1 → Multicastadresse
- Bei einer 802.x-Standard-Netzwerkadapteradresse gilt:
  - U/L-Bit = I/G-Bit = 0 → universell verwaltete MAC-Unicastadresse

# Für Interessierte: EUI-64-Adresse (Schicht 2)

- IEEE EUI-64-Adresse (Extended Unique Identifier) ist ein neuerer Standard in der Adressierung von Netzwerkschnittstellen (siehe RFC 2373). Zwei Adressteile:
  - Firmenkennung ist 24 Bit lang
  - Erweiterungskennung ist 40 Bit → größerer Adressbereich für Hersteller von Netzwerkadaptersn
- Die IEEE EUI-64-Adresse verwendet die U/L- und I/G-Bits auf dieselbe Art wie die IEEE 802.3-Adresse

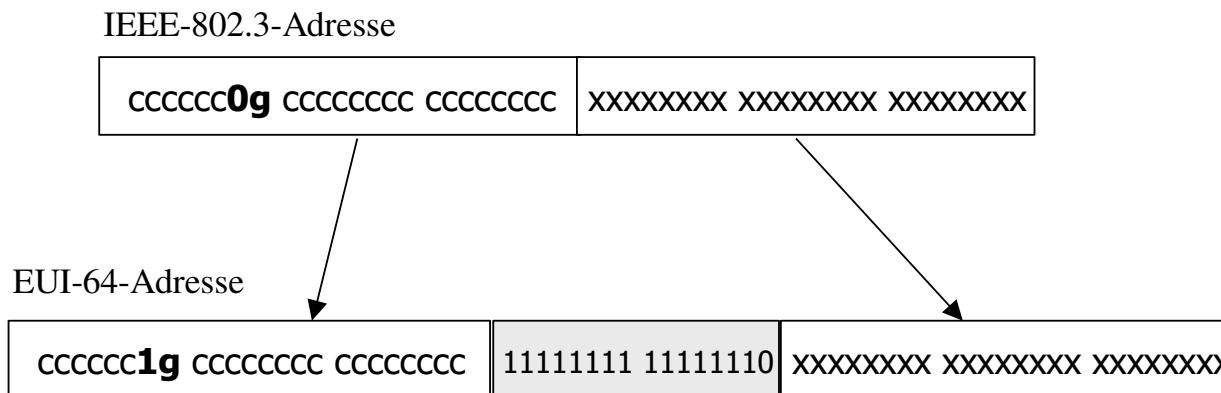


siehe RFC 4291

# Für Interessierte:

## Mapping IEEE 802.3-Adresse → EUI-64-Adresse

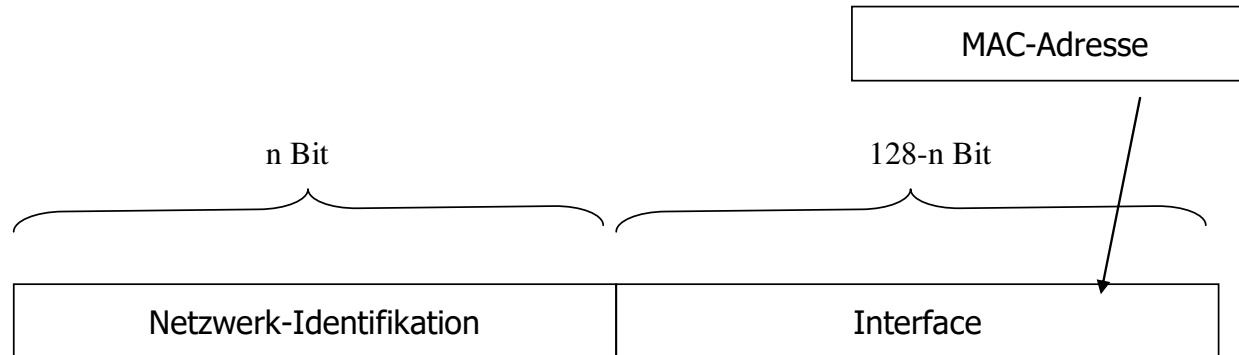
- IEEE EUI-64-Adresse ist länger, also nicht direkt abbildungbar
- 16 Bit 0b1111111111111110 = 0xFFFF werden zwischen der Firmenkennung und der Erweiterungskennung in die IEEE 802.3-Adresse eingefügt
- u-Bit wurde für IPv6 wegen einfacherer Schreibweise einer link-lokalen Adresse invertiert :
  - Beispiel: FE80::124 einfacher als FE80::200:0:0:124



# IPv6-Adressaufbau: Netzwerk-Id

---

- MAC-Adresse wird als Interface-Identifikation übernommen
  - Sicherheitsproblem, konnte man leicht manipulieren
  - Umfangreiche Diskussionen dazu
  - Daher RFC 4941 (privacy Extensions), um zufällige Interface Identifier zu erzeugen
- Abbildung IEEE-803.3-Adresse auf IEEE EUI-64-Adresse wird dabei vorgenommen



# IPv6-Adressaufbau und Regeln

- Adressen-Notation mit 8 Gruppen zu je vier Hex-Zahlen abgetrennt durch Doppelpunkte, CIDR-Notation (x/y) auch zulässig

Beispiel:

8000:0000:0000:0000:**0123**:5555:89AB:CDEF

- 
- Führende Nullen können in jeder Gruppe weggelassen werden und Gruppen mit lauter Nullen können durch einen Doppelpunkt ersetzt werden, aber „::“ nur an einer Stelle möglich:

8000::123:5555:89AB:CDEF

- 
- IPv4-Adressen können mit speziellen Unicast-Adressen (Präfix ::FFFF/96) abgebildet werden (Mapping-Adressen)

Beispiel: 192.168.0.1 → ::FFFF:C0A8:1

# Besondere IPv6-Adressen

## Sonderformen

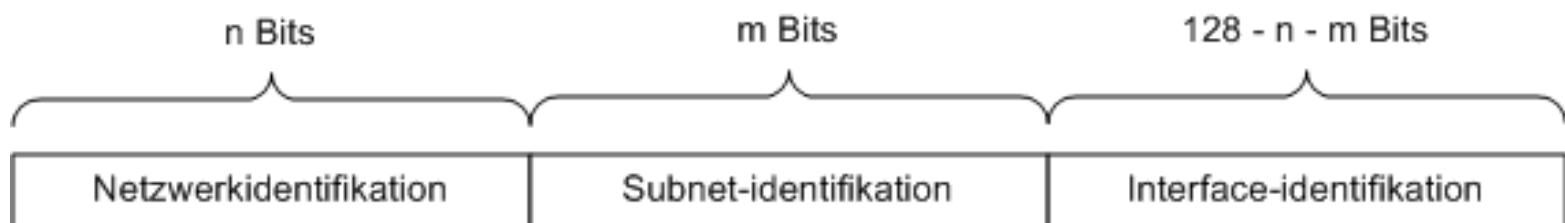
---

- ::0 entspricht 0.0.0.0 in IPv4 (undefinierte Adresse)
  - Synonym: 0:0:0:0:0:0:0:0 oder ::/128
- ::1 entspricht der Loopback-Adresse 127.0.0.1 in IPv4
  - Synonym: 0:0:0:0:0:0:1 oder ::1/128
- FF00::/8 weist auf eine Multicast-Adresse hin
- FE80::/10 weist auf eine Link-Local-Adresse hin

# Globale Unicast-Adressen: Aufbau, Struktur

---

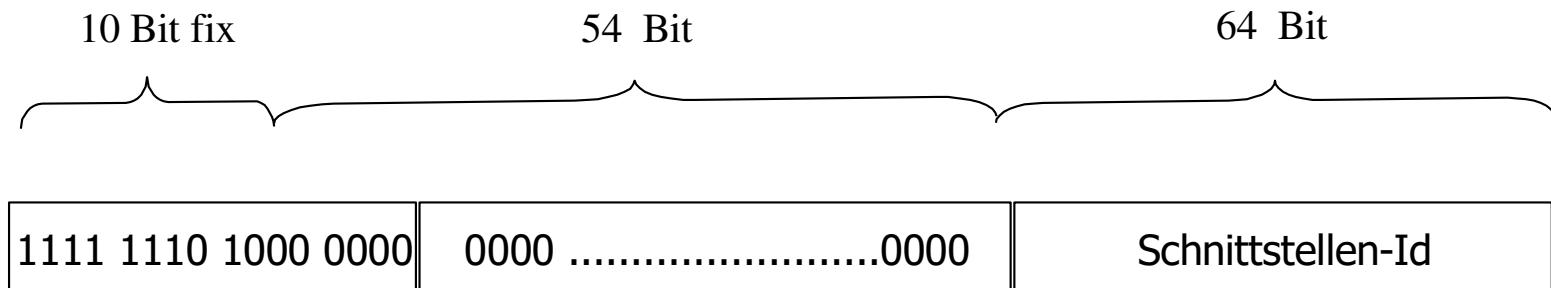
- Dienen dazu, einen Host (Knoten) im Internet **global** eindeutig zu identifizieren → **öffentliche** Adressen
- Hierarchiebildung möglich → Provider-/Netzbetreiberzuordnung
- Adresspräfix binär: 001 -> Hex: 20 .. 3F
- Eine Unicast-Adresse hat z.B. folgenden Aufbau:



