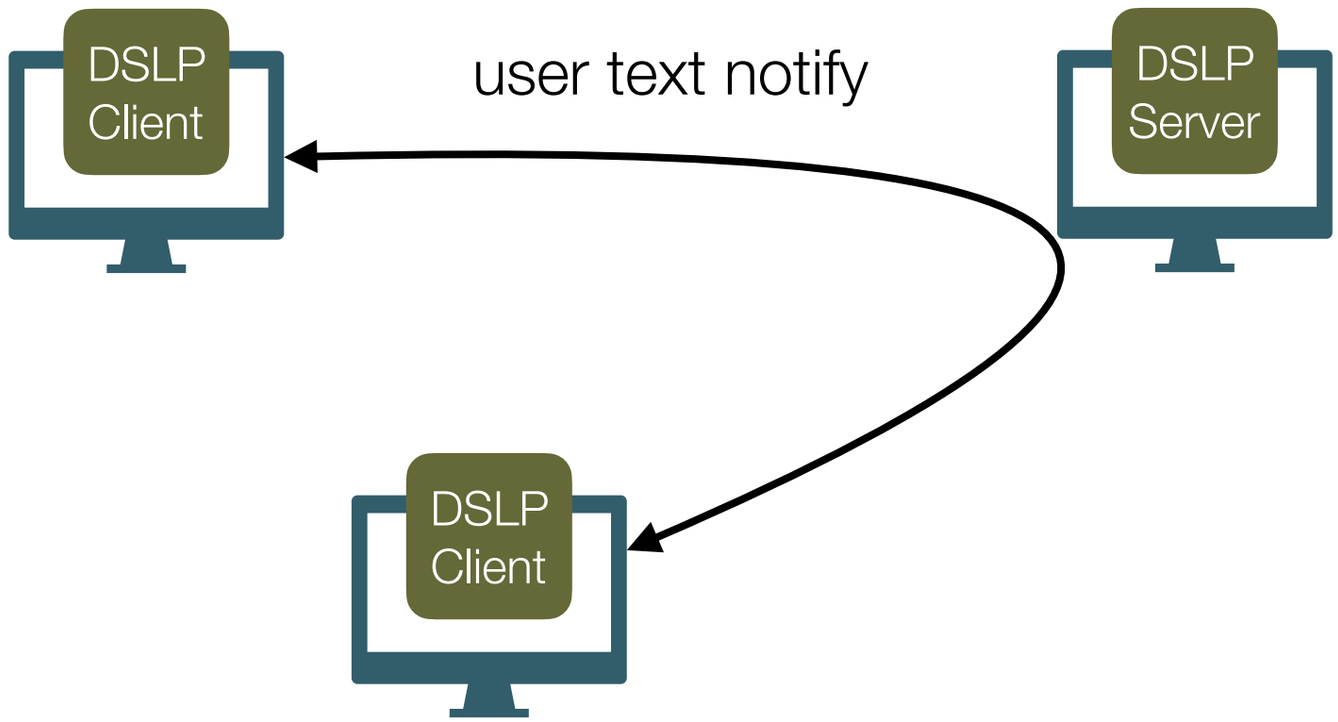
Modul Verteilte Systeme

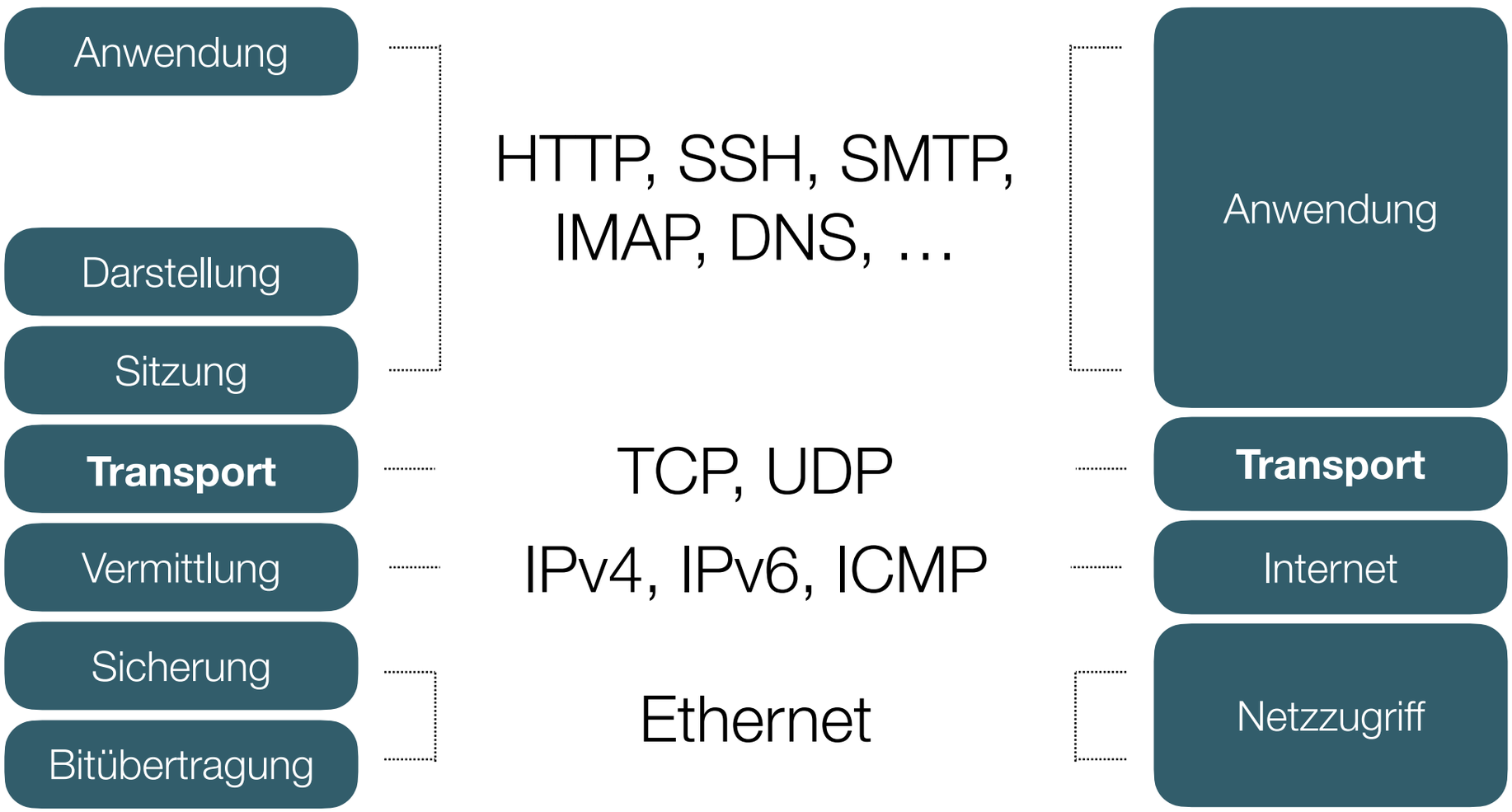
Transport

Peter Tröger
Beuth Hochschule für Technik Berlin
Sommersemester 2020
(Version 1)

