

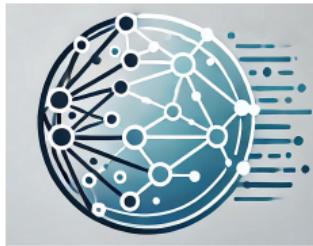
---

# Netze

Modul 8: Transportschicht und UDP

👤 Prof. Dr. Michael Rademacher

---



20. Mai 2025

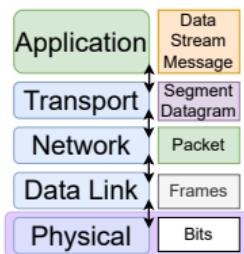
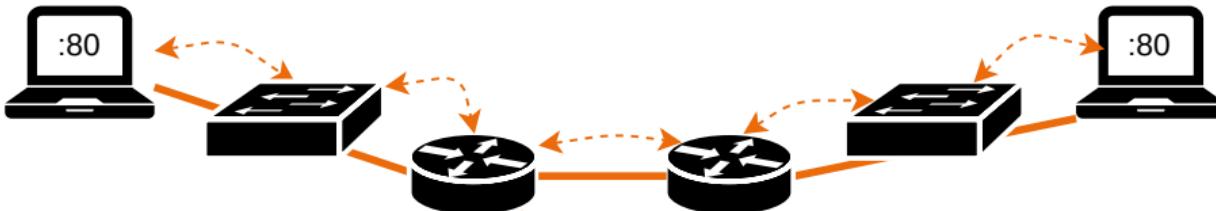
# ISO/OSI und TCP/IP Modell

ISO/OSI	TCP/IP	This Course	Application -oriented
7 Application Layer	Application Layer	Application Layer	
6 Presentation Layer			
5 Session Layer			
4 Transport Layer	Transport Layer	Transport Layer	Transport -oriented
3 Network Layer	Internet Layer	Network Layer	
2 Data Link Layer	Link Layer	Data Link Layer	
1 Physical Layer		Physical Layer	

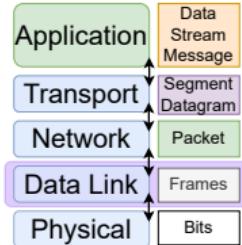
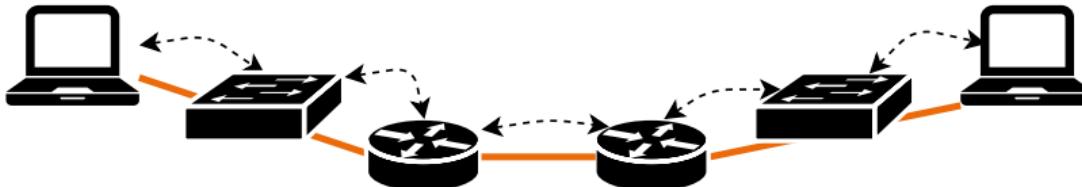
- Das OSI-Modell (7 Schichten) dient als **Referenzmodell**.
- Das Internet folgt praktisch dem **TCP/IP-Schichtenmodell** (oft 4–5 Schichten).
- Abbildungen des OSI-Modells helfen beim systematischen Denken über Funktionen/Interfaces zwischen Schichten.

# Physical Layer

- Physikalische Übertragung der Bits über ein Medium.
- Umsetzung von Bits in Signale:
  - elektrische Signale (z. B. Kupferkabel)
  - optische Signale (z. B. Glasfaser)
  - elektromagnetische Signale (z. B. Funk)
- Typische Aspekte: Leitungscodierung, Modulation, Dämpfung, Bandbreite.



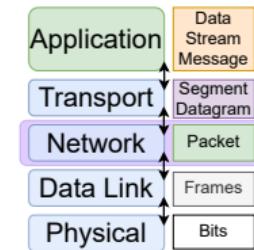
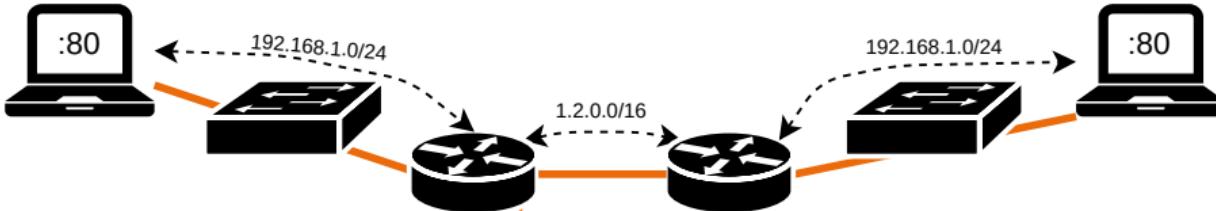
- **Lokale** Zustellung von Rahmen (Frames) zwischen Knoten im **gleichen** Netzwerksegment.
- Layer-2-Geräte (z. B. Switches, Bridges, Access Points) leiten Frames weiter.
- Weitestgehend fehlerfreie Übertragung über das physikalische Medium (z. B. FEC, Automatic Repeat reQuest (ARQ)).
- Zugriffskontrolle auf das Übertragungsmedium (z. B. CSMA/CD, CSMA/CA).
- Adressierung über **MAC-Adressen** der Netzwerkschnittstellen.