# Link-lokale Adresse

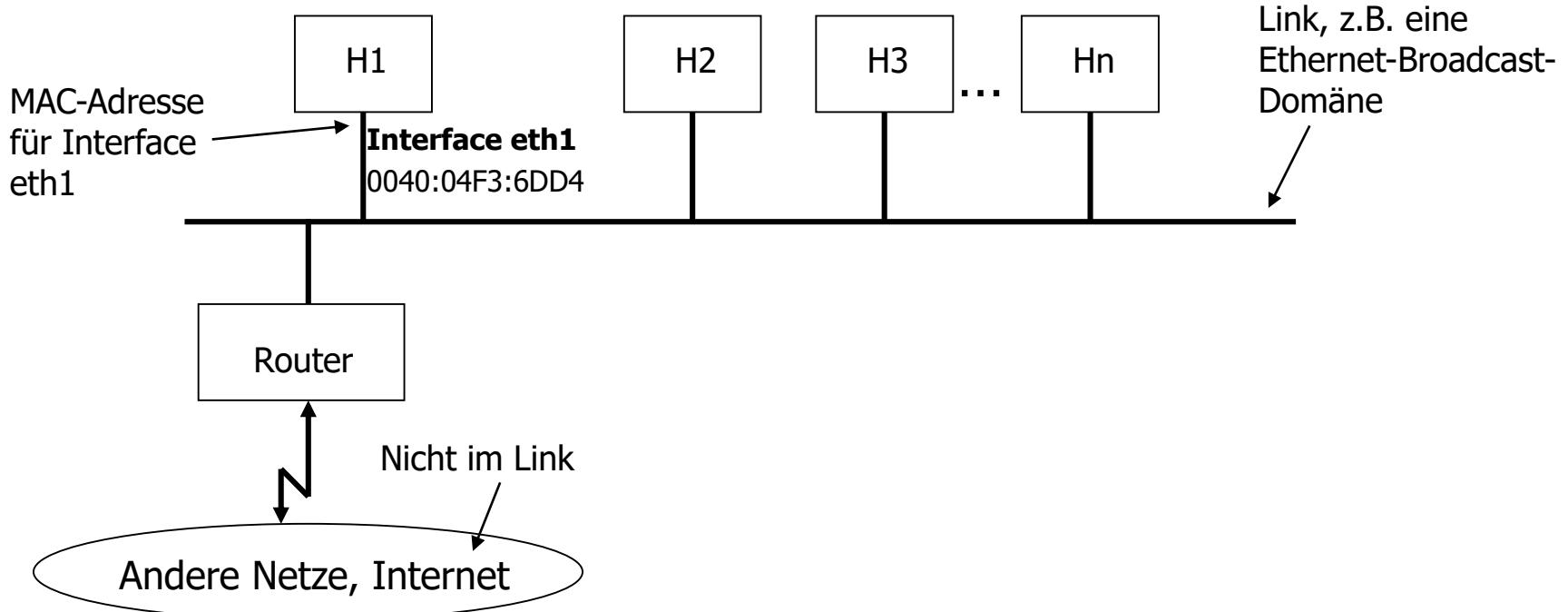
---

- Link-Local bezieht sich auf das lokale Netz (auf einen Link)
  - Jedes Interface an einem Link generiert sich selbst eine Link-Local-Adresse
  - Nutzung zur Kommunikation mit anderen Knoten auf demselben Link
- Präfix der Adressen: 1111 1110 10 → FE80::/10 – FEBF::/10
- Typischer Aufbau einer link-lokalen Adresse:



# Link-lokale Adressierung

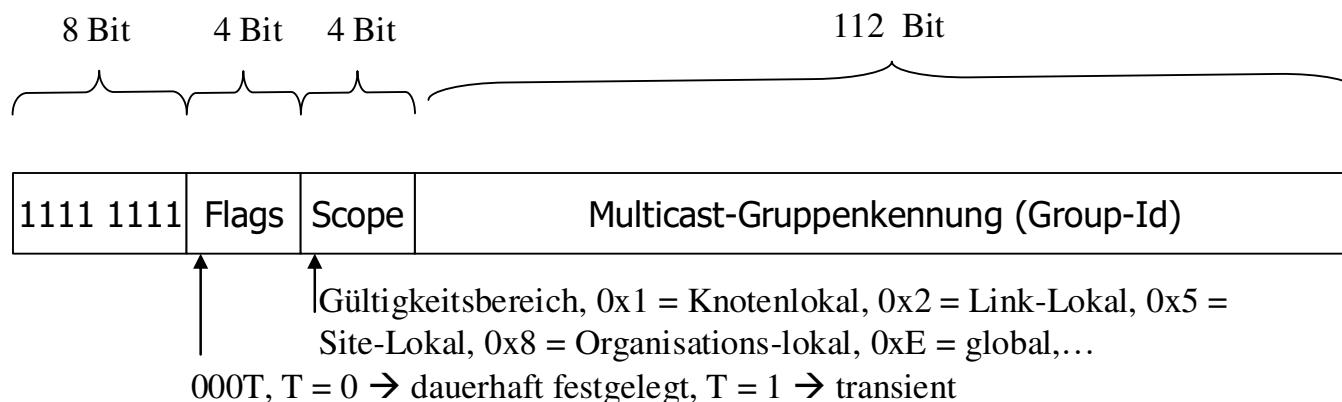
- MAC-Adresse wird abgebildet auf link-lokale Adresse (EUI)
- Beispiel für Host H1:
  - Ethernet-MAC-Adresse 0040:04F3:6DD4
  - IP-Adresse FE80::0240:04FF:FEF3:6DD4
  - ping6 FE80::0240:04FF:FEF3:6DD4%eth1



# IPv6-Multicast-Adressen

## ■ Multicast-Adressen

- Multicast-Adressen beginnen mit 0xFF
- Knotenlokale Adresse **FF01**:... → knotenlokal, Nachricht verlässt Knoten nicht
- Für das gleiche Link-Segment **FF02**:... → Nachricht verlässt Knoten, bleibt aber im Subnetz
- **FF0E**: Entspricht der IPv4-Broadcast-Adresse (limited)
- ...

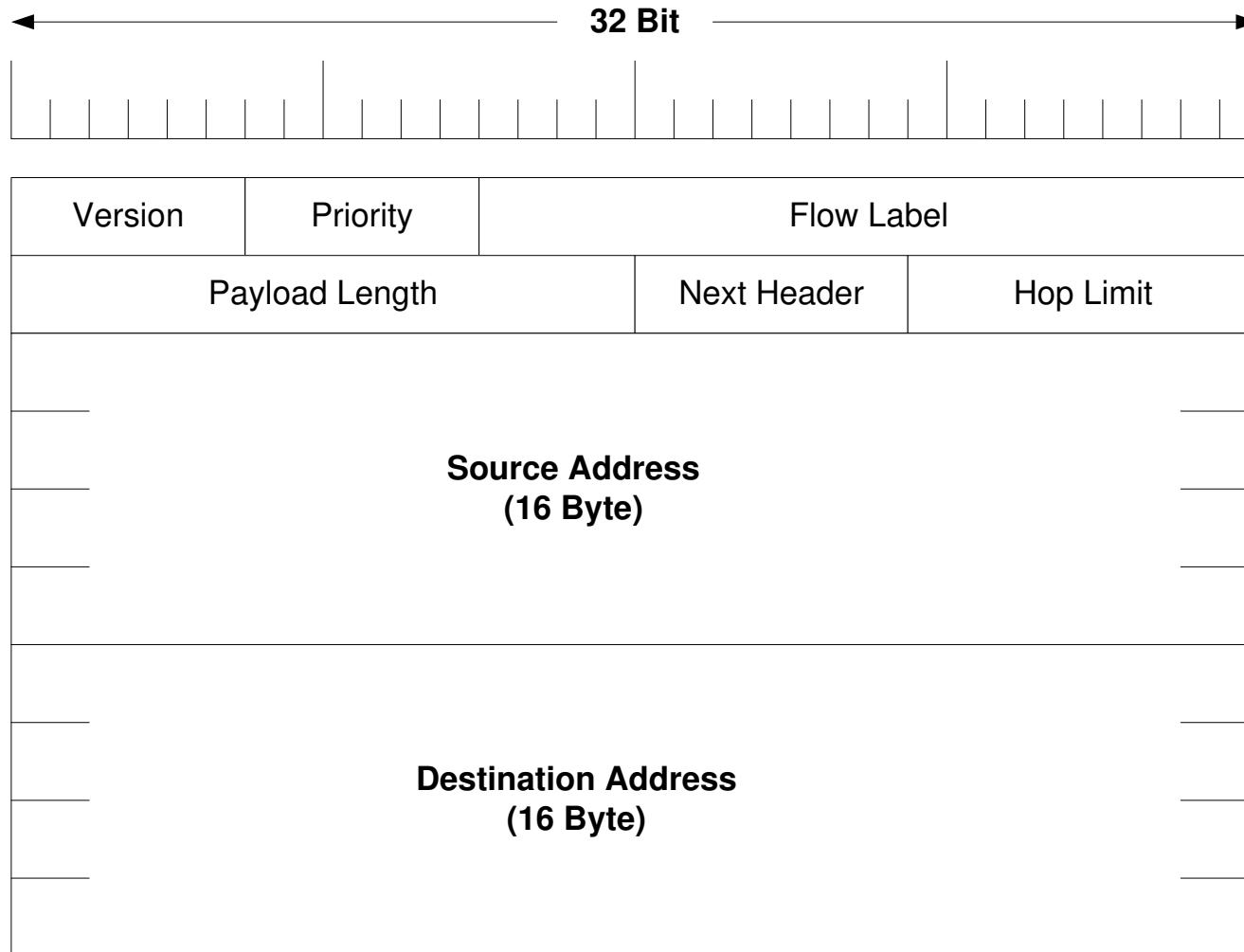


# Überblick

---

1. IPv6-Grundlagen und Adressierung
- 2. IPv6-PDU**
3. Discovery-Automatismen

# IPv6-Header



**Keine Prüfsumme!**  
→ TCP, UDP, ...

# IPv6-Header

---

Version	Priorität	Flow Label (Flussmarke)
---------	-----------	-------------------------

