PVK Technische Informatik 2

(Teil 1)

Michael Wolf

20.06.2016

Programm

Link Security Network Transport Application Markov

Montag

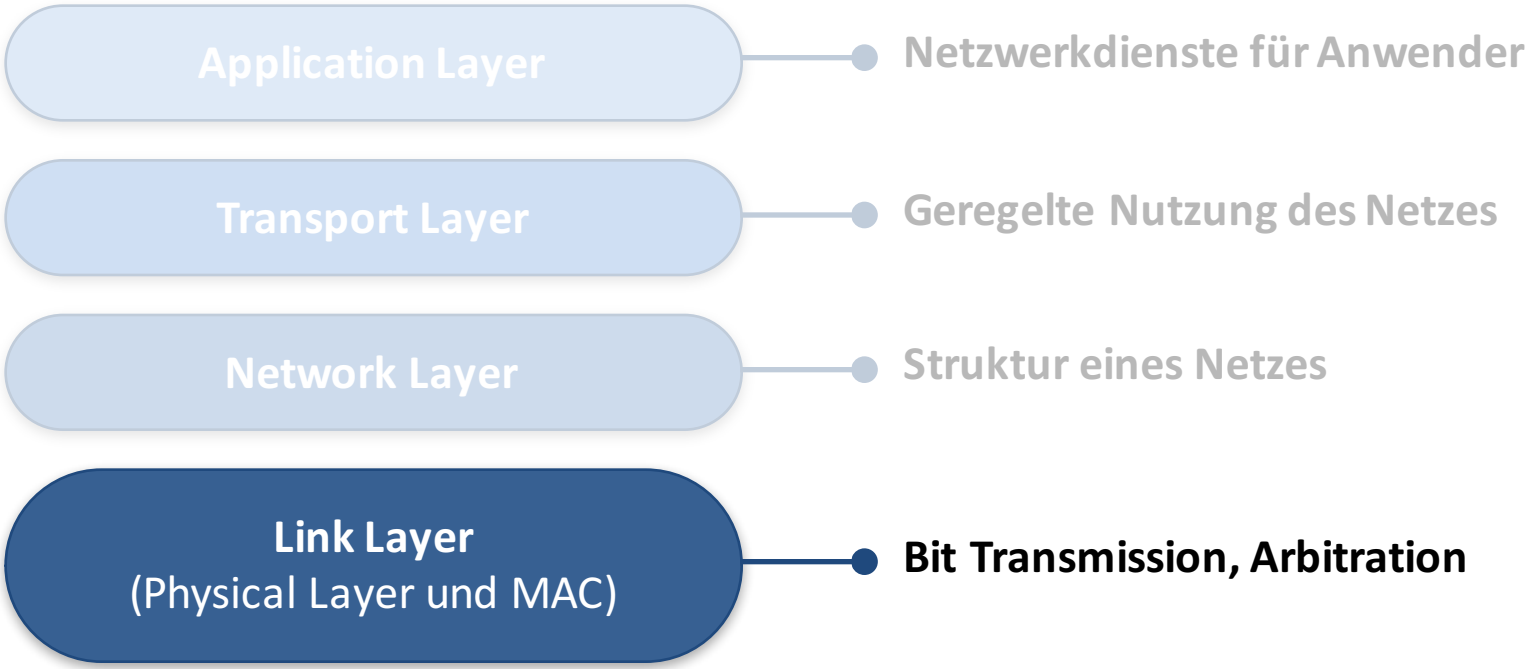
- Kommunikation
  - Link Layer (+ Security)
  - Network Layer
  - Transport Layer
  - Application Layer
- Markov-Ketten

Dienstag

- Speicher
  - Storage
  - Dictionaries & Hashing
  - Datenbanken & SQL
- Concurrency & Locks

Grafisches Material, welches nicht selber erstellt wurde und keine weitere Referenzangabe besitzt, stammt aus dem Vorlesungsmaterial von Prof. Dr. R. Wattenhofer.

Link Layer


Application Layer
Netzwerkdienste für Anwender
Transport Layer
Geregelte Nutzung des Netzes
Network Layer
Struktur eines Netzes
Link Layer
(Physical Layer und MAC)
Bit Transmission, Arbitration

Link Layer

Inhalt

- Physical Layer
- Adressierung
- Wireless Link Phenomena
- Medium Access Control (MAC)

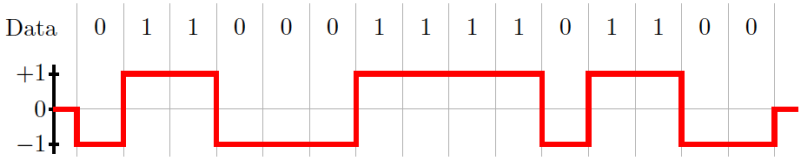
Physical Layer (PHY)

- **Line Coding** → Physikalische Codierung

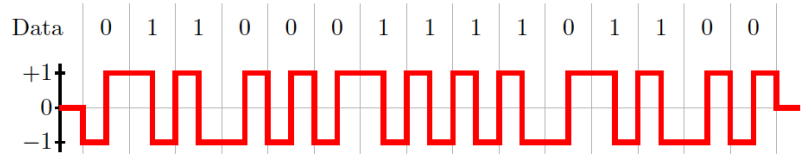
  - Bildet den Daten-Bitstrom (0 & 1) auf die Werte -1, 0, +1 ab

- **Non-Return to Zero (NRZ)** → Bilde 0 auf -1 ab und 1 auf +1

  - Nachteil: Synchronisation und Mittelwert



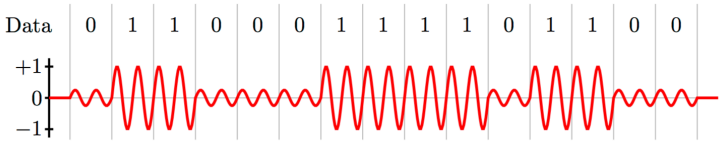
- **Manchester Coding** → Bit wird auf 2 Werte abgebildet (0 auf -1, +1 und 1 auf +1, -1)



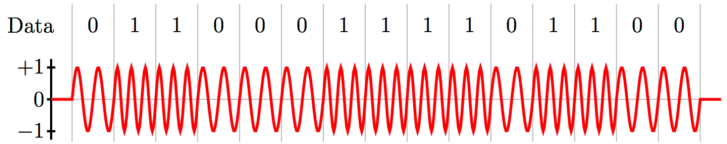
Physical Layer (PHY)

- **Modulation** → Bits auf periodische Wellenform (Carrier Signal) übertragen

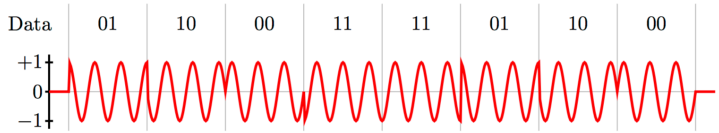
  - Amplitude Modulation (AM)



  - Frequency Modulation (FM)

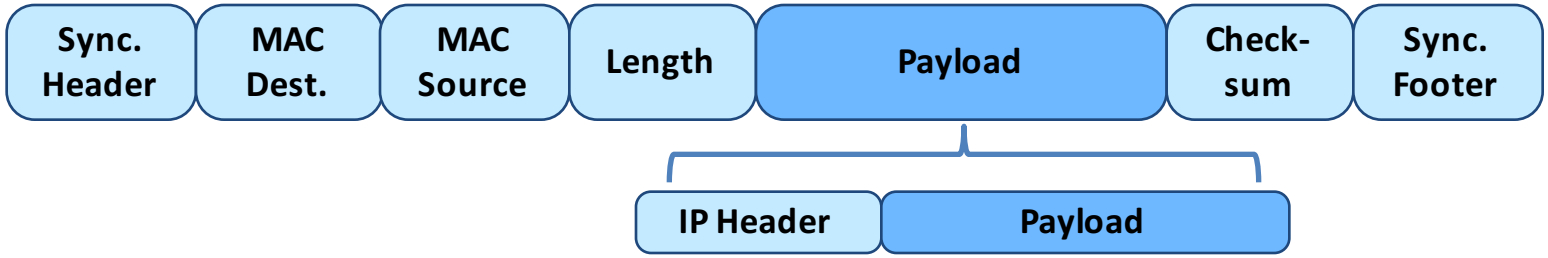


  - Phase Modulation (PM)



Physical Layer (PHY)

- **Pakete / Frames**



- **Payload**

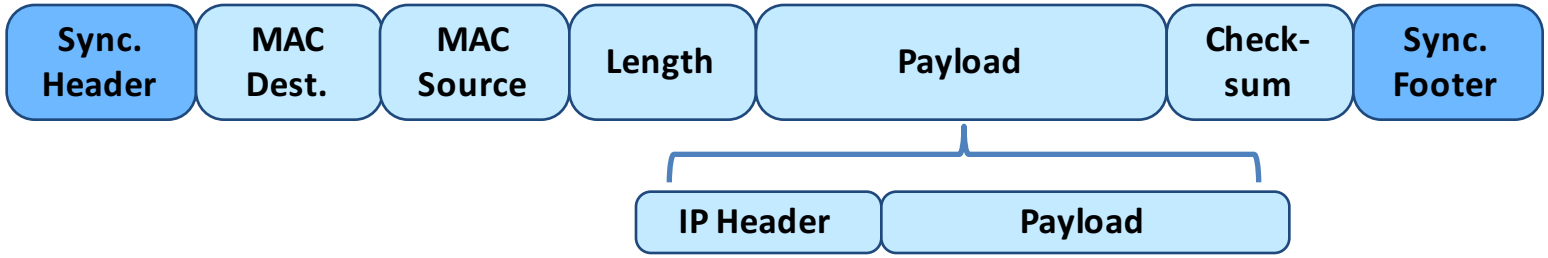
  - IP Packet wird zur Payload für das Link Layer Frame

- **Checksum**

  - Prüfsumme für Header oder Payload

Physical Layer (PHY)

- **Pakete / Frames**



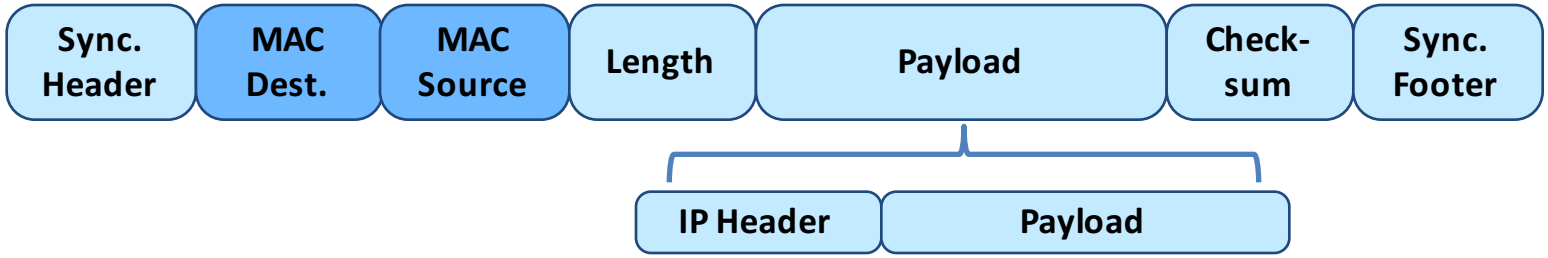
- **Header & Footer / Syncword / Preamble**
  - Bitsequenz, welche Start von Paket markieren
  - Ende signalisiert durch Länge oder „Footer“
  - **Bit / Byte Stuffing** → Verhindert auftreten von Syncword in Daten

Physical Layer (PHY)

- **Bit / Byte Stuffing** → Kritische Sequenz = X
  - Problem: XABCXABXAYBXCXABC
  - Escape Sequence Y: Y → YA und X → YB
  - Lösung: XABCXABXAYABYBCXABC
  - Nachteil: Daten können sich verdoppeln (Worst Case)
- **Consistent Overhead Byte Stuffing** → Sende kritische Sequenz nicht mit
  - Original: s_0, X, s_1, X, s_2
  - Übertragung: len(s_0), s_0, len(s_1), s_1, len(s_2), s_2
  - Codierte Nachricht nur 1 Byte länger als Original

Physical Layer (PHY)

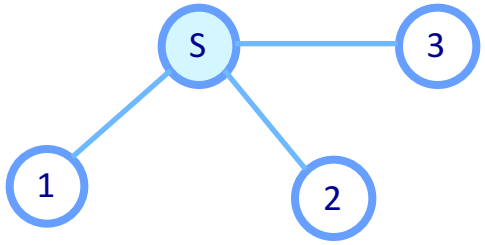
- **Pakete**



- **MAC Destination / MAC Source**
- **MAC Adresse** → 00:00:00:00:00:00
  - 6 Bytes (Hexadezimalziffern)
  - Identifizierung von Knoten unterhalb des Network Layers
  - IP-Adressen sind auf dieser Hierarchie-Ebene nicht bekannt

Adressierung

- **Switch** → Netzwerkknoten, welcher Verkehr seiner Nachbarn regelt
  - Hat keine Kenntnisse über IP Adressen
  - Operiert somit ausschliesslich auf Link Layer
  - Switches besitzen keine eigene MAC Adresse
  - Können lernen, auf welchen Ports welche Adressen erreichbar sind



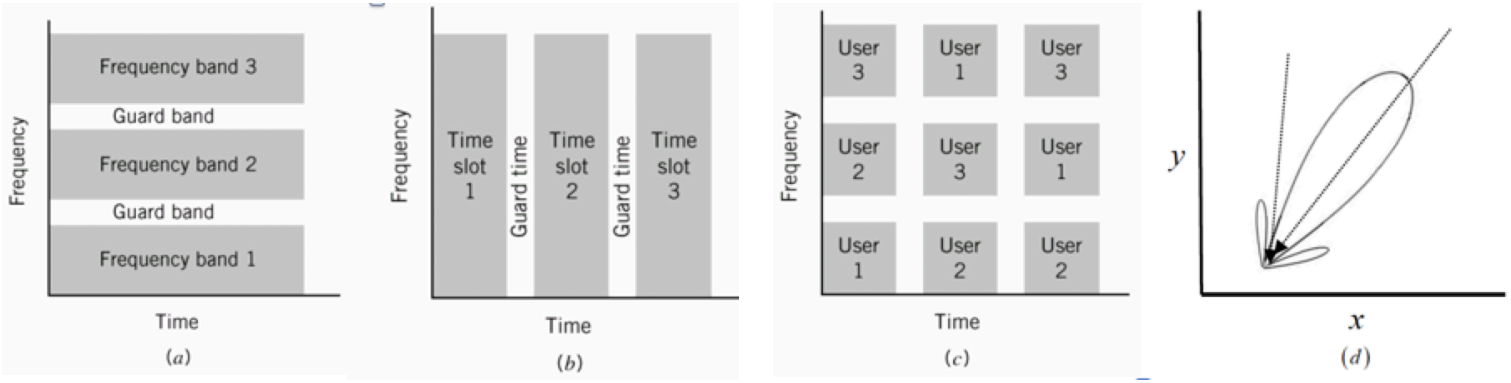
Wireless Link Phenomena

- **Half-Duplex** → Geräte können nicht gleichzeitig senden und empfangen
  - Kabelbasierte Kommunikation meistens full-duplex
- Transmission ist immer broadcast in gewissem Range
- **Signal-to-interference-plus-noise ratio (SINR)**

$$\mathrm{SINR} = \frac{S}{I + N} > \beta$$

Medium Access Control (MAC)

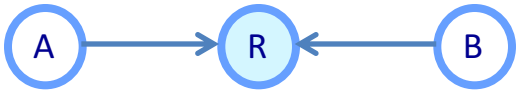
- **Multiple Access** → Mehrere Geräte kommunizieren über gemeinsames Medium
- **Collisions** → Entstehen vor allem bei Wireless-Kommunikation
- **Medium Division** → Jeder Domain kann nur von einem Gerät benutzt werden
  - Frequency Division
  - Time Division
  - Code Division
  - Space Division



Aus den Folien von Prof. Dr. A. Wittneber

Medium Access Control (MAC)

- **Carrier Sensing (CS)** → Prüfe Kanal vor Transmission
  - Falls Kanal frei, sende direkt
  - Falls Kanal besetzt, warte eine Zeit und prüfe nochmals
- **Hidden Terminal Problem** → Verschiedene Sender können sich nicht „hören“



- **Exposed Terminal Problem** → Sender „hören“ sich, obwohl keine Interferenz

Medium Access Control (MAC)

- **Slotted ALOHA** → Collision Response

  - Zeit ist in diskrete „Slots“ eingeteilt

  - n Sender

Algorithm 10.25 Slotted Aloha

1: **Every node** v executes the following code:
2: **repeat**
3: transmit with probability $1/n$
4: **until** one node has transmitted alone

- Wahrscheinlichkeit einer erfolgreichen Übertragung pro Slot:

$$\Pr[X = 1] = n \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1}$$

Medium Access Control (MAC)

- **Request To Send (RTS) / Clear To Send (CTS)**

  - Statusnachrichten der kommunizierenden Knoten

- **Backoff Time / Backoff Strategies** → Wie lange warten bis nächster Versuch

- **Random Exponential Backoff**

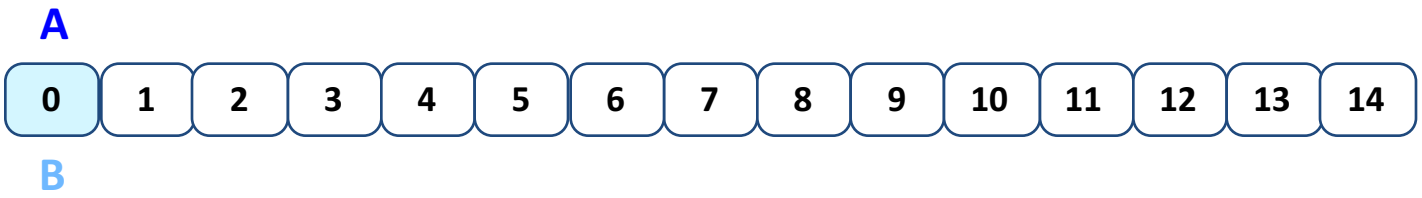
  - Je mehr Kollisionen, desto länger soll gewartet werden

Algorithm 10.28 Random exponential backoff

1: $i \leftarrow 0$
2: Attempt sending.
3: **while** sending unsuccessful **do**
4: $\quad i \leftarrow i + 1$
5: $\quad$ Pick a value from the interval $[0, c^i]$ uniformly at random and wait that many time units.
6: $\quad$ Attempt sending again.
7: **end while**

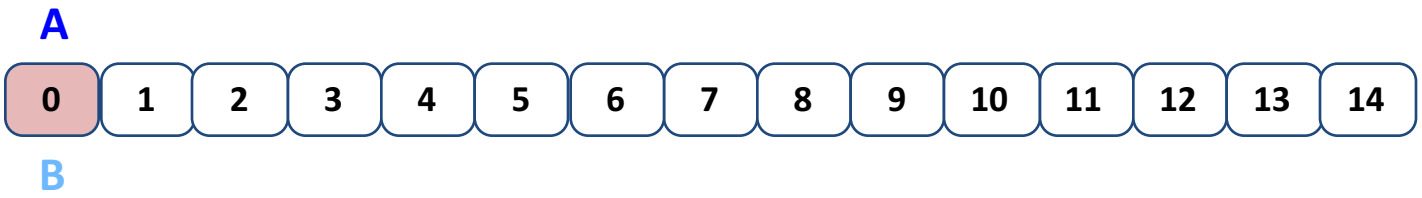
Medium Access Control (MAC)