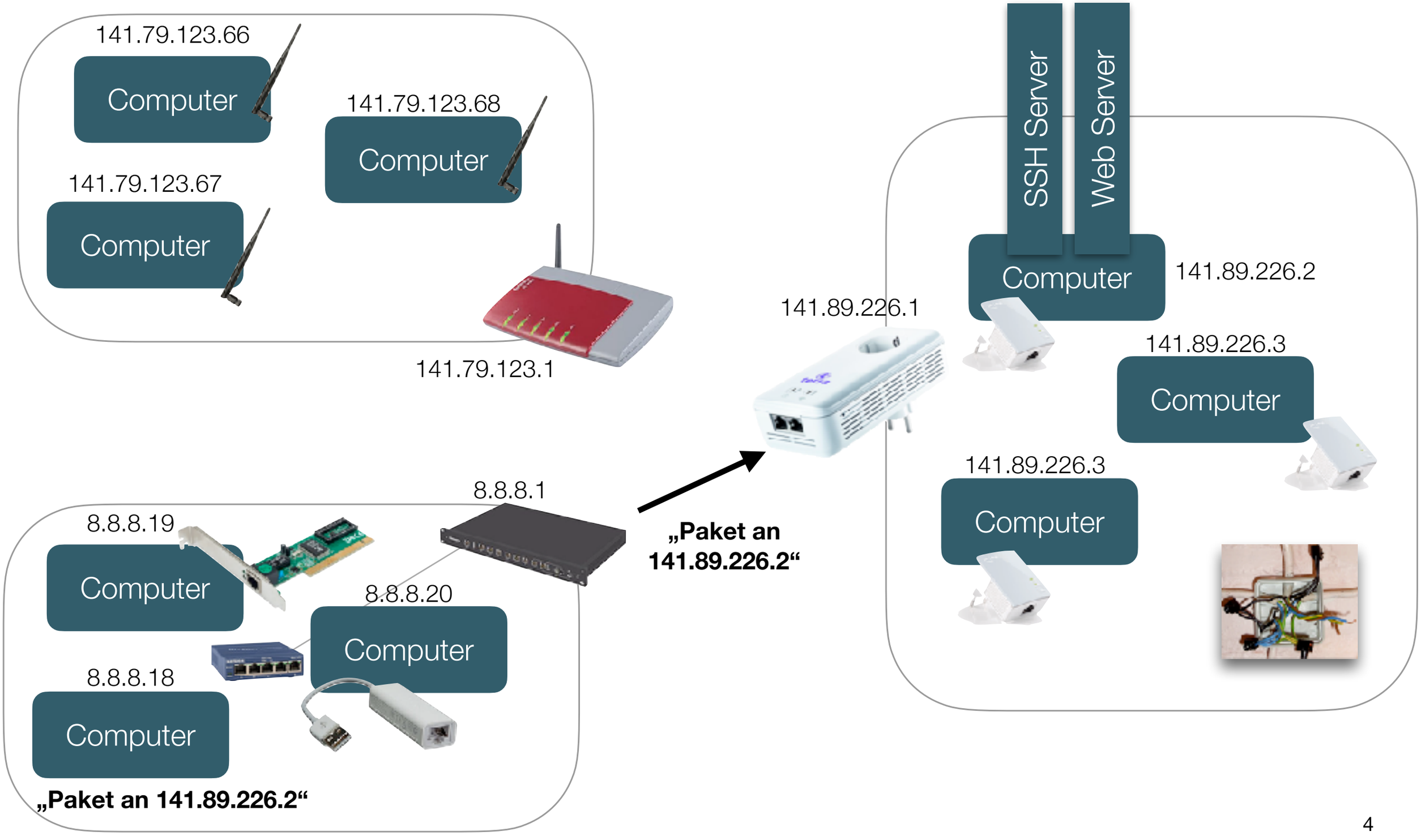


Neue Übung „DSLP Persönlicher Chat“

Netzwerke heute



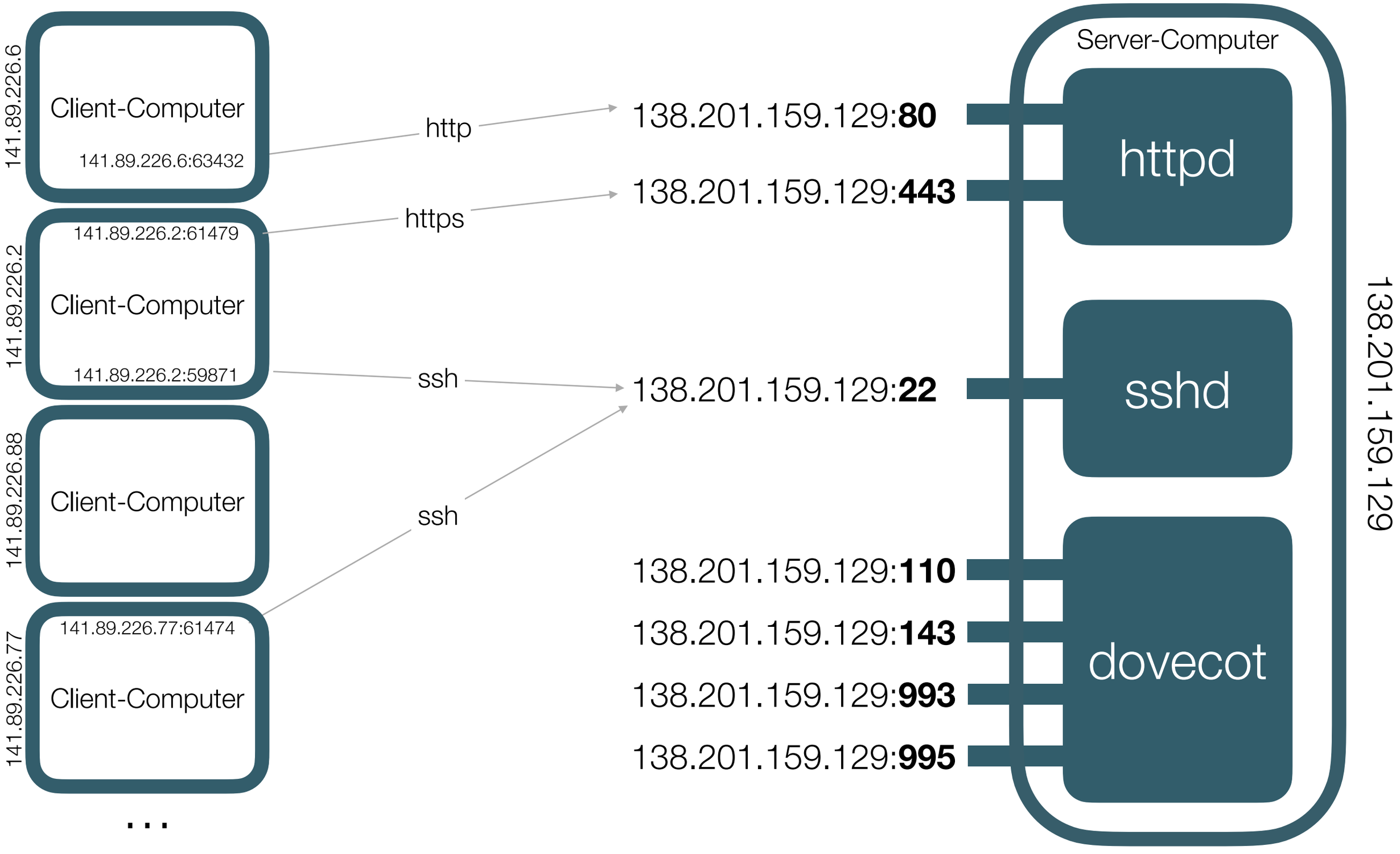
Vermittlungsschicht = Routing



Ports

- Verschiedene Prozesse auf Server-Maschine mit gleicher IP-Adresse
- Wie kann ein Client einen Server-Prozess direkt ansprechen?
- **Port:** Endpunkt einer Kommunikation aus Sichtweise des Betriebssystems
- **Port-Nummer** auf dem Server legt potentiell das **Protokoll** schon fest
  - Liste von festgelegten (*well-known*) Port-Nummern für bestimmte Anwendungen
  - https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers
  - https://www.iana.org/assignments/service-names-port-numbers/
- Client nutzt dynamisch Ports aus dem Bereich 49152 - 65535 (*ephemeral ports*)

