

Technische Grundlagen der Informatik 2

– Teil 6:

Netzwerkschicht (Layer 3)

Philipp Rettberg / Sebastian Harnau

Block 12/18

Netzwerkschicht (Layer 3)

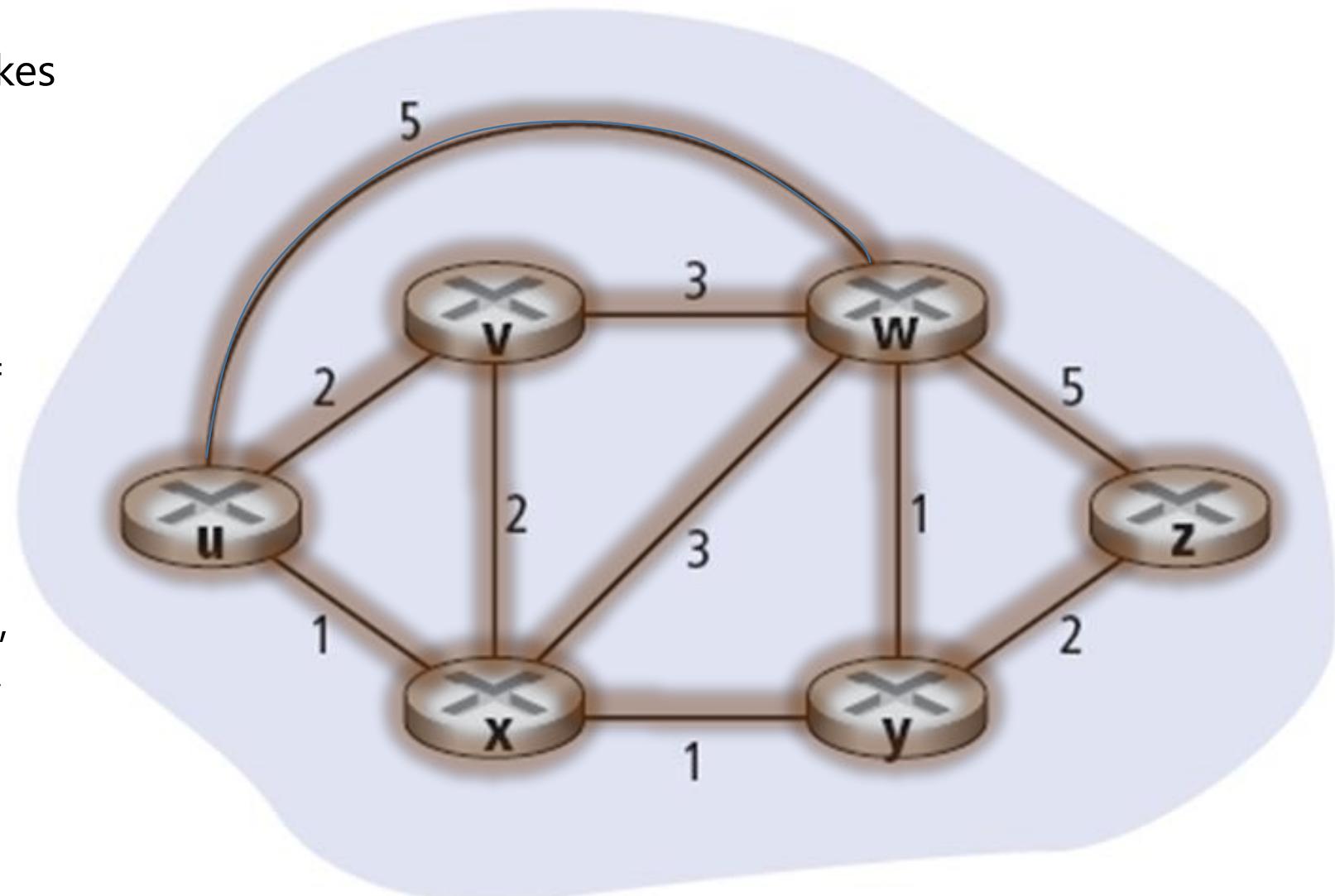
IP, Router, Routing...

