Netze

Modul 7: Routing

Prof. Dr. Hannes Tschofenig

13. November 2025

Semesterplanung — Vorlesungen

Modul	Dozent	Datum	Thema
1	Rademacher	2. Oktober 2025	Einführung, OSI-Referenzmodell und Topologien
2	Rademacher	9. Oktober 2025	Übertragungsmedien und Verkabelung
3	Rademacher	16. Oktober 2025	Ethernet und WLAN
4	Tschofenig	23. Oktober 2025	IPv4, Subnetze, ARP, ICMP
5	Tschofenig	30. Oktober 2025	IPv6 und Autokonfiguration
6	Tschofenig	6. November 2025	Netzwerksegmentierung
7	Tschofenig	13. November 2025	Routing
8	Rademacher	20. November 2025	Transportschicht und UDP
9	Rademacher	27. November 2025	TCP
10	Rademacher	4. Dezember 2025	DNS und HTTP 1
11	Tschofenig	11. Dezember 2025	HTTP 2 und QUIC
12	Tschofenig	18. Dezember 2025	TLS und VPN
/	/	8. Januar 2026	`Bei Bedarf / TBA`
13	Tschofenig	15. Januar 2026	Messaging
14	Rademacher	22. Januar 2026	Moderne Netzstrukturen

Semesterplanung — Übungen und Praktika

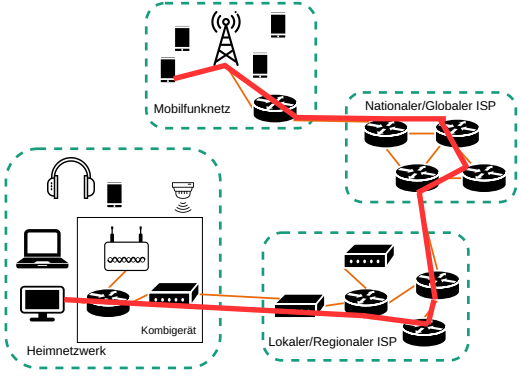
ID	KW	Art	Thema
	40	/	/
UE-1	41	Übung	Topologien und OSI
UE-2	42	Übung	Übertragungen bspw. Kabel
P-1	43	Praktikum	Laboreinführung und Netzwerktools
S-1	44	Video	IPv4
P-2	45	Praktikum	Adressierung
P-3	46	Praktikum	IPv4 und Autokonfiguration
P-4	47	Praktikum	IPv6 und Autokonfiguration
P-5	48	Praktikum	Routing
P-6	49	Praktikum	Switching
P-7	50	Praktikum	Transportprotokolle
S-2	51	Experiment	VPN
S-2	52	Experiment	VPN
	2	/	/
P-8	3	Praktikum	DNS
P-9	4	Praktikum	Webkommunikation

UE - Übung laut Stundenplan in den Seminarräumen

P - Praktikum in C055

S - Selbststudium KEINE Präsenz

Routing



- Routing: Ermittlung „günstiger“ Pfade (Routen) von sendenden Hosts zu empfangenden Hosts durch ein Netz von Routern.
- Pfad: Abfolge von Routern, die Pakete durchlaufen.
- „günstig“ je nach Metrik: geringe Kosten, kurze Verzögerung, hohe Bandbreite, geringe Überlastung, …

Übersicht über Routingprotokolle

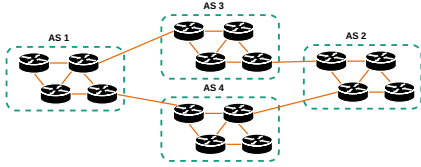
Autonomous System

- Im Internet werden oft Netze zu größeren Einheiten zusammengeschlossen.

Autonomous System

Ein Autonomous System ist eine Gruppe von Routern unter der Kontrolle einer einzigen Administration [5].

- Verwaltung der Autonomous System Numbers: `https://www.iana.org/assignments/as-numbers/as-numbers.xhtml`
- Internes Routing: Routing **innerhalb** eines Autonomous Systems (statisch oder dynamisch via Routing-Protokollen).
- Externes Routing: Routing **zwischen** unterschiedlichen Autonomous Systems — immer via Routing-Protokollen.



Routing-Tabellen

- Routing-Informationen werden i. d. R. als Tabelle dargestellt.
- Bewertungskriterien von Pfaden heißen „Metrik“.
- Metriken: Hop-Count, Bandbreite, Verzögerung, Zuverlässigkeit, Kosten.
- Beispiel (hier ein Host, kein Router):