Ports


Client-Computer
141.89.226.6
141.89.226.6:63432
Client-Computer
141.89.226.2
141.89.226.2:61479
141.89.226.2:59871
Client-Computer
141.89.226.88
Client-Computer
141.89.226.77
141.89.226.77:61474
...
http
https
ssh
ssh
138.201.159.129:80
138.201.159.129:443
138.201.159.129:22
138.201.159.129:110
138.201.159.129:143
138.201.159.129:993
138.201.159.129:995
Server-Computer
httpd
sshd
dovecot
138.201.159.129

User Datagram Protocol (UDP)

- Erste Standardisierung in RFC 768
- Nur Datenintegrität und Port-Konzept —> Fokus auf Performanz
- Fehlerbehebung wird auf die Anwendungsebene verschoben
- Verbindungslos, daher auch Rundruf (*broadcast*) unterstützt
- Beispiele: SNMP, Audio Streaming

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	Quell-Port																Ziel-Port															
	Länge der Daten																Prüfsumme															
	Daten (max. 65507 Bytes)																															

UDP im IPv4 - Paket

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	Version				IHL				DSCP						ECN		Länge des Pakets															
	Identifikation bei Fragmentierung																Flags			Offset Fragmentierung											
	Time To Live								Protokoll								Prüfsumme für IPv4 Header															
	IP Adresse Sender																															
	IP Adresse Empfänger																															
	IPv4 Optionen																															

Quell-Port	Ziel-Port
Länge der Daten	Prüfsumme
Daten (max. 65507 Bytes)	

Matroschka - Modell

Anwendung

Socket-API *Bind*: 141.64.89.32:45342
Socket-API *Connect*: 8.8.8.8:53

DNS-Anfrage

Transport

Absenderport: 45342
Zielport: 53

UDP-Paket	DNS-Anfrage	

Internet

Absenderadresse: 141.64.89.32
Zieladresse: 8.8.8.8

IP-Paket	UDP-Paket	DNS-Anfrage		

Netzzugriff

Zieladresse: a8:20:66:3a:e4:8e (Router)

Ethernet-Frame	IP-Paket	UDP-Paket	DNS-Anfrage			

Transportschicht - UDP und TCP

- UDP/IP - *User Datagram Protocol*
  - UDP Datagram = IP Paket + Port-Nummer, noch immer **verbindungslos**
  - Keine Qualitätszusicherungen bei der Übertragung
- TCP/IP - *Transmission Control Protocol*
  - Komplexeres Protokoll für bidirektionales **Verbindungskonzept**
  - Qualitätszusicherungen bei der Übertragung
- Beide Protokolle: Defekte Pakete über Prüfsumme erkennen
- Beide Protokolle: Port-Nummer bei Sender und Empfänger

Transmission Control Protocol (TCP)

- Erste Standardisierung in RFC 793 (September 1981)
- Aufbau einer **bidirektionalen** Verbindung zwischen **Endpunkten (*endpoints*)**
  - Endpunkt auf beiden Seiten durch IP-Adresse + Portnummer
  - Programmiererin bekommt Illusion einer eigenen Übertragungs“leitung“ (*pipe*), analog zu I/O mit Dateien
  - Lesen und Schreiben von Daten innerhalb der Verbindung
  - Entsprechende Funktionen in der Socket-API (*connect(), read(), close(), …*)
- Grundlage für die populärsten Anwendungsprotokolle im Internet

Zuverlässigkeit

- TCP implementiert neue Eigenschaften, die in reinem IP nicht vorhanden sind
  - Konzept einer Verbindung (Aufbau, Abbau, Paketreihenfolge, Timeout)
  - Bidirektionale Kommunikation (Senden + Empfangen)
  - Automatischer Umgang mit Paketverlust und -verdoppelung
  - „Empfangen wie gesendet“
- TCP/IP-Verbindung kann natürlich trotzdem gestört sein
- Typisch: Transparenter erneuter Verbindungsaufbau durch die Anwendung

TCP Segmente

- Anwendung stellt **Datenstrom** (***data stream***) für die Übertragung bereit
- Betriebssystem implementiert TCP/IP Protokollstapel und bietet API
- Unterteilung in **Datenblöcke** (***chunks***)
- Datenblock + TCP Header = **TCP Segment**
- Übertragung in einem IP-Datagram, Größe typischerweise max. 1500 Bytes, je nach *Maximum Segment Size (MSS)*
- TCP Header speichert für jedes Segment eine Sequenznummer, eine Prüfsumme und die Port-Nummern

TCP Paket

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Quell-Port																	Ziel-Port															
Sequenznummer																																
Bestätigungsnummer (für ACK)																																
Offset Daten			reserviert					CWR	ECE	URG	**ACK**	PSH	**RST**	**SYN**	**FIN**		Größe Empfangsfenster															
Prüfsumme																	Urgent Pointer															
Optionen …																																