- **Version** Versionsnummer des Internet-Protokolls (6)
- **Priority:** auch **Traffic Class / DS** = Differentiated Service Field (neu) mit neuen Werten (RFC 2474, 2478)
  - Information für Router, interessant bei Überlastsituationen
    - stoßartiger Verkehr (ftp, NFS) → hoher Durchsatz
    - interaktiver Verkehr (telnet) → geringe Verzögerung
    - Verkehrsarten ohne Staukontrolle (z.B. für Videoanwendungen)
- **Flow Label (Flussmarke):** Fluss-Id, falls ungleich 0
  - Zweck: Zusammengehörige Datenflüsse End-to-End (Video/Audio) auf Netzebene speziell behandeln
  - Quelladresse+Zieladresse+Flussmarke kennzeichnen einen Fluss
  - Flussmarken werden im Quellknoten in die IPv6-PDU eingetragen

# IPv6-Header

---

Payload Length	Next Header	Hop Limit
Source und Destination Address	...	

- **Payload Length:** Nutzdatenlänge **ohne** die 40 Bytes des IPv6-Headers, es gibt aber auch Jumbo-Pakete
- **Next Header:** Verweis auf ersten Erweiterungs-Header
  - Letzter Header verweist auf Protokolltyp der nächst höheren Schicht (siehe IPv4-Feld **Protokoll**)
- **Hop Limit:** Verbleibende Lebenszeit des Pakets in Hops
  - Jeder Router zählt Hop Limit um 1 herunter
  - Entspricht dem TTL-Feld in IPv4
  - Name entspricht jetzt der eigentlichen Nutzung im Internet
- **Source und Destination Adresse:** IPv6-Adressen der Quelle und des Ziels

# IPv6-Header: Erweiterungs-Header - Überblick

---

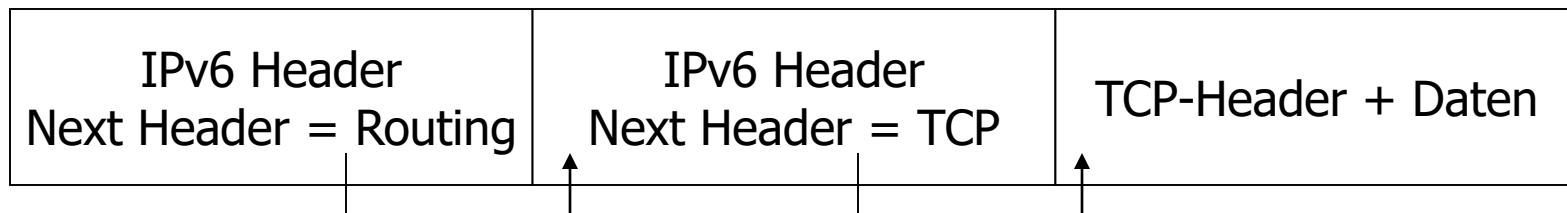
- Kodierung im Next-Header-Feld
  - EGP = 0x08
  - Routing = 0x2B
  - Fragment = 0x44
  - TCP = 0x06 ...

<b>Erweiterungs-Header</b>	<b>Beschreibung</b>
Optionen für Teilstrecken (Hop-by-Hop)	Verschiedene Informationen für Router
Routing	Definition einer vollen oder teilweisen Route
Fragmentierung	Verwaltung von Datagrammfragmenten
Authentifikation	Echtheitsüberprüfung des Senders
Verschlüsselte Sicherheitsdaten	Informationen über den verschlüsselten Inhalt
Optionen für Ziele	Zusätzliche Informationen für das Ziel

# IPv6-Header: Erweiterungs-Header - Verkettung

---

- Header und Erweiterungs-Header sind miteinander **verkettet**, jeder Typ **max. einmal**
- Die Erweiterungen werden **nicht** in den Routern bearbeitet, nur in den Endsystemen
- Eine **Ausnahme**: Routing-Erweiterungs-Header
- Reihenfolge der Header ist festgelegt
  
- Beispiel eines IPv6-Headers mit einem Erweiterungs-Header und einer anschließenden TCP-PDU



# IPv6-Header: Erweiterungs-Header - Fragmentierung

---

- Der **Fragmentierungs-Header** wird verwendet, um größere Dateneinheiten zu senden, als zugelassen
  - PDU-Länge > MTU des Pfades (MTU = Maximum Transmission Unit)
  - Minimum in IPv6 1280 Bytes gemäß RFC 2460
- Fragmentierung erfolgt bei IPv6 **nur im Quellknoten**, Router fragmentieren nicht → geringe Routerbelastung

Next Header	reserviert	Fragment Offset	00M
Identifikation			

- Fragment Offset:** Position der Nutzdaten relativ zum Beginn der PDU (Ursprungs-Dateneinheit) → 13 Bit (wie IPv4)
- Identifikation:** Id der PDU (wie IPv4)
- M:** More-Flag, M=1 → weitere Fragmente folgen (wie IPv4)

# IPv6: Sicherheitsaspekte

---

- Im Gegensatz zu IPv4 sind in IPv6 schon Sicherheitsmechanismen im Protokoll spezifiziert (siehe IPv4+**IPsec**)
  - Authentifizierung
  - Verschlüsselung
- **MD5-Algorithmus** (Message Digest) kann zur Authentifizierung der Partner verwendet werden
- Verschlüsselung des Nutzdatenteils wird mit einer Variante des **DES- oder AES-Verschlüsselungsalgorithmus** unterstützt
  - DES = Data Encryption Standard
  - AES = Advanced Encryption Standard
  - Symmetrisches Verschlüsselungsverfahren

## Einschub: ICMPv6

---

- ICMPv6 ist **zwingend** für IPv6 notwendig
- Viele neue Fehlernachrichten und Informationsnachrichten

<b>ICMP-Typ</b>	<b>Beschreibung</b>
132	Multicast Listener Done
133	Router Solicitation
134	Router Advertisement
135	Neighbor Solicitation
136	Neighbor Advertisement
137	Redirect
138	Router Renumbering
139	ICMP Node Information Query
140	ICMP Node Information Response

...

1. IPv6-Grundlagen und Adressierung
2. IPv6-PDU
- 3. Discovery-Automatismen**

# Autokonfiguration: Einige wichtige Features

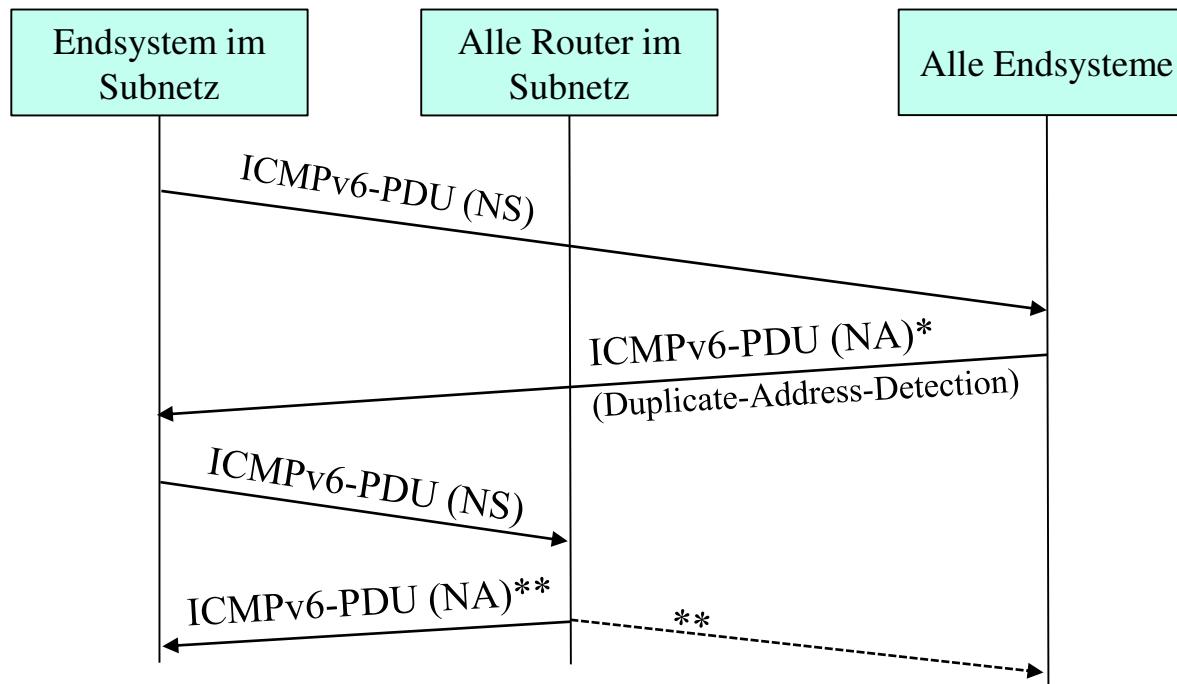
---

- **Selbstkonfiguration:** Host konfiguriert seine eigene Adresse dynamisch
    - Die automatische IP-Adress-Konfiguration für Interfaces
  - **Neighbor Discovery Protocol – NDP** als Ersatz für ARP
    - Automatische und dynamische Adress-Auflösung für Layer-2-Adressen
  - **Stateless Address Autoconfiguration (SLAAC)** als Ersatz für DHCP
  - **Router Discovery:** Automatisches Auffinden von Routern im gleichen Link (Subnetz)
  - **Parameter Discovery:** Die dynamische Zuordnung von Konfigurationsparametern
  - Die Suche nach der optimalen MTU-Size zwischen Sender und Empfänger (**Path MTU Discovery**)
  - **Multicast Listener Discovery - MLD** als Ersatz für IGMP
-