Protokolle (z. B.):

- Ethernet (IEEE 802.3)
- PPP (RFC 1661)
- WLAN (IEEE 802.11)

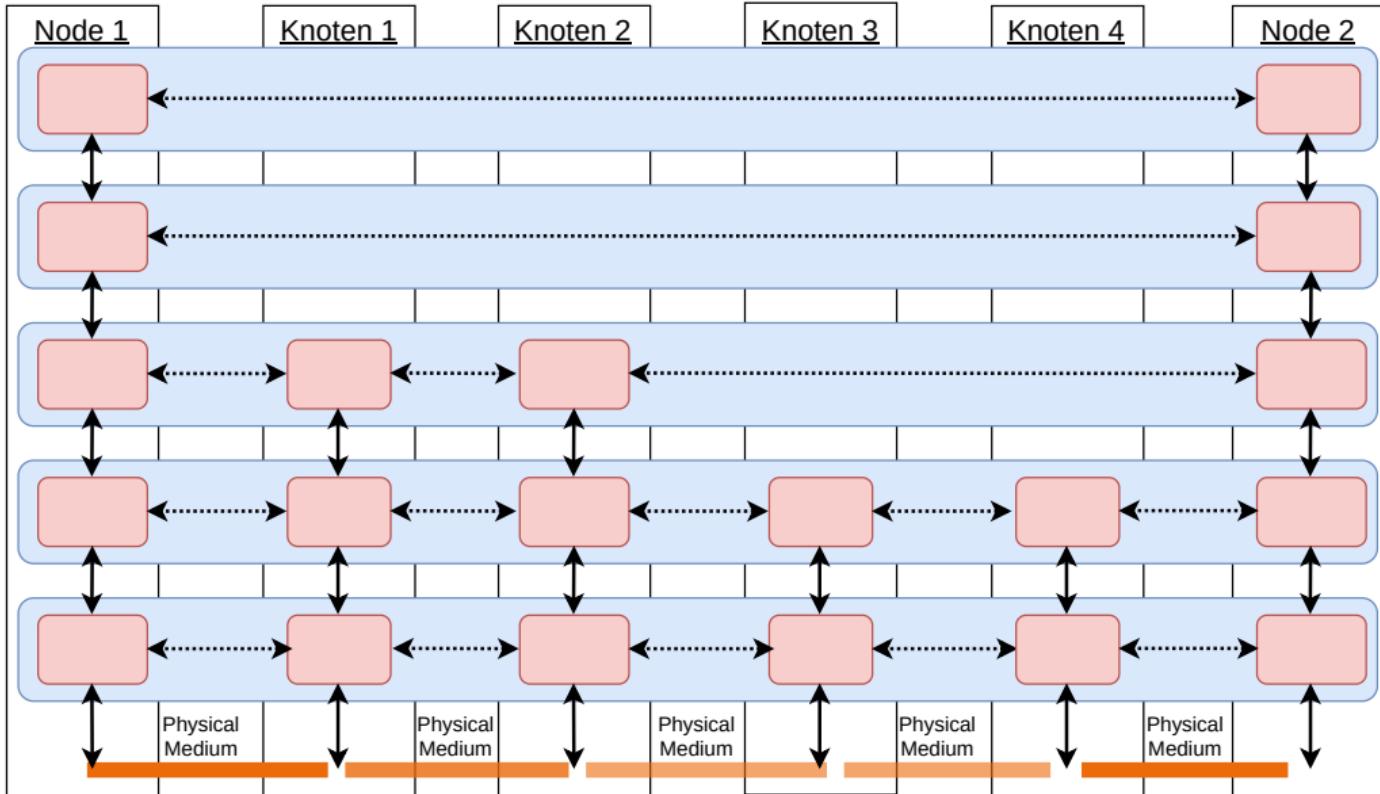
- Weiterleitung von Paketen/Datagrammen **zwischen** unterschiedlichen Netzwerken.
- Router verbinden Netze und treffen Weiterleitungsentscheidungen.
- Der Pfad wird durch **Routing** bestimmt; der Sender kennt die Ziel-IP (Wegfindung erfolgt im Netz).
- **Logische Adressierung** (IP-Adressen), Fragmentierung (bei IPv4).



Protokolle (z. B.):

- IP (IPv4/IPv6)