- **Random Exponential Backoff**



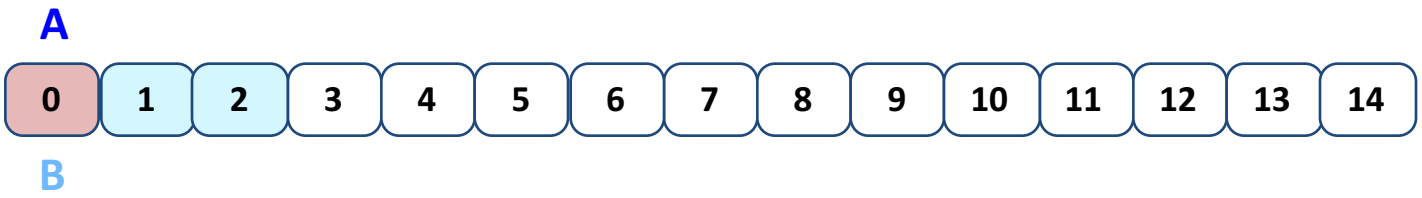
Medium Access Control (MAC)

- **Random Exponential Backoff**



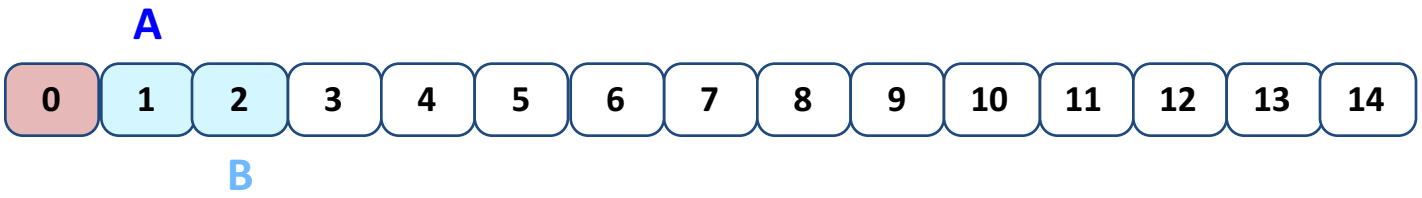
Medium Access Control (MAC)

- **Random Exponential Backoff**



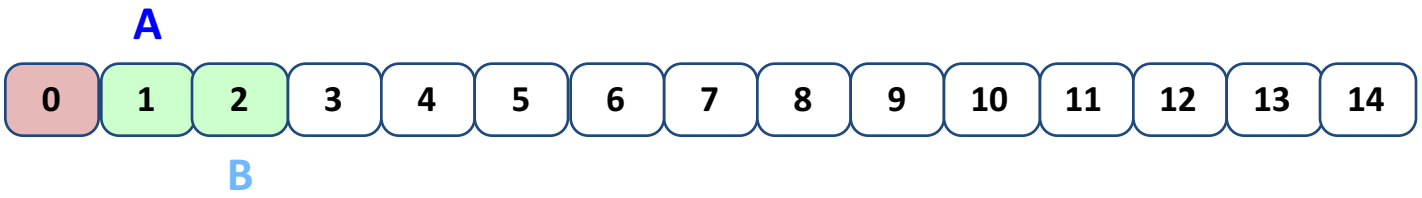
Medium Access Control (MAC)

- **Random Exponential Backoff**



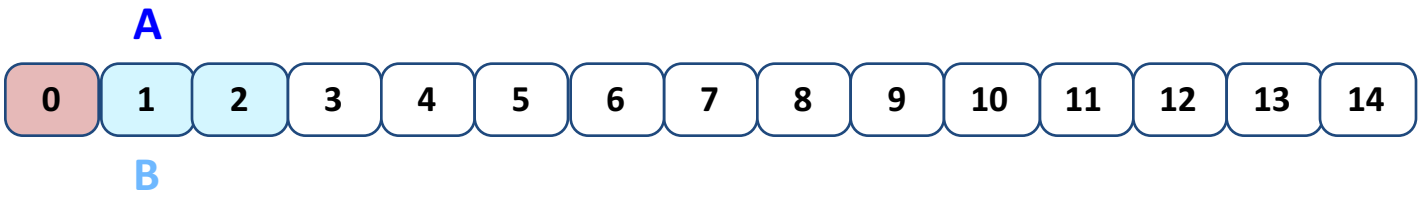
Medium Access Control (MAC)

- **Random Exponential Backoff**



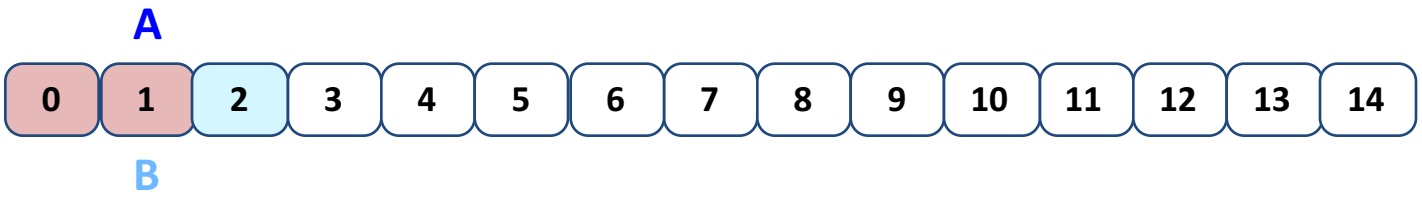
Medium Access Control (MAC)

- **Random Exponential Backoff**



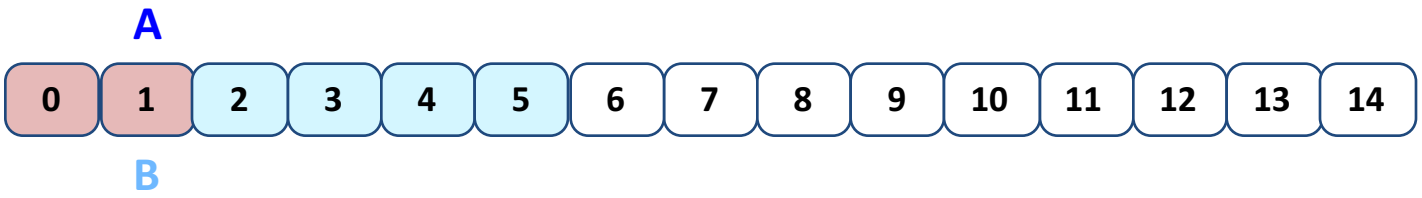
Medium Access Control (MAC)

- **Random Exponential Backoff**



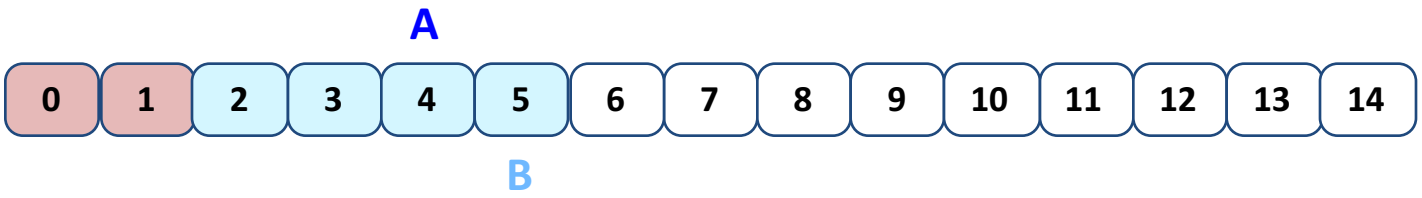
Medium Access Control (MAC)

- **Random Exponential Backoff**



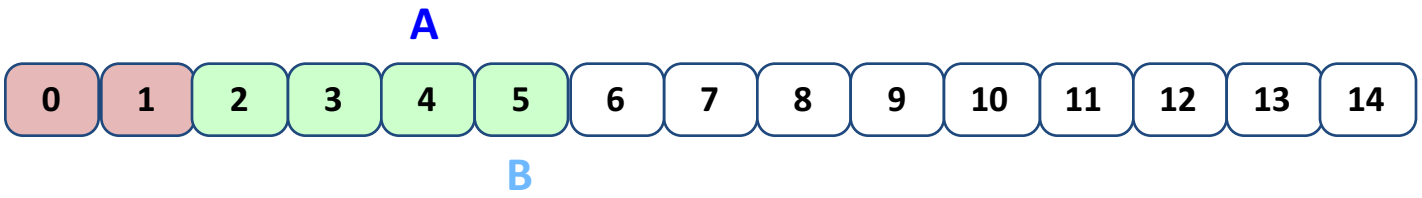
Medium Access Control (MAC)

- **Random Exponential Backoff**



Medium Access Control (MAC)

- **Random Exponential Backoff**



Empfohlene Übungen

- **Serie 10**
  - Aufgabe 2.c (Basic)
  - Aufgabe 3 (Basic)
  - Aufgabe 4 (Advanced)
  - Aufgabe 5 (Advanced)

Security

Inhalt

- Key Exchange
- Man in the Middle Attack
- Public Key Cryptography
- Secret Sharing and Bulk Encryption
- Message Authentication and Passwords

Key Exchange

- Aufgabe → Finde Schlüssel ohne einander zu treffen (**Public Key**)
- **Primitive Root g** → Für alle h: $1 \leq h < p$ → $g^k = h \bmod p$

  - p = Primzahl

  - k = Natürliche Zahl

p = 5
g = 2 → Primitive root of p

$2^1 = 2 \bmod 5$
$2^2 = 4 \bmod 5$
$2^3 = 3 \bmod 5$
$2^4 = 1 \bmod 5$

p – 1 = 4
Teilerfremde Zahlen: 1, 3

$g_1 = 2$
$g_2 = 2^3 \bmod 5 = 3$

Key Exchange

Algorithm 12.3 Diffie-Hellman Key Exchange

Input: Publicly known prime p and a primitive root g of p.
Result: Alice and Bob agree on a common secret key.

1: Alice picks k_A, with $1 \leq k_A \leq p-2$ and sends $g^{k_A} \mod p$ to Bob
2: Bob picks k_B, with $1 \leq k_B \leq p-2$ and sends $g^{k_B} \mod p$ to Alice
3: Alice calculates $\left(g^{k_B}\right)^{k_A} \mod p = g^{k_B k_A} \mod p$
4: Bob calculates $\left(g^{k_A}\right)^{k_B} \mod p = g^{k_A k_B} \mod p$
5: Alice & Bob have a common secret key $g^{k_A k_B} \mod p = g^{k_B k_A} \mod p$

- **Alice**
  - Wählt $k_A = 2$
  - Sendet 2^2 mod 5 (also 4) an Bob
  - Rechnet 3^2 mod 5 = **4**
- **Bob**
  - Wählt $k_B = 3$
  - Sendet 2^3 mod 5 (also 3) an Bob
  - Rechnet 4^3 mod 5 = **4**

Key Exchange

- Challenge → Finde möglichst grosse Primzahl für Diffie-Hellman Key Exchange

Algorithm 12.5 Probabilistic Primality Testing

Input: An odd number $p \in \mathbb{N}$.
Result: Is p a prime?

1: Let $j, r \in \mathbb{N}$ and j odd with $p - 1 = 2^r j$
2: Select $x \in \mathbb{N}$ uniformly at random, $1 \leq x < p$
3: Set $x_0 = x^j \mod p$
4: **if** $x_0 = 1$ or $x_0 = p - 1$ **then**
5: Output "p `is probably prime`" and **stop**
6: **end if**
7: **for** $i = 1, \ldots r - 1$ **do**
8: Set $x_i = x_{i-1}^2 \mod p$
9: **if** $x_i = p - 1$ **then**
10: Output "p `is probably prime`" and **stop**
11: **end if**
12: **end for**
13: Output "p `is not prime`"

Key Exchange

r = 3
j = 3
p – 1 = 24 → p = 25

x = 4
$x_0 = x^j \bmod p = 14$

14 != 1 and 14 != p - 1
→ Don‘t break!

For i = 1, … , r - 1

$x_1 = x_0^2 \bmod 25 = 46$ → Don‘t break

$x_2 = x_1^2 \bmod 25 = 16$ → Don‘t break

„p is not prime“

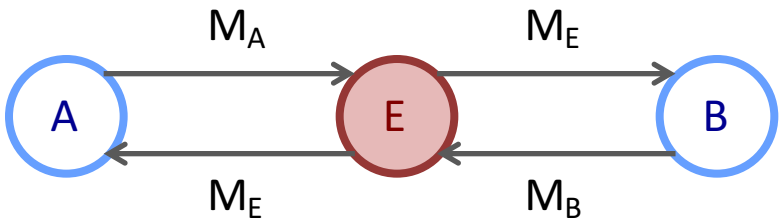
Key Exchange

- Falls Output „p is probably prime“ → Zu 75% wahr
- Falls Output „p is not prime“ → Zu 100% wahr

- Wahrscheinlichkeit für korrekte Aussage erhöhen
  - Wähle anders x (unabhängige Resultate)
  - Fehlerwahrscheinlichkeit = 0.25^r für r Durchläufe
  - Solange laufen lassen bis Pr[error] < ε

Man in the Middle Attack

- Kann Diffie-Hellman Key Exchange angreifen
- Angreifer entschlüsselt oder ändert Nachrichten zwischen Bob und Alice
  - Ohne dass die beiden dies bemerken



- Lösung → Privater Schlüssel / Private Key
- **Forward Secrecy**
  - Abfangen von (Private) Key entschlüsselt vorhergehende Runden nicht
  - Zukünftige Runden können jedoch entschlüsselt werden

Forward Secrecy

- Secret Key muss im Voraus definiert werden

Algorithm 12.11 Diffie-Hellman Key Exchange with Forward Secrecy

Input: Alice's and Bob's common secret key $k_{A,B}$.
Result: A Diffie-Hellman key exchange not vulnerable to a man in the middle attack, and with forward secrecy.

1: Alice and Bob run Algorithm 12.3 to obtain round key $g^{k_A k_B}$
2: Alice sends a random number $1 \leq x_A \leq p-2$ encrypted with $k_{A,B}$ as c_A to Bob, and Bob sends Alice a random number $1 \leq x_B \leq p-2$ encrypted with $k_{A,B}$ as c_B a challenge, respectively
3: Alice and Bob decrypt the respective messages, and Alice sends $x_B + 1$ encrypted with $k_{A,B}$ to Bob as a response (and Bob as well with $x_A + 1$)
4: If decryption yields $x_A + 1$ for Alice, or $x_B + 1$ for Bob, respectively, they accept the round key $g^{k_A k_B}$

Forward Secrecy

- Secret Key $k_{A,B} = 7$
- Round Key = 4

- $m_{A1} = 5 \rightarrow c_{A1} = 35$
- $m_{B1} = 12 \rightarrow c_{B1} = 84$

- $m_{A2} = 13 \rightarrow c_{A2} = 91$
- $m_{B2} = 6 \rightarrow c_{B2} = 42$

Secure Communication! → Use Round Key

Public Key Cryptography

- Jeder hat 2 Arten von Schlüssel
  - Secret / Private Key → Decodieren
  - Public Key → Codieren
- **Finde Private und Public Key**

Algorithm 12.14 Elgamal Public Secret Key Generation

Input: Publicly known prime p and a primitive root g of p.
Result: Alice generates a public and a secret key

1: Alice picks random k_s with $1 \leq k_s \leq p-2$ as her secret key
2: Alice calculates $k_p = g^{k_s} \mod p$ as her public key

p = 5
g = 2

$k_s = 3$
$k_p = 2^3 \mod 5 = 3$

Public Key Cryptography

➢ Verschlüsseln und Entschlüsseln

Algorithm 12.15 Elgamal Public Key Encryption and Decryption

Input: Alice and Bob know p, g, k_p, Alice knows k_s.
Result: Bob sends Alice an encrypted message, which she can decrypt.

1: Bob picks a message $1 \leq m \leq p-2$ and a random $1 \leq x \leq p-2$
2: Bob sends $g^x \mod p$ and $c = m \cdot k_p^x \mod p$ to Alice
3: Alice first calculates $y = (g^x)^{p-k_s-1} \mod p$
4: Alice then obtains $m = y \cdot c \mod p$

p = 5 g = 2	$k_s = 3$ $k_p = 3$	m = 1 x = 2	Send: $2^2 \bmod 5 = 4$ $c = 1 * 3^2 \bmod 5 = 4$	$y = 2^{2(5-3-1)} \bmod 5 = 4$ $m = 4 * 4 \bmod 5 = 1$

Public Key Cryptography

- **Web of Trust** → Ungerichteter oder gerichteter Graph
  - Kante repräsentiert Vertrauen zwischen zwei Knoten
  - **Certificate Authority (CA)** → Knoten mit vielen Kanten zu Nachbarn
  - Vertrauen muss nicht auf Gegenseitigkeit beruhen

Secret Sharing and Bulk Encryption

- **Perfect Secrecy** → Verschlüsselte Nachricht trägt keine Information
  - Ausser Paketlänge
- **(t,n)-Threshold Secret Sharing**
  - Secret an n Teilnehmer schicken
  - t müssen zusammenarbeiten zur Wiederherstellung
  - Besitzt „Perfect Secrecy“

Secret Sharing and Bulk Encryption

Algorithm 12.26 (n, n)-Threshold Secret Sharing

Input: A secret k, encoded in binary representation of length $l(k)$.

Secret distribution

1: Generate $n-1$ random binary numbers k_i of length $l(s)$ and distribute them among $n-1$ participants
2: Give participant n the value k_n as the result of XOR of k and $k_1, \ldots, k_{n-1}$, i.e., $k_n = k_1 \oplus k_2 \oplus \cdots \oplus k_{n-1}$

Secret recovery

1: Collect all n values $k_1, \ldots, k_n$ and obtain $k = k_1 \oplus k_2 \oplus \cdots \oplus k_{n-1} \oplus k_n$

n = 4

Secret
k = 1111111
l(k) = 7

k_1 = 1110000
k_2 = 0000111
k_3 = 1010101

k_n = k XOR k_1 XOR k_2 XOR k_3 = 1011101

k = k_n XOR k_1 XOR k_2 XOR k_3 = 1111111

Secret Sharing and Bulk Encryption

Algorithm 12.28 (t, n)-Threshold Secret Sharing

Input: A secret k, represented as a real number.

Secret distribution

1: Generate $t-1$ random $a_1, \ldots, a_{t-1} \in \mathbb{R}$
2: Obtain a polynomial f of degree $t-1$ with $f(x) = k + a_1 x + \cdots + a_{t-1} x^{t-1}$
3: Generate n distinct $x_1, \ldots, x_n \in \mathbb{R} \setminus 0$
4: Distribute $(x_1, f(x_1))$ to participant P_1, $\ldots$, $(x_n, f(x_n))$ to P_n

Secret recovery

1: Collect t pairs $(x_i, f(x_i))$ from at least t participants
2: Use Lagrange's interpolation formula to obtain $f(0) = k$

t = 3, n = 4

Secret
k = 5

$a_1 = 2$
$a_2 = 6$
$f(x) = 5 + 2x + 6x^2$

$x_1 = 1$ $x_2 = 4$
$x_3 = 3$ $x_4 = 5$

$f(x_1) = 13$ $f(x_2) = 109$
$f(x_3) = 65$ $f(x_4) = 165$

Secret Sharing and Bulk Encryption

Algorithm 12.28 (t, n)-Threshold Secret Sharing

Input: A secret k, represented as a real number.

Secret distribution

1: Generate $t-1$ random $a_1, \ldots, a_{t-1} \in \mathbb{R}$
2: Obtain a polynomial f of degree $t-1$ with $f(x) = k + a_1 x + \cdots + a_{t-1} x^{t-1}$
3: Generate n distinct $x_1, \ldots, x_n \in \mathbb{R} \setminus 0$
4: Distribute $(x_1, f(x_1))$ to participant P_1, $\ldots$, $(x_n, f(x_n))$ to P_n

Secret recovery

1: Collect t pairs $(x_i, f(x_i))$ from at least t participants
2: Use Lagrange's interpolation formula to obtain $f(0) = k$

$$x_0 = 1 \quad f(x_0) = 1$$
$$x_1 = 2 \quad f(x_1) = 4$$
$$x_2 = 3 \quad f(x_2) = 9$$

$$L(x) = 1 \cdot \frac{x-2}{1-2} \cdot \frac{x-3}{1-3} + 4 \cdot \frac{x-1}{2-1} \cdot \frac{x-3}{2-3} + 9 \cdot \frac{x-1}{3-1} \cdot \frac{x-2}{3-2}$$

Secret Sharing and Bulk Encryption

- **Bulk Encryption Algorithm** → Verschlüsselt Nachricht beliebiger Länge sicher
- **One-Time Pad** → Perfect Secrecy

Algorithm 12.30 One-Time Pad

Input: A message m known to Alice, and a symmetric key k (as a random bitstring) of length $l(s)$ known by both Alice and Bob.