# Beispiel

## Stateless Address Autoconfiguration (SLAAC)

- Ablauf einer Stateless Address Autoconfiguration (SLAAC)
  - Ersatz für DHCP
  - Nutzung von ICMPv6-Nachrichten



## Beispiel Router-Discovery

---

- Wenn ein Endsystem seinen nächsten Router sucht, sendet es eine ***Router-Solicitation-Nachricht*** über Multicast an die Adresse **FF02::2**
- Router antworten mit einer ***Router-Advertisement-Nachricht***
- Damit unterstützt das ND-Protokoll das Auffinden des verantwortlichen Routers zur Laufzeit
  - DHCP kann auch wegfallen
- Mehrere Router können aktiv sein
- Das ND-Protokoll nutzt zur Abwicklung seiner Aufgaben einige ICMPv6-Nachrichten

# Beispiel

## Parameter-Discovery

---

- Netzwerkparameter werden vom Host zum Startzeitpunkt auch über ***Router-Solicitation-Nachricht*** besorgt (DHCP-Aufgaben)
- Nachricht geht an Multicast-Adresse **FF02::2**
- Ein Router antwortet mit einer ***Router-Advertisement-Nachricht*** an die Link-Adresse des Endsystems
- Folgende Parameter kann eine *Router-Advertisement*-Nachricht u.a. übertragen:
  - *Max-Hop-Limit*: Dies ist der Wert „Hop-Limit“ der in die IPv6-PDUs eingetragen wird
  - *Retransmission-Timer*: Zeit in Millisekunden, die seit dem Absenden der *Solicitation*-Nachricht ablaufen darf, bevor wiederholt wird
  - ...
  - Über ein *Optionsfeld* wird z.B. vom Router auch die *MTU-Size* übermittelt

## Rückblick und Weiterführendes

---

- ✓ IPv6-Grundlagen und Adressierung
- ✓ IPv6-PDU
- ✓ Discovery-Automatismen

- 1 Kommunikationssysteme und verteilte Anwendungen
- 2 Grundlagen der Transportschicht
- 3 Transportprotokolle TCP und UDP
- 4 Grundlagen der Vermittlungsschicht
- 5 Internet und Internet Protocol (IP)
- 6 Routing-Verfahren und -Protokolle
- 7 Internet-Steuerprotokolle und DNS
- 8 Internet Protocol Version 6 (IPv6)
- 9 **Netzwerkschnittstelle**

# Überblick über das Kapitel

---

## 1. Bitübertragungsschicht

- Aufgaben, Begriffe und Definitionen
- Kodierung (Leitungskodierung)
- Laufzeit, Übertragungszeit, Transferzeit, Jitter
- Datenübertragungsmedien und Verkabelung
- Netzzugangstechnologien

## 2. Sicherungsschicht

- Aufgaben, Framing und Übertragung
- Fallbeispiel CSMA/CD und Ethernet
- Fehlererkennung am Beispiel von Ethernet

# Bitrate, Schrittgeschwindigkeit und Bandbreite

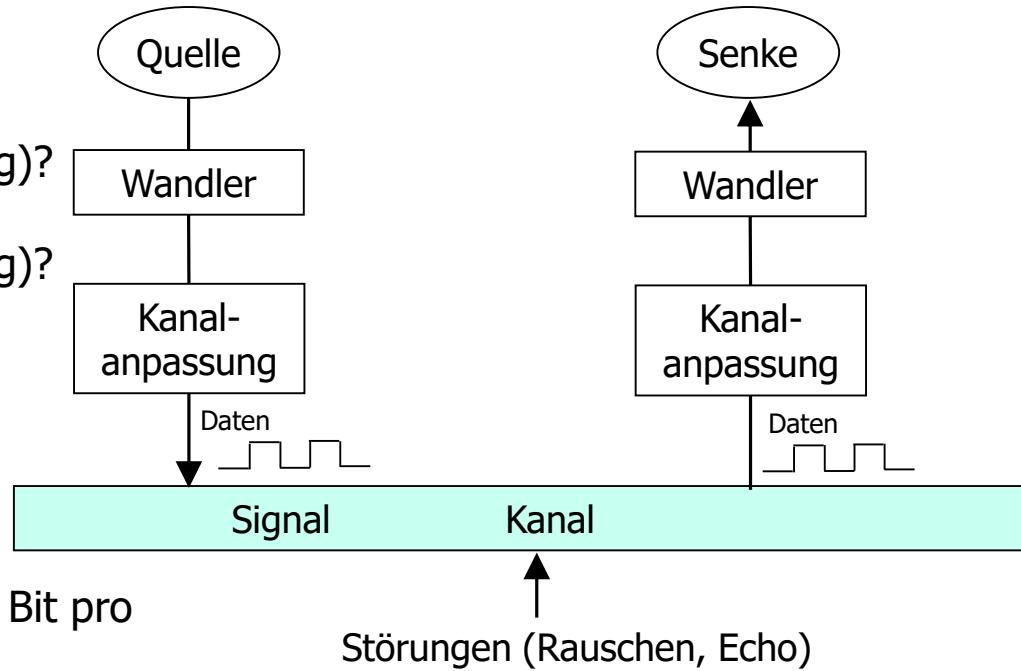
## ■ Schrittgeschwindigkeit S

- Die Anzahl der Zustandsänderungen eines Signals pro Zeiteinheit
- Einheit: baud = bd = 1/s (Hz)

- **Daten:** Was wird übertragen? (digital, analog)?
- **Signal:** Wie wird es übertragen? (digital, analog)?

## ■ Bandbreite B

- Physikalische Eigenschaft des Mediums,
- Analoge Welt, Einheit: Hz
- Digitale Welt, Einheit: Bit/s