# Protokollstapel entlang des Weges

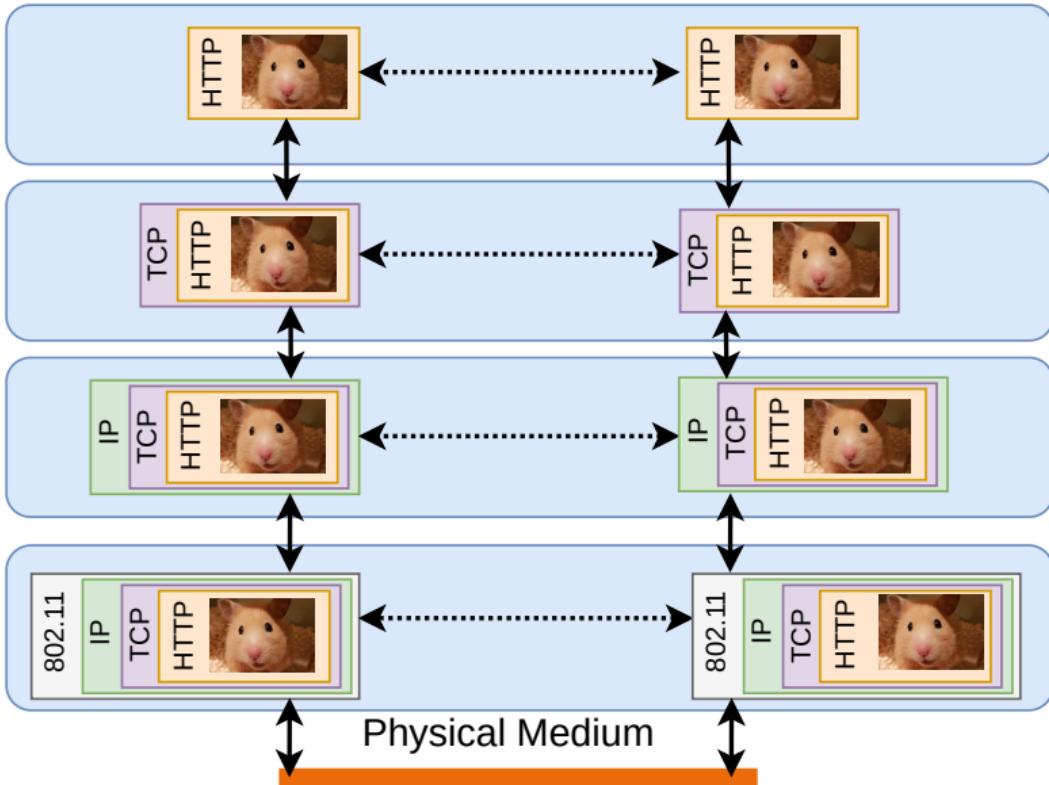


# Datenkapselung (engl. Encapsulation)

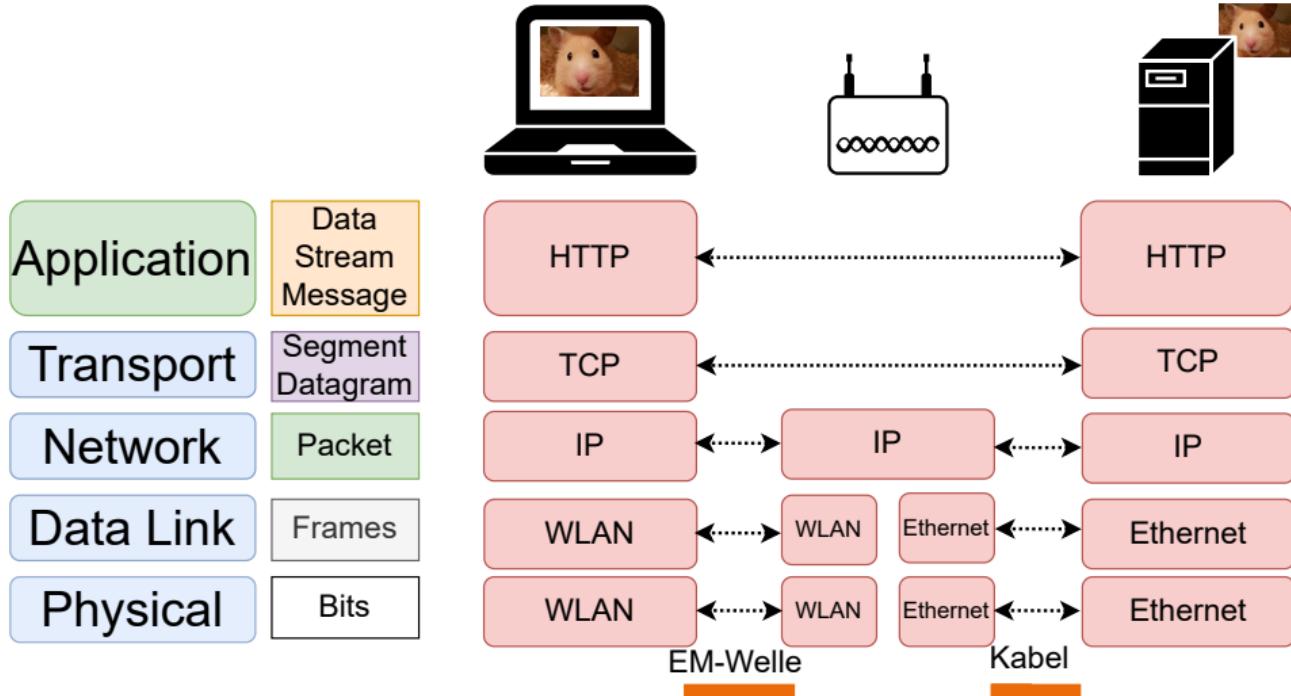
## Datenkapselung

Schichtung wird durch Datenkapselung realisiert. Untere Schichten **umhüllen** den Inhalt der höheren Schichten und fügen eigene Informationen hinzu.

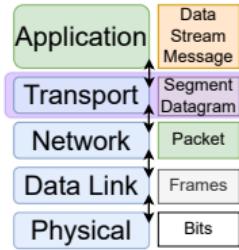
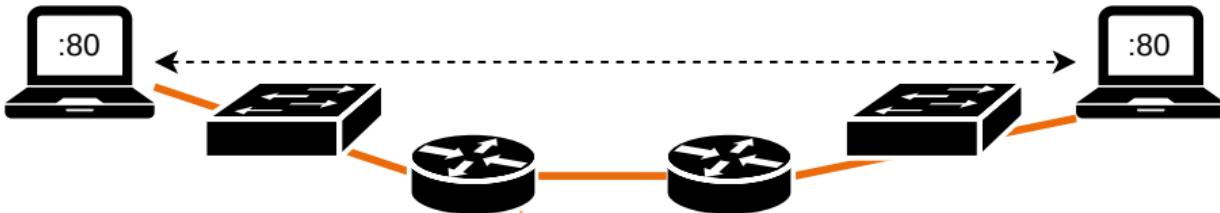
- Beispiel: Abruf einer Website über WLAN (Schichten fügen sukzessive Header hinzu).