Routing / Wegewahl - Graph

Darstellung des Netzwerkes
als Graph:

Graph: $G = (N, E)$

- $N =$
Menge von Routern =
 $\{ u, v, w, x, y, z \}$
- $E =$
Menge von Links =
 $\{ (u,v), (u,x), (u,w), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) \}$



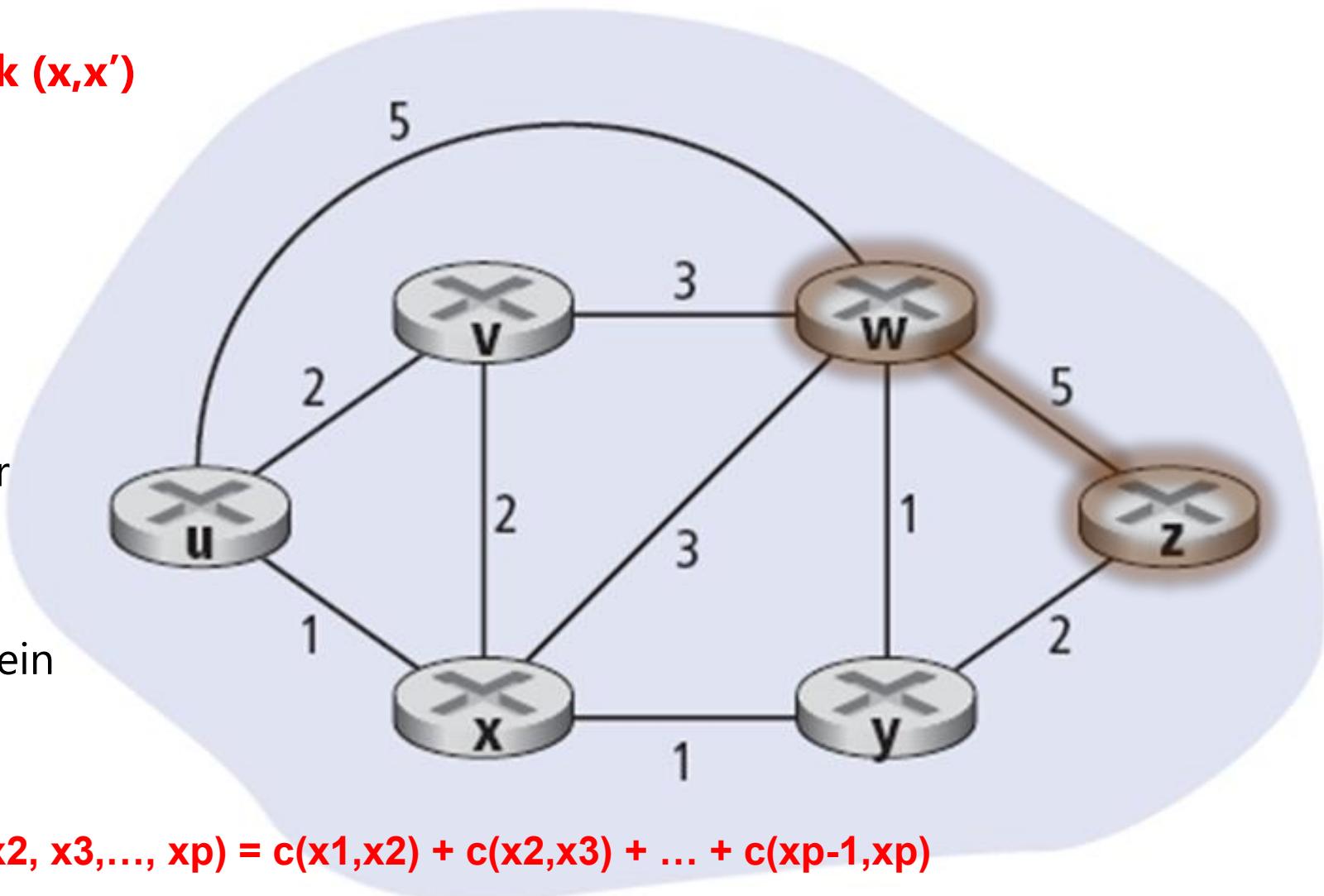
Routing / Wegewahl - Kosten

$c(x,x')$ = Kosten von Link (x,x')

- z.B. $c(w,z) = 5$

Kosten können:

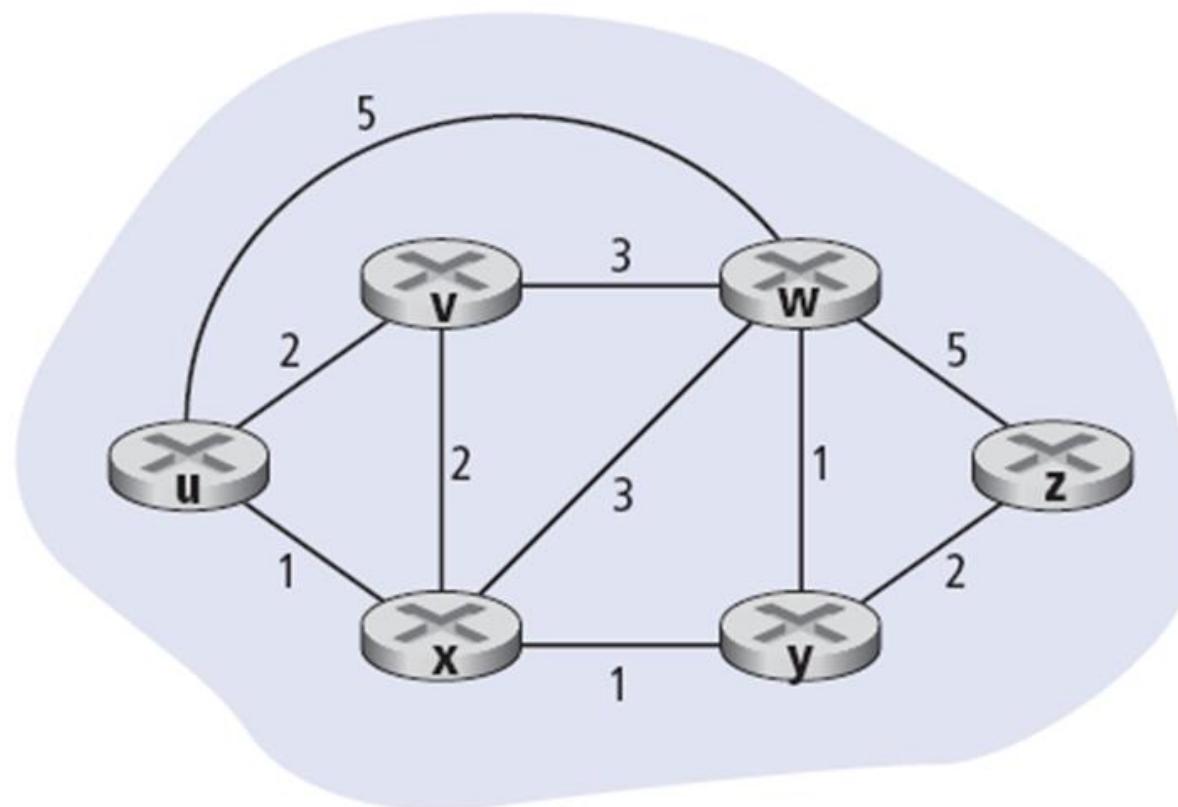
- immer 1 sein
- invers proportional zur Datenrate sein
- proportional zum Verkehrsaufkommen sein
- ...



Kosten eines Pfades $(x_1, x_2, x_3, \dots, x_p)$ = $c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$

Routing / Wegewahl - Beispiel

Frage: Was ist der günstigste Weg von u nach z?



Um die günstigsten Wege zu ermitteln, benötigen wir Routing-Algorithmen!

Klassifikation von Routing-Algorithmen

Globale Informationen



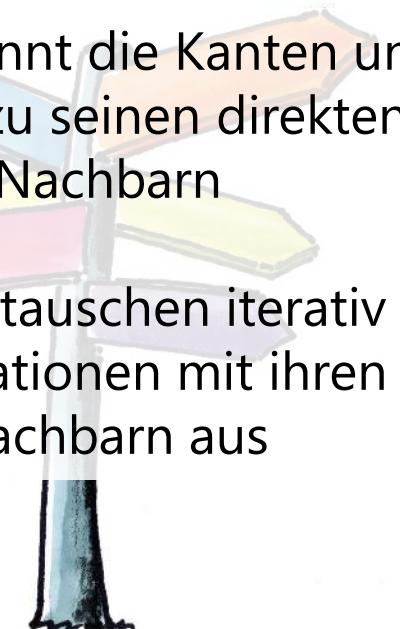
Alle Router kennen die vollständige Topologie des Graphen (alle Knoten, alle Kanten, alle Kosten)