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.27.176.0    0.0.0.0         255.255.240.0   U     256    0        0 eth1
172.27.176.1    0.0.0.0         255.255.255.255 U     256    0        0 eth1
172.27.191.255  0.0.0.0         255.255.255.255 U     256    0        0 eth1
224.0.0.0       0.0.0.0         240.0.0.0       U     256    0        0 eth1
255.255.255.255 0.0.0.0         255.255.255.255 U     256    0        0 eth1
192.168.56.0    0.0.0.0         255.255.255.0   U     256    0        0 eth2
192.168.56.1    0.0.0.0         255.255.255.255 U     256    0        0 eth2
192.168.56.255  0.0.0.0         255.255.255.255 U     256    0        0 eth2
224.0.0.0       0.0.0.0         240.0.0.0       U     256    0        0 eth2
255.255.255.255 0.0.0.0         255.255.255.255 U     256    0        0 eth2
127.0.0.0       0.0.0.0         255.0.0.0       U     256    0        0 lo
127.0.0.1       0.0.0.0         255.255.255.255 U     256    0        0 lo
127.255.255.255 0.0.0.0         255.255.255.255 U     256    0        0 lo
224.0.0.0       0.0.0.0         240.0.0.0       U     256    0        0 lo
255.255.255.255 0.0.0.0         255.255.255.255 U     256    0        0 lo
0.0.0.0         192.168.178.1   255.255.255.255 U     0      0        0 wifi0
192.168.178.0   0.0.0.0         255.255.255.0   U     256    0        0 wifi0
192.168.178.28  0.0.0.0         255.255.255.255 U     256    0        0 wifi0
192.168.178.255 0.0.0.0         255.255.255.255 U     256    0        0 wifi0
224.0.0.0       0.0.0.0         240.0.0.0       U     256    0        0 wifi0
255.255.255.255 0.0.0.0         255.255.255.255 U     256    0        0 wifi0
```

Linux: `route -n`

Statisches Routing und Dynamisches Routing

Statisches Routing

- Einträge der Routing-Tabelle werden manuell **eingegeben**, d. h. fest vorgegeben.
- Geeignet für kleinere Netze mit wenigen Routern.
- Änderungen müssen von Hand gepflegt werden.
- Bestimmte Adressen gehen immer an festgelegte Interfaces.
- Unbekannte Adressen gehen an ein „Standard-Gateway“ (meist Internetzugang).

Dynamisches Routing

- Routing-Tabellen werden mit **Routing-Protokollen** verwaltet.
- Geeignet für größere Netze.
- Routing-Informationen werden regelmäßig über dieselben Verbindungen übertragen wie Nutzdaten.

Grundfrage: Ab wann lohnt sich der Einsatz eines Routing-Protokolls?

Routing-Protokolle (Überblick mit Beispielen)