# Beispiel: Web-Browser



- **Ende-zu-Ende-Kommunikation** zwischen Prozessen auf Endsystemen.
- Abstrakte, **einheitliche Schnittstelle** zur Anwendungsschicht (Sockets).
- Zerlegung/Wiederzusammenführung der Daten in Einheiten der Transportschicht:  
**Segmente** (TCP) bzw. **Datagramme** (UDP).



Protokolle (z. B.):

- TCP
- UDP
- QUIC (auf UDP, u. a. für HTTP/3)

# Ende-zu-Ende-Prinzip in Computernetzwerken

## Kernaussage

**Funktionalität, Zuverlässigkeit und Sicherheit** sollen nach dem **End-to-End Principle** dort implementiert werden, wo sie semantisch überprüfbar sind – in den **kommunizierenden Endsystemen**.

Saltzer, Reed, Clark: "End-to-End Arguments in System Design", MIT 1984 [8]



David D. Clark

- Viele Netzfunktionen (z. B. Fehlerkorrektur, Verschlüsselung) sind im Netzkern *ineffizient* oder *nicht überprüfbar*.
- Nur Endsysteme kennen die Anwendungslogik und können korrekte Ende-zu-Ende-Garantien bieten.

# Zwei zentrale Internet-Transportprotokolle

## TCP

RFC 793 [3] aktualisiert durch RFC 9293 [5]

- Verbindungsorientiert
- Zuverlässige Übertragung (Sequenznummern, ACKs, Retransmissions)
- Empfangsreihenfolge korrekt (Bytestrom)
- Flusskontrolle (Flow Control)
- Überlastkontrolle (Congestion Control)

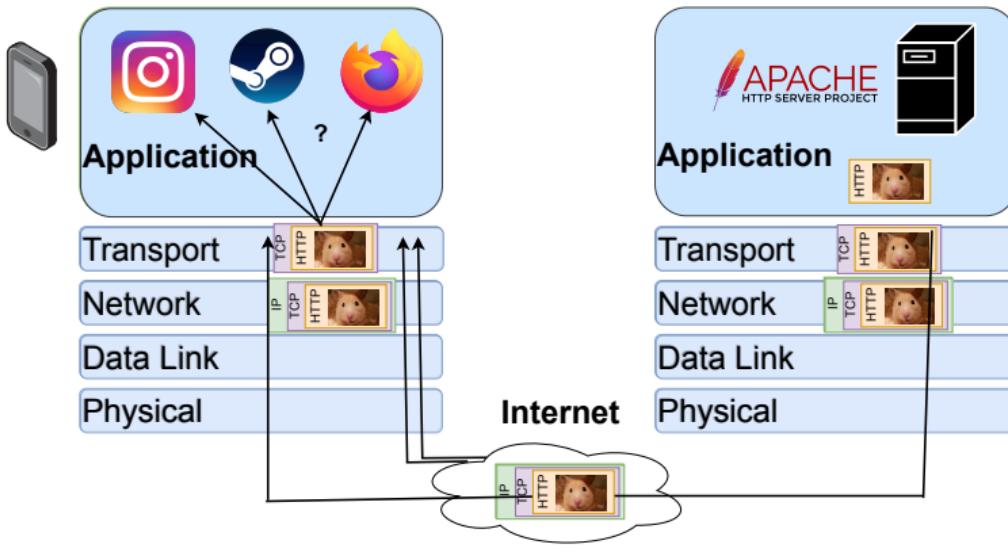
## UDP

RFC 768 [2]

- Verbindungslos
- Schlanke Ergänzung des „best effort“ von IP
- Keine Zustandsverwaltung im Netz, geringe Latenz
- Kein Garantie für Reihenfolge oder Zuverlässigkeit

**Beide Protokolle** gehören zur Transportschicht des Internet-Protokollstapels und bauen direkt auf der Netzwerkschicht (IP) auf.

# Adressierung auf der Transportschicht

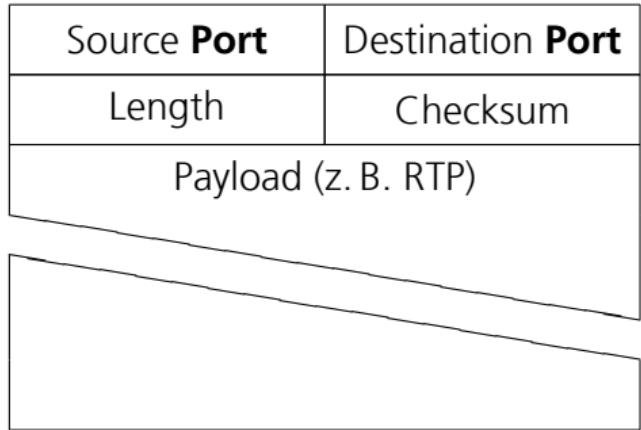
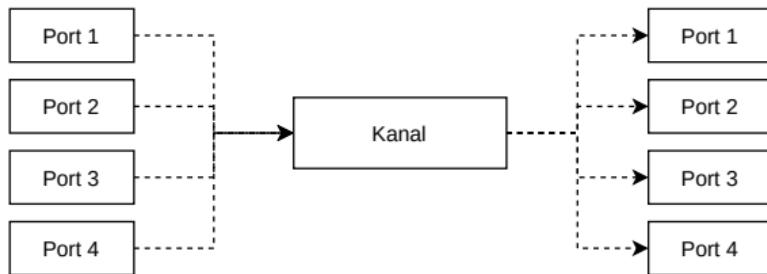


## Kernaussage

- Wie findet die Transportschicht die korrekte Anwendung, die die Nachricht empfangen soll?
- Wie werden die Adressen auf der Transportschicht vergeben? [4] [7]