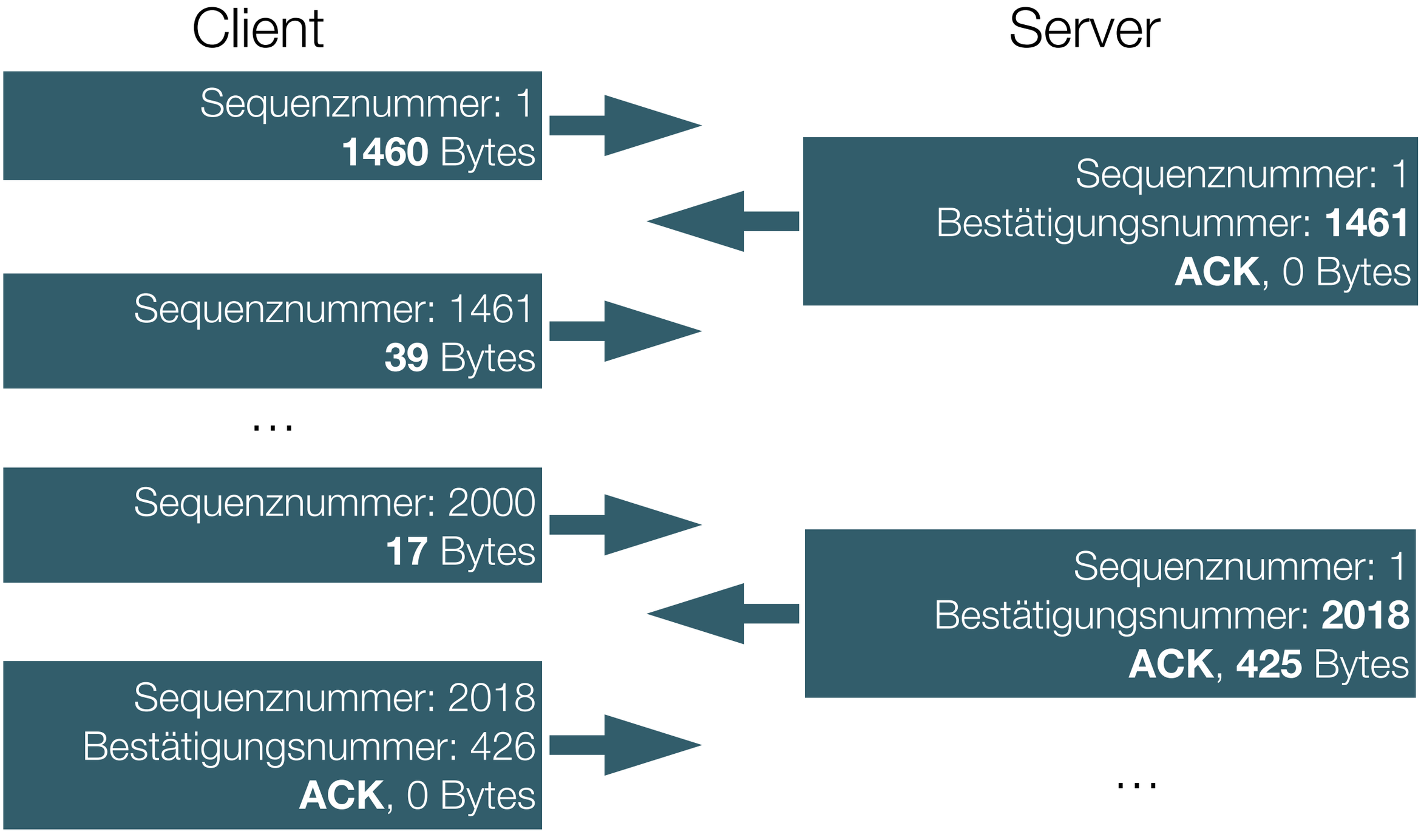
TCP Sequenznummer

- **Sequenznummer** dient zur Sortierung der Segmente beim Empfänger
- **Bestätigungsnummer** (bei gesetztem ACK-Flag) gibt an, wie viele Daten der Empfänger bereits erhalten hat
- Sender hat Timer für jedes Paket
- Zeitlich begrenztes Warten auf ACK der Gegenstelle
- Zu niedrig: Viele duplizierte Pakete
- Zu hoch: Langsame Neuübertragung bei Paketverlust
- Moderne TCP/IP - Implementierungen justieren den Timeout dynamisch

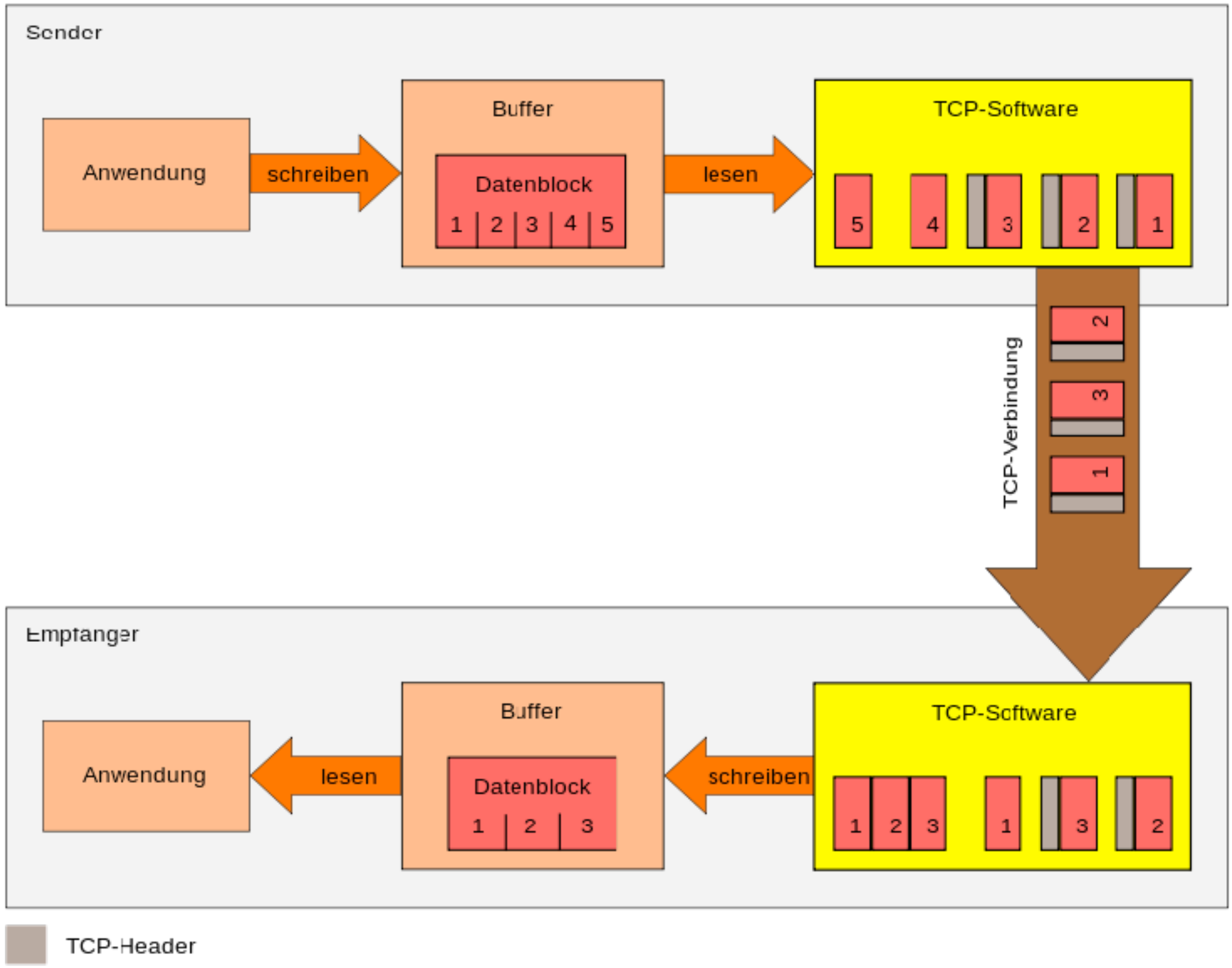
TCP Sequenznummer

- Initiale Sequenznummer ist auf beiden Seiten zufällig
  - Sicherheitsmechanismus, um Kapern der Verbindung zu verhindern
  - *Wireshark* zeigt zur besseren Lesbarkeit relative Sequenznummer an, beginnend bei 0
- Nach dem TCP-Verbindungsaufbau hat die relative Sequenznummer auf beiden Seiten den Wert 1
- Ab hier können Client und Server jeweils Daten senden, welche die Gegenseite jeweils bestätigen muss
- Bestätigungsnummer ist die nächste erwartete Sequenznummer

Beispiel (etablierte Verbindung)