```
Routing-Protokolle
+-- Interne Routing-Protokolle (IGP)
|   +-- Distance-Vector
|   |   +-- Routing Information Protocol (RIP) [2]
|   |   `-- Enhanced Interior Gateway Routing Protocol (EIGRP) [6]
|   `-- Link-State
|       +-- Open Shortest Path First (OSPF) [3]
|       `-- Intermediate System to Intermediate System (IS-IS) [4]
`-- Externe Routing-Protokolle (EGP)
    `-- Path-Vector
        `-- Border Gateway Protocol (BGP) [5]
```

Neben etablierten Protokollen wie RIP und OSPF werden in speziellen Szenarien auch andere Routing-Protokolle eingesetzt, etwa das Routing Protocol for Low-Power and Lossy Networks (RPL) [1], das bei Matter über Thread im Smart Home genutzt wird.

Link-State Routing

Link-State Routing

- **Prinzip:**
  Jeder Router kennt die Kosten seiner **direkten Verbindungen** und teilt diese Informationen mit allen anderen Routern in der Area.
- **Gespeicherte Informationen:**
  Jeder Router führt eine vollständige **Topologie-Datenbank** (LSDB) mit allen Links und ihren Kosten.
- **Austauschmechanismus:**
  Jeder Router erstellt **Link-State Advertisements (LSA)**. Diese LSAs enthalten:
  - welche Nachbarn der Router hat (also welche anderen Router direkt verbunden sind),
  - welche Netzwerke an ihn angeschlossen sind, und
  - die Kosten (Metriken) dieser Links.

  Diese LSAs werden durch **Flooding** an alle Router in der Area verteilt.
- **Berechnung:**
  Jeder Router berechnet mit dem **Dijkstra-Algorithmus** die kürzesten Pfade und erstellt daraus seine Routing-Tabelle.
- **Vertreter: OSPF**, **IS-IS**.

Ablauf des Link-State Routings (1)

1. **Nachbarn ermitteln (Hello-Prozess)**
   Router senden regelmäßig *Hello-Nachrichten* auf ihren Schnittstellen. Kommt eine Antwort, entsteht eine Nachbarschaft. Bleiben Hellos aus ⇒ Nachbar gilt als ausgefallen.

Ablauf des Link-State Routings (1)

1. **Nachbarn ermitteln (Hello-Prozess)**
   Router senden regelmäßig *Hello-Nachrichten* auf ihren Schnittstellen. Kommt eine Antwort, entsteht eine Nachbarschaft. Bleiben Hellos aus ⇒ Nachbar gilt als ausgefallen.
2. **Link-Kosten bestimmen**
   Für jede Verbindung werden Metriken (z. B. Bandbreite, Delay, Hop-Anzahl) berechnet oder konfiguriert.

Ablauf des Link-State Routings (1)

1. **Nachbarn ermitteln (Hello-Prozess)**
   Router senden regelmäßig *Hello-Nachrichten* auf ihren Schnittstellen. Kommt eine Antwort, entsteht eine Nachbarschaft. Bleiben Hellos aus ⇒ Nachbar gilt als ausgefallen.
2. **Link-Kosten bestimmen**
   Für jede Verbindung werden Metriken (z. B. Bandbreite, Delay, Hop-Anzahl) berechnet oder konfiguriert.