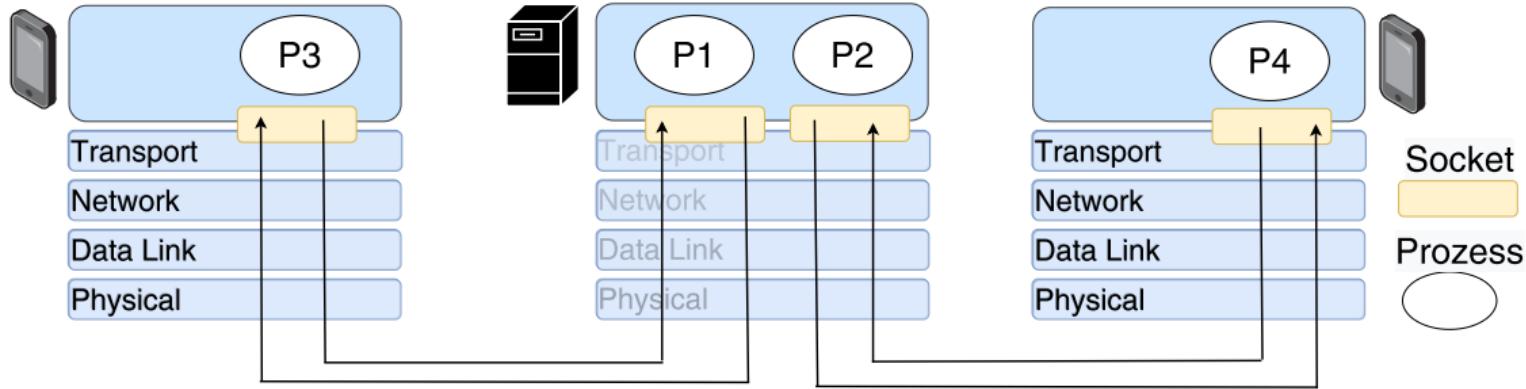
# Adressierung über Ports

- Neben MAC- und IP-Adressen führt die Transportschicht **Ports** als Adressierung für Prozesse ein.
- Ports sind ein zentraler Bestandteil des **Multiplexing/Demultiplexing** am Sender/Empfänger.



(Abstrakte Darstellung eines Schicht-4-Protokolls)

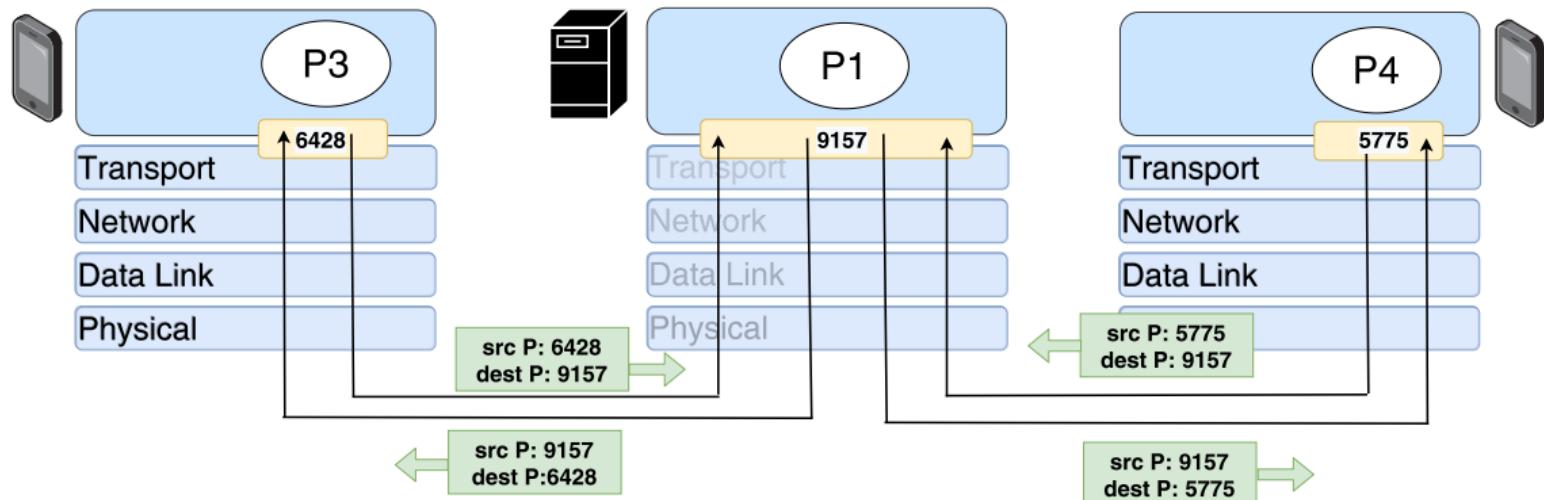
# Multiplexing/Demultiplexing



- Prozesse verwenden „Sockets“ zum Senden und Empfangen von Daten.
- Multiplexing (Sender): Hinzufügen eines Transportschicht-Headers.
- Demultiplexing (Empfänger): Auswertung der Header-Informationen, um Segmente/Datagramme über den Socket dem korrekten Prozess zuzuordnen.