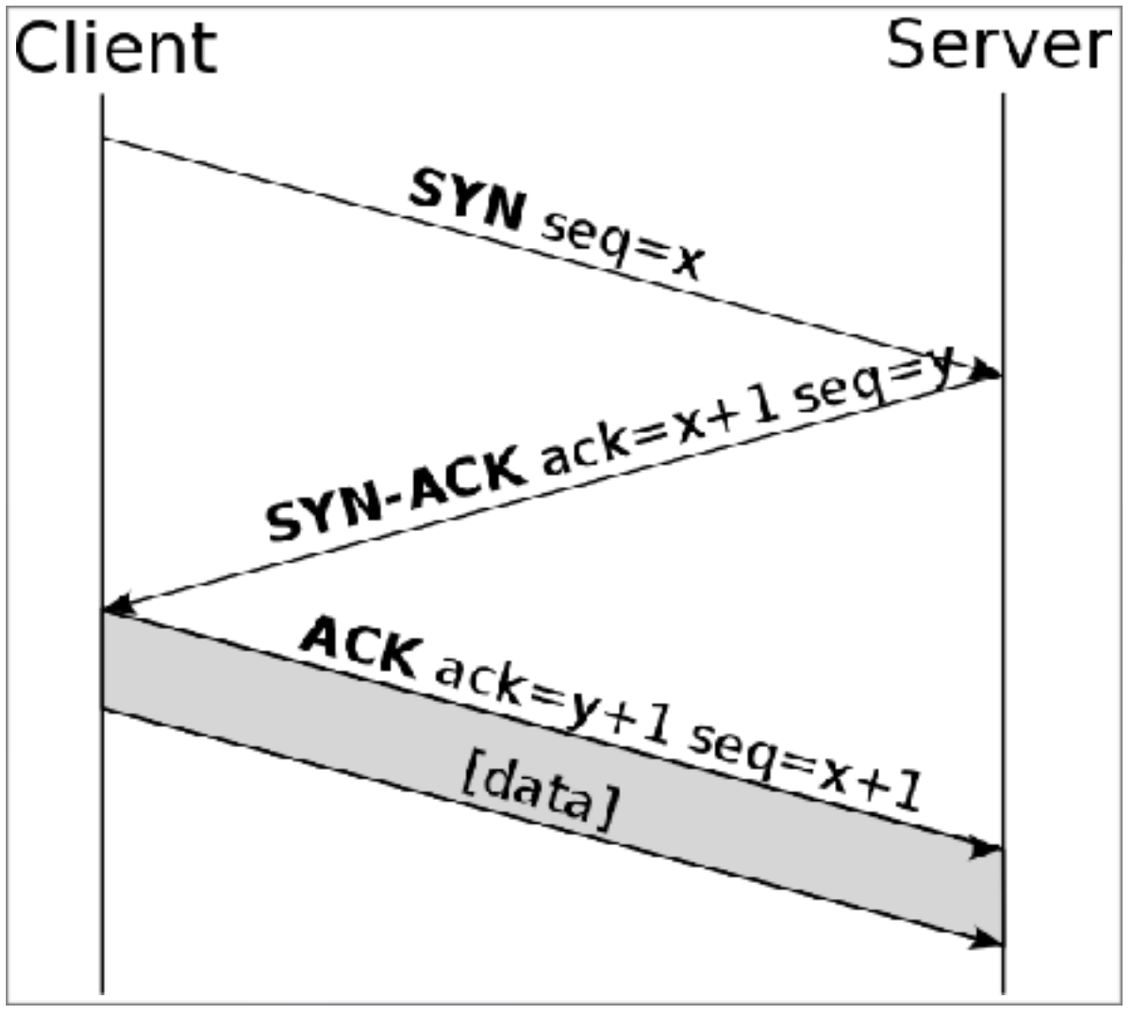


Sortierung



[de.wikipedia.org]

Verbindungsaufbau



[de.wikipedia.org]

- X und Y sind beim Start zufällig, relativer Wert ist 0
- Gesetztes SYN-Bit signalisiert Wunsch nach Verbindungsaufbau
- Server bestätigt den Verbindungsaufbau mit SYN-ACK, Client bestätigt dies wieder
- Pakete mit SYN zählen als Pseudo-Datenbyte
- ***Three-Way-Handshake***

Verbindungsaufbau

- Schritt 1: *Passive Open*
  - Server bindet sich an eine lokale Port-Nummer + Adresse
  - Schaltet in **Listen**-Modus, um eingehende Anfragen entgegenzunehmen
- Schritt 2: *Active Open*
  - Client sendet **SYN**-Paket zum Server, Sequenznummer A zufällig gewählt
  - Server antwortet mit **SYN-ACK**, Sequenznummer B des Servers wird zufällig gewählt, Bestätigungsnummer A+1 (SYN zählt als ein Pseudo-Datenbyte)
  - Client antwortet mit **ACK**, Sequenznummer A+1, Bestätigungsnummer B+1 (SYN zählt als ein Pseudo-Datenbyte)

Verbindungsabbau

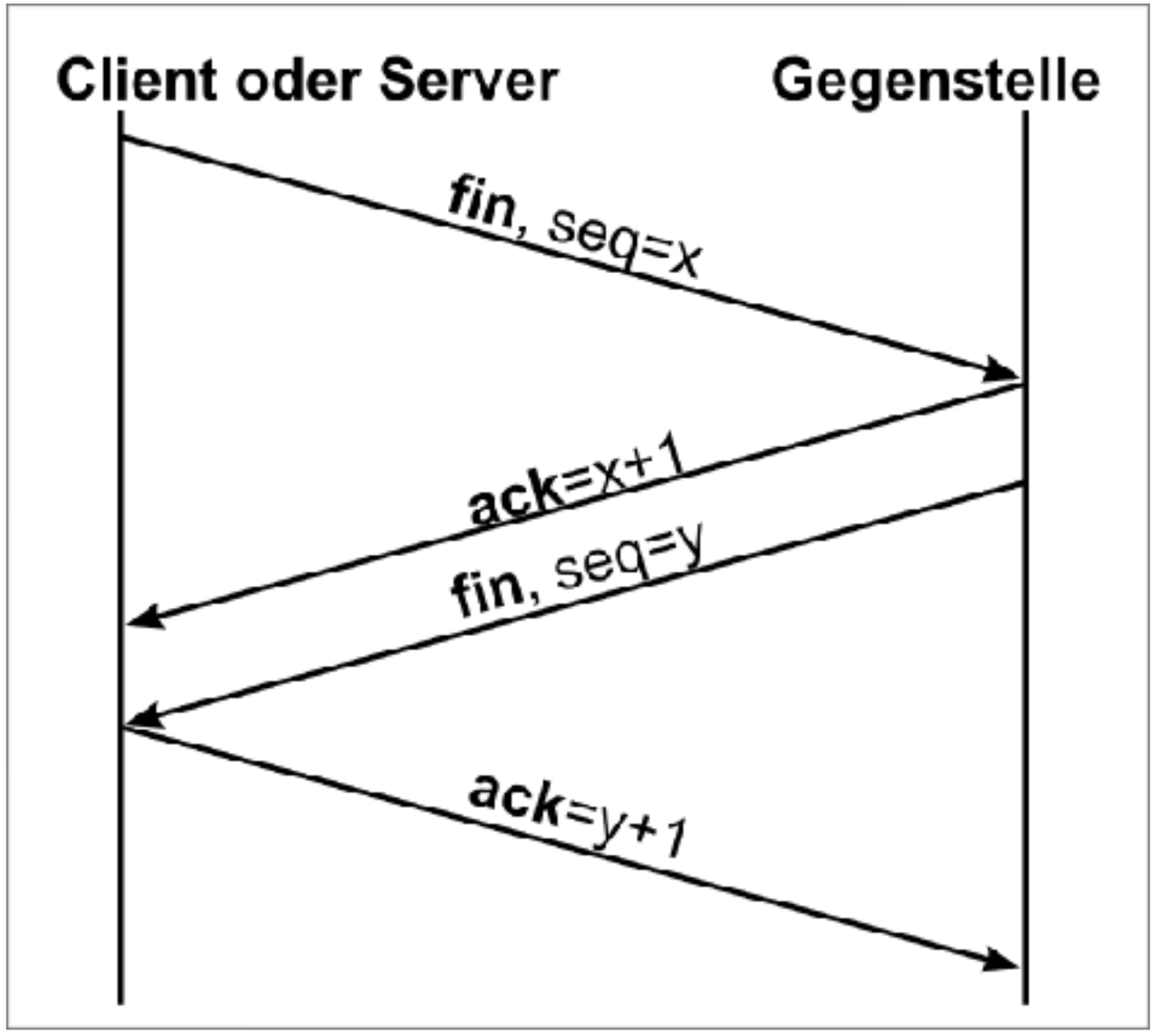
- Komplizierter als vermutet, beide Seiten müssen Einigkeit über Verbindungsabbau erzielen