Encryption

1: Alice sends $c = m \oplus k$ to Bob

Decryption

1: Bob obtains m by $m = c \oplus k$

Secret Sharing and Bulk Encryption

- **Electronic Code Book (ECB)** → Jeden Block einzeln verschlüsseln mit XOR

  - r Blöcke → Je x Bits

$m_1 = 1001011$ $m_2 = 0101101$ $m_3 = 1110001$	XOR	$k = 1010101$	=	$c_1 = 0011110$ $c_2 = 1111000$ $c_3 = 0100100$

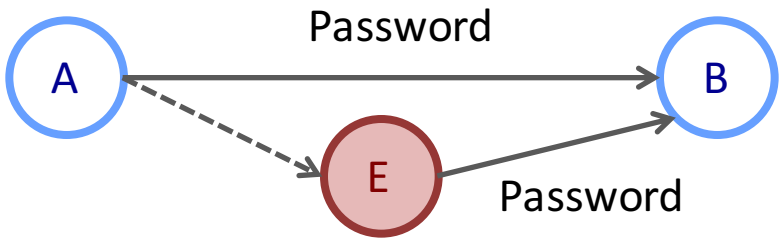
- **Cipher Block Chaining (CBC)** → Jeden Block mit Vorgänger verschlüsseln (XOR)

  - Erster Block wird zufällig gewählt

$m_1 = 1001011$ $m_2 = 0101101$ $m_3 = 1110001$	CBC =	$c_1 = 1111111$ (random) $c_2 = f(m_2 \text{ XOR } c_1)$ $c_3 = f(m_3 \text{ XOR } c_2)$

Message Authentication and Passwords

- **Replay Attack**
  - Vorherig gültige Nachricht wird von Eve erneut gesendet
  - Verhindern durch Zeitstempel



- **Malleability**
  - Ciphertext c wird zu c‘ verfälscht
  - Empfänger entschlüsselt c‘ fälschlicherweise zu m‘
  - Fehler wird nicht bemerkt

Message Authentication and Passwords

- **One-Way Hash Function** → Nachrichten können nicht mehr verfälscht werden
  - Hash Function, bei welcher es für h(x) = z schwer ist, x zu ermitteln
- **Collision Resistant Hash Function**
  - Schwierig 2 Elemente zu finden, die auf gleichen Bucket gemappt werden
- **Message Authentication Code (MAC)** → Bitstring
  - Nachricht kommt von bestimmtem Sender
  - Nachricht wurde nicht verfälscht

Empfohlene Übungen

- **Serie 12**
  - Aufgabe 3 (Basic to Advanced)
  - Aufgabe 4 (Basic to Advanced)

Network Layer

Application Layer — Netzwerkdienste für Anwender

Transport Layer — Geregelte Nutzung des Netzes

Network Layer — **Struktur eines Netzes**

Link Layer
(Physical Layer und MAC) — Bit Transmission, Arbitration

Network Layer

Inhalt

- Graphen
- Adressen
- Packets
- Routing

Graphen

- n Knoten und m Kanten
- **Adjazenzmatrix** → Charakterisiert Graphen
- Knoten können **inzident** zu Kanten sein
- Knoten können **adjazent** zueinander sein
- **Gewichteter Graph** → Jede Kante hat Gewicht ω
  - Totales Gewicht ω(T) → Summe aller Kantengewichte
- **Path / Weg** → Abfolge von Knoten
  - Länge = Anzahl Kanten
- **Tree / Baum** → Kreisfreier Graph

Graphen

- **Teilgraph** → Enthält nur eine Auswahl von Knoten & Kanten
- **Spanning Tree** → Teilgraph mit Baumeigenschaften
  - Enthält alle Knoten des ursprünglichen Graphen
  - Enthält eine Auswahl von Kanten
- **Minimum Spanning Tree (MST)** → Spanning Tree mit kleinstem totalen Gewicht

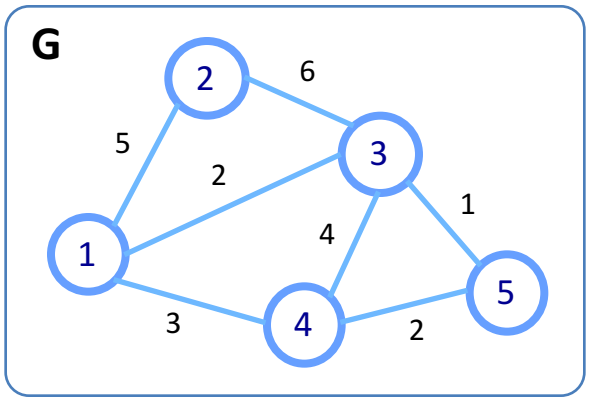
Algorithm 1.11 MST Algorithm

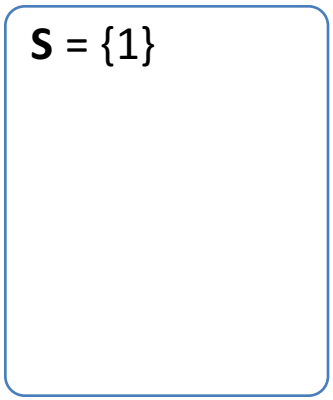
1: Given a weighted graph $G = (V, E, \omega)$
2: Let $S = \{u\}$ be a set of visited nodes, initialized with any node $u \in V$
3: Let T be a tree just consisting of the single node $u \in S$, no edges
4: **while** $S \neq V$ **do**
5: Find minimum weight edge $e = \{v, w\}$ with $v \in S$ and $w \in V \setminus S$
6: Add node w to S
7: Add edge e to T
8: **end while**

Graphen

Algorithm 1.11 MST Algorithm

1: Given a weighted graph $G = (V, E, \omega)$
2: Let $S = \{u\}$ be a set of visited nodes, initialized with any node $u \in V$
3: Let T be a tree just consisting of the single node $u \in S$, no edges
4: **while** $S \neq V$ **do**
5: Find minimum weight edge $e = \{v, w\}$ with $v \in S$ and $w \in V \setminus S$
6: Add node w to S
7: Add edge e to T
8: **end while**

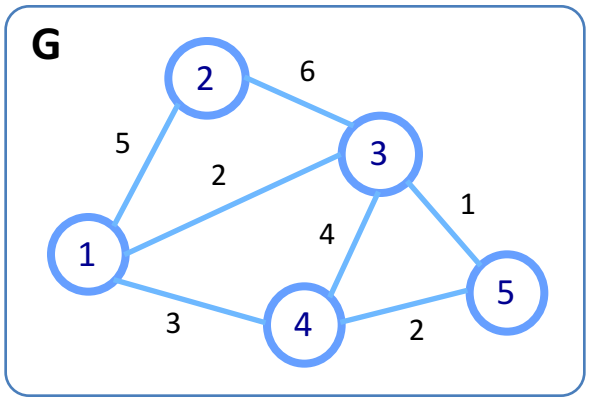




Graphen

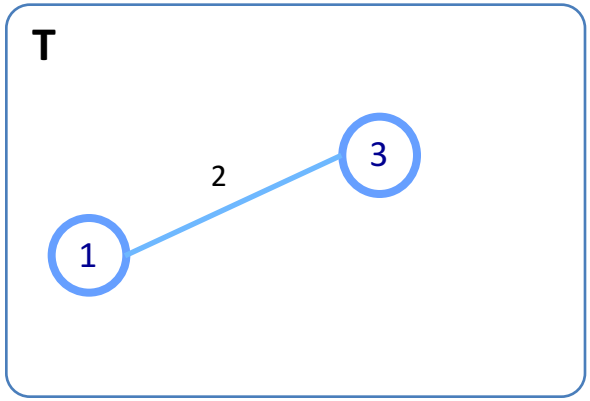
Algorithm 1.11 MST Algorithm

1: Given a weighted graph $G = (V, E, \omega)$
2: Let $S = \{u\}$ be a set of visited nodes, initialized with any node $u \in V$
3: Let T be a tree just consisting of the single node $u \in S$, no edges
4: **while** $S \neq V$ **do**
5: Find minimum weight edge $e = \{v, w\}$ with $v \in S$ and $w \in V \setminus S$
6: Add node w to S
7: Add edge e to T
8: **end while**



S = {1}

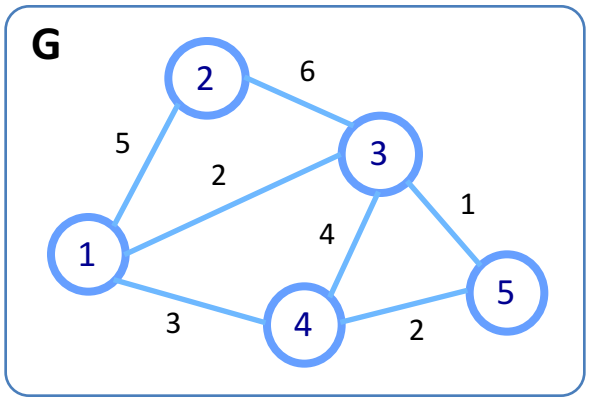
S = {1,3}



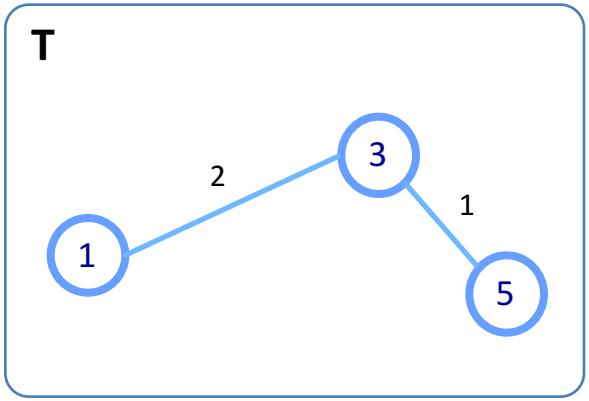
Graphen

Algorithm 1.11 MST Algorithm

1: Given a weighted graph $G = (V, E, \omega)$
2: Let $S = \{u\}$ be a set of visited nodes, initialized with any node $u \in V$
3: Let T be a tree just consisting of the single node $u \in S$, no edges
4: **while** $S \neq V$ **do**
5: Find minimum weight edge $e = \{v, w\}$ with $v \in S$ and $w \in V \setminus S$
6: Add node w to S
7: Add edge e to T
8: **end while**



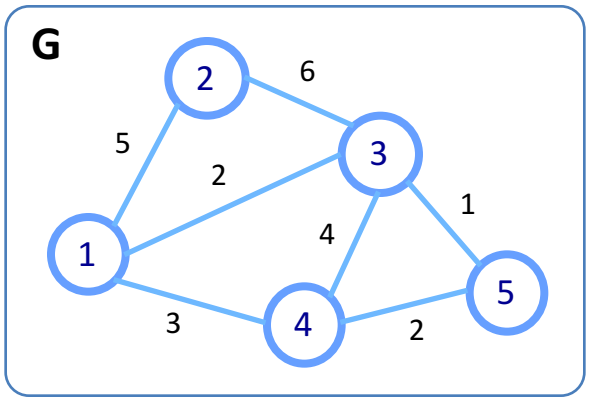
S = {1}
S = {1,3}
S = {1,3,5}



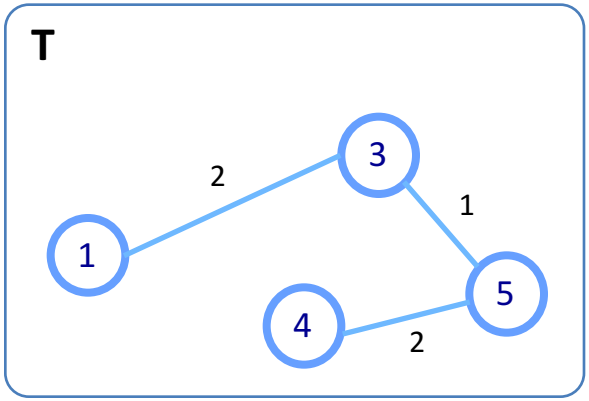
Graphen

Algorithm 1.11 MST Algorithm

1: Given a weighted graph $G = (V, E, \omega)$
2: Let $S = \{u\}$ be a set of visited nodes, initialized with any node $u \in V$
3: Let T be a tree just consisting of the single node $u \in S$, no edges
4: **while** $S \neq V$ **do**
5: Find minimum weight edge $e = \{v, w\}$ with $v \in S$ and $w \in V \setminus S$
6: Add node w to S
7: Add edge e to T
8: **end while**



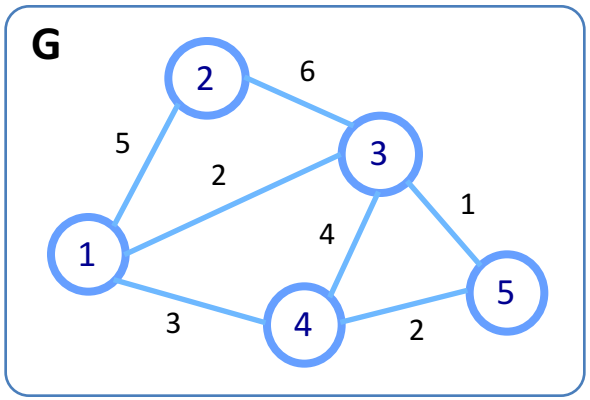
S = {1}
S = {1,3}
S = {1,3,5}
S = {1,3,5,4}



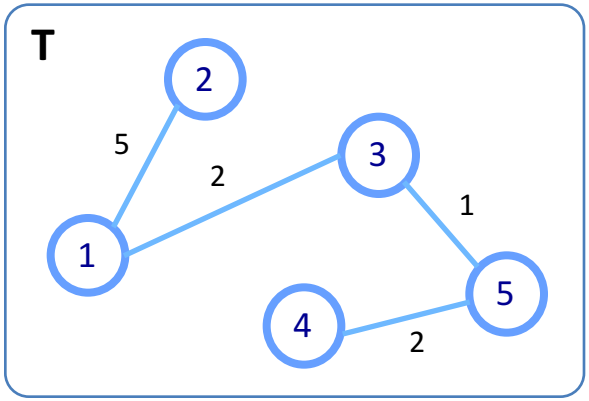
Graphen

Algorithm 1.11 MST Algorithm

1: Given a weighted graph $G = (V, E, \omega)$
2: Let $S = \{u\}$ be a set of visited nodes, initialized with any node $u \in V$
3: Let T be a tree just consisting of the single node $u \in S$, no edges
4: **while** $S \neq V$ **do**
5: Find minimum weight edge $e = \{v, w\}$ with $v \in S$ and $w \in V \setminus S$
6: Add node w to S
7: Add edge e to T
8: **end while**



S = {1}
S = {1,3}
S = {1,3,5}
S = {1,3,5,4}
S = {1,3,5,4,2}



Graphen

- **Shortest Path** → Weg zwischen 2 Knoten mit minimalem totalem Gewicht ω(P)
- **Distanz d(u,v)** → ω(P) wobei P der kürzeste Weg zwischen u & v ist

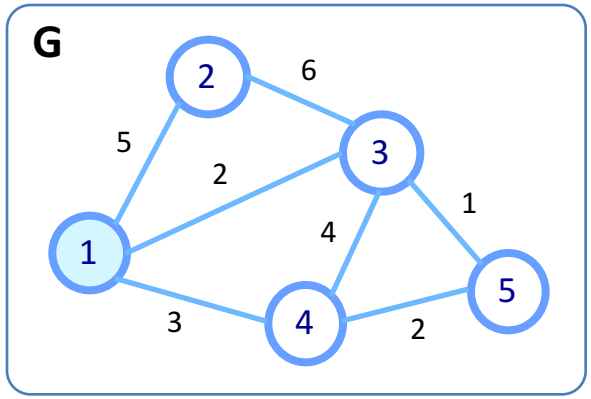
Algorithm 1.16 SPT Algorithm

1: Given a weighted graph $G = (V, E, \omega)$ and a node $r \in V$
2: Set a parent node $p_v = \text{null}$ for every node $v \in V$
3: Set $d_r = 0$ and $d_v = \infty$ for every node $r \neq v \in V$
4: Let $S = \{r\}$ be the set of visited nodes
5: **while** $S \neq V$ **do**
6: Find edge $e = \{v, w\}$ with $v \in S$ and $w \in V \setminus S$ with minimum $d_v + \omega(e)$
7: Set $p_w = v$
8: Set $d_w = d_v + \omega(e)$
9: $S = S \cup \{w\}$
10: **end while**

SPT Algorithmus

Algorithm 1.16 SPT Algorithm

1: Given a weighted graph $G = (V, E, \omega)$ and a node $r \in V$
2: Set a parent node $p_v = \text{null}$ for every node $v \in V$
3: Set $d_r = 0$ and $d_v = \infty$ for every node $r \neq v \in V$
4: Let $S = \{r\}$ be the set of visited nodes
5: **while** $S \neq V$ **do**
6: Find edge $e = \{v, w\}$ with $v \in S$ and $w \in V \setminus S$ with minimum $d_v + \omega(e)$
7: Set $p_w = v$
8: Set $d_w = d_v + \omega(e)$
9: $S = S \cup \{w\}$
10: **end while**



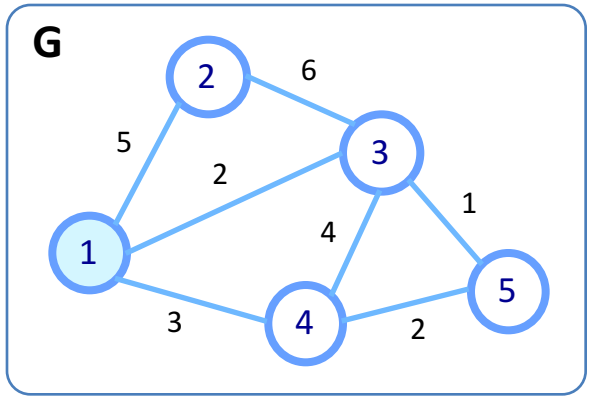
S = {1}

1: p = Null, d = 0
2: p = Null, d = inf
3: p = Null, d = inf
4: p = Null, d = inf
5: p = Null, d = inf

SPT Algorithmus

Algorithm 1.16 SPT Algorithm

1: Given a weighted graph $G = (V, E, \omega)$ and a node $r \in V$
2: Set a parent node $p_v = \text{null}$ for every node $v \in V$
3: Set $d_r = 0$ and $d_v = \infty$ for every node $r \neq v \in V$
4: Let $S = \{r\}$ be the set of visited nodes
5: **while** $S \neq V$ **do**
6: Find edge $e = \{v, w\}$ with $v \in S$ and $w \in V \setminus S$ with minimum $d_v + \omega(e)$
7: Set $p_w = v$
8: Set $d_w = d_v + \omega(e)$
9: $S = S \cup \{w\}$
10: **end while**



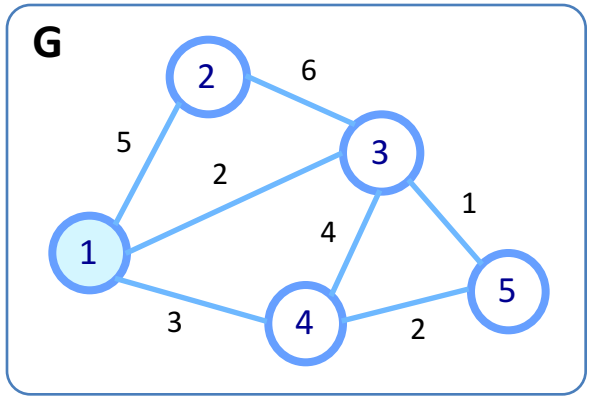
S = {1}
S = {1,3}

1: p = Null, d = 0
2: p = Null, d = inf
3: p = 1, d = 2
4: p = Null, d = inf
5: p = Null, d = inf