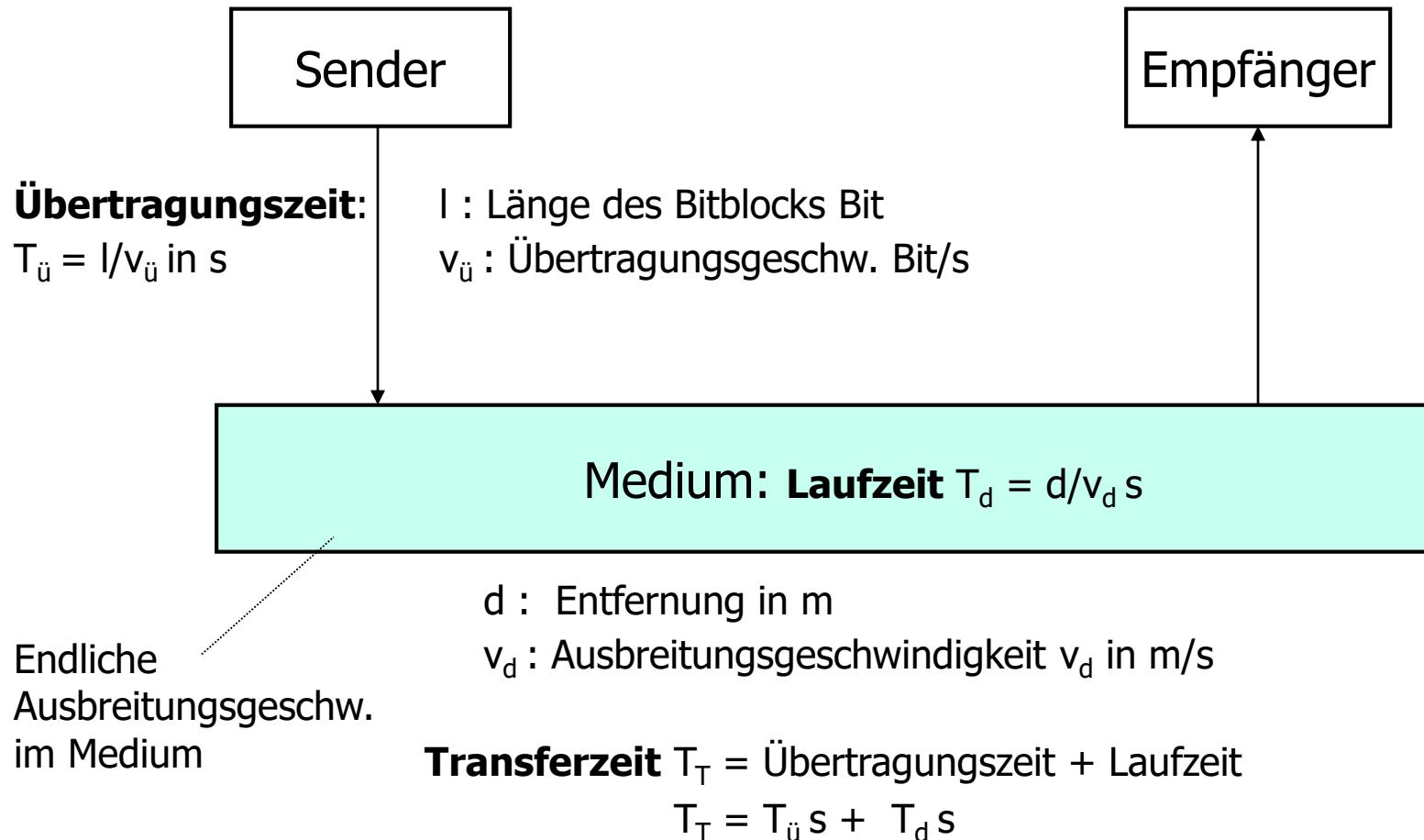
## ■ Bitrate R

- Anzahl der übertragbaren Bit pro Zeiteinheit
- Gemessen in Bit/s

Für unsere Betrachtung: Datenrate = Übertragungsrate = Bitrate

# Laufzeit, Übertragungszeit, Transferzeit

---



# Laufzeit, Übertragungszeit, Transferzeit

---

- Typische Ausbreitungsgeschwindigkeiten und dazu gehörige Laufzeiten:

<b>Medium</b>	<b>Ausbreitungsgeschw. <math>v_d</math> [m/s]</b>	<b>Laufzeit <math>T_d</math> [<math>\mu</math>s/km]</b>
Funkkanal	$3 * 10^8$ (näherungsweise)	3,33
Freiraum-Infrarot	$3 * 10^8$ (näherungsweise)	3,33
Glasfaserleitung (Quarzglas)	$2 * 10^8$	5
Basisband-Koaxialkabel (50/75 Ohm)	$2,3 * 10^8$	4,33
Zweidrahtleitung (verdrillt)	$2,5 * 10^8$	4

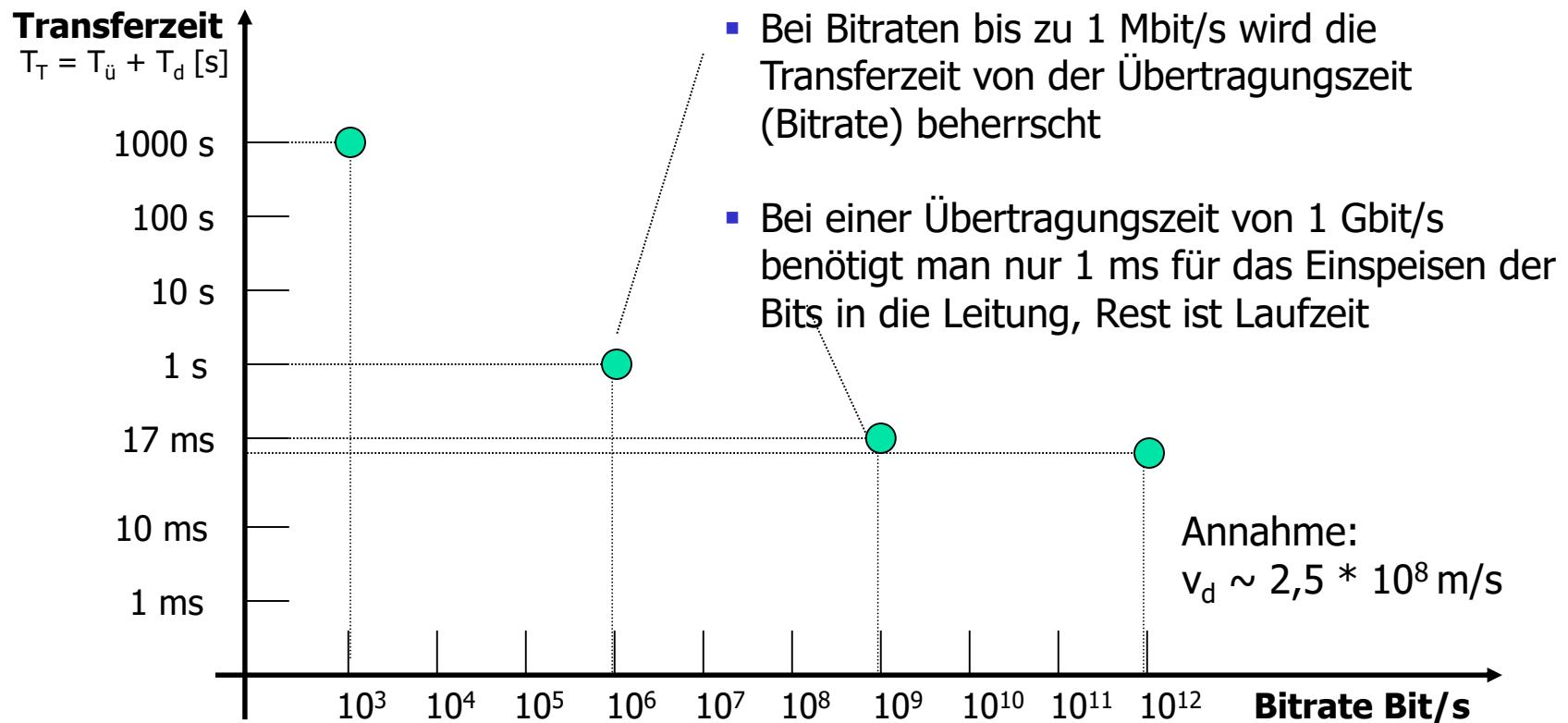
# Laufzeit und Übertragungszeit

---

- Bei niedrigen Übertragungsgeschwindigkeiten ist die Laufzeit gegenüber der Übertragungszeit vernachlässigbar:
  - Übertragungszeit  $T_{ü}$  entspricht ungefähr der Transferzeit  $T_T$  da Laufzeit  $T_d$  verschwindend klein ist
  - $T_{ü} \sim T_T$
- Bei sehr großen Entfernungen (mehrere Tausend km) spielt die Laufzeit eine große Rolle

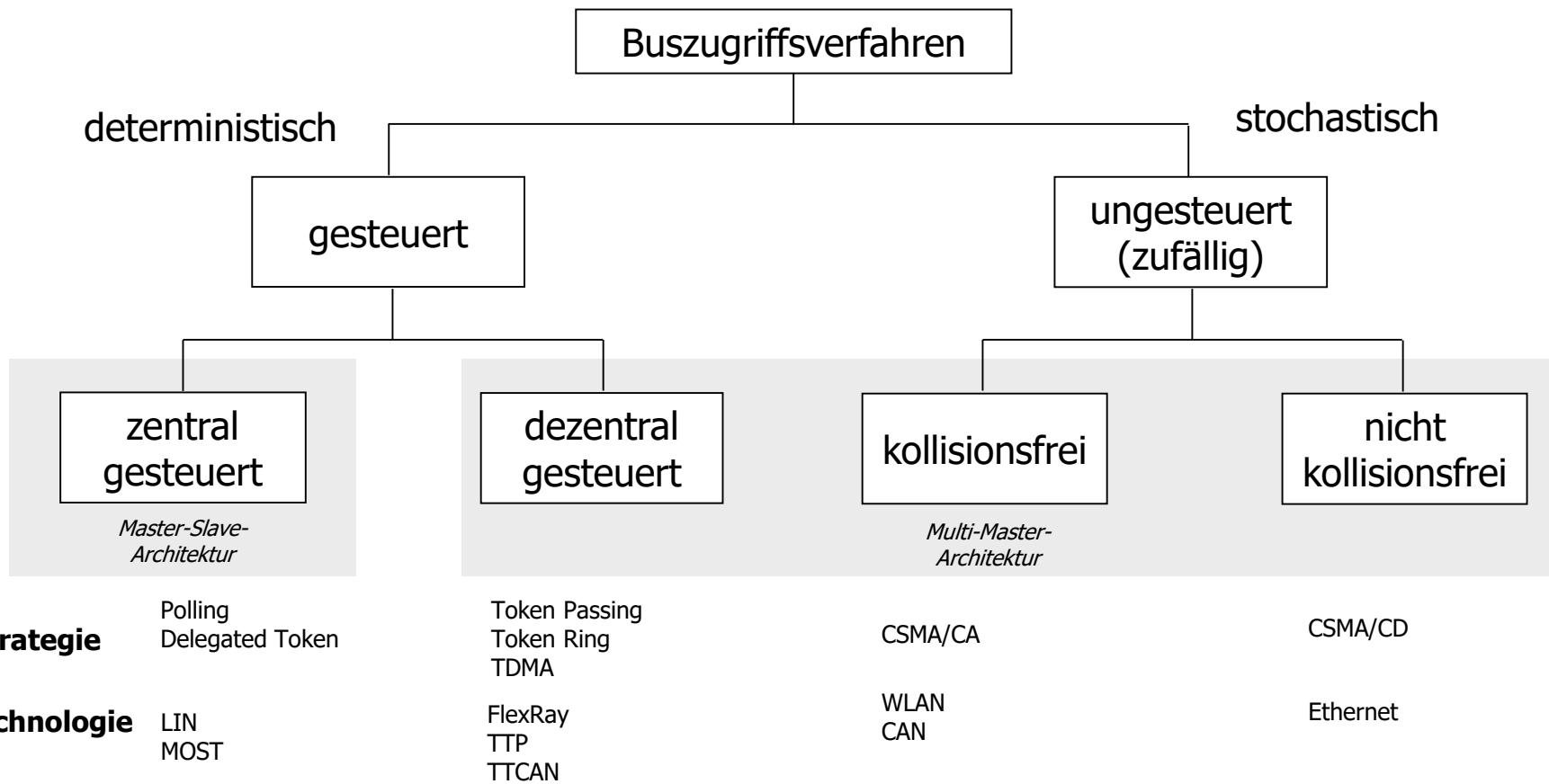
# Übertragung bei großen Distanzen

- Übertragung einer Datei mit  $I = 1000000$  Bit ( $10^6$ ) über 4.000 Km ( $4 * 10^6$  m)