- Es steht aber nur ein unsicherer Kanal zur Verfügung

- Abbaubestätigungen können verloren gehen

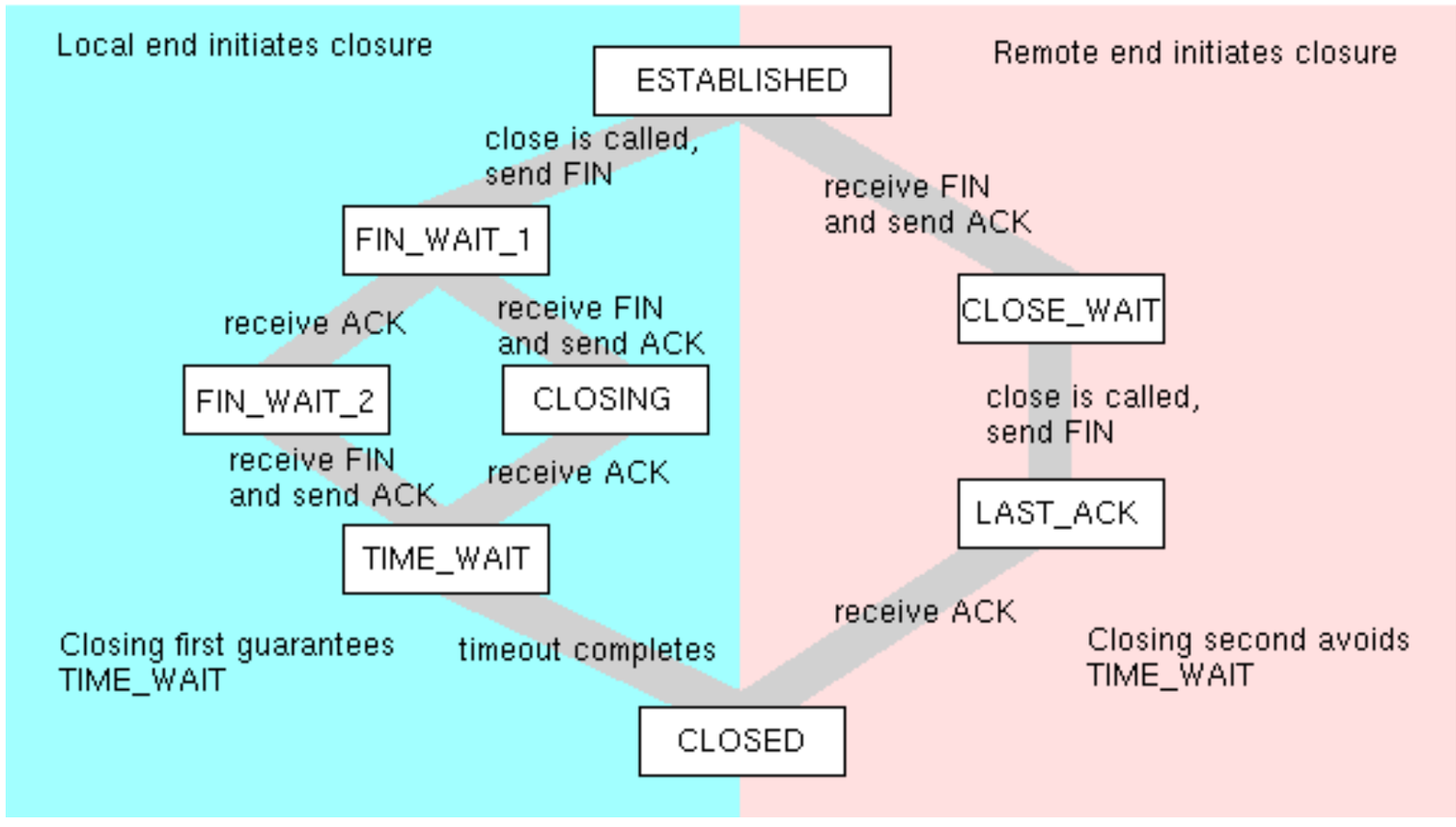
- In der Praxis pragmatisch durch *timeout* gelöst

- Zeit ist so gewählt, das möglichst keine Zombie-Pakete entstehen



[de.wikipedia.org]

„Address already in use“



- Typisches Problem, wenn Server abstürzt oder terminiert wird
- Ursache: ACK für FIN wird vom Betriebssystem nach *socket.close()* behandelt
- Da kein FIN gesendet wird, bleibt der Socket in TIME_WAIT bis zum Timeout

TCP Flags

- **`RST`**: Verbindung soll zurückgesetzt werden
  (Beispiel: Server-Port nicht geöffnet)

- **`SYN`**: Erstes Paket des Client oder des Servers

  - Aktiviert Durchführung von Drei-Wege-Handshake

- **`ACK`**: Markiert Gültigkeit der Bestätigungsnummer

- **`FIN`**: Letztes Paket von diesem Absender

  - Leitet Vier-Weg-Handshake für Verbindungsabbau ein, per ACK bestätigt

Programmierung

- TCP/IP und UDP/IP sind die dominanten Protokolle im Internet
- Jedes (!) Betriebssystem muss sie implementieren
  - Aufgabe des Betriebssystems ist u.a. die Definition standardisierter Systemrufe für die Anwendungen
  - Erlaubt portable Anwendungen
- **Socket-API**: Ursprünglich in den 80er Jahren für BSD-Unix entwickelt
  - Menge von C-Funktionen, die durch das Betriebssystem geboten werden
  - Entsprechende Abbildung in höheren Programmiersprachen