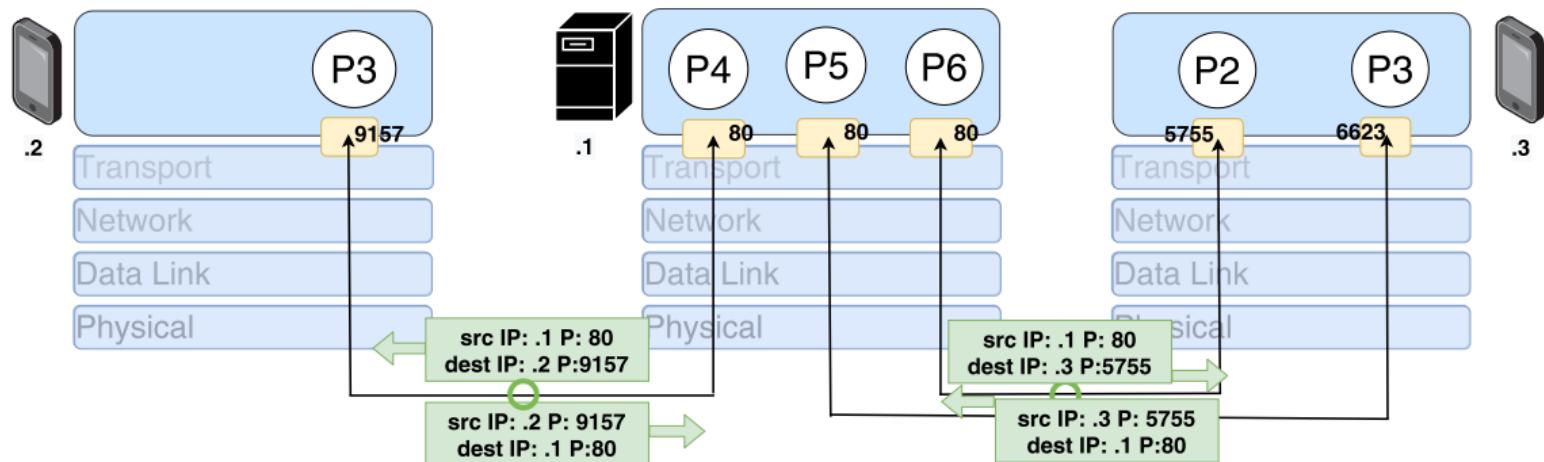
# Verbindungsloses Demultiplexing

- Beispiel: UDP [2]
- UDP ist **verbindungslos**: Es gibt keine Zustandsinformation über Verbindungen.
- Der Empfänger (Transportschicht) nutzt die Kombination aus **Ziel-IP-Adresse** und **Zielportnummer**, um das Datagramm dem passenden **Socket** zuzuordnen.
- Alle IP/UDP-Datagramme mit derselben Zieladresse und Zielportnummer landen im selben Socket (und damit Prozess).



# Verbindungsorientiertes Demultiplexing

- Beispiel: TCP [3]
- Der Empfängerprozess wird über das **5-Tuple** identifiziert: (Quell-IP, Quellport, Transportprotokoll, Ziel-IP, Zielport).
- Praktisch legt jedoch der Socket-Typ bereits das Transportprotokoll (z. B. TCP) fest, sodass die eindeutige Identifikation meist durch das **4-Tuple** (Quell-IP, Quellport, Ziel-IP, Zielport) erfolgt.
- Für jede angenommene Verbindung (`accept()`) wird ein eigener Socket erzeugt.



# Socket „API“

- Betriebssysteme stellen Anwendungen eine einheitliche Transportschnittstelle bereit:  
engl. Socket (BSD Sockets API, seit 1983).
- Zentrale Funktionen (Auswahl):

`socket()` Erzeugt ein neues Socket eines bestimmten Typs und alloziert Systemressourcen.

`bind()` Bindet das Socket an eine Adressinformation (IP-Adresse und Port).

`read()` Liest Daten **aus** einem Socket.

`write()` Schreibt Daten **in** ein Socket.

- Zusätzlich für **verbindungsorientierte** Kommunikation (TCP):

`listen()` Versetzt ein Socket in den Zustand für Verbindungsanfragen.

`connect()` Initiiert eine Verbindungsanfrage.

`accept()` Akzeptiert eine eingehende Verbindung und liefert ein neues, verbundenes Socket.

# Das „Hallo Welt“ der Webanwendung — Verbindungslos

## Server:

```
#!usr/bin/python
import socket

# UDP Socket
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)

host = "10.20.111.207"
port = 1337

sock.bind((host,port))

while True:
    print("Waiting for client...")
    msg,addr = sock.recvfrom(1024)
    msg = msg.decode('utf-8')
    print("Received Messages:",msg,"from",addr)
```

## Client:

```
#!usr/bin/python
import socket

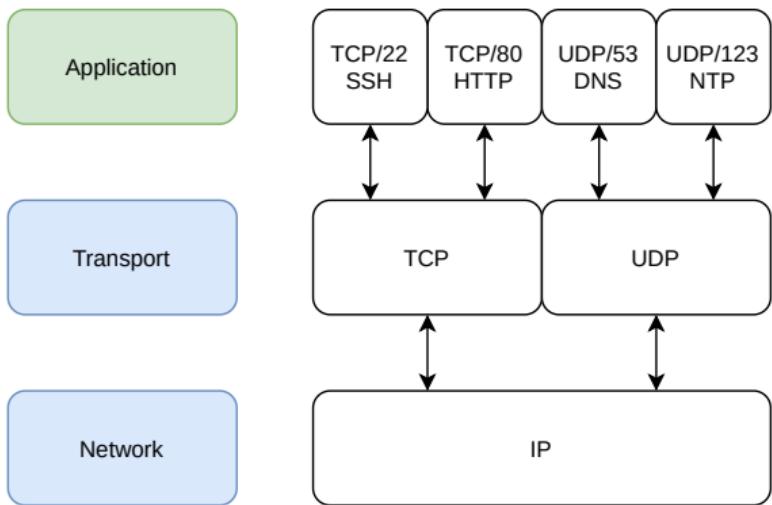
# UDP Socket
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)

host = "10.20.111.207"
port = 1337
msg = "Hallo\u00d7Netze\u00d7Vorlesung!!!!!!"
msg = str.encode(msg, 'utf-8')

print("UDP\u00d7target\u00d7IP:", host)
print("UDP\u00d7target\u00d7Port:", port)

sock.sendto(msg,(host,port))
```

# Beispiele für Ports und Transportprotokolle



- Portnummern **0–1023** sind *well-known ports* (IANA [7]), reserviert für verbreitete Dienste.