SPT Algorithmus

Algorithm 1.16 SPT Algorithm

1: Given a weighted graph $G = (V, E, \omega)$ and a node $r \in V$
2: Set a parent node $p_v = \text{null}$ for every node $v \in V$
3: Set $d_r = 0$ and $d_v = \infty$ for every node $r \neq v \in V$
4: Let $S = \{r\}$ be the set of visited nodes
5: **while** $S \neq V$ **do**
6: Find edge $e = \{v, w\}$ with $v \in S$ and $w \in V \setminus S$ with minimum $d_v + \omega(e)$
7: Set $p_w = v$
8: Set $d_w = d_v + \omega(e)$
9: $S = S \cup \{w\}$
10: **end while**



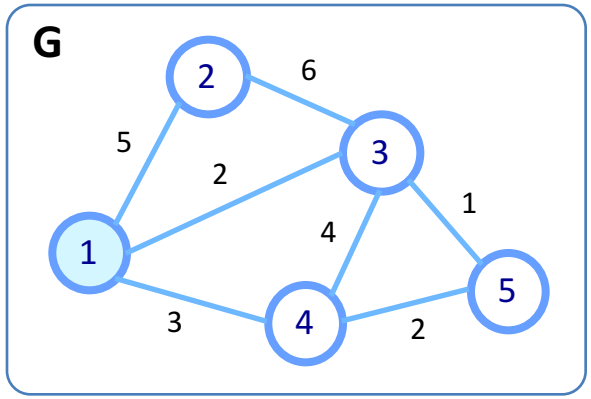
S = {1}
S = {1,3}
S = {1,3,5}

1: p = Null, d = 0
2: p = Null, d = inf
3: p = 1, d = 2
4: p = Null, d = inf
5: p = 3, d = 3

SPT Algorithmus

Algorithm 1.16 SPT Algorithm

1: Given a weighted graph $G = (V, E, \omega)$ and a node $r \in V$
2: Set a parent node $p_v = \text{null}$ for every node $v \in V$
3: Set $d_r = 0$ and $d_v = \infty$ for every node $r \neq v \in V$
4: Let $S = \{r\}$ be the set of visited nodes
5: **while** $S \neq V$ **do**
6: Find edge $e = \{v, w\}$ with $v \in S$ and $w \in V \setminus S$ with minimum $d_v + \omega(e)$
7: Set $p_w = v$
8: Set $d_w = d_v + \omega(e)$
9: $S = S \cup \{w\}$
10: **end while**



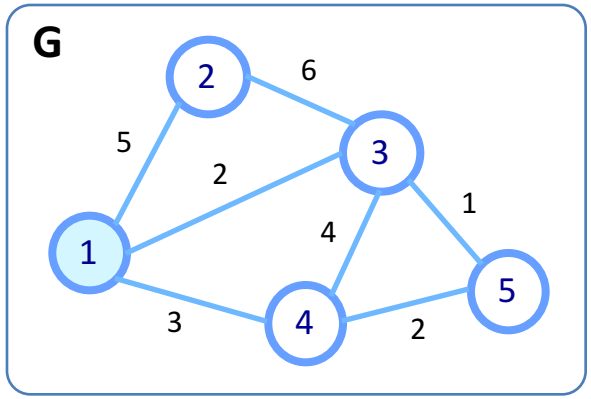
S = {1}
S = {1,3}
S = {1,3,5}
S = {1,3,5,4}

1: p = Null, d = 0
2: p = Null, d = inf
3: p = 1, d = 2
4: p = 1, d = 3
5: p = 3, d = 3

SPT Algorithmus

Algorithm 1.16 SPT Algorithm

1: Given a weighted graph $G = (V, E, \omega)$ and a node $r \in V$
2: Set a parent node $p_v = \text{null}$ for every node $v \in V$
3: Set $d_r = 0$ and $d_v = \infty$ for every node $r \neq v \in V$
4: Let $S = \{r\}$ be the set of visited nodes
5: **while** $S \neq V$ **do**
6: Find edge $e = \{v, w\}$ with $v \in S$ and $w \in V \setminus S$ with minimum $d_v + \omega(e)$
7: Set $p_w = v$
8: Set $d_w = d_v + \omega(e)$
9: $S = S \cup \{w\}$
10: **end while**



S = {1}
S = {1,3}
S = {1,3,5}
S = {1,3,5,4}
S = {1,3,5,4,2}

1: p = Null, d = 0
2: p = 1, d = 5
3: p = 1, d = 2
4: p = 1, d = 3
5: p = 3, d = 3

Network Layer

Adressen

- **IPv4** → 0.0.0.0 bis 255.255.255.255
  - 32 Bit
  - 4 Chunks mit je 8 Bits
  - Dezimale Angabe
- **Präfix** → 10.10.0.0/16
- **IPv6** → 0000:0000:0000:0000:0000:0000:0000:0000
  - 128 Bits
  - 8 Chunks mit je 16 Bits
  - Hexadezimale Angabe

Adressen

- **IPv6**
  - Pro „Chunk“ können die führenden Nullen weggelassen werden
  - Chunks aus lauter Nullen können durch ein weiteres : ersetzt werden (einmalig)
  - Jede IPv4 Adresse kann in IPv6 übersetzt werden
  - ab.cd.ef.gh → ::ffff:abcd:efgh

Network Layer

Packets

- **Header** → Information
  - Source und Destination
  - Version (v4 oder v6)
  - Länge Header
  - Länge Payload
  - **Time-to-live** (TTL) → 8 Bit
- **Payload** → Eigentliche Daten

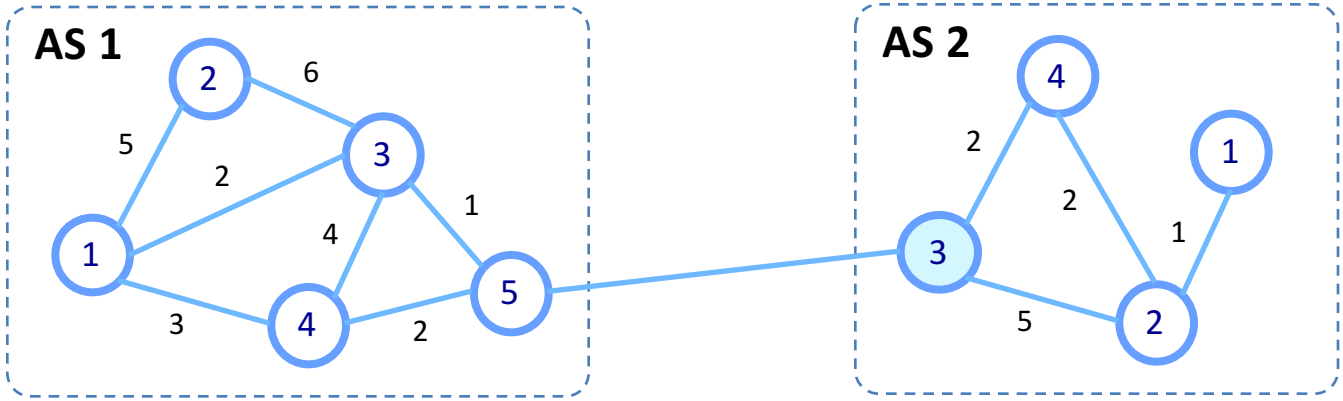
Routing

- **Routing Table** → Für jeden Knoten
  - Wohin schicken, um bestimmten Knoten zu erreichen
- **Autonomous System** (AS) → Menge von Knoten, die alle das ganze Netz kennen
  - Jedes AS hat eindeutige ASN
  - Teilen gemeinsamen Präfix
- **Link-State (LS) Routing Algorithmus** → Zum Beispiel OSPF
  - Nur für „kleine“ Netzwerke, da Knoten ganze Topologie kennen müssen (Intra-Domain)

Algorithm 1.26 Link-State (LS) Routing Algorithm.

1: Given a weighted graph $G = (V, E, \omega)$
2: Learn $\omega(e)$ for every edge $e \in E$
3: Compute shortest paths to between all nodes, e.g., by using Algorithm 1.16

Routing



1.0.0.0/8

2.0.0.0/8

2.0.0.1	→ 2.0.0.4
2.0.0.4	→ 2.0.0.4
2.0.0.2	→ 2.0.0.4
1.0.0.0/8	→ 1.0.0.5

Routing

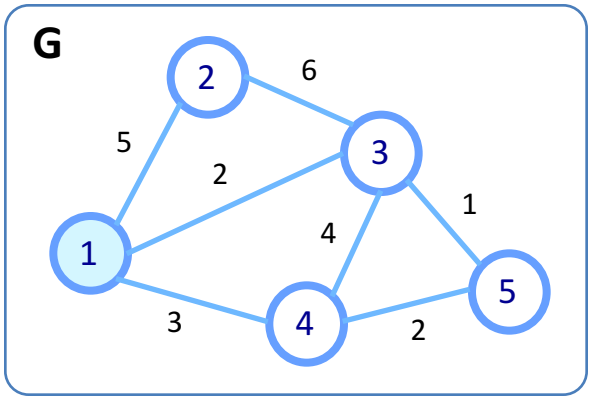
➢ Distance-Vector (DV) Routing Algorithmus

Algorithm 1.27 Distance-Vector (DV) Routing Algorithm.

1: Given a weighted graph $G = (V, E, \omega)$ and a node $u \in V$
2: Initialize a distance estimate $D(u \to v) = \omega(\{u, v\})$ for all neighbors $N(u)$ and $D(u \to w) = \infty$ for all other nodes
3: Send distance vector $\mathcal{D}(u) = \{D(u \to v) \mid v \in N(u)\}$ to all neighbors $N(u)$
4: **while** true **do**
5: Upon receiving a distance vector $\mathcal{D}(v)$ from a neighbor v, update the distance estimate to all destinations accordingly
6: **if** $D(u \to w)$ changed for any w **then**
7: Send the updated distance vector $\mathcal{D}(u)$ to all neighbors
8: **end if**
9: **end while**

Routing

- **Distance-Vector (DV) Routing Algorithmus**



D(1→2) = 5
D(1→3) = 2
D(1→4) = 3
D(1→5) = inf

} Alles in Vektor

Routing

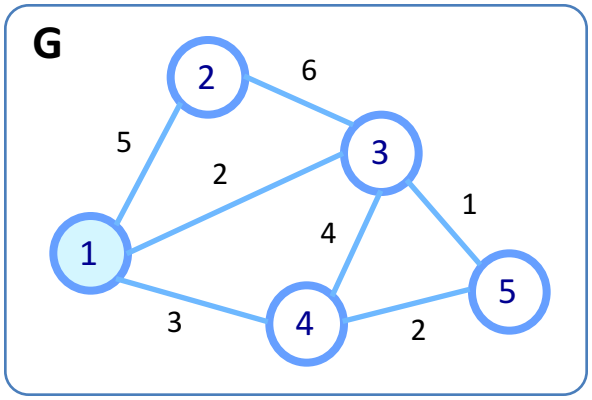
- **Distance-Vector (DV) Routing Algorithmus**

Algorithm 1.27 Distance-Vector (DV) Routing Algorithm.

1: Given a weighted graph $G = (V, E, \omega)$ and a node $u \in V$
2: Initialize a distance estimate $D(u \to v) = \omega(\{u, v\})$ for all neighbors $N(u)$ and $D(u \to w) = \infty$ for all other nodes
3: Send distance vector $\mathcal{D}(u) = \{D(u \to v) \mid v \in N(u)\}$ to all neighbors $N(u)$
4: **while** true **do**
5: Upon receiving a distance vector $\mathcal{D}(v)$ from a neighbor v, update the distance estimate to all destinations accordingly
6: **if** $D(u \to w)$ changed for any w **then**
7: Send the updated distance vector $\mathcal{D}(u)$ to all neighbors
8: **end if**
9: **end while**

Routing

- **Distance-Vector (DV) Routing Algorithmus**



D(1→2) = 5
D(1→3) = 2
D(1→4) = 3
D(1→5) = inf

Vektor bei 4

D(1→2) = 5
D(1→3) = 2
D(1→4) = 3
D(1→5) = 5

Vektor bei 3

D(1→2) = 5
D(1→3) = 2
D(1→4) = 3
D(1→5) = 3

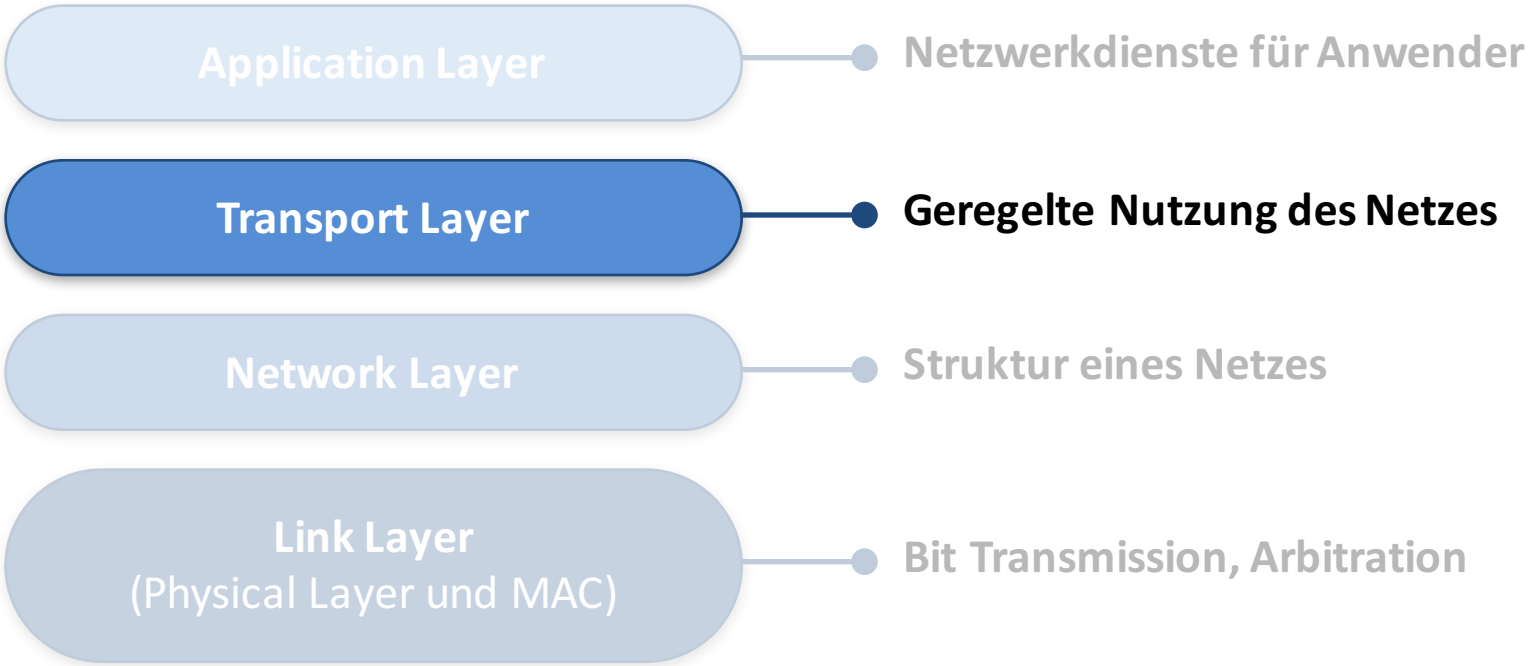
Routing

- **Intra-Domain** → Routing innerhalb eines AS
- **Inter-Domain** → Routing zwischen verschiedenen AS
- Unterschiedliche Protokolle verwendet
- **Border Gateway Protocol** (BGP) für inter-domain Routing

Empfohlene Übungen

- **Serie 1**
  - Aufgabe 1 (Quiz)
  - Aufgabe 4 (Advanced)

Transport Layer



Transport Layer

Inhalt

- Flows / Flüsse
- Linear Programming
- Fairness
- UDP & TCP

Flows / Flüsse

- **Source** s
- **Destination** t
- Gewichte der Kanten sind Kapazitäten → Maximal möglicher Fluss
- Capacity Constraint → Fluss muss auf allen Kanten Kapazität einhalten
- Flow Conservation → In-Flow = Out-Flow für jeden Knoten
- **F(e)** → Rate von F auf Kante e
- **F** → Rate von F, Netto-Outflow der Quelle
- **Multi-Commodity Flow** → $F = (F_1, \ldots, F_k)$
  - Capacity Constraint gilt nun für Summe der Flüsse

Linear Programming (LP)

- **Zweck** → Lösen von Optimierungsproblemen
- m **Ungleichungen** und eine **lineare Funktion f(x)**
- **Kanonische Form**

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &\leq b_2 \\ \vdots \quad\quad \vdots \quad\quad\quad\quad \vdots \quad\quad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &\leq b_m \end{aligned}$$

$$f(\mathbf{x}) = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

- Umformen in kanonische Form

  ax = b → ax ≤ b & ax ≥ b

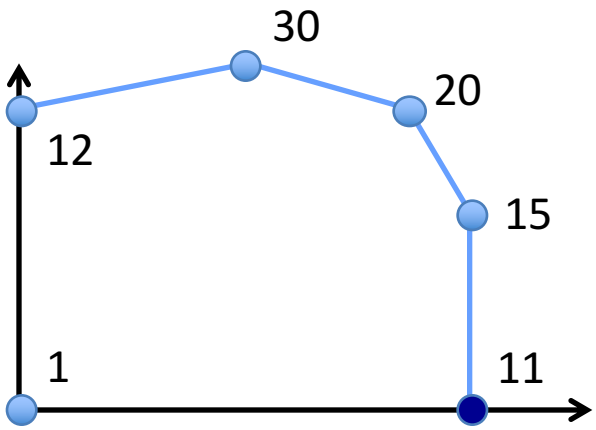
  ax ≥ b → -ax ≤ -b

Linear Programming (LP)

- **Simplex Algorithmus** → Lösen eines LP-Problems

Algorithm 2.6 Simplex Algorithm

1: choose a vertex $\mathbf{x}$ of the polytope
2: **while** there is a neighboring vertex $\mathbf{y}$ such that $f(\mathbf{y}) > f(\mathbf{x})$ **do**
3: $\quad \mathbf{x} := \mathbf{y}$
4: **end while**
5: **return** $\mathbf{x}$

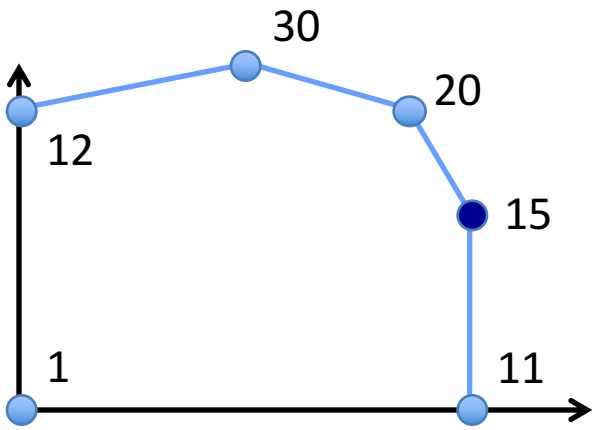


Linear Programming (LP)

- **Simplex Algorithmus** → Lösen eines LP-Problems

Algorithm 2.6 Simplex Algorithm

```
1: choose a vertex x of the polytope
2: while there is a neighboring vertex y such that f(y) > f(x) do
3:     x := y
4: end while
5: return x
```