3. **LSAs erstellen und fluten**
   Jeder Router beschreibt in einer *Link-State Advertisement (LSA)* alle seine **direkten Verbindungen (Links)** zu anderen Routern oder Netzen inklusive der jeweiligen **Kosten**. Diese werden per **Flooding** an alle Router derselben Area verteilt.

Ablauf des Link-State Routings (2)

4. **Topologie-Datenbank aufbauen**
   Router speichern nur **neue oder geänderte LSAs**; doppelte oder alte Informationen werden ignoriert. Ergebnis: alle Router teilen die gleiche Topologie.

Ablauf des Link-State Routings (2)

4. **Topologie-Datenbank aufbauen**
   Router speichern nur **neue oder geänderte LSAs**; doppelte oder alte Informationen werden ignoriert. Ergebnis: alle Router teilen die gleiche Topologie.
5. **Kürzeste Wege berechnen**
   Mit dem *Dijkstra-Algorithmus* berechnet jeder Router die besten Pfade zu allen Zielen und aktualisiert seine Routing-Tabelle. Obwohl alle Router dieselbe Topologie sehen, berechnet jeder Router eine eigene Routing-Tabelle.

Ergebnis: Schnelle und stabile Konvergenz, jedoch höherer Speicher- und Rechenbedarf als beim Distance-Vector-Routing.

Dijkstra-Algorithmus (Shortest Path First)

Ziel: Für jeden Router die kürzesten Pfade (minimale Gesamtkosten) zu allen anderen Knoten bestimmen.

Grundidee: Jeder Router kennt die gesamte Topologie (LSDB) und berechnet daraus mit dem *Dijkstra-Algorithmus* einen Baum der kürzesten Wege.

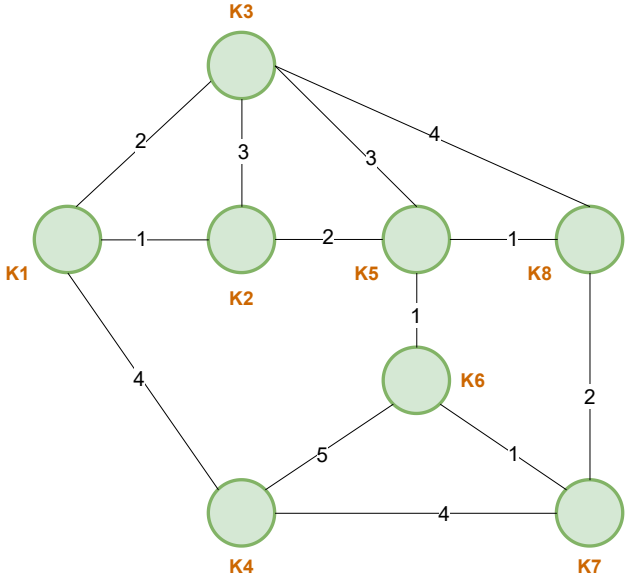
1. **Initialisierung:** Setze Distanz zum Startknoten = 0, zu allen anderen = ∞.
2. **Knoten wählen:** Wähle den Knoten mit der aktuell kleinsten Distanz (noch nicht besucht).
3. **Nachbarn prüfen:** Berechne für jeden Nachbarn die neuen Kosten:

   $$D_{\text{neu}} = D_{\text{aktuell}} + \text{Kosten(Kante)}$$

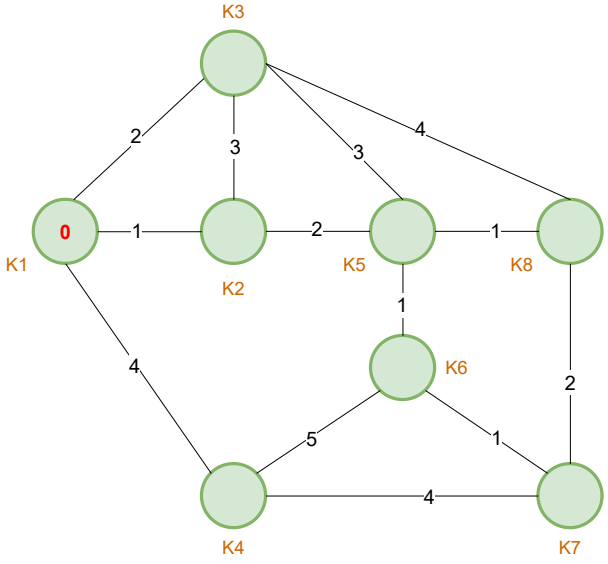
   Wenn D_{neu} kleiner ist als der bisherige Wert, aktualisiere Distanz und Vorgänger.
4. **Wiederholen:** Wiederhole Schritt 2–3, bis alle Knoten „besucht“ sind.

Ergebnis: Der *Shortest Path Tree* (SPT) zeigt, über welche Nachbarn die kürzesten Pfade zu jedem Ziel verlaufen.

Beispiel: Netzwerktopologie

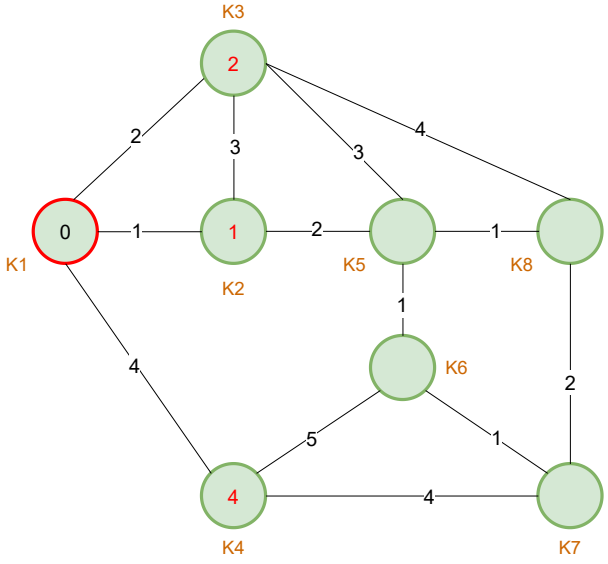


Iteration 0 — Initialisierung



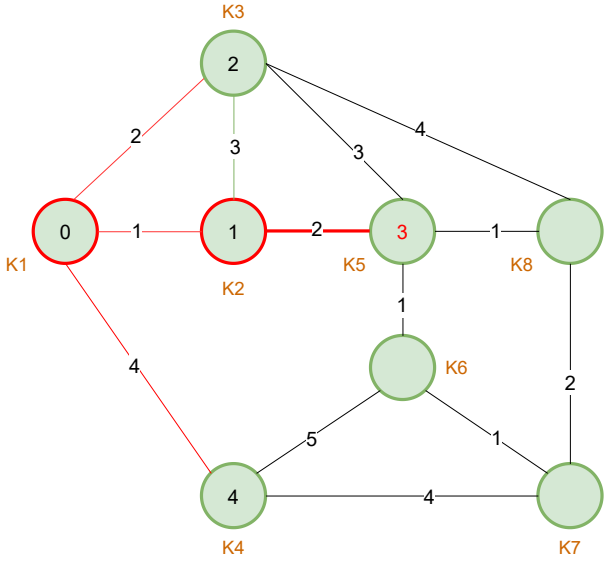
- Distanz: $d(K_1) = 0$, alle anderen $d = \infty$.
- Vorgänger: $\pi(K_i)$ zunächst undefiniert.

Iteration 1 — Auswahl von K_1 (Distanz 0)



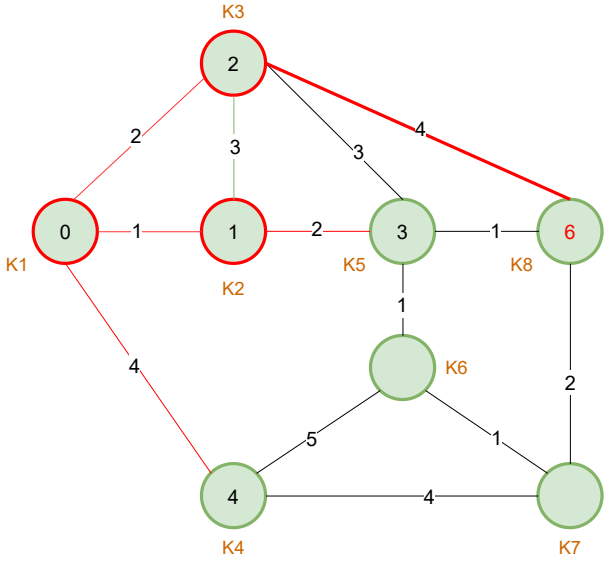
- $K_2: 0+1=1 \Rightarrow d(K_2)=1,\ \pi(K_2)=K_1$
- $K_3: 0+2=2 \Rightarrow d(K_3)=2,\ \pi(K_3)=K_1$