## Praxis

Öffnen Sie ein Terminal:  
`ss -tn` (socket statistics)

# Sockets untersuchen mit ss

## Beispielaufruf

```
ss -rempl
```

- -r – Auflösen numerischer Adressen/Ports versuchen
- -e – Erweiterte Socket-Infos anzeigen (detailliert)
- -m – Speicherbelegung des Sockets zeigen
- -p – Prozess anzeigen, der das Socket nutzt
- -t – Nur TCP-Sockets anzeigen
- -l – Nur lauschende (listening) Sockets anzeigen

```
hannes@hannes-pc:~$ ss -rempl
State      Recv-Q      Send-Q          Local Address:Port          Peer Address:Port
Process
LISTEN      0            1000          10.255.255.254:domain        0.0.0.0:*
    ino:10250 sk:2001 cgroup:/ <-->
        skmem:(r0,rb131072,t0,tb16384,f0,w0,o0,b10,d0)
LISTEN      0            4096          127.0.0.53%lo:domain        0.0.0.0:*
    uid:101 ino:17464 sk:2002 cgroup:/system.slice/systemd-resolved.service <-->
        skmem:(r0,rb131072,t0,tb16384,f0,w0,o0,b10,d0)
LISTEN      0            10            localhost:4433           0.0.0.0:*
    users:(("ssl_server2",pid=57892,fd=3)) uid:1000 ino:1011813 sk:4001 cgroup:/ <-->
        skmem:(r0,rb131072,t0,tb16384,f0,w0,o0,b10,d0)
```

# Wireshark Beispiel

Datei Bearbeiten Ansicht Navigation Aufzeichnen Analyse Statistiken Telefonie Wireless Tools Hilfe

tcp.port == 4711

No.	Time	Source	Destination	Protocol	Length	Info
53	4.270401608	192.168.0.53	192.168.0.107	TCP	74	44670 → 4711 [SYN] Seq=0 Win=6424
54	4.270939319	192.168.0.107	192.168.0.53	TCP	74	4711 → 44670 [SYN, ACK] Seq=0 Ack=1
55	4.270965688	192.168.0.53	192.168.0.107	TCP	66	44670 → 4711 [ACK] Seq=1 Ack=1 Wi
75	7.217246412	192.168.0.53	192.168.0.107	TCP	78	44670 → 4711 [PSH, ACK] Seq=1 Ack=1
76	7.217783902	192.168.0.107	192.168.0.53	TCP	66	4711 → 44670 [ACK] Seq=1 Ack=13 W
85	8.185194621	192.168.0.53	192.168.0.107	TCP	66	44670 → 4711 [FIN, ACK] Seq=13 Ack=14
86	8.186425154	192.168.0.107	192.168.0.53	TCP	66	4711 → 44670 [FIN, ACK] Seq=14 Ack=15
87	8.186455892	192.168.0.53	192.168.0.107	TCP	66	44670 → 4711 [ACK] Seq=14 Ack=2 W

Frame 53: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0  
Ethernet II, Src: AsrockIn\_a7:a9:ce (70:85:c2:a7:a9:ce), Dst: 66:a3:8e:e6:1a:df (66:a3:8e:e6:1a:df)  
Internet Protocol Version 4, Src: 192.168.0.53, Dst: 192.168.0.107  
Transmission Control Protocol, Src Port: 44670, Dst Port: 4711, Seq: 0, Len: 0

# UDP – Eigenschaften

- „Schlankes“ Transportprotokoll ohne Verbindungszustand.
- Datagramme können verloren gehen, dupliziert werden oder in anderer Reihenfolge ankommen.
- **Verbindungslos** → kein „Handshake“

## Warum UDP?

- Kein Verbindungsaufbau (geringe Latenz)
- **Einfach:** Kein Zustandsmanagement in Endsystemen erforderlich
- Kleiner Header (dadurch geringer Overhead)



David Reed

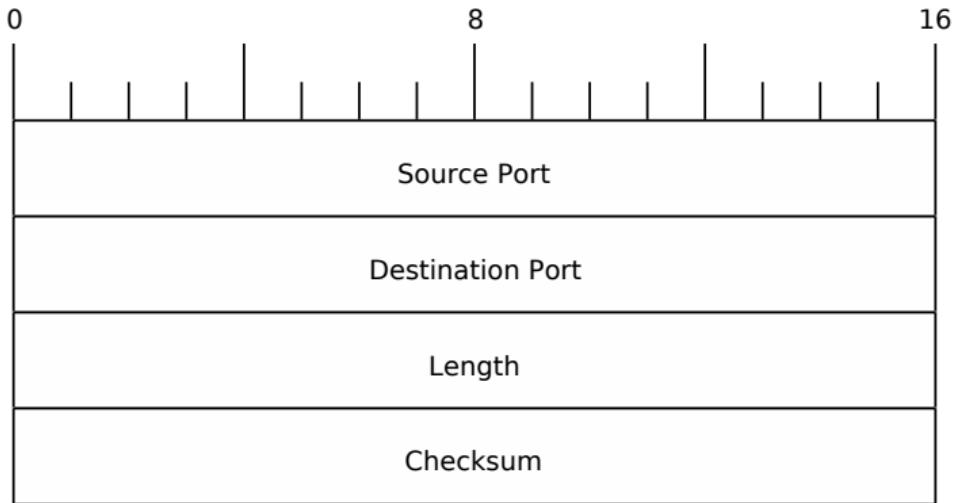
„[...] it's a little embarrassing to be the inventor of something so simple [...]“ [1]