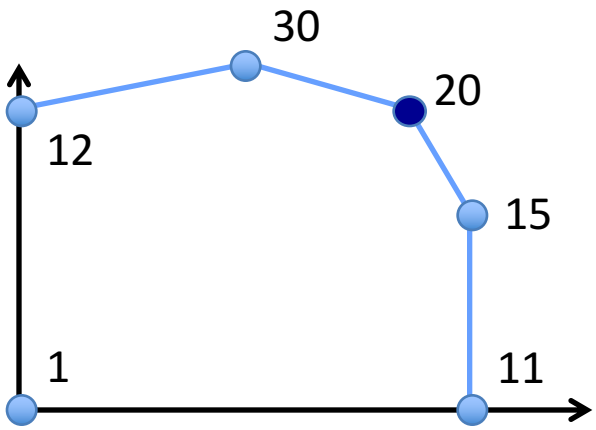


Linear Programming (LP)

- **Simplex Algorithmus** → Lösen eines LP-Problems

Algorithm 2.6 Simplex Algorithm

1: choose a vertex $\mathbf{x}$ of the polytope
2: **while** there is a neighboring vertex $\mathbf{y}$ such that $f(\mathbf{y}) > f(\mathbf{x})$ **do**
3: $\quad \mathbf{x} := \mathbf{y}$
4: **end while**
5: **return** $\mathbf{x}$

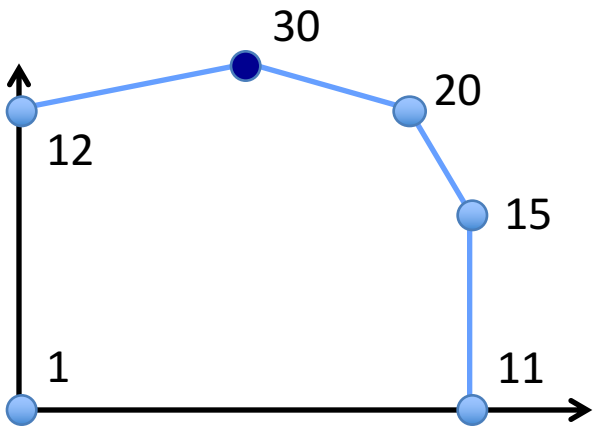


Linear Programming (LP)

- **Simplex Algorithmus** → Lösen eines LP-Problems

Algorithm 2.6 Simplex Algorithm

1: choose a vertex $\mathbf{x}$ of the polytope
2: **while** there is a neighboring vertex $\mathbf{y}$ such that $f(\mathbf{y}) > f(\mathbf{x})$ **do**
3: $\quad \mathbf{x} := \mathbf{y}$
4: **end while**
5: **return** $\mathbf{x}$



Linear Programming (LP)

- **LP für maximalen Single-Commodity Fluss**

Maximize $f(\mathbf{x}) = \sum_{e \in \text{out}(s)} x_e$
subject to

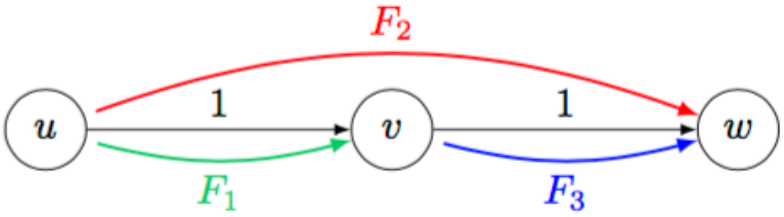
1. $x_e \geq 0$ for all $e \in E$
2. $x_e \leq c(e)$ for all $e \in E$
3. $\sum_{e \in \text{in}(v)} x_e = \sum_{e \in \text{out}(v)} x_e$ for all $v \in V \setminus \{s, t\}$
4. $\sum_{e \in \text{in}(s)} x_e = 0$

- **Unsplittable Flow** → Fluss formt einen einzelnen Pfad von s nach t

Fairness

- **Demand d_i** → Gewünschte Rate mit welcher F_i übertragen möchte

  - Fluss ist maximal so gross wie der Demand



Fairness

- **Max-Min-Fairness** → Eindeutig

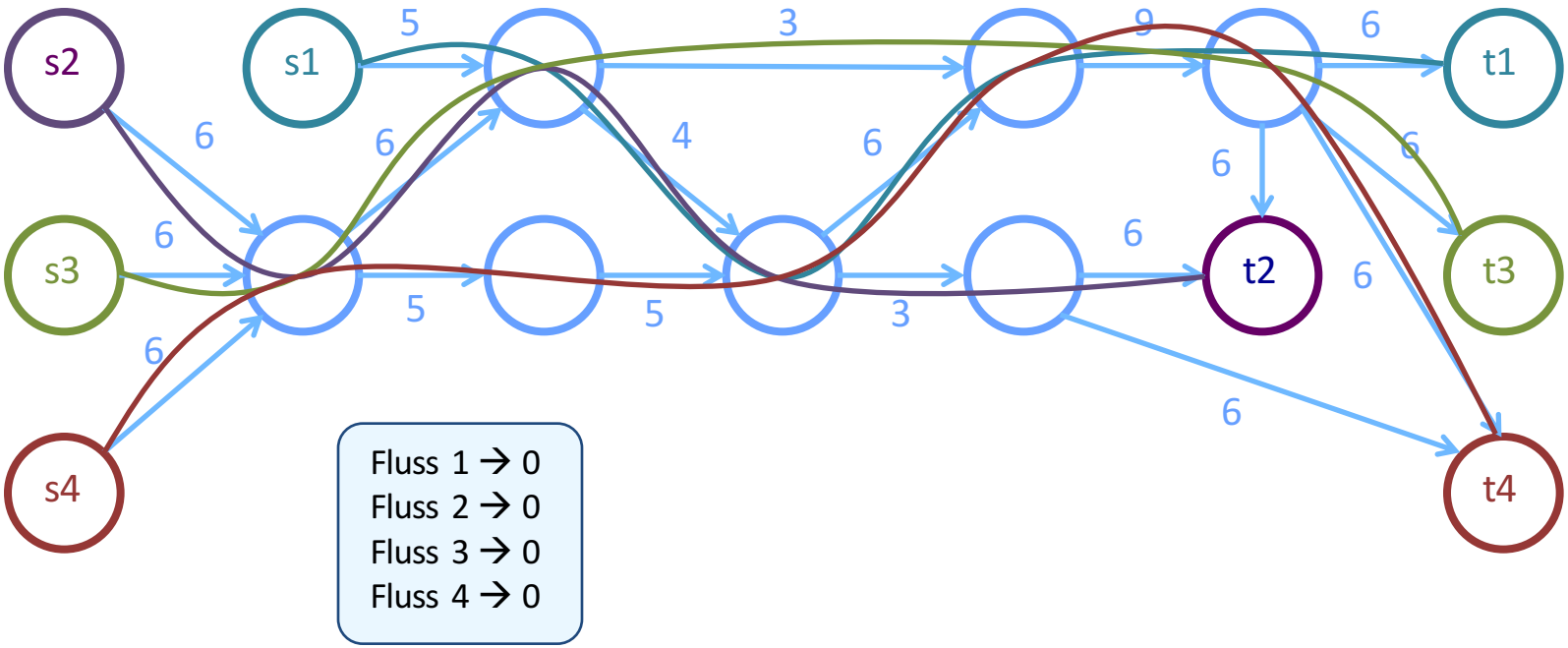
  - Bandbreite so verteilt, dass weitere Erhöhung eines Flusses, anderen Fluss von gleicher oder kleinerer Grösse vermindern würde

Algorithm 2.12 Max-Min-Fair Allocation

Given a graph G, a set $\mathcal{F} = \{F_1, \ldots, F_k\}$ of flows with initial rate 0 on all edges, paths $p_1, \ldots, p_k$ along which the respective flows are to be routed and demands $d_1, \ldots, d_k$
while $\mathcal{F} \neq \emptyset$ **do**
  repeat
    increase rate of all flows in $\mathcal{F}$ evenly, but at most up to the respective demands
  until there is an edge $e \in E$ such that $\sum_{i:e \in p_i} F_i = c(e)$
  for all such edges e **do**
    for all i such that $e \in p_i$ **do**
      $\mathcal{F} := \mathcal{F} \setminus \{F_i\}$
    end for
    $E := E \setminus \{e\}$
  end for
end while