- $K_4: 0+4=4 \Rightarrow d(K_4)=4,\ \pi(K_4)=K_1$
- **Fest:** $\{K_1\}$

Iteration 2 — wähle K_2 (1)



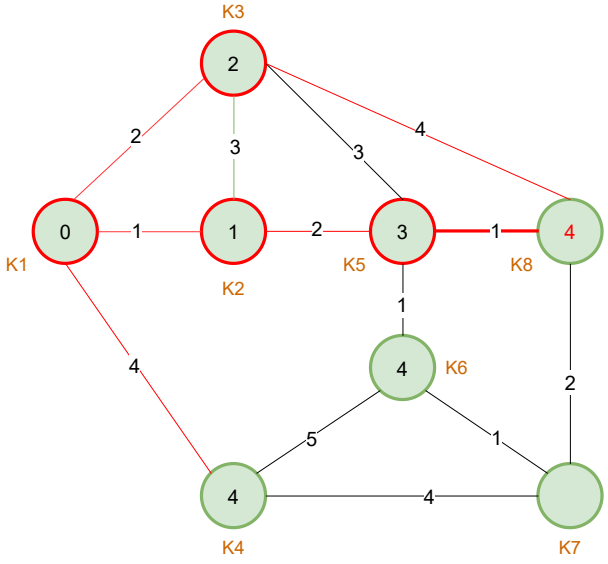
- K_3 über K_2 : $1 + 3 = 4 > 2 \Rightarrow$ keine Änderung
- K_5 : $1 + 2 = 3 \Rightarrow d(K_5) = 3,\ \pi(K_5) = K_2$
- **Fest:** $\{K_1, K_2\}$

Iteration 3 — wähle K_3 (2)



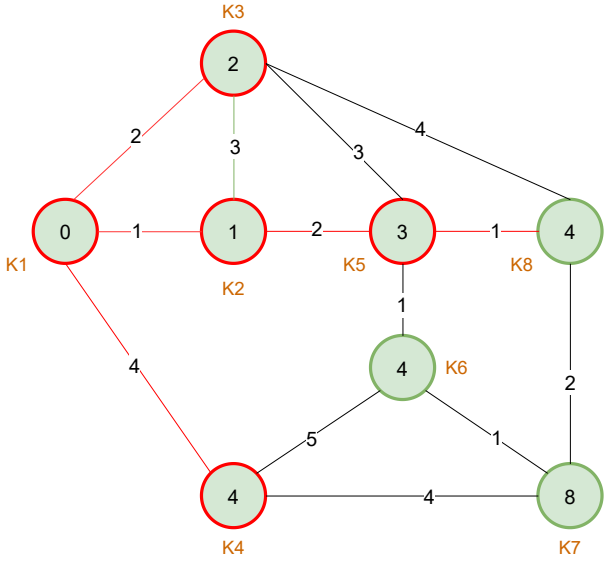
- K_5 über K_3 : $2 + 3 = 5 > 3 \Rightarrow$ keine Änderung
- K_8 : $2 + 4 = 6 \Rightarrow d(K_8) = 6,\ \pi(K_8) = K_3$
- **Fest:** $\{K_1, K_2, K_3\}$

Iteration 4 — wähle K_5 (3)



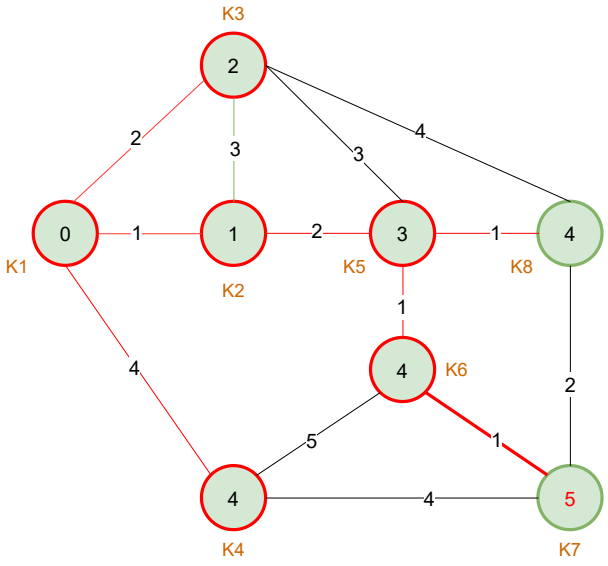
- $K_6: \ 3+1=4 \Rightarrow d(K_6)=4, \ \pi(K_6)=K_5$
- $K_8: \ 3+1=4<6 \Rightarrow d(K_8)=4, \ \pi(K_8)=K_5$
- **Fest:** $\{K_1, K_2, K_3, K_5\}$

Iteration 5 — wähle K_4 (4)



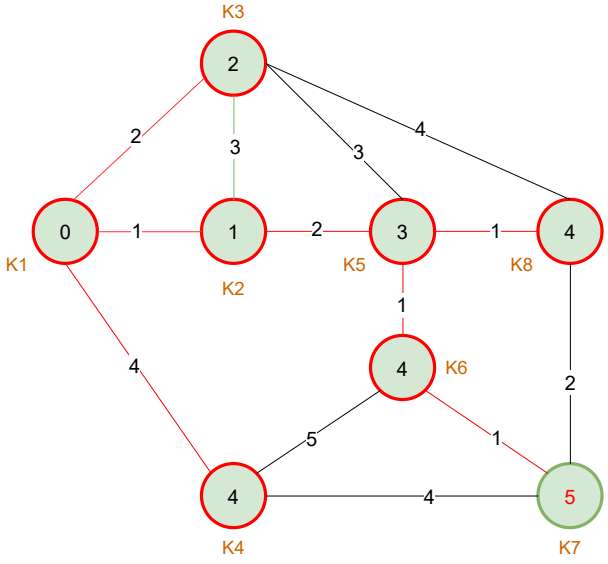
- K_6 über K_4 : $4 + 5 = 9 > 4 \Rightarrow$ keine Änderung
- K_7 : $4 + 4 = 8 \Rightarrow d(K_7) = 8,\ \pi(K_7) = K_4$
- **Fest:** $\{K_1, K_2, K_3, K_5, K_4\}$

Iteration 6 — wähle K_6 (4)



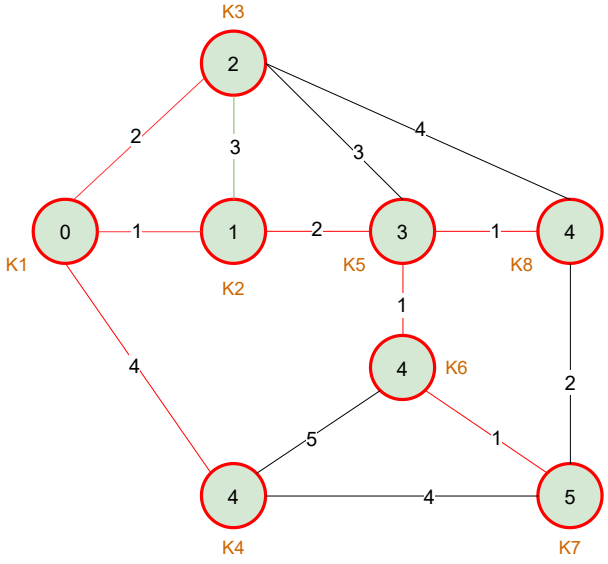
- $K_7: \ 4+1=5<8 \Rightarrow d(K_7)=5, \ \pi(K_7)=K_6$
- **Fest:** $\{K_1, K_2, K_3, K_5, K_4, K_6\}$

Iteration 7 — wähle K_8 (4)



- K_7 über K_8 : $4 + 2 = 6 > 5 \Rightarrow$ keine Änderung
- **Fest:** $\{K_1, K_2, K_3, K_5, K_4, K_6, K_8\}$

Iteration 8 — wähle K_7 (5)



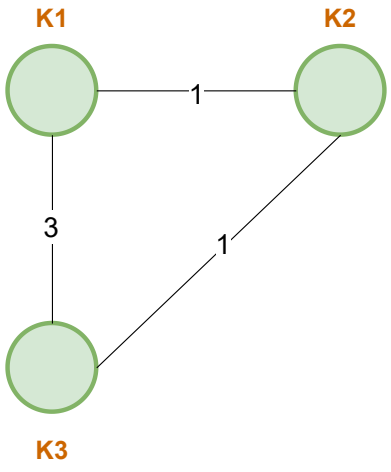
- Fertig: alle Knoten fest.
- **Fest:**
  $\{K_1, K_2, K_3, K_5, K_4, K_6, K_7, K_8\}$

Distance-Vector Routing

Distance-Vector Routing

- **Prinzip:**
  Jeder Router kennt nur seine direkten Nachbarn und die Kosten dorthin. Er erfährt die restlichen Routen **indirekt** durch periodischen Austausch mit Nachbarn.
- **Gespeicherte Informationen:**
  Für jedes Zielnetz speichert der Router:
  1. Zieladresse oder -netz,
  2. Kosten (Metrik, meist Hop-Count),
  3. Next Hop (nächster Router-Port).
- **Austauschmechanismus:**
  Jeder Router sendet seinen **Distance-Vector** (Liste aller bekannten Ziele mit Kosten) regelmäßig an seine Nachbarn.
- **Berechnung:**
  Router aktualisieren ihre Tabellen mit dem **Bellman-Ford-Algorithmus** und wählen pro Ziel den günstigsten Next Hop.
- **Vertreter: RIP**, **EIGRP** (Cisco).

Initialisierung



Netzwerktopologie mit Kosten

K1

Destination	Next Hop	Cost
K1	Local	0

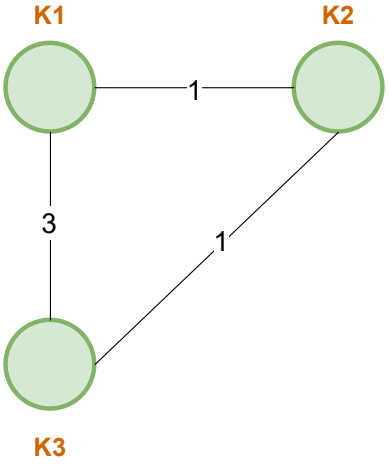
R2

Destination	Next Hop	Cost
K2	Local	0

R3

Destination	Next Hop	Cost
K3	Local	0

Direkte Nachbarn Lernen



K1

Destination	Next Hop	Cost
K1	Local	0
K2	K2	1
K3	K3	3

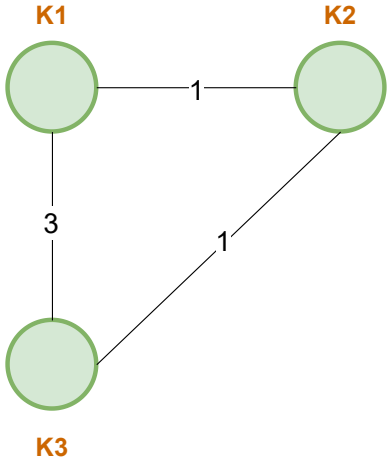
K2

Destination	Next Hop	Cost