# Buszugriffsverfahren

- Bus als gemeinsam genutztes Medium
- Einteilung der Zugriffsverfahren



# CSMA-Protokolle

---

## ■ **CSMA** (Carrier Sense Multiple Access)

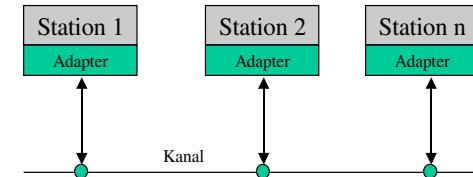
- Trägererkennungsprotokoll (Carrier Sense)
- Prüfung des Kanals vor dem Senden
- Nicht kollisionsfrei

## ■ **Non-Persistent CSMA**

- Kanal frei → Senden
- Kanal belegt → Zufällige Zeit warten, dann erneut versuchen

## ■ **p-persistent CSMA**

- Wenn Kanal frei ist, wird mit Wahrscheinlichkeit  $p$  gesendet und mit Wahrscheinlichkeit  $1-p$  eine zufällige Zeit  $t$  gewartet und dann erneut gesendet
- Bei belegtem Kanal beobachtet Station zunächst den Kanal



- Ethernet wurde Anfang der 70er Jahre von **Bob Metcalfe** entwickelt und als **IEEE 802.3-Standard** bekannt
- Die Architektur basiert auf der Definition von Funktionen der **beiden untersten Schichten** des ISO/OSI-Referenzmodells für
  - die Festlegung der physikalischen Eigenschaften der benötigten Komponenten
  - die Zugriffsverfahren der Stationen auf das Netz und
  - den Aufbau der versendeten Nachrichten

# Ethernet-Medienzugriffsverfahren

## CSMA/CD, Grundprinzip (1)

---

- CSMA/CD-Zugriffsverfahren mit **dezentraler Steuerung**
  - Medium wird von allen Stationen unabhängig abgehört, wenn Medium frei (keine Signalenergie) darf Station senden
  - **Kollision** möglich → Sendungen werden in diesem Fall eingestellt
- Genaue Bezeichnung des Verfahrens: **1-persistent CSMA/CD mit exponentiellem Backoff:**
  - Bei freiem Medium wird sofort gesendet (1-persistent)
  - Bei Kollision wird zufällige Zeit gewartet (Rückzieher) → verhindert erneute Kollision:
    - . Wenn Station schon  $i$  Kollisionen hatte, wird  $x$  aus dem Intervall  $[0, 2^{i-1}]$  gewählt
    - . Wenn Medium frei, dann  $x * \text{Zeitslot}$  (51,2 Mikrosekunden) berechnen und diese Zeit warten
  - **Stauauflösungsmechanismus!** (binär exponentielles Wachstum der Wahlmöglichkeiten)
  - Nach 16 Versuchen erfolgt Abbruch

# Ethernet-Medienzugriffsverfahren

## CSMA/CD, Grundprinzip (2)

---

- **MA** = „mehrfacher Zugriff“ von Rechnern auf ein Übertragungsmedium (Multiple Access)
- **CS** = „Befühlen des Mediums“: (Carrier Sense)
  - Sendewillige Station prüft, ob Kabel nicht gerade von einem anderen Rechner benutzt wird
  - Sendewillige Stationen hören den Bus ab und belegen ihn, wenn er frei ist (wenn keine andere Station bereits sendet)
- **CD** = Im **Kollisionsfall** Abbruch des Sendevorgangs und Wiederholung
- Stochastisches Verfahren
  - für zeitkritische Anwendungen nicht geeignet
  - nicht deterministisch

# Kollisionen im Ethernet

---

Station A



Paket startet  
bei Zeit 0

Station B



Szenario 1

Station A



Paket ist fast  
bei B

Station B



Szenario 2

Station A



Station B



Szenario 3

Station A



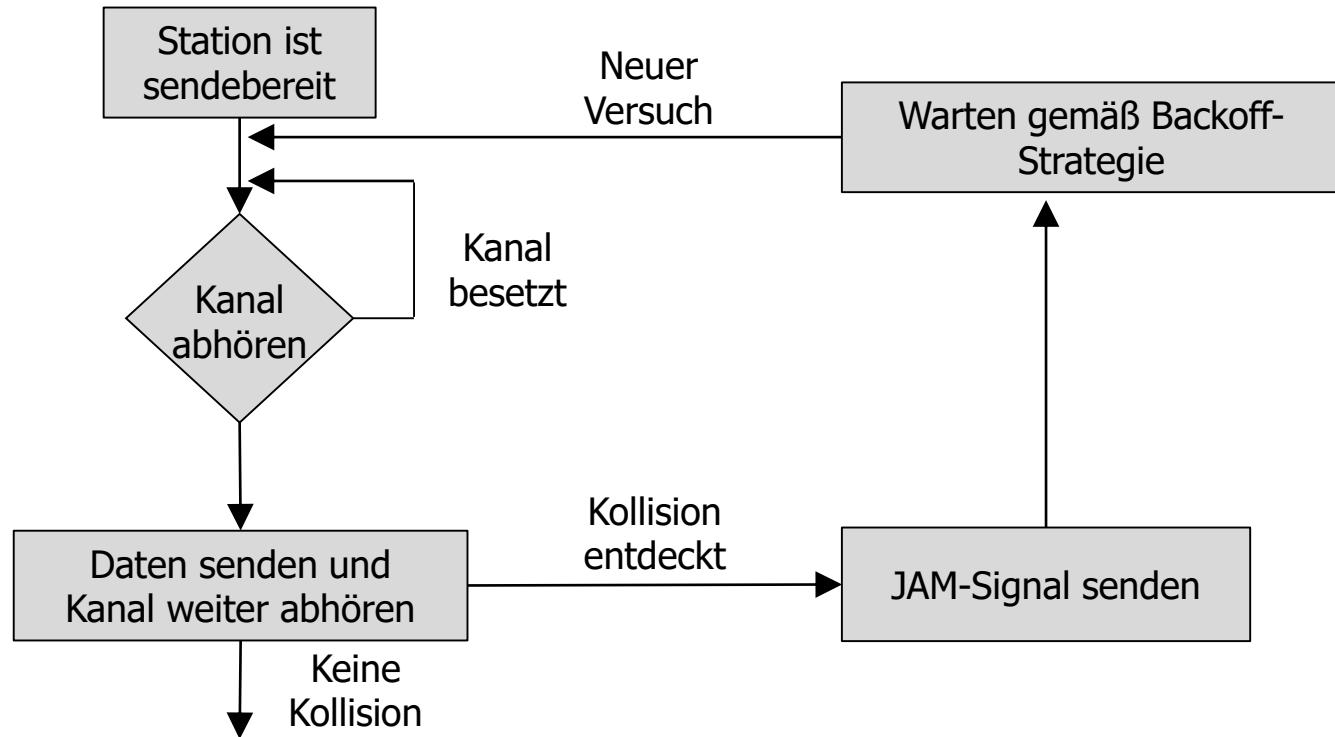
Kollision  
→ ←

Station B



Szenario 4

# Ethernet: Ablauf



## Backoff-Algorithmus:

- Algorithmus bestimmt nach einer Kollision eine Zeitspanne zum Warten, bevor sie einen neuen Sendevorschlag startet
- Die Zeitspanne ist ein Vielfaches von einem so genannten „Slot“, der 51,2 µs (bis 100 Mbit/s, danach kürzer) lang ist

# Ethernet-Paketaufbau

- Die Struktur des Ethernet-Frames ist grundsätzlich für alle Übertragungsraten gleich:

Präambel	7 Byte	Dient der Synchronisation der Station auf dem gemeinsamen Kabel
Start Frame Delimiter	1 Byte	SFD markiert den Anfang des Pakets
Zieladresse	6 Byte	Zur Identifikation des Empfängers: z. B. 00 00 0C 60 50 01 <sub>(16)</sub>
Quelladresse	6 Byte	HW-Adresse des Senders z. B. 00 06 7C 67 45 31 <sub>(16)</sub>
Pakettyp oder Längenfeld	2 Byte	IP 0800 (16) ARP 0806 <sub>(16)</sub>
Nutzdaten und Padding	0 Byte – 1500 Byte	Falls weniger als 46 Byte Nutzdaten, wird mit Füllbyte aufgefüllt (Padding)
Prüfsumme CRC	4 Byte	Cyclic Redundancy Check

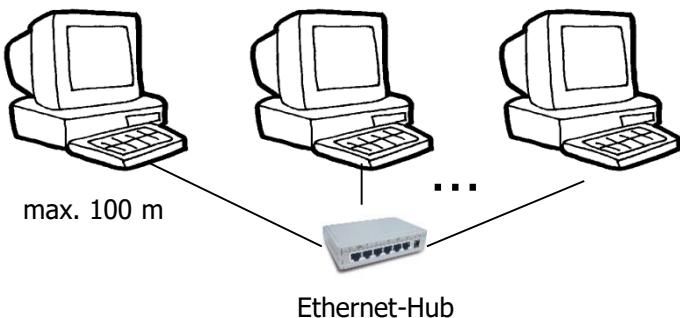
Eigene  
MAC-Adresse

- ARP=Address Resolution Protocol übersetzt die IP-Adresse eines Rechners in eine MAC-Adresse → siehe Internet-Protokolle

# Ethernet-Hubs

---

- **Verbindet mehrere Segmente** eines LANs, mehrere Ports
- Kommt ein Paket an einem Port an, wird es an alle anderen Ports weitergeleitet
- Hubs sind **nicht** vollduplex-fähig, Kollisionsbehandlung notwendig
- Hub-Varianten:
  - Ein **passiver Hub** überträgt Daten von einem Port an alle anderen
  - Ein **intelligenter Hub** beinhaltet Features, die es dem Administrator ermöglichen, den Verkehr des Hub zu überwachen und jeden Port im Hub zu konfigurieren
  - Ein **Switching Hub** liest die Zieladresse und gibt das Paket an den richtigen Port weiter

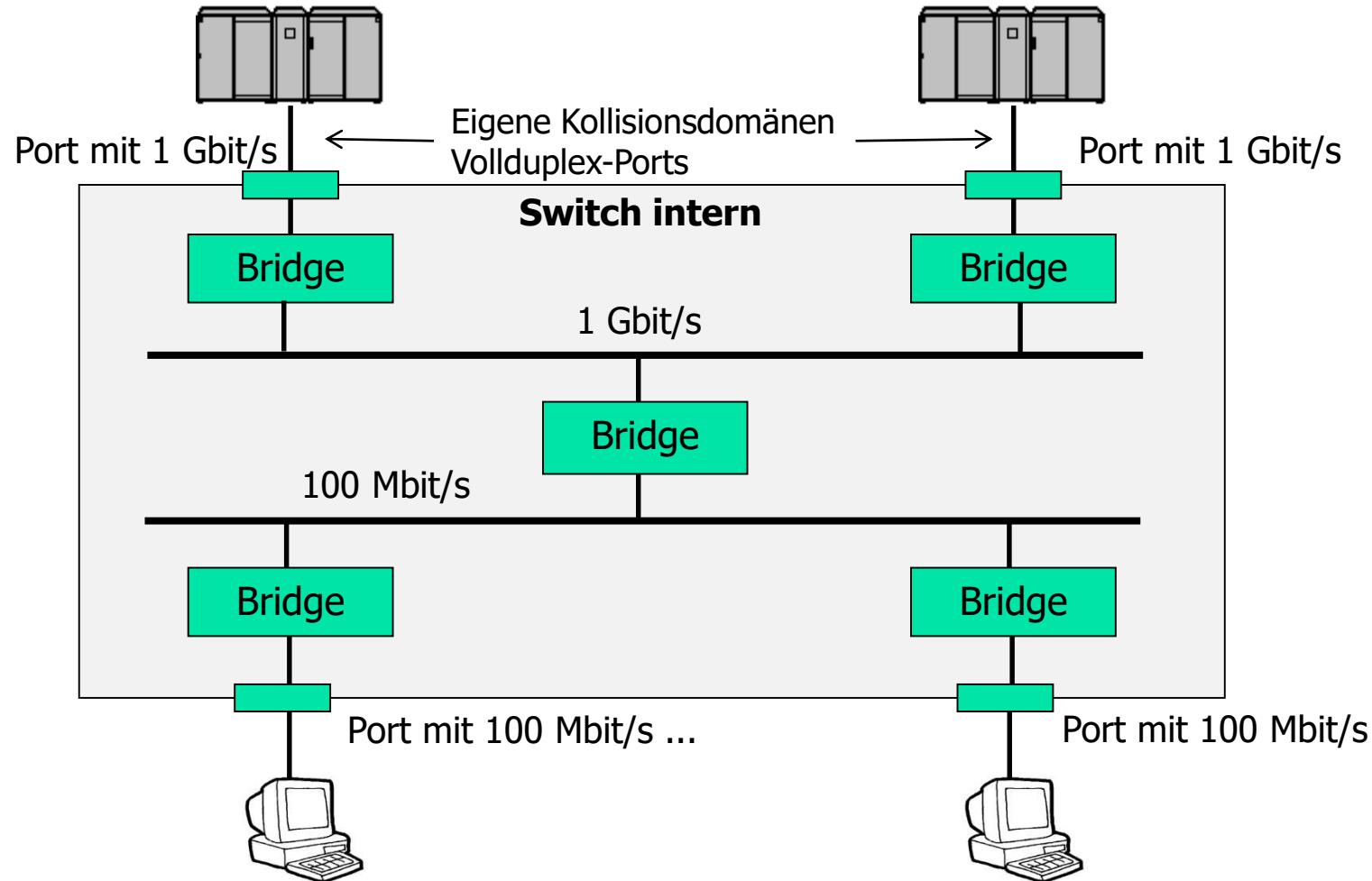


# Ethernet-Switches

---

- Switches arbeiten meist auf der **Schicht 2**, verbinden mehrere Segmente, auch **Multiport-Bridges** genannt
- Exklusive Leitung je Port möglich:
  - Jeder Port ist eine **eigene Kollisionsdomäne**, Verzicht auf „shared Medium“
  - **Keine Kollisionsbehandlung** mehr erforderlich
  - Ports können unabhängig voneinander empfangen und senden, **Vollduplexverbindungen** zwischen Stationen
  - Trotzdem noch CSMA/CD → eigentlich nicht mehr notwendig
- MAC-Schicht hat zusätzliche Flusssteuerung
  - Empfänger sendet Pausenrahmen zur Vermeidung von Pufferüberläufen im Switch
- Ein Switch kann in einem Ethernet-LAN verschiedene Gruppen schalten
  - Z.B. können 100 Mbit/s-Segmente mit 10 Mbit/s-Segmente verbunden werden

# Switch-Innenleben: Interne Bridges und Hochleistungsbus



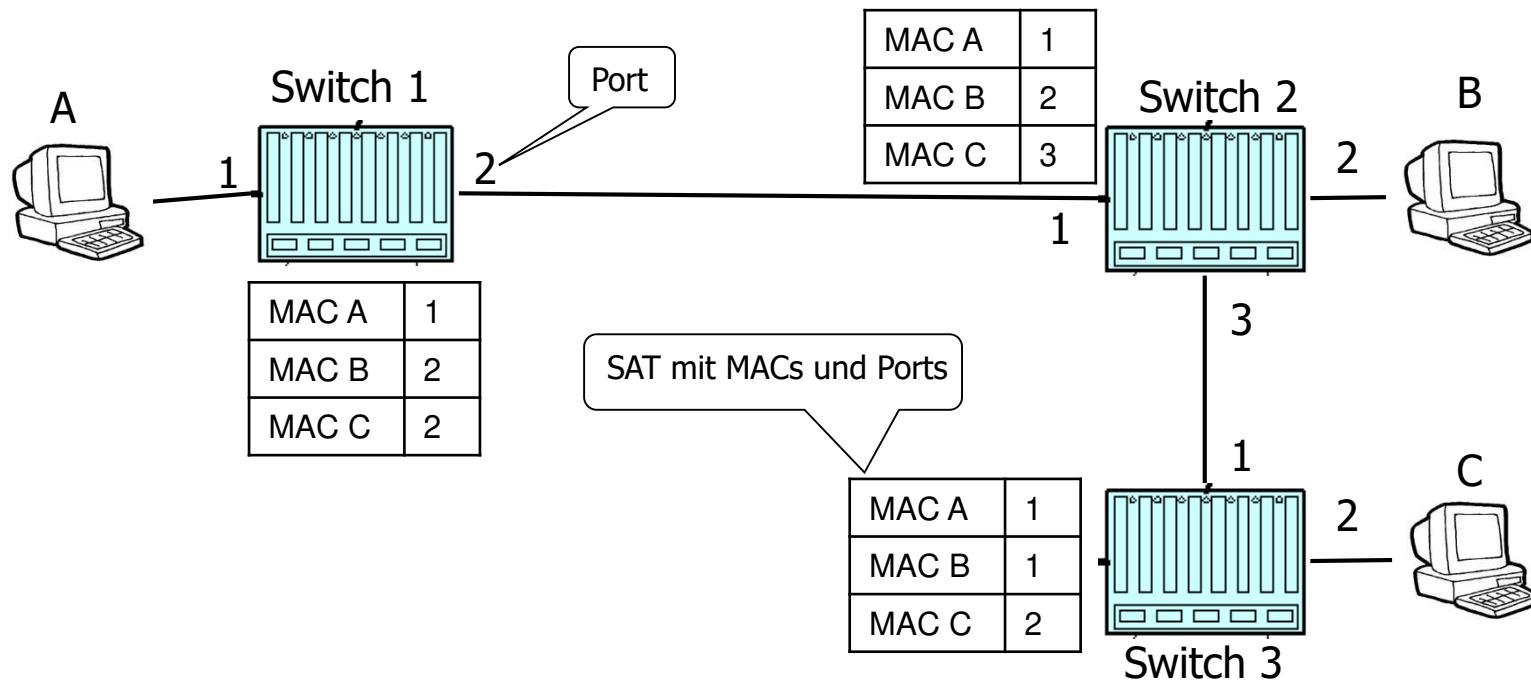
## Lernfähige Switches: Source Address Table (1)