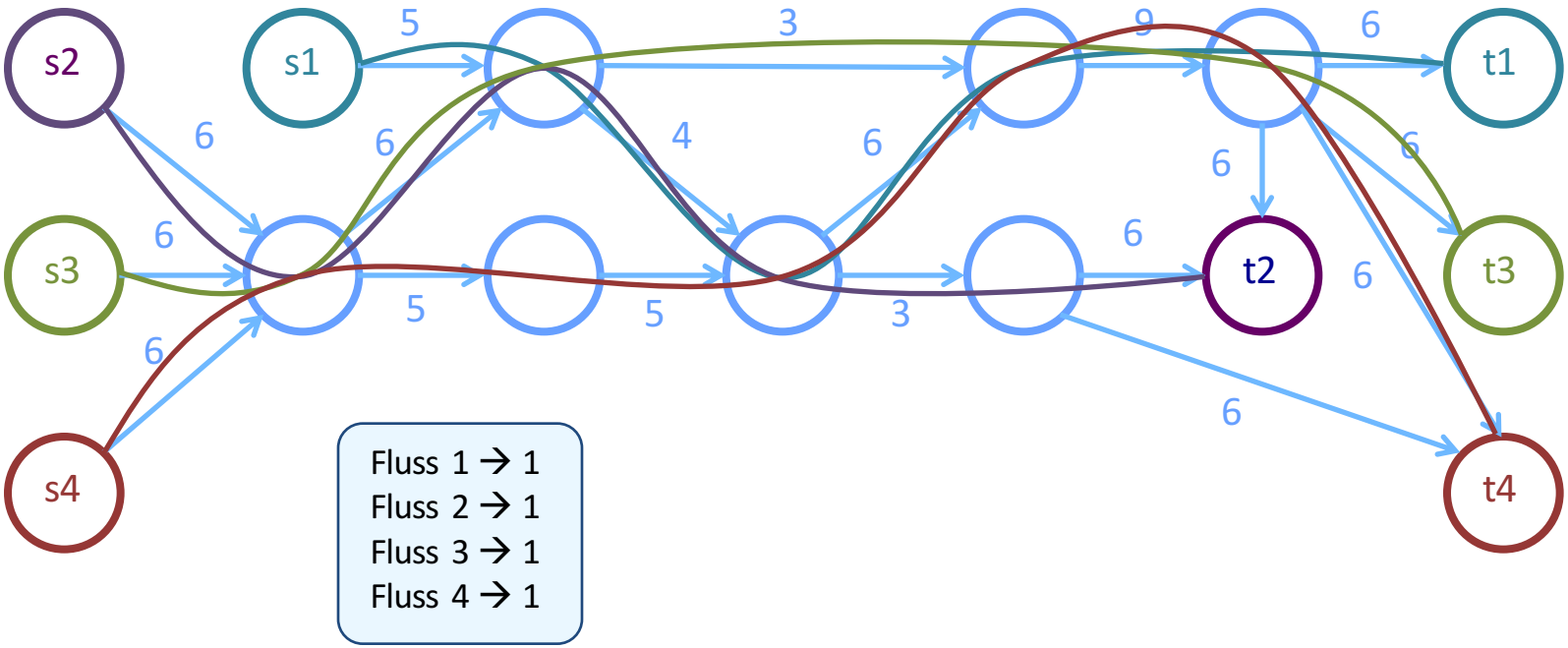
Fairness

- **Max-Min-Fairness**



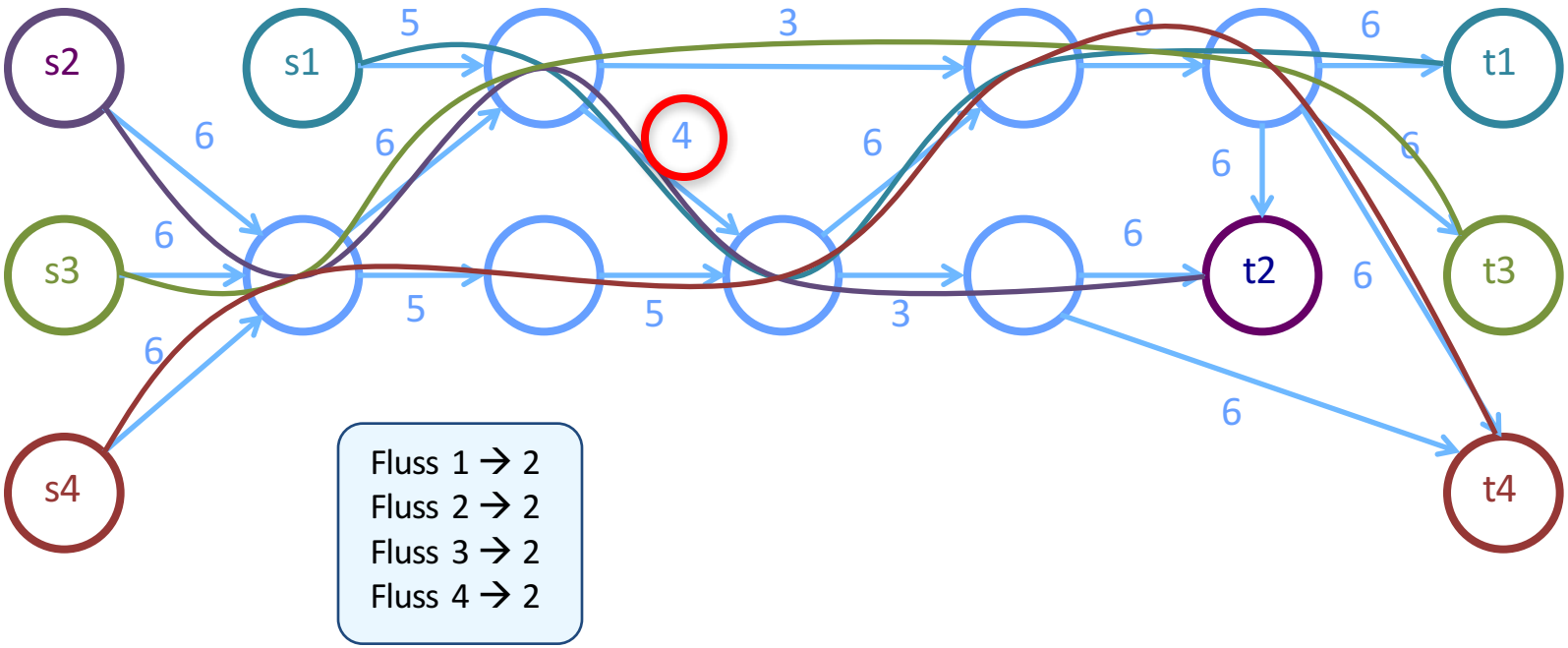
Fairness

- **Max-Min-Fairness**



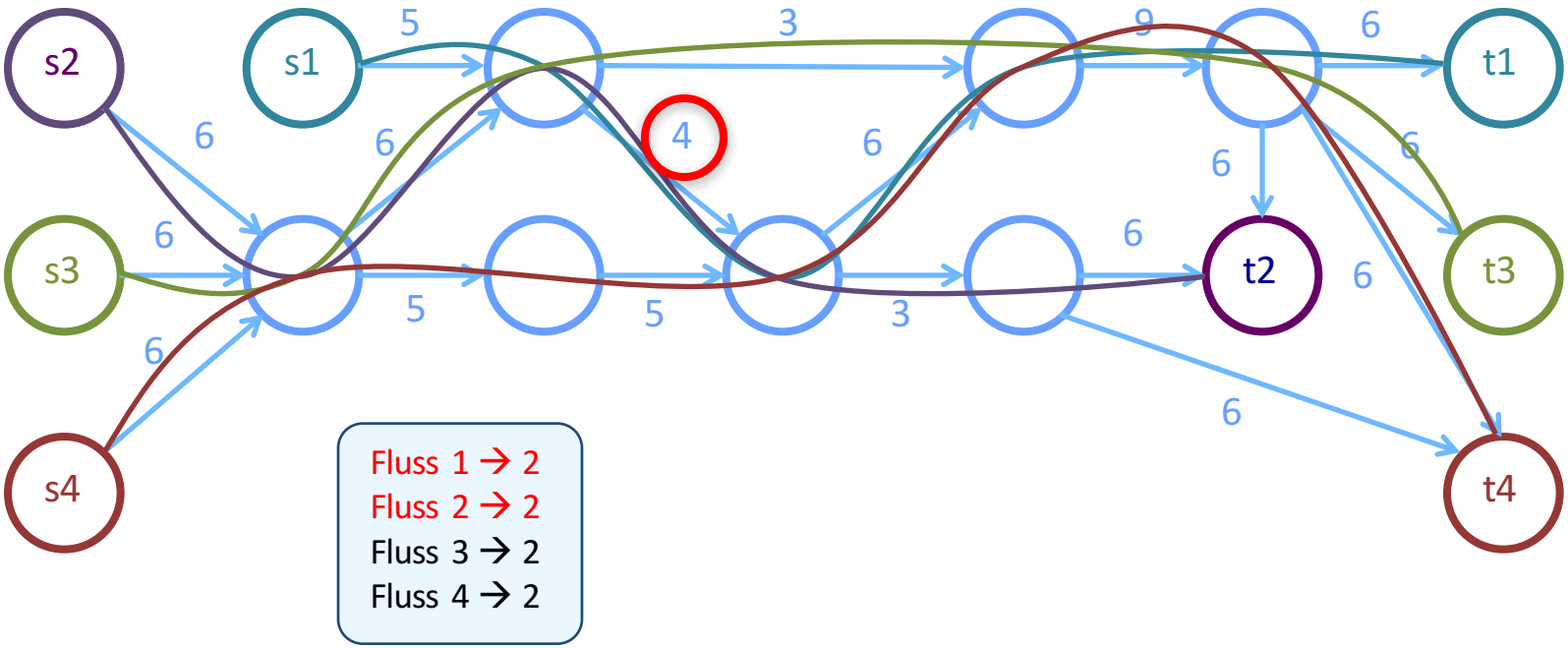
Fairness

- **Max-Min-Fairness**



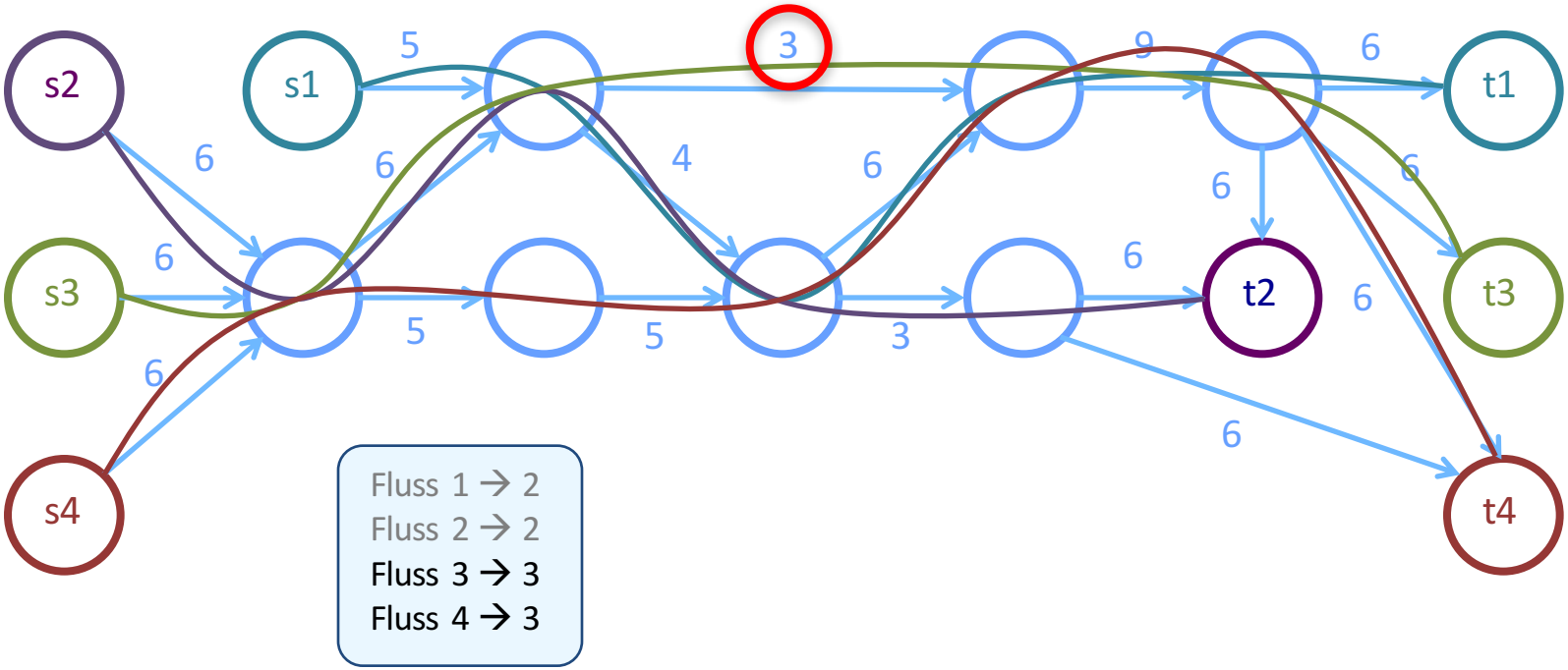
Fairness

- **Max-Min-Fairness**



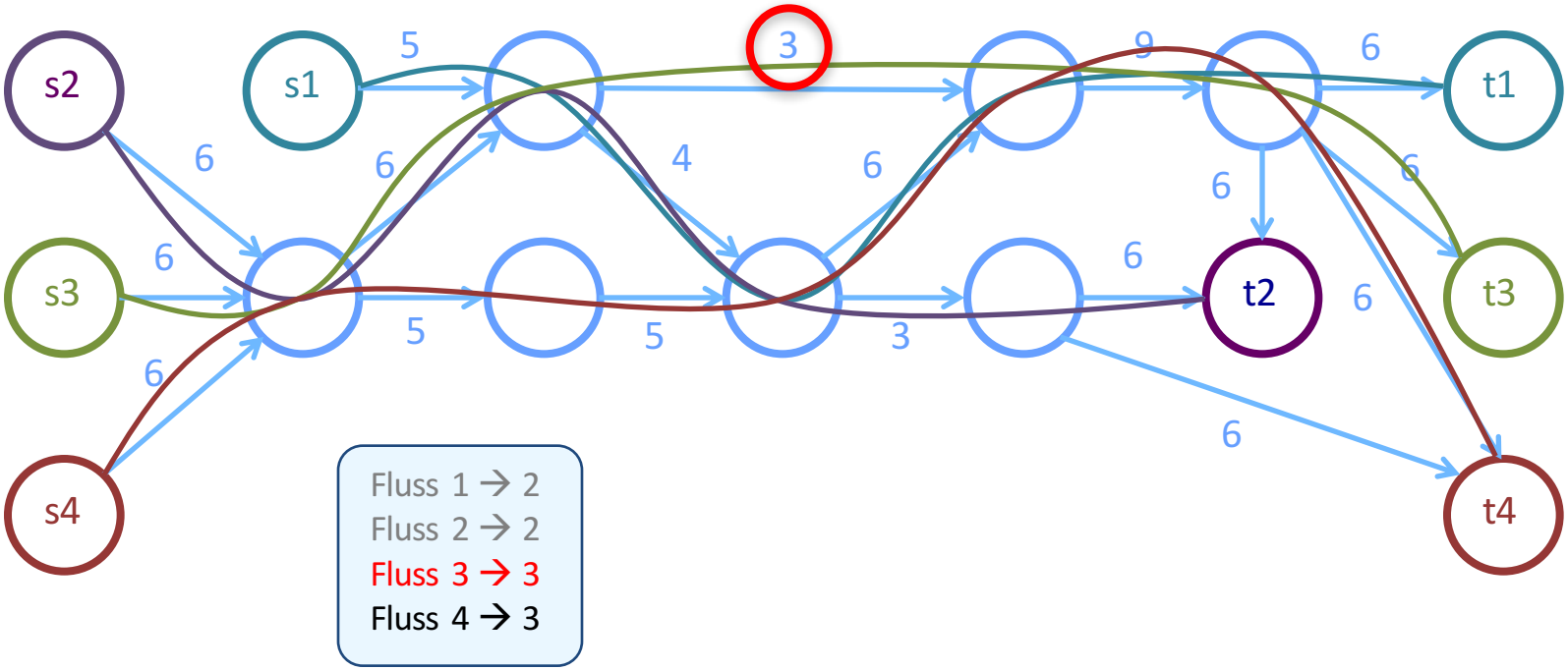
Fairness

- **Max-Min-Fairness**



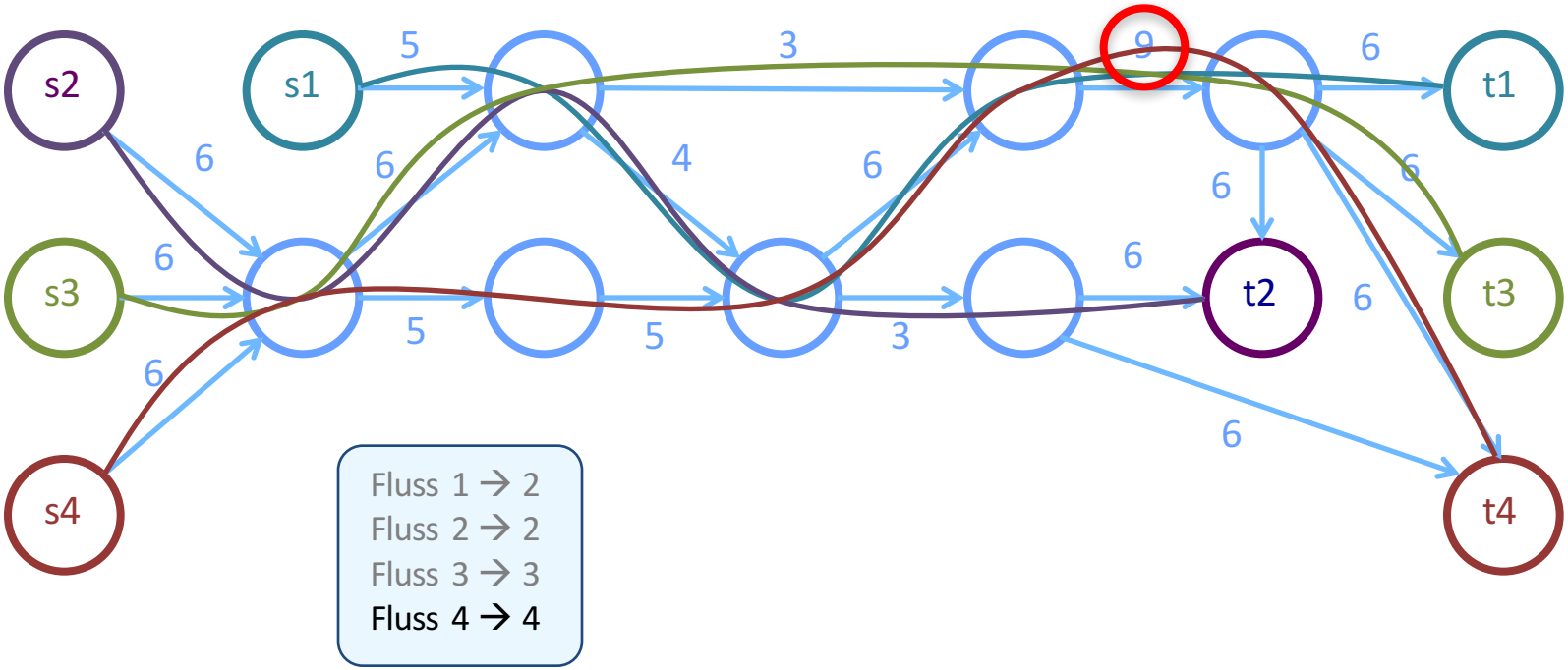
Fairness

- **Max-Min-Fairness**



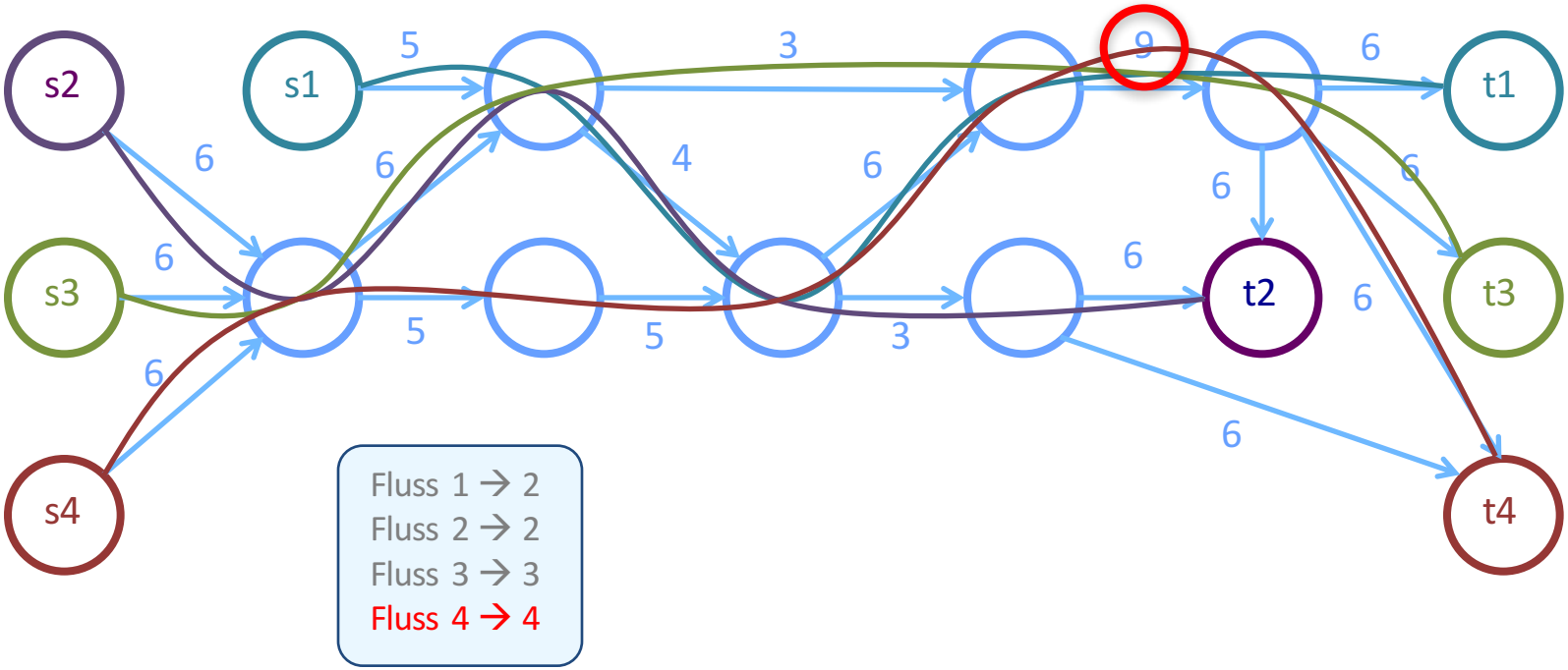
Fairness

- **Max-Min-Fairness**



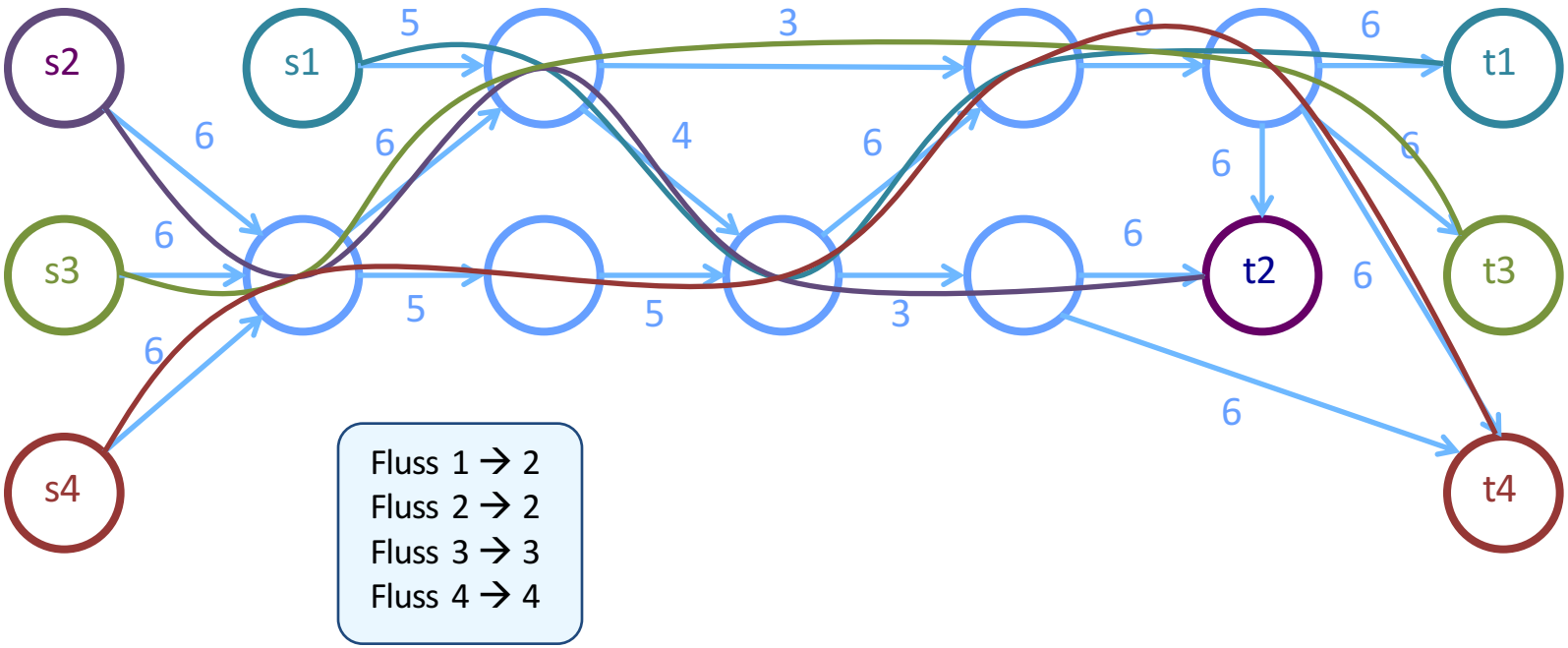
Fairness

- **Max-Min-Fairness**



Fairness

- **Max-Min-Fairness**



User Datagram Protocol (UDP)

- **UDP** → Transport-Protokoll, welches Pakete von Client zu Server sendet
  - Keine Garantie für Paketverluste
  - Keine Garantie für korrekte Reihenfolge der Pakete
- **Client** → Sender, Konsument
- **Server** → Empfänger, Dienstleister

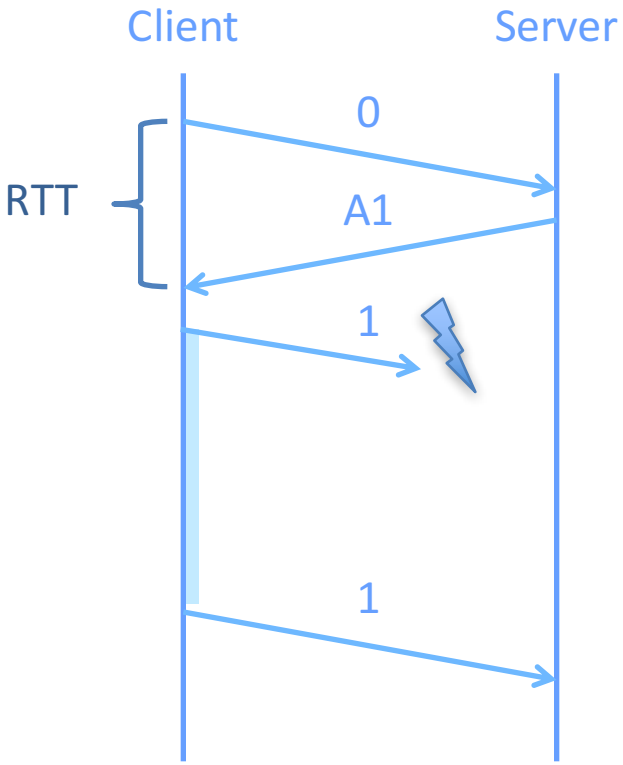
Transmission Control Protocol (TCP)

- **Connection-Oriented**
  - Garantiert richtige Reihenfolge und Retransmission
  - Connection = Bidirektionale Langzeit-Verbindung zwischen Client und Server
- **Segments** → TCP packets
- **Acknowledgement (ACK)** → Bestätigung für Empfang von Segment
  - ACK hat immer Nummer des als nächstes erwarteten Segments
- **Sequenznummer** → Jedes Segment wird nummeriert
- **Round-Trip Time (RTT)** → Zeit von Absenden des Segments bis Erhalt des ACK

Transmission Control Protocol (TCP)

- **Flow Control** → Verhindere Überlastung des Empfängers
  - Server teilt dies dem Client im Header mit
  - Client passt Senderate dementsprechend an
- **Congestion Control** → Verhindere Überlastung des Mediums (Routers)
  - Congestion window
  - Am Anfang **Slow Start** → Exponentielles Wachstum bis zu gewissem Threshold
  - **AIMD** → Ab Threshold
  - Verlorene Pakete werden durch Timeouts erkannt

Transmission Control Protocol (TCP)



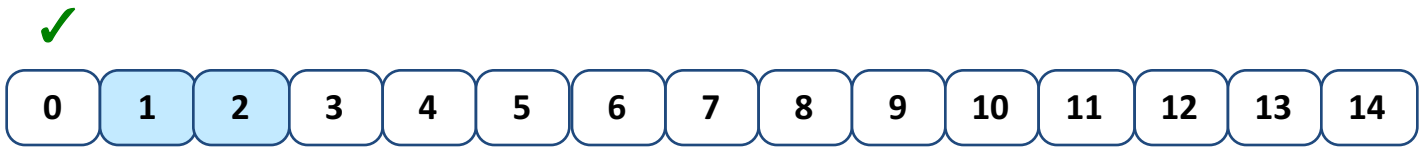
Transmission Control Protocol (TCP)

- Slow Start

- AIMD (Window Size = 3)

Transmission Control Protocol (TCP)

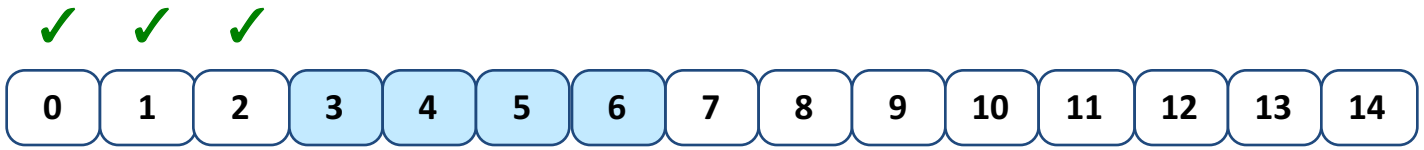
- Slow Start



- AIMD (Window Size = 3)

Transmission Control Protocol (TCP)

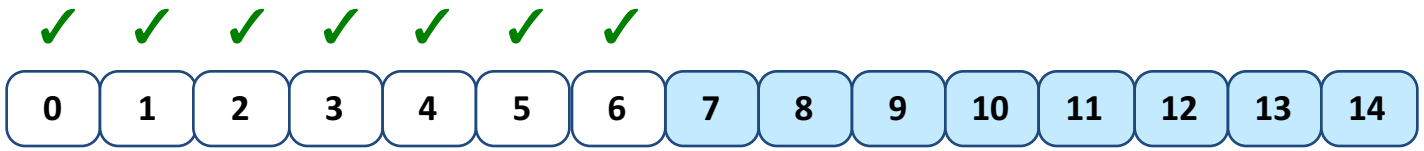
- Slow Start



- AIMD (Window Size = 3)