K2	Local	0
K1	K1	1
K3	K3	1

K3

Destination	Next Hop	Cost
K3	Local	0
K2	K2	1
K1	K1	3

Indirekte Nachbarn Lernen



K1

Destination	Next Hop	Cost
K1	Local	0
K2	K2	1
K3	K2	2

K2

Destination	Next Hop	Cost
K2	Local	0
K1	K1	1
K3	K3	1

K3

Destination	Next Hop	Cost
K3	Local	0
K2	K2	1
K1	K2	2

Wie lernt K1 indirekte Nachbarn? – Ausgangssituation

K1 – Routing-Tabelle

Destination	Next Hop	Cost
K1	Local	0
K2	K2	1
K3	K3	3

Prinzip:

K1 kennt nur seine direkten Nachbarn: K2 (Kosten 1) und K3 (Kosten 3).

Wie lernt K1 indirekte Nachbarn? – Schritt 1

K2 – Distance Vector

Destination	Cost (aus Sicht K2)
K2	0
K1	1
K3	1

Schritt 1:

K2 sendet seinen Distance Vector an K1.

K1 vergleicht diese Werte mit seiner eigenen Tabelle.

Wie lernt K1 indirekte Nachbarn? – Schritt 2

K1 berechnet Kosten über K2

Destination	Kosten über K2	Bisher	Besser?
K2	1 + 0 = 1	1	–
K1	1 + 1 = 2	0	–
K3	1 + 1 = 2	3	ja

Erkenntnis:

Der Weg zu K3 über K2 ist günstiger (**Kosten 2 statt 3**).

K1 aktualisiert seinen Eintrag für K3.

Wie lernt K1 indirekte Nachbarn? – Ergebnis

K1 – neue Routing-Tabelle

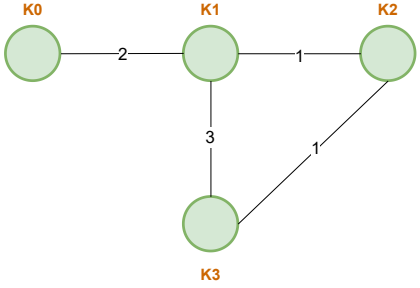
Destination	Next Hop	Cost
K1	Local	0
K2	K2	1
K3	K2	2

Ergebnis:

K1 hat über K2 einen kürzeren Weg zu K3 gefunden.

Indirektes Lernen ist abgeschlossen.

Good News: Neuer Router hinzugefügt



K1

Destination	Next Hop	Cost
K1	Local	0
K0	K0	2
K2	K2	1
K3	K3	3

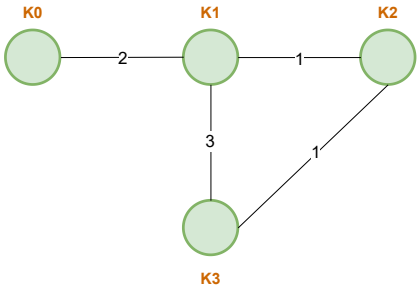
K2

Destination	Next Hop	Cost
K2	Local	0
K1	K1	1
K3	K3	1

K3

Destination	Next Hop	Cost
K3	Local	0
K2	K2	1
K1	K2	2

Routing-Tabellen nach Austausch



K1

Destination	Next Hop	Cost
K1	Local	0
K0	K0	2
K2	K2	1
K3	K3	3

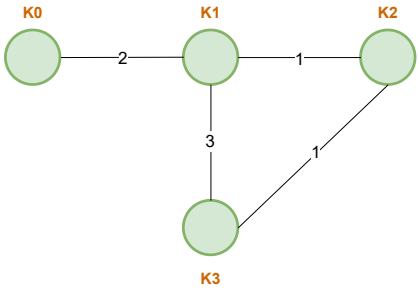
K2

Destination	Next Hop	Cost
K2	Local	0
K1	K1	1
K3	K3	1
K0	K1	3

K3

Destination	Next Hop	Cost
K3	Local	0
K2	K2	1
K1	K2	2
K0	K1	5

Aktualisierung nach weiterer Verbreitung



K1

Destination	Next Hop	Cost
K1	Local	0
K0	K0	2
K2	K2	1
K3	K3	3

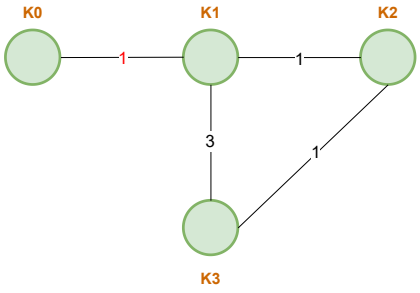
K2

Destination	Next Hop	Cost
K2	Local	0
K1	K1	1
K3	K3	1
K0	K1	3

K3

Destination	Next Hop	Cost
K3	Local	0
K2	K2	1
K1	K2	2
K0	K2	4

Good News: Link-Gewicht sinkt



K1

Destination	Next Hop	Cost
K1	Local	0
K0	K0	1
K2	K2	1
K3	K3	3

K2

Destination	Next Hop	Cost
K2	Local	0
K1	K1	1
K3	K3	1
K0	K1	2

K3

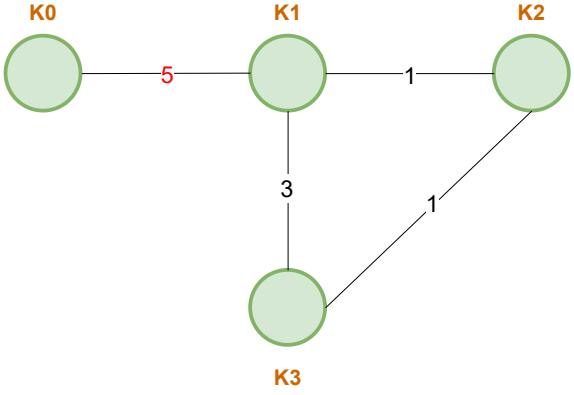
Destination	Next Hop	Cost
K3	Local	0
K2	K2	1
K1	K2	2
K0	K2	3

„Good news travel fast!“

Wie schnell verteilen sich gute und schlechte Nachrichten?

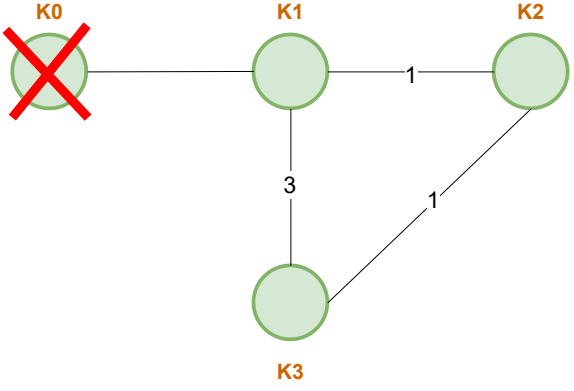
Wenn N die maximale Pfadlänge in Hops ist, brauchen wir max. N Schritte.