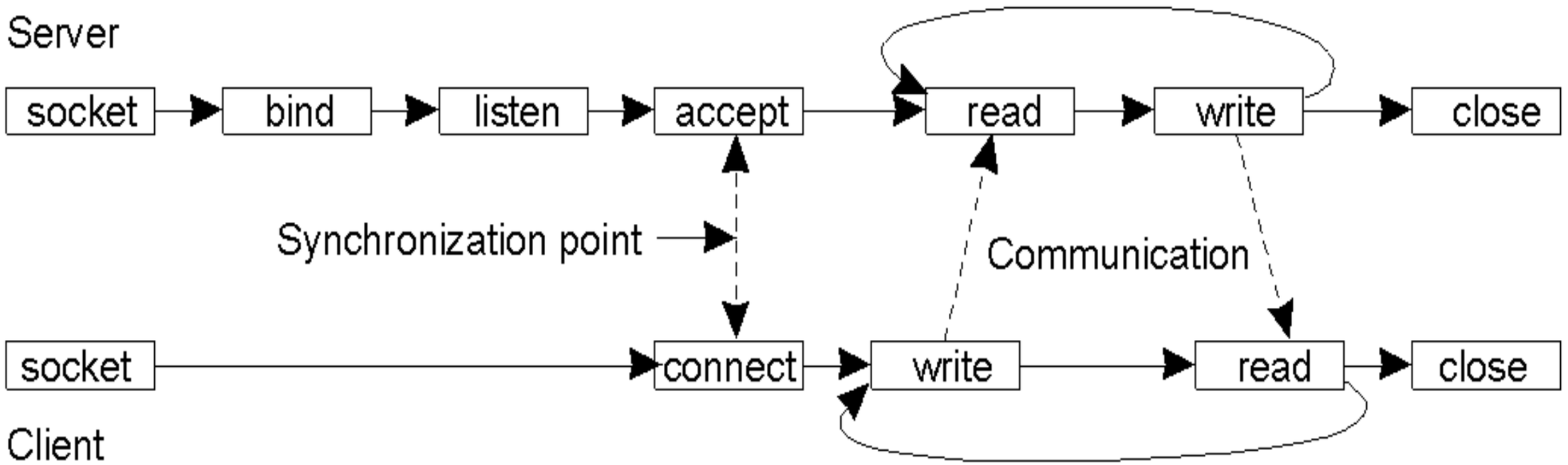
Socket - API

- Socket = Repräsentation eines Endpunkts durch das Betriebssystem
  - Lokale Ressource
  - Historisch für reine IP-Verbindung, mittlerweile bzgl. Layer 1 / 2 flexibel
  - Eingeführt mit RFC 147 (1971)
  - Heutige Version sind Berkeley Sockets (1983)
- **Datagram Socket** - verbindungslos, typischerweise für UDP/IP
- **Stream Socket** - verbindungsorientiert, typischerweise für TCP/IP
- **Raw Socket** - verbindungslos, typischerweise für reines IP

Berkeley Sockets

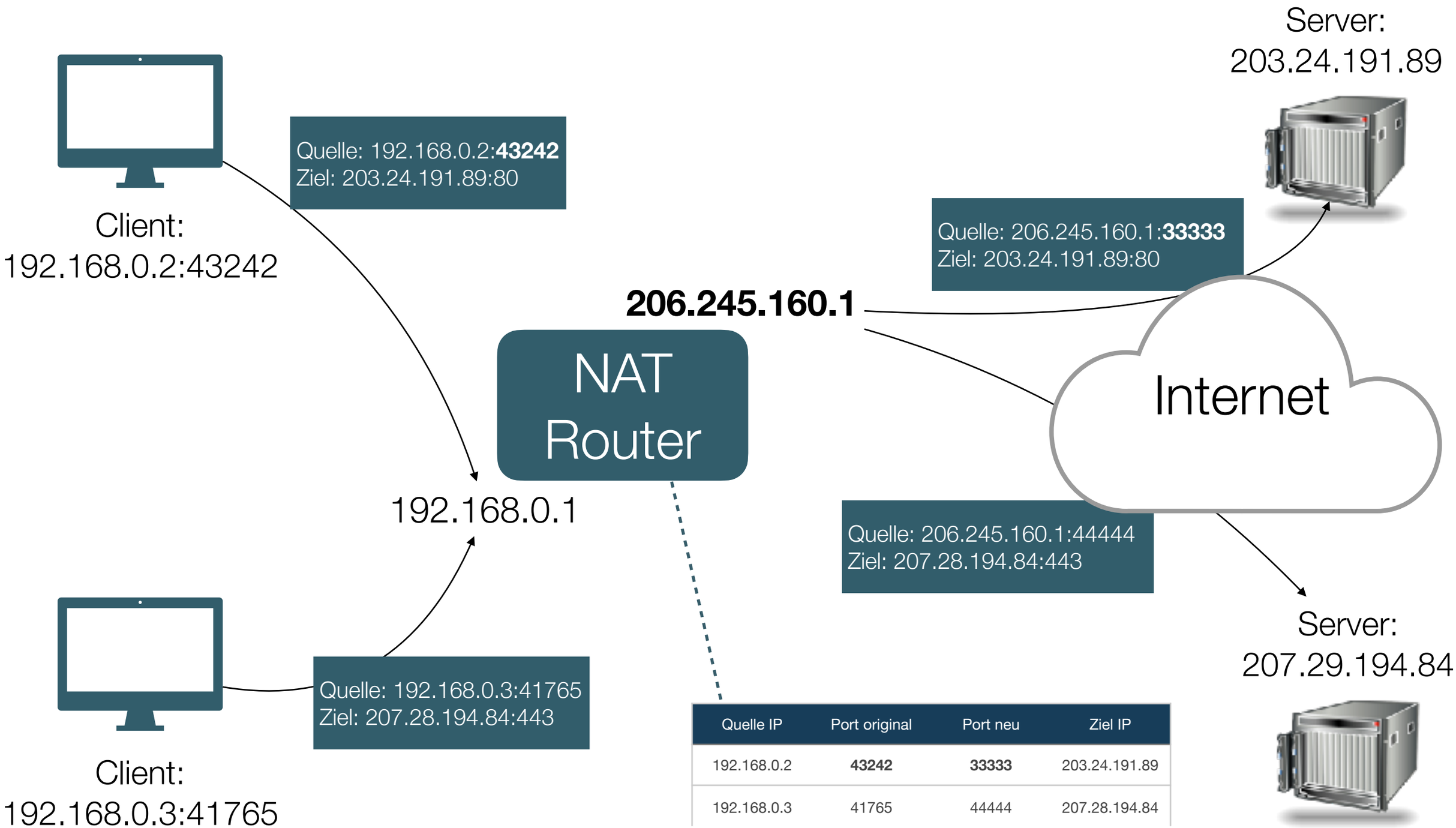
- Idee der netzwerkweiten Umleitung von Ein- und Ausgabekanälen
- Operationen
  - Neuen Kommunikationsendpunkt anlegen (*socket*)
  - Lokale Adresse (+ Portnummer) dem Socket zuweisen (*bind*)
  - Verbindung aufbauen (*connect*), Verbindung beenden (*close*)
  - Beginn der Entgegennahme eingehender Verbindungen (*listen*)
  - Empfänger blockieren, bis Verbindungswunsch eintrifft (*accept*)
  - Daten senden und empfangen (*send, receive*)