---

- Switches lernen die MAC-Adressen der angeschlossenen Segmente automatisch
- Bei empfangenem Frame speichert ein Switch die MAC-Adresse und den Port in der **Source Address Table (SAT)**
- SAT-Eintragung wird üblicherweise bei ARP-Addressabfragen vorgenommen
- Broadcast-Adressen (FF:FF:FF:FF:FF:FF) werden **nicht** in die SAT eingetragen
- Unterscheidung:
  - **Port-Switch**
    - Je Port ein SAT-Eintrag, nur für Endgeräte
  - **Segment-Switch** (moderner)
    - Je Port mehrere MAC-Adressen in der SAT, unterstützt auch Ports, die weitere LAN-Segmente anbinden

# Lernfähige Switches: Source Address Table (2)

- Beispielnetz im eingeschwungenen Zustand



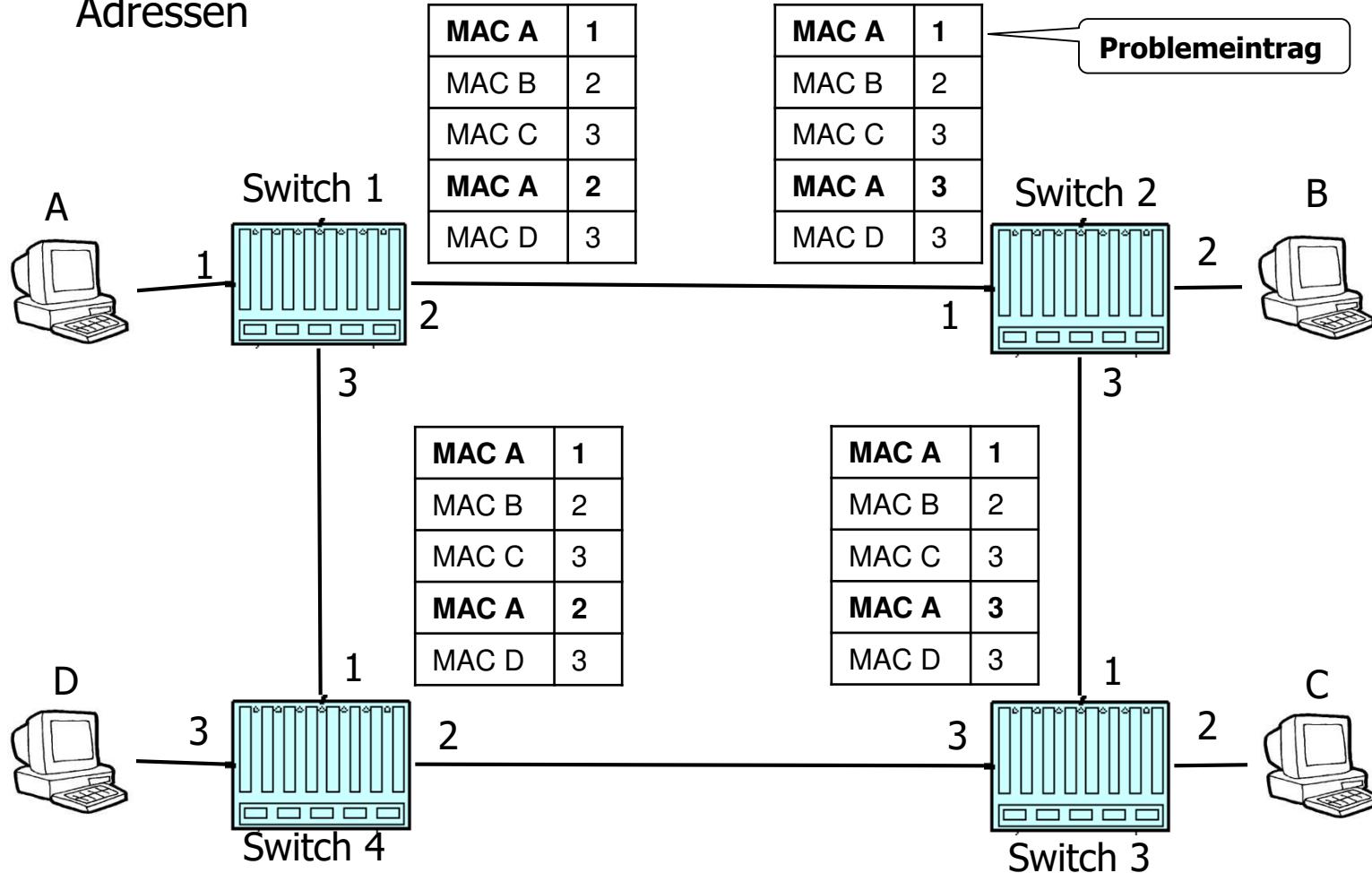
# Switches: Problemfälle

---

- Verbindung von mehreren Switches ist durch die **SAT-Aufnahmefähigkeit begrenzt** (abhängig vom Hersteller)
  - Üblich: > 500 bis zu mehreren Tausend SAT-Einträgen
  - Kleinster Switch im Netz bestimmt die Netzwerkgröße
- Redundante Verbindungen zur Ausfallsicherheit können zu **Switching-Schleifen** führen
  - Schleifen verursachen **Broadcast-Stürme**
  - Vermeiden z.B. durch **Spanning Tree Protocol (STP)**
  - Weiterentwicklung von STP mit Anleihen aus Routing-Protokollen der Schicht 3: **Shortest Path Bridging (SPB)**

# Switching-Loops

- Hier: MAC A mehrfach eingetragen, gilt auch für alle anderen MAC-Adressen



# Spanning-Tree Algorithmen für Switches

## Überblick

---

- Spanning-Tree-Protokoll (STP), IEEE 802.1D
- Für jede Kommunikationsbeziehung zwischen zwei Stationen darf es nur einen Weg geben, auch bei vielen Switches
  - Switches nutzen STP, um Schleifen in Netzen zu vermeiden
  - Ports werden blockiert, wenn sie Schleifen zulassen würden
  - Selbstlernender Algorithmus
- Varianten
  - RSTP (Rapid Spanning Tree Protocol): schnelle Berechnungszeiten bei Topologieänderungen
  - MSTP (Multiple Spanning Tree Protocol): Erweiterung, auch für VLANs geeignet → mehrere Spanning-Trees parallel

---

Weitere Informationen: Odom, W.; Sequeira, A.: Cisco CCNA Routing und Switching ICND2 200-101. Das offizielle Handbuch zur erfolgreichen Zertifizierung. Ciscopress.com, 2013

# Spanning-Tree Algorithmen:

## Funktionsweise

---

- **Root-Bridge** (Master) wird gewählt
  - Switch mit der niedrigsten BID (Bridge ID) gewinnt
  - Initial und bei jeder Änderung wird der Spanning Tree über den Master ermittelt und verteilt
  - Master bestimmt alle Pfade, es ergibt sich ein Baum (Spanning Tree)
- Zyklischer Austausch der Konfiguration, über sog. **BPDUs** (Bridge Protocol Data Unit) über Multicast, z. B. alle 2 s
- Switches **blockieren** Ports, die zu Schleifen führen könnten
- **Portzustände** werden vom Switch verwaltet:
  - *Blocking*: verwirft Frames, hört aber BPDUs ab
  - *Listening*: verwirft Frames, hört aber BPDUs ab und sendet sie weiter
  - *Learning*: verwirft Frames, hört BPDUs ab, sendet sie weiter, lernt MAC-Adressen
  - *Forwarding*: leitet zusätzlich zu *Learning* auch Frames weiter
  - *Disabled*: nichts wird gemacht

# Überblick

---

- 1 ✓ Kommunikationssysteme und verteilte Anwendungen
- 2 ✓ Grundlagen der Transportschicht
- 3 ✓ Transportprotokolle TCP und UDP
- 4 ✓ Grundlagen der Vermittlungsschicht
- 5 ✓ Internet und Internet Protocol (IP)
- 6 ✓ Routing-Verfahren und -Protokolle
- 7 ✓ Internet-Steuerprotokolle und DNS
- 8 ✓ Internet Protocol Version 6 (IPv6)
- 9 ✓ Netzwerkschnittstelle