Transmission Control Protocol (TCP)

- Slow Start



- AIMD (Window Size = 3)

Transmission Control Protocol (TCP)

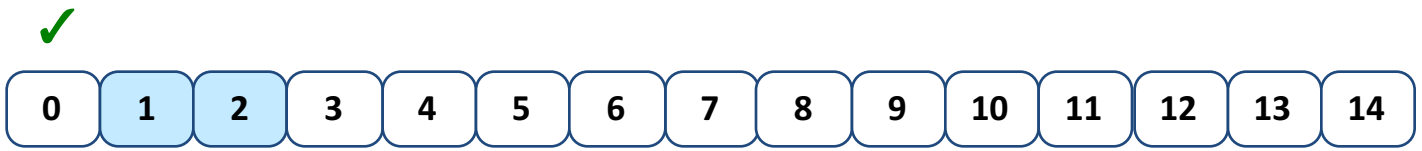
- Slow Start

- AIMD (Window Size = 3)

Transmission Control Protocol (TCP)

- Slow Start

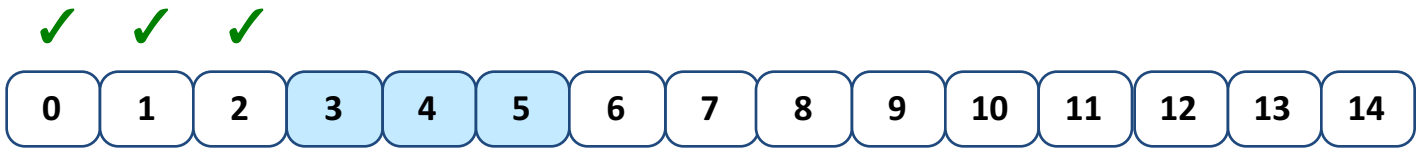
- AIMD (Window Size = 3)



Transmission Control Protocol (TCP)

- Slow Start

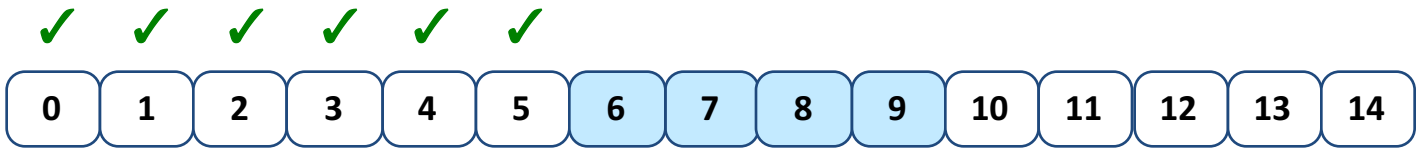
- AIMD (Window Size = 3)



Transmission Control Protocol (TCP)

- Slow Start

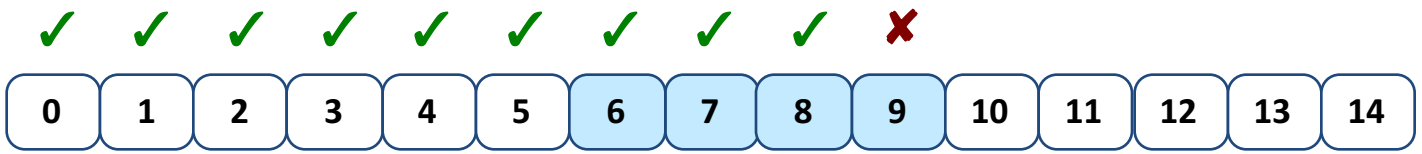
- AIMD (Window Size = 3)



Transmission Control Protocol (TCP)

- Slow Start

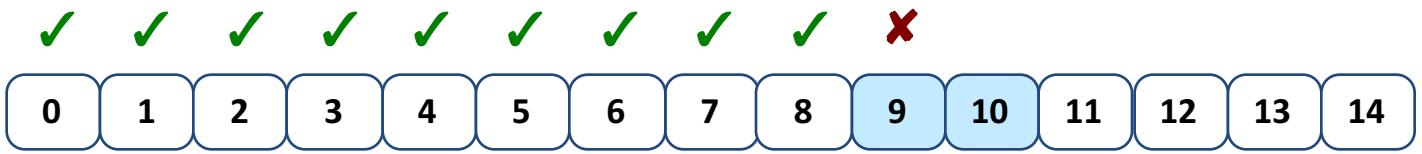
- AIMD (Window Size = 3)



Transmission Control Protocol (TCP)

- Slow Start

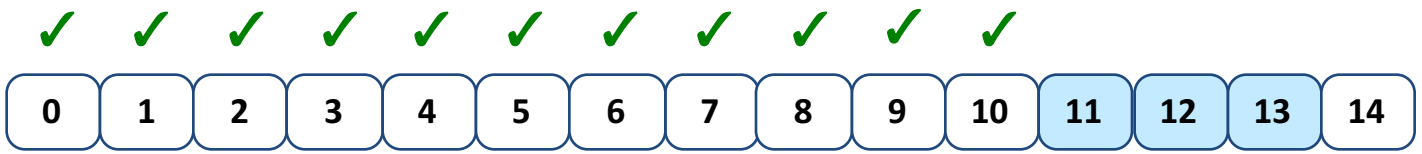
- AIMD (Window Size = 3)



Transmission Control Protocol (TCP)

- Slow Start

- AIMD (Window Size = 3)



Empfohlene Übungen

- **Serie 2**
  - Aufgabe 1 (Quiz)
  - Aufgabe 2 (Basic)
  - Aufgabe 3 (Advanced)
  - Aufgabe 4 (Advanced)

Application Layer

Application Layer — **Netzwerkdienste für Anwender**

Transport Layer — **Geregelte Nutzung des Netzes**

Network Layer — **Struktur eines Netzes**

Link Layer
(Physical Layer und MAC) — **Bit Transmission, Arbitration**

Application Layer

Inhalt

- Hypertext Transport Protocol (HTTP)
- Hypertext Markup Language (HTML)
- Domain Name System (DNS)
- Mailserver

Application Layer

Hypertext Transport Protocol (HTTP)

- Netzwerk-Protokoll auf Application Layer
  - Interaktion mit Ressourcen auf Server
  - Request (Method) & Response
- **Uniform Resource Locator (URL)** → String, der Ressource auf Server identifiziert
  - Verwendetes Protokoll: **HTTP** = Unencrypted, **HTTPS** → Encrypted
  - Host Identifier: Server IP oder Domain Name
  - Path zur Location auf dem Server
- Beispiel: https://moodle-app2.let.ethz.ch/course/view.php?id=2040

Application Layer

Hypertext Transport Protocol (HTTP)

- Mögliche Methoden
  - GET → Ressourcen holen
  - PUT / POST → Ressource verändern / erstellen
  - DELETE → Ressource löschen
- Header und Payload (Content Length) → Getrennt durch Leerzeile (\n\n)
- **Return Codes** → Information von Server an Client
  - 2xx: Success
  - 4xx: Client Error
  - 5xx: Server Error

Hypertext Transport Protocol (HTTP)

```
# telnet pc-10367.ethz.ch 80
Trying 82.130.102.226...
Connected to pc-10367.ethz.ch.
PUT / HTTP/1.1
Host: pc-10367.ethz.ch
Content-Length: 11

hello world
HTTP/1.1 201 Created
Transfer-Encoding: chunked
Date: Wed, 09 Mar 2016 15:30:37 GMT
Content-Type: text/html
Server: TwistedWeb/15.5.0

18
Comment added with ID 3

0
```

Hypertext Markup Language (HTML)

- Verwendung → Erstellen von strukturierten Dokumenten
- **HTML-Element** → Bestandteil eines HTML-Dokuments

  - Umschlossen von Tags: < ... >

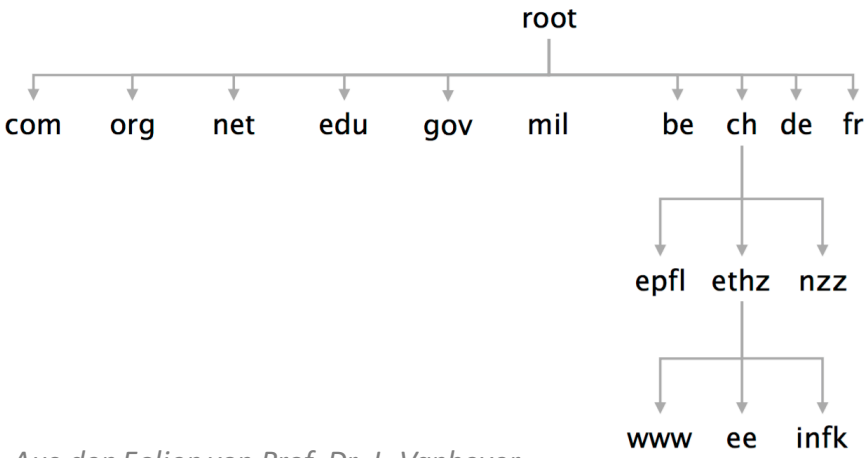
```
<html>
   <head>
      <title>This is a title</title>
   </head>
   <body>
      <p><strong>Hello</strong> world!</p>
   </body>
</html>
```

Domain Name System (DNS)

- Übersetzt (**Resolve**) Domain-Namen in IP Adressen der Server
- **Nameserver** → Server im Domain Name System
  - Client sendet Request an Nameserver
  - Nameserver übersetzt und antwortet Client
  - Kein Nameserver enthält sämtliche IP Adressen
  - Hierarchische Organisation
- **Authoritative Nameserver** → Verantwortlich für Domain Name
- **Root Nameserver** → Kontaktiert, falls kein besserer bekannt
  - Weltweit 13 Root Nameserver
  - Können nicht immer kontaktier werden → Flaschenhals

Domain Name System (DNS)

- **Regional Nameserver** → Erste Stufe unter den Root Nameserver
  - Verantwortlich für Top Level Domain
- Clients und Nameserver können Responses cachen
  - Responses enthalten **Time-To-Live (TTL)**



Aus den Folien von Prof. Dr. L. Vanbever

Application Layer

Mailserver

- Transferiert Nachrichten von Sender zu Empfänger
- Speichert eingehende Nachrichten für User → **Spooled Messages**
- **Username** und **Domain** → Getrennt durch @
- Senden von User zu Mailserver mit SMTP
  - User kontaktiert outgoing Mailserver
  - Outgoing Mailserver kontaktiert Ziel-Mailserver
- **Simple Mail Transfer Protocol (SMTP)**
  - HELO
  - RCPT TO / MAIL FROM
  - DATA

Application Layer

Mailserver

- **Post Office Protocol (POP)** → Erlaubt Client Zugang zu spooled Mails
  - Sehr ähnlich zu SMTP
  - STAT
  - LIST
  - RETR
  - DELE
- **Internet Message Access Protocol (IMAP)** → Alternatives Protokoll zu POP

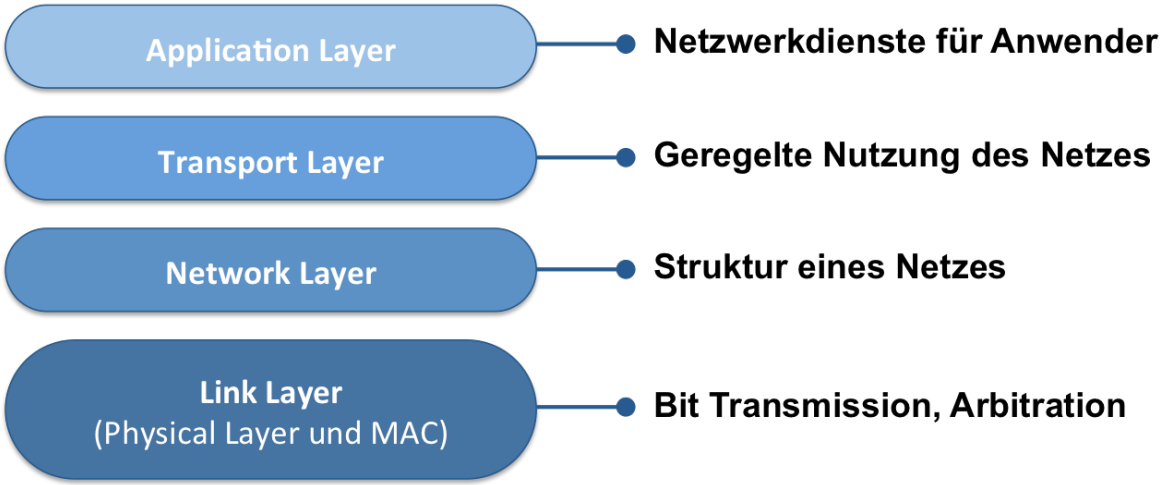
Application Layer

Empfohlene Übungen

- **Serie 3** → Protokolle repetieren
  - Aufgabe 2 (Basic)
  - Aufgabe 3 (Basic)

Kommunikation Überblick

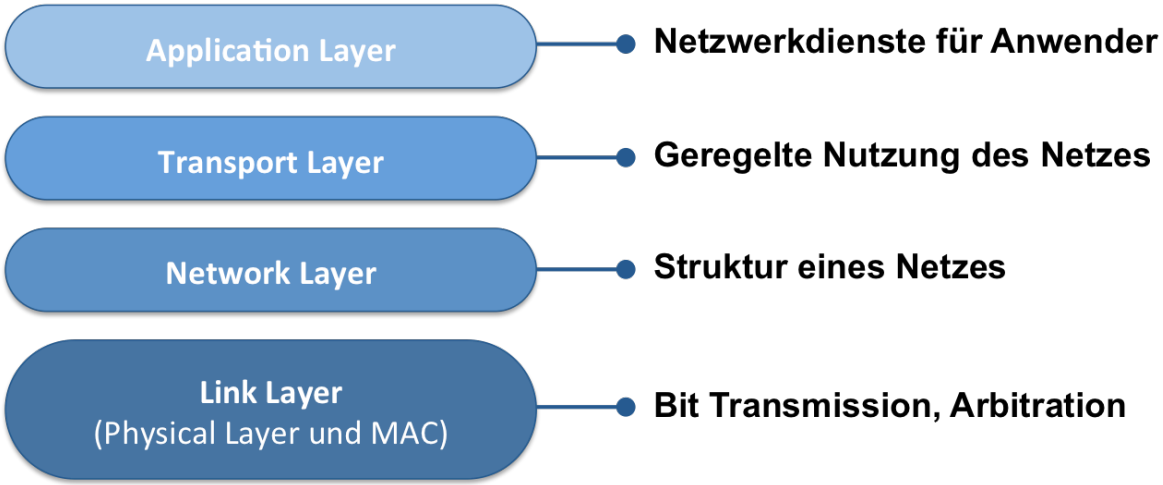
Übersicht Protokoll-Layers



Übertragene Daten

Kommunikation Überblick

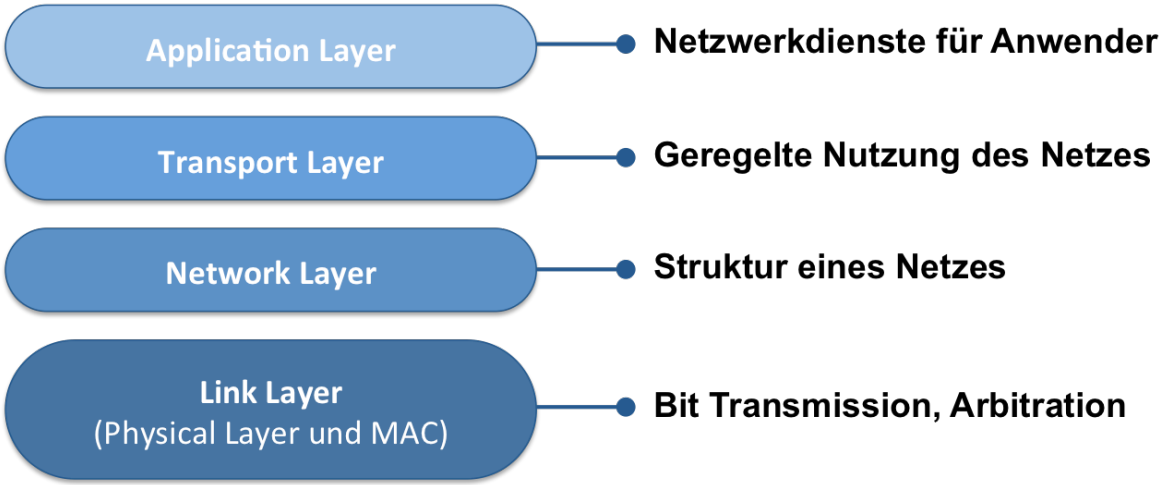
Übersicht Protokoll-Layers



Übertragene Daten

Kommunikation Überblick

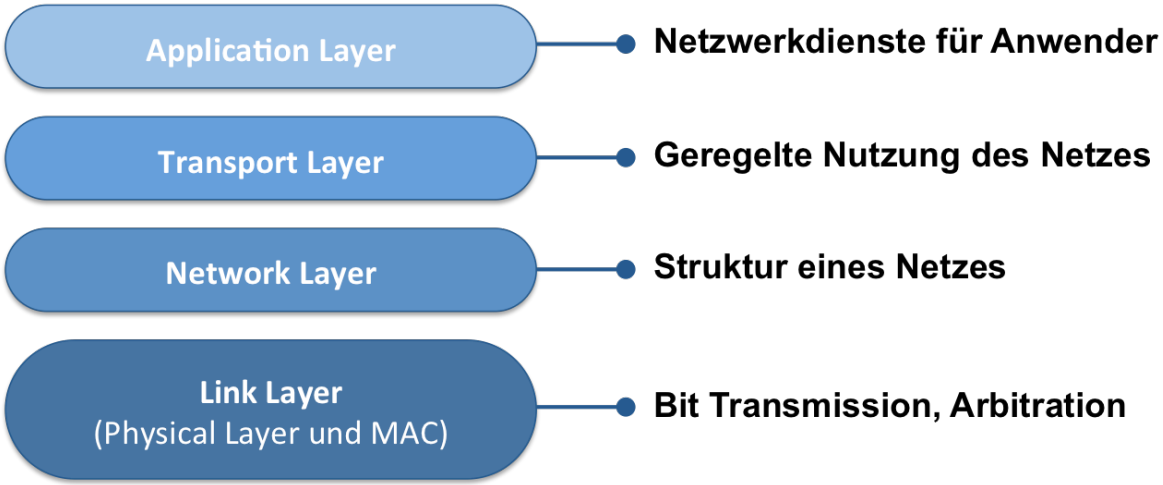
Übersicht Protokoll-Layers



Übertragene Daten

Kommunikation Überblick

Übersicht Protokoll-Layers



Übertragene Daten

Markov-Ketten

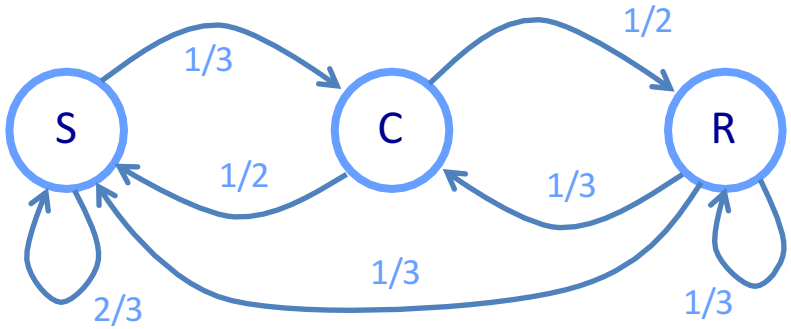
Inhalt

- Grundlagen
- Transitions
- Stationary Distributions
- Periodizität
- Page Rank Algorithm