Bad News: Link-Gewicht steigt



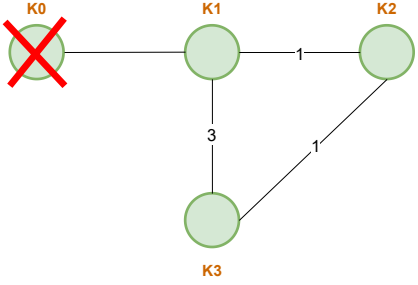
Was passiert, wenn K1 ein Update von K2 zu K0 erhält?

Bad News: Router fällt aus



Was passiert beim Ausfall eines Routers?

Count-to-Infinity (1): K0 fällt aus



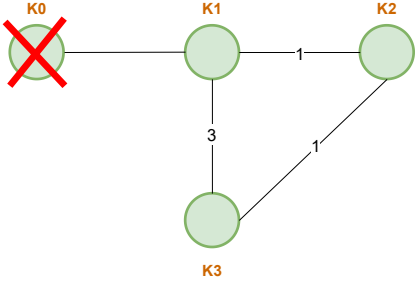
K1

Destination	Next Hop	Cost
K1	Local	0
K2	K2	1
K3	K3	3
K0	–	∞

K2

Destination	Next Hop	Cost
K2	Local	0
K1	K1	1
K3	K3	1
K0	K1	3

Count-to-Infinity (2): Falsches Update



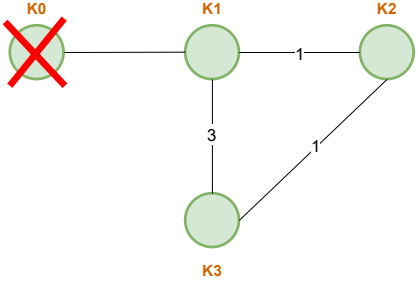
K1

Destination	Next Hop	Cost
K1	Local	0
K2	K2	1
K3	K3	3
K0	K2	4

K2

Destination	Next Hop	Cost
K2	Local	0
K1	K1	1
K3	K3	1
K0	K1	5

Count-to-Infinity (3): Kosten steigen weiter



K1

Destination	Next Hop	Cost
K1	Local	0
K2	K2	1
K3	K3	3
K0	K2	6

K2

Destination	Next Hop	Cost
K2	Local	0
K1	K1	1
K3	K3	1
K0	K1	7

Wie vermeiden wir Schleifen?

Split Horizon

Route wird nicht über den Weg bekanntgegeben, über den sie gelernt wurde.

Split Horizon mit Poisoned Reverse

Route über den Lernweg wird mit Kosten „unendlich“ (∞) angekündigt.

Funktioniert dieser Ansatz für alle Topologien?

Was passiert bei komplexeren oder zyklischen Netzen?

Können Schleifen oder falsche Kosten dennoch auftreten?

Wie vermeiden wir Schleifen?

Split Horizon

Route wird nicht über den Weg bekanntgegeben, über den sie gelernt wurde.

Split Horizon mit Poisoned Reverse

Route über den Lernweg wird mit Kosten „unendlich“ (∞) angekündigt.

Vordefinierte maximale Kosten

„Bad news travel slow!“

Wie schnell verteilen sich gute und schlechte Nachrichten?

Wenn N die maximal möglichen Kosten sind,
brauchen wir höchstens N Schritte.

Warum ist Count-to-Infinity ein fundamentales Problem?

Beobachtung:

Distance-Vector-Router lernen nur über ihre Nachbarn – Informationen über Ausfälle verbreiten sich daher *langsam und indirekt*.

Problem:

Beim Ausfall eines Pfads glauben Router, dass Nachbarn den Weg noch kennen.
$\Rightarrow$ Sie lehren sich gegenseitig falsche Routen.

Folgen:

- Kosten steigen schrittweise bis ∞ (Count-to-Infinity)
- Routing-Schleifen, Pakete zirkulieren
- Langsame Konvergenz

Lösungen:

- **Split Horizon:** Keine Rückgabe über den Lernweg
- **Poisoned Reverse:** Route über Lernweg mit ∞-Kosten ankündigen
- **Kostenlimit:** z. B. 16 bei RIP

Fazit:

Count-to-Infinity ist eine Folge der *lokalen, verteilten Sicht* im Distance-Vector-Ansatz.

Zusammenfassung

Distance-Vector vs. Link-State

Merkmal	Distance-Vector	Link-State
Prinzip	Kennt nur Nachbarn und deren Distanzen	Kennt komplette Topologie der Area