Berkeley Sockets



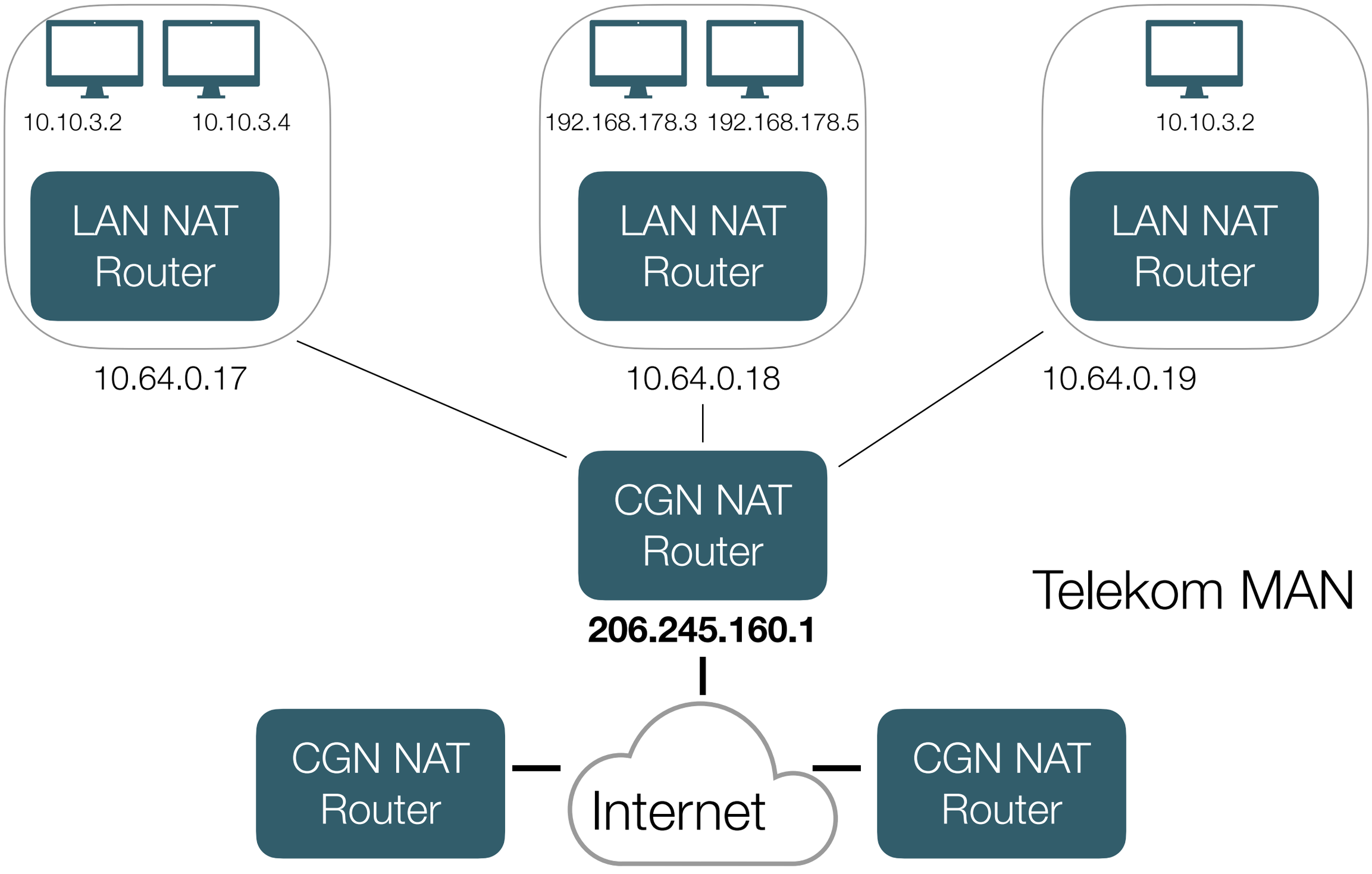
Aus “Computer Networks”, 4th Edition, Tanenbaum, Prentice Hall 2003

Network Address Translation (NAT)



Quelle IP	Port original	Port neu	Ziel IP
192.168.0.2	**43242**	**33333**	203.24.191.89
192.168.0.3	41765	44444	207.28.194.84

Carrier Grade NAT (CGN)



Carrier Grade NAT (CGN)

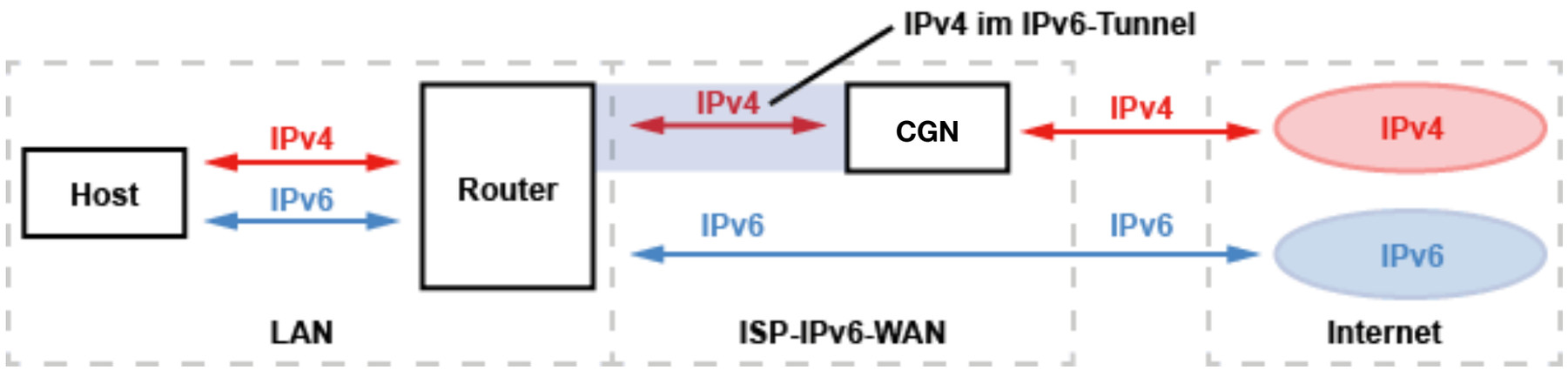
- Internet Providern (ISPs) können auch die öffentlichen IP-Adressen ausgehen
- Router bekommt private Adresse aus dem 100.64.0.0/10 Netz (RFC 6598)
  - LAN kann keine Server mehr enthalten
  - Doppelte „Übersetzung“ führt zu Verzögerungen
  - Ca. 150 Kunden teilen sich eine öffentliche Adresse
  - Protokollierung nötig, um Strafverfolgung noch zu ermöglichen
  - IP-basierte Server-Sperren betreffen plötzlich mehrere Clients

Begriffe

- Carrier-Grade NAT (CGN) = Large-Scale NAT (LSN)
- NAT44 = IPv4 NAT, Router hat öffentliche Adresse
- NAT444 = IPv4 NAT + IPv4 CGN, Router hat private Adresse
  - Probleme mit vielen Anwendungen
    (siehe https://tools.ietf.org/html/draft-donley-nat444-impacts-01)
  - Performanz hängt vom CGN-Router ab
- NAT64 = IPv6 Client kommuniziert mit IPv4 Server über IPv6->IPv4 Router
- IPv6 Tunnel = IPv4 Pakete werden in IPv6 Paketen transportiert

Dual-Stack Lite (DS-Lite)

- ISP hat nur ein IPv6 Netz
- Router verpackt IPv4 - Pakete in IPv6 - Pakete
- Private IPv4 - Absenderadresse für Router, deswegen trotzdem CGN nötig
- IPv6 - Anfragen aus dem LAN werden direkt vermittelt (*dual stack*)
- Zunehmend bei den ISPs verbreitet



https://www.elektronik-kompendium.de/sites/net/2010211.htm

Zusammenfassung

- TCP/IP und UDP/IP sind der Standard im Internet
- Große Anzahl an Anwendungsprotokollen basiert auf dieser Grundlage
- UDP = verbindungslos, keine Garantien, Broadcast möglich
- TCP = verbindungsorientiert, *at-most-once,* Reihenfolge sicherstellen
  - Handshake für Verbindungsabbau
  - Handshake für Verbindungsabbau
- Programmierung mit Socket - API
- Einsatz von Portnummern bei NAT-Routern