Markov-Ketten

Grundlagen

- Abgeschlossener Menge S von Zuständen
- Wahrscheinlichkeitsverteilung q_t (Vektor) zum Zeitpunkt t
  - Zeilensumme immer 1
  - Initial Distribution q_0
- **Markov Property**: q_{t+1} hängt nur von q_t ab

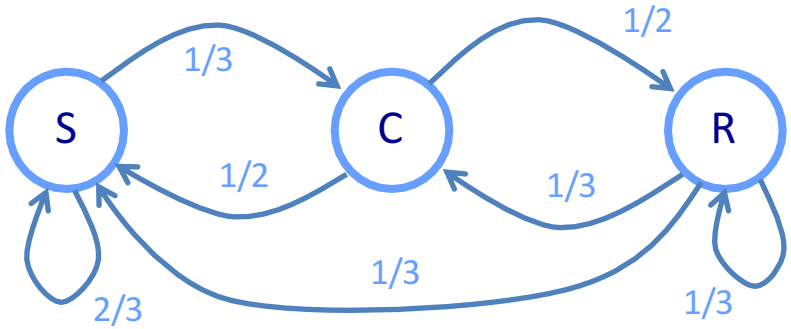


$q_0 = (1, 0, 0)$

Markov-Ketten

Grundlagen

- Abgeschlossener Menge S von Zuständen
- Wahrscheinlichkeitsverteilung q_t (Vektor) zum Zeitpunkt t
  - Zeilensumme immer 1
  - Initial Distribution q_0
- **Markov Property**: q_{t+1} hängt nur von q_t ab

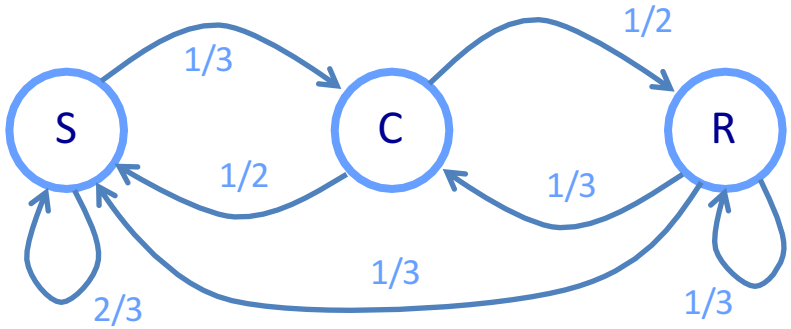


$q_1 = (2/3, 1/3, 0)$

Markov-Ketten

Grundlagen

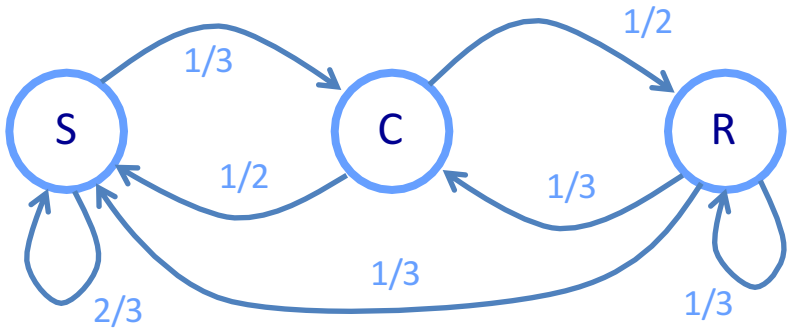
- Abgeschlossener Menge S von Zuständen
- Wahrscheinlichkeitsverteilung q_t (Vektor) zum Zeitpunkt t
  - Zeilensumme immer 1
  - Initial Distribution q_0
- **Markov Property**: q_{t+1} hängt nur von q_t ab



$q_2 = (11/18, 4/18, 3/18)$

Grundlagen

- **Transition Matrix P**
  - Zeilen = Startknoten
  - Spalten = Endknoten
- Eintrag $P^t_{i,j}$ → Wahrscheinlichkeit, von i in t Schritten j zu erreichen



$$P = \begin{pmatrix} 2/3 & 1/3 & 0 \\ 1/2 & 0 & 1/2 \\ 1/3 & 1/3 & 1/3 \end{pmatrix}$$

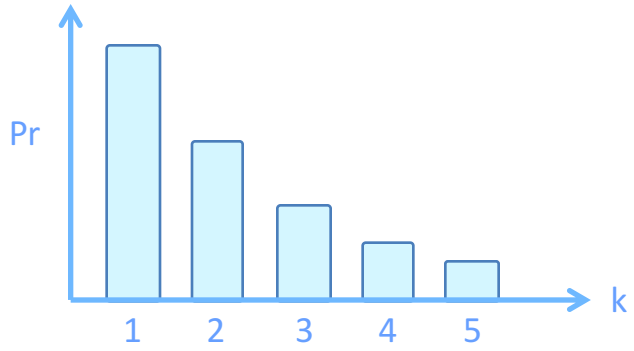
$$q_{t+1} = q_t \cdot P$$
$$q_t = q_0 \cdot P^t$$

Transitions / Übergänge

- **Soujourn Time T_i** → Wie lange bleibt man in Zustand i (Anzahl Zyklen)

  - Geometrische Verteilung

$$\Pr[T_i = k] = p_{i,i}^{k-1} \cdot (1 - p_{i,i})$$



- **Arrival Probability $f_{i,j}$** → Wahrscheinlichkeit, dass wir in einem Zustand enden

  - Nach langer Zeit

  - System linearer Gleichungen

$$f_{i,j} = \Pr[T_{i,j} < \infty] = p_{i,j} + \sum_{k \neq j} p_{i,k} \cdot f_{k,j}$$

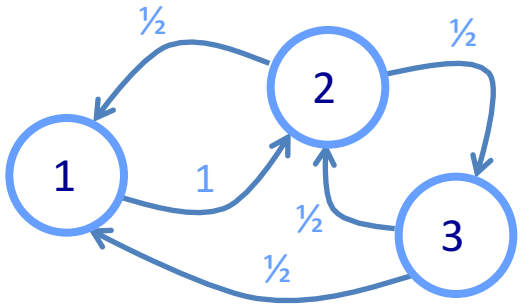
Transitions / Übergänge

- **Hitting Time $h_{i,j}$** → Wie viele Schritte bis wir von i nach j gelangen
  - System linearer Gleichungen
  - Prüfungsfrage: Wie oft geschieht ein gewisses Ereignis?
- **Commute Time $c_{i,j}$** → Anzahl Schritte von i nach j und zurück

$$h_{i,j} = E[T_{i,j}] = 1 + \sum_{k \neq j} p_{i,k} \cdot h_{k,j}$$

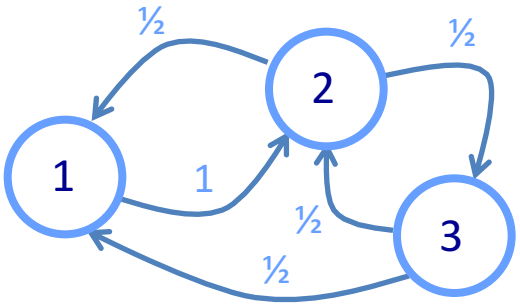
$h_{1,1} = 1 + p_{1,2}h_{2,1}$
$h_{1,2} = 1$
$h_{1,3} = 1 + p_{1,2}h_{2,3}$
…

$$c_{i,j} = h_{i,j} + h_{j,i}$$



Stationary / Steady State Distribution

- π ist ein Vektor (mit „Wahrscheinlichkeiten“)
- π ist ein „spezielles“ p
- Nicht zwingend eindeutig
- System linearer Gleichungen



$$\pi = \pi \cdot P$$

$$(\pi_1 \quad \pi_2 \quad \pi_3)\begin{pmatrix} p & p & p \\ p & p & p \\ p & p & p \end{pmatrix} = (\pi_1 \quad \pi_2 \quad \pi_3)$$

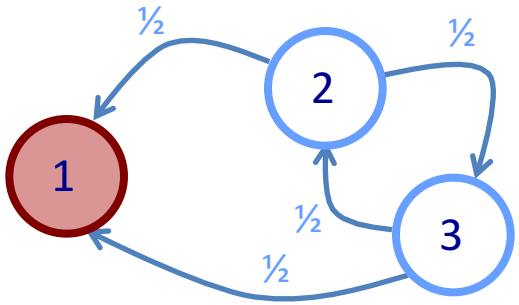
$$\pi_i \geq 0 \text{ and } \sum_i \pi_i = 1$$

$\pi_1 = ½\pi_2 + ½\pi_3$
$\pi_2 = \pi_1 + ½\pi_3$
$\pi_3 = ½\pi_2$
$\pi_1 + \pi_2 + \pi_3 = 1$

$\pi_1 = 3/9$
$\pi_2 = 4/9$
$\pi_3 = 2/9$

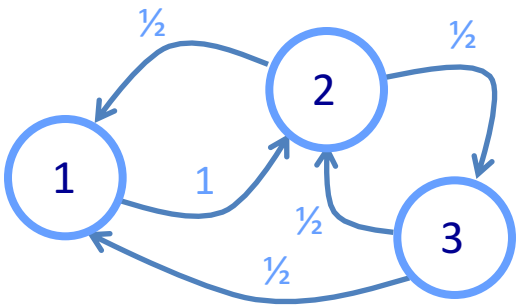
Stationary / Steady State Distribution

- **Irreduzible Markov Chains**
  - Alle Zustände können von überall her erreicht werden
  - $h_{i,j}$ beschränkt für alle Zustände
  - $f_{i,j} = 1$ für alle Zustände
- **Absorbing / Terminal States**



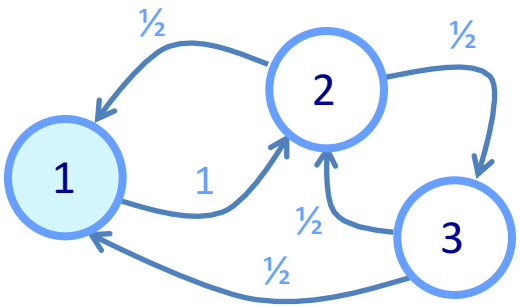
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



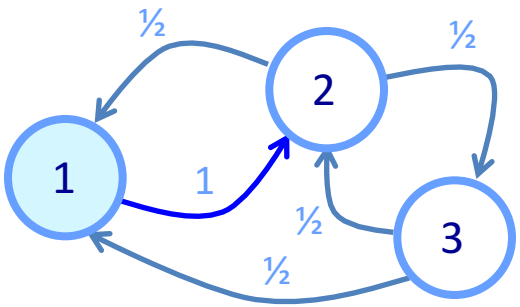
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



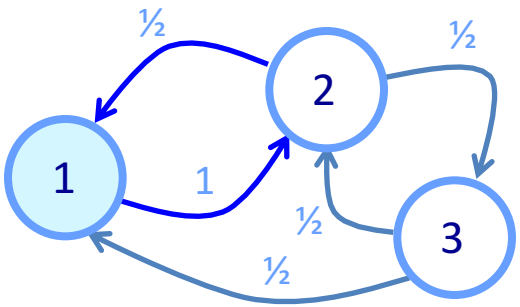
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



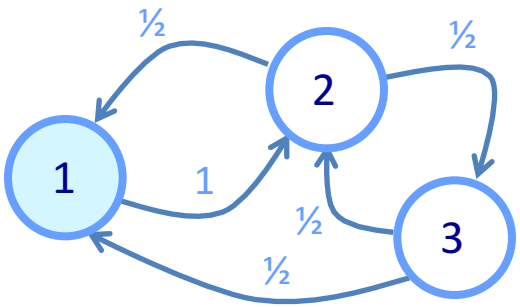
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



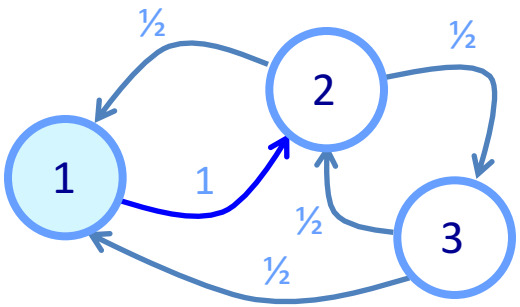
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



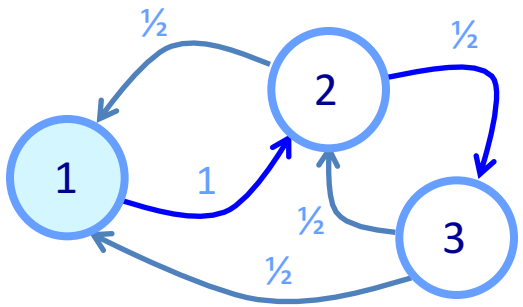
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



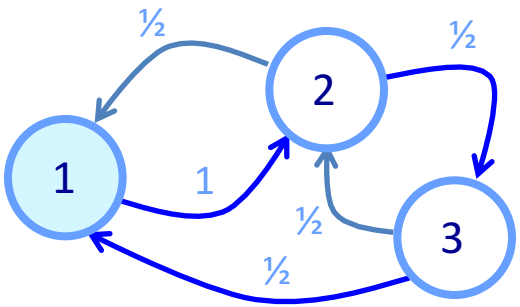
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



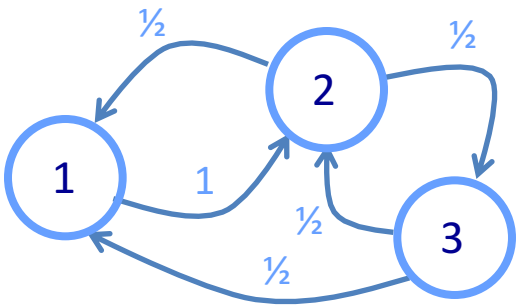
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



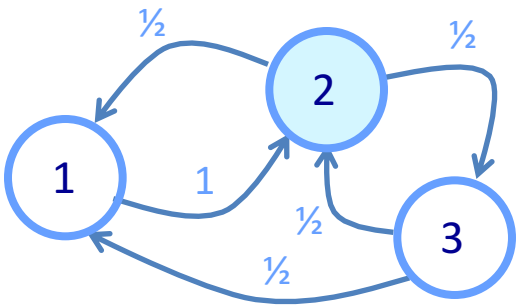
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



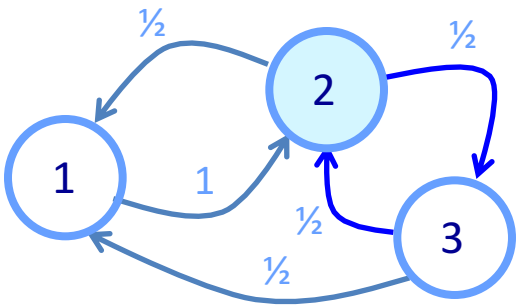
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



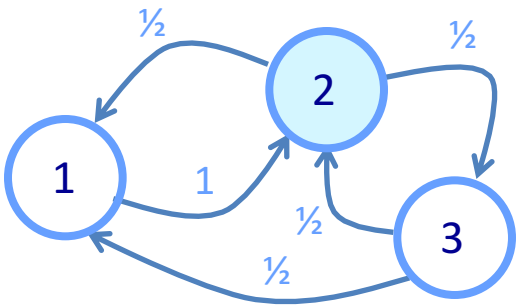
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



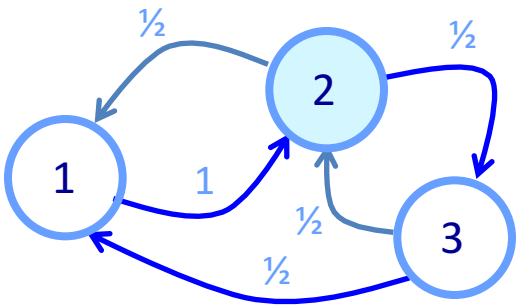
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



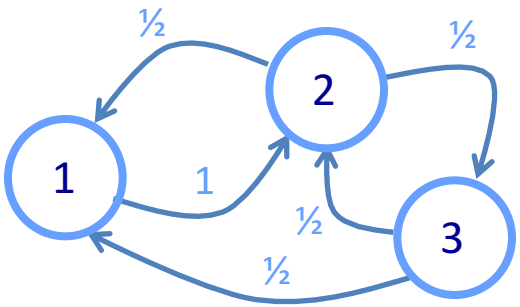
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



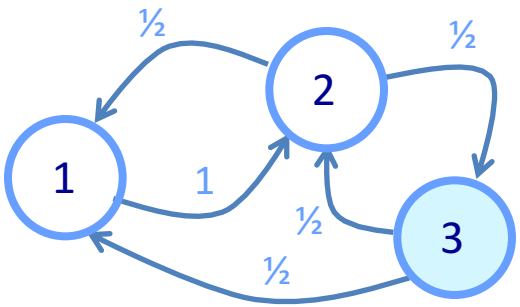
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



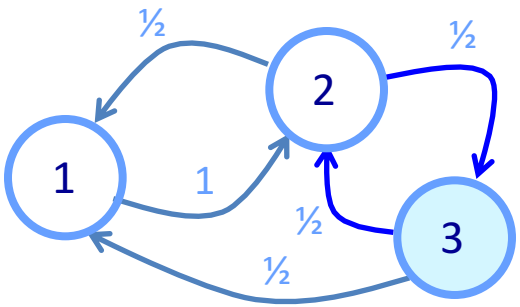
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



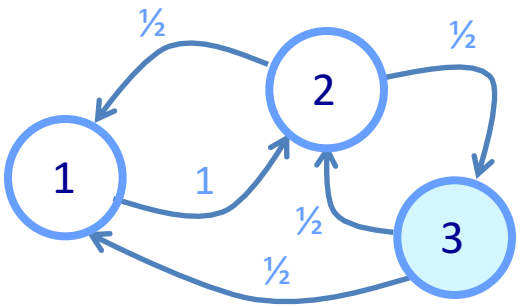
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



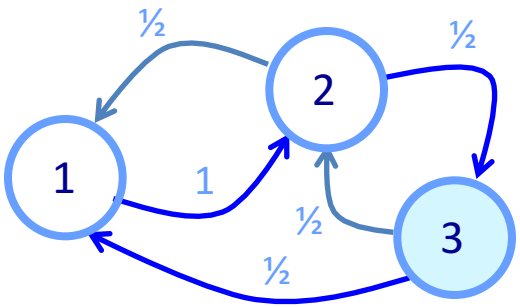
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



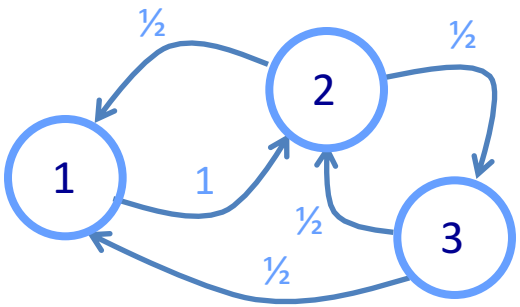
Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



Periodizität

- **Periode** → Periode k, falls alle Return-Trips Vielfache von k sind
- Zustand ist **aperiodisch**, falls Periode des Zustands 1 ist
- Markov-Kette ist **aperiodisch**, falls alle Zustände aperiodisch sind
- Markov-Kette ist **ergodisch**, falls sie irreduzible und aperiodisch ist
- Markov-Kette ist symmetrisch, falls π_i für alle absorb. Zustände identisch sind



Aperiodisch

Page Rank Algorithmus

- **Naiver Ansatz** → Adjazenz- / Transition Matrix verwenden
  - Webseiten nach der Anzahl eingehender Hyperlinks bewerten
  - **Page-Rank-Vector** v

$$v = (1, 1, 1, \dots, 1) \cdot A$$

- **Google Matrix**
  - **Random Surfer Matrix W** → Transition Matrix
  - α ist eine Gewichtung im Intervall [0,1] → Sollte nahe an 1 sein
  - R ist eine Matrix mit den Einträgen 1/n → Macht Problem irreduzibel

$$M = \alpha \cdot W + (1 - \alpha) \cdot R$$

Empfohlene Übungen

- **Serie 11**

  - Aufgabe 4 (Advanced)