# UDP – Beispielhafte Anwendungen

- **Media-Streaming:** Sprach- und Videokonferenzen. *Einzelne Verluste sind tolerierbar.*
- **Gaming:** Reaktionskritische Spiele. *Latenz ist wichtiger als Vollständigkeit.*
- **Virtual Private Networks (VPNs):**
  - Anforderungen ergeben sich aus dem getunnelten Protokoll.
  - Das Übertragen von TCP-Verkehr über ein TCP-basiertes VPN ist zwar möglich, aber ineffizient.
- **IoT-Kommunikation:**
  - Das *Constrained Application Protocol (CoAP)* basiert auf UDP und ermöglicht ressourcenschonende Kommunikation für eingebettete Geräte.
- **Domain Name System (DNS):**
  - Verwendet standardmäßig UDP-Port 53 für schnelle Anfragen; bei großen Antworten oder Zonenübertragungen kann auf TCP gewechselt werden.
- **QUIC (Neu):**
  - Modernes Transportprotokoll, läuft über UDP (Port 443).
  - Bietet Zuverlässigkeit, Multiplexing und Verschlüsselung auf Anwendungsebene (z. B. für HTTP/3).

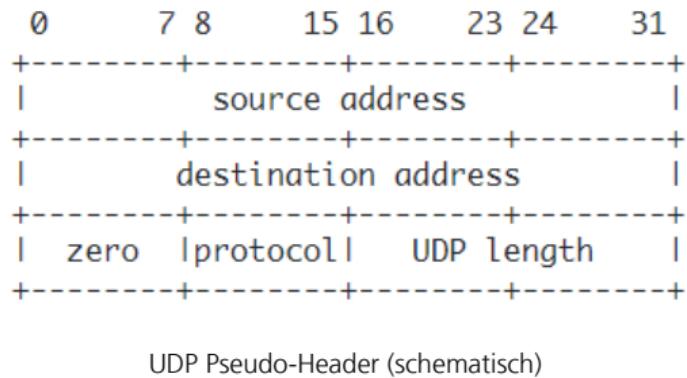


## UDP — RFC 768: Headerstruktur



**Minimaler Header: 8 Byte (4 × 16 Bit)**

# UDP — RFC 768: Pseudo-Header & Checksumme



## Internet Checksum (Einserkomplement):

- Fehlererkennung per **Einserkomplement-Summe** über 16-Bit-Wörter mit **End-Around-Carry**.
- Eingabe = **Pseudo-Header (IP)** + **UDP-Header + Daten** (+ ggf. Padding).
- **Pseudo-Header** wird nur zur Berechnung genutzt — **nicht übertragen**.

## Pseudo-Header-Felder:

- **IPv4:** Src IP, Dst IP, 0, Protokoll (=17), UDP-Länge
- **IPv6:** Src IP, Dst IP, UDP-Länge, Next Header (=17), 3x reservierte 0-Bytes

## Pflicht zur Berechnung:

- **IPv4:** UDP-Checksumme ist **optional** (0 = „nicht berechnet“).
- **IPv6:** UDP-Checksumme ist **verpflichtend**.

# UDP in der Praxis — Empfehlungen für Entwickler

## UDP ist einfach, aber nicht trivial:

- Kein Verbindungsaufbau, keine Zustandsverwaltung, keine Zuverlässigkeit.
- Anwendungen müssen viele Aufgaben selbst übernehmen:
  - Paketverlust, Reihenfolge, Duplikaterkennung
  - Fluss- und Überlastkontrolle (Congestion Control)
  - Fragmentierung und MTU-Handling
  - Port-Management und Firewalls/NAT
  - Sicherheit (Authentifizierung, Verschlüsselung)
- Fehlende Mechanismen → mehr Flexibilität, aber auch mehr Verantwortung.

**Referenz:** RFC 8085 — UDP Usage Guidelines [6]

## Kernaussage:

*„UDP provides minimal services. Application designers are responsible for safe, congestion-aware, and interoperable use of the Internet.“*

— RFC 8085, Section 3



Lars Eggert — Autor von RFC 8085,  
ehemaliger Transport Area Director  
und Chair der IETF.

# Quellen I

- [1] udp and me – deep plum research.  
[Online; accessed 2025-11-16].
- [2] User Datagram Protocol.  
RFC 768, August 1980.
- [3] Transmission Control Protocol.  
RFC 793, September 1981.
- [4] Community, S.  
How are source ports determined and how can i force it to use a specific port?, 2016.  
SuperUser Q&A, abgerufen am 1. November 2025.
- [5] Eddy, W.  
Transmission Control Protocol (TCP).  
RFC 9293, August 2022.
- [6] Eggert, L., Fairhurst, G., and Shepherd, G.  
UDP Usage Guidelines.  
RFC 8085, March 2017.
- [7] Internet Assigned Numbers Authority (IANA).  
Service name and transport protocol port number registry, 2025.  
Abgerufen am 1. November 2025.
- [8] Saltzer, J. H., Reed, D. P., and Clark, D. D.  
End-to-end arguments in system design.  
*ACM Transactions on Computer Systems (TOCS)* 2, 4 (1984), 277–288.