Informationsaustausch	Austausch von **Distance-Vectoren** mit Nachbarn	Flooding von **Link-State Advertisements (LSA)**
Update-Verhalten	Periodisch oder bei größeren Änderungen	Ereignisgesteuert bei Topologieänderungen
Algorithmus	**Bellman–Ford** (iterativ, Nachbar-basiert)	**Dijkstra** (global, kürzeste Pfade)
Konvergenz	Langsam; Gefahr von Schleifen (Count-to-Infinity)	Schnell; konsistente Sicht bei allen Routern
Skalierbarkeit	Begrenzt (einfache Metriken, geringe Hierarchie)	Hoch (Areas, hierarchische Struktur)
Beispiele	**RIP**, **EIGRP**	**OSPF**, **IS-IS**

Vergleich: Dijkstra, Bellman–Ford und Spanning Tree Algorithm

Dijkstra-Algorithmus

- **Typ:** Kürzeste-Pfade (Link-State)
- **Einsatz:** Routing (z. B. OSPF, IS-IS)
- **Ziel:** Kürzeste Wege von einer Quelle zu allen Zielen
- **Arbeitsweise:** Kennt gesamte Topologie und berechnet *Shortest Path Tree (SPT)*.
- **Ergebnis:** Routing-Tabelle (Next Hops)
- **Eigenschaft:** Schnelle, deterministische Berechnung

Bellman–Ford-Algorithmus

- **Typ:** Distance-Vector
- **Einsatz:** Routing (z. B. RIP, EIGRP)
- **Ziel:** Kürzeste Wege zu allen Zielen
- **Arbeitsweise:** Austausch von Distanzvektoren mit Nachbarn:

$$D_x(y) = \min_v(c(x,v) + D_v(y))$$

- **Ergebnis:** Routing-Tabelle (Next Hops)
- **Eigenschaft:** Langsame Konvergenz, Count-to-Infinity-Problem, aber geringer Speicherbedarf

Spanning Tree Algorithm

- **Typ:** Minimaler-Spannbaum (Topologiekontrolle)
- **Einsatz:** Minimalen Spannbaum berechnen, der alle Knoten verbindet, keine Zyklen enthält und die Gesamtkosten minimiert.
- **Ziel:** Schleifenfreien Baum über alle Knoten erzeugen
- **Arbeitsweise:** Wahl einer Root-Bridge, Berechnung kürzester Pfade dorthin, Blockierung redundanter Links.
- **Ergebnis:** Minimaler Spannbaum (keine Zyklen)
- **Eigenschaft:** Verteilte Variante des Minimum Spanning Tree-Problems

Quellen I

[1] Alexander, R., Brandt, A., Vasseur, J., Hui, J., Pister, K., Thubert, P., Levis, P., Struik, R., Kelsey, R., and Winter, T.
RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks.
RFC 6550, March 2012.

[2] Malkin, G. S.
RIP Version 2.
RFC 2453, November 1998.

[3] Moy, J.
OSPF Version 2.
RFC 2328, April 1998.

[4] Parker, J.
Recommendations for Interoperable Networks using Intermediate System to Intermediate System (IS-IS).
RFC 3719, February 2004.

[5] Rekhter, Y., Hares, S., and Li, T.
A Border Gateway Protocol 4 (BGP-4).
RFC 4271, January 2006.

[6] Savage, D., Ng, J., Moore, S., Slice, D., Paluch, P., and White, R.
Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP).
RFC 7868, May 2016.