Link-State-Routing

Dezentrale Informationen

Router kennt die Kanten und Kosten zu seinen direkten Nachbarn

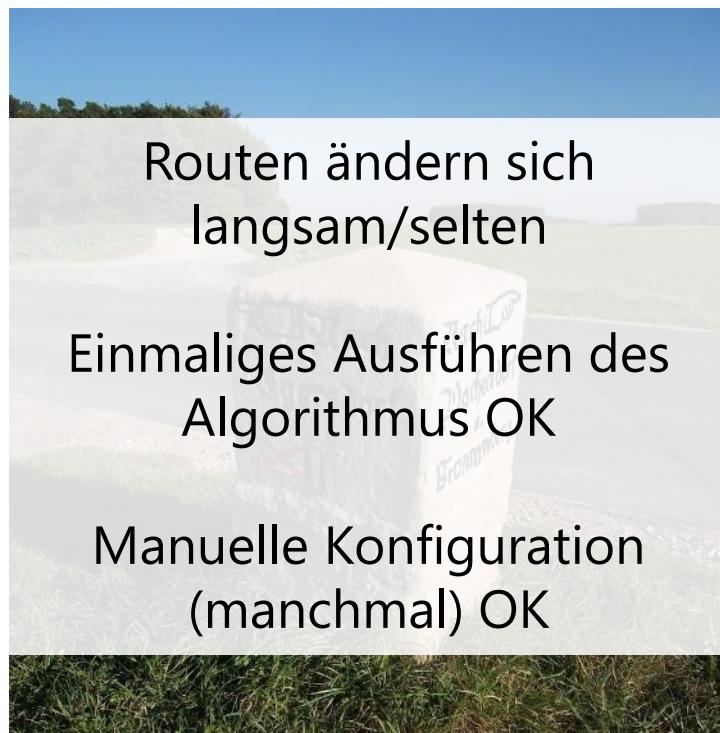
Router tauschen iterativ Informationen mit ihren Nachbarn aus



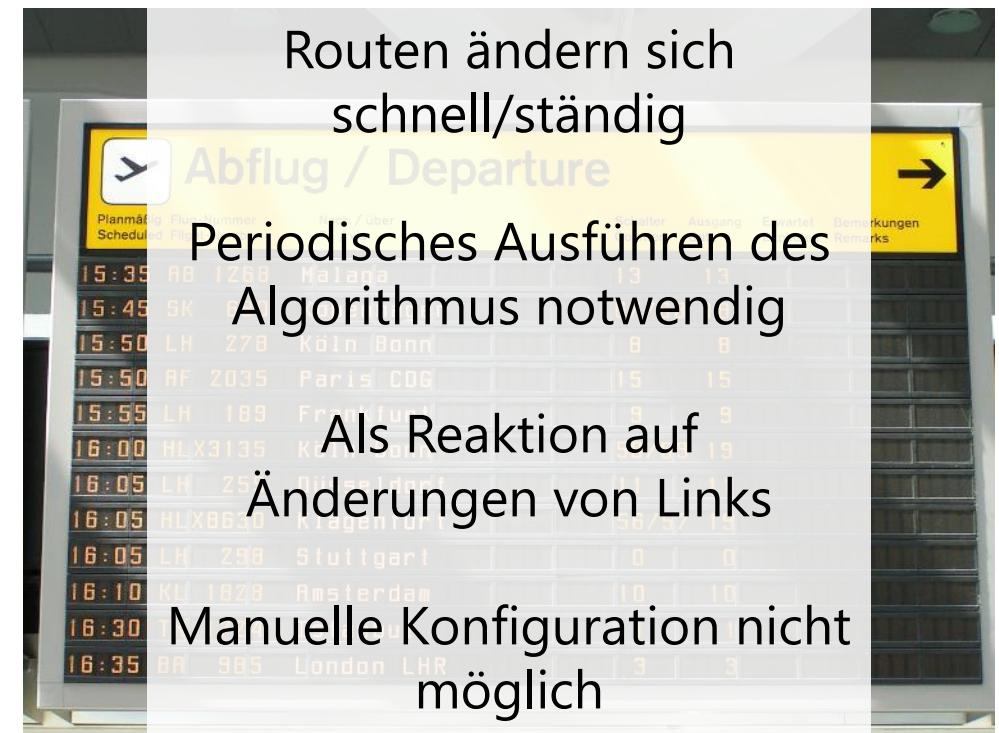
Distanzvektor-Routing

Klassifikation von Routing-Algorithmen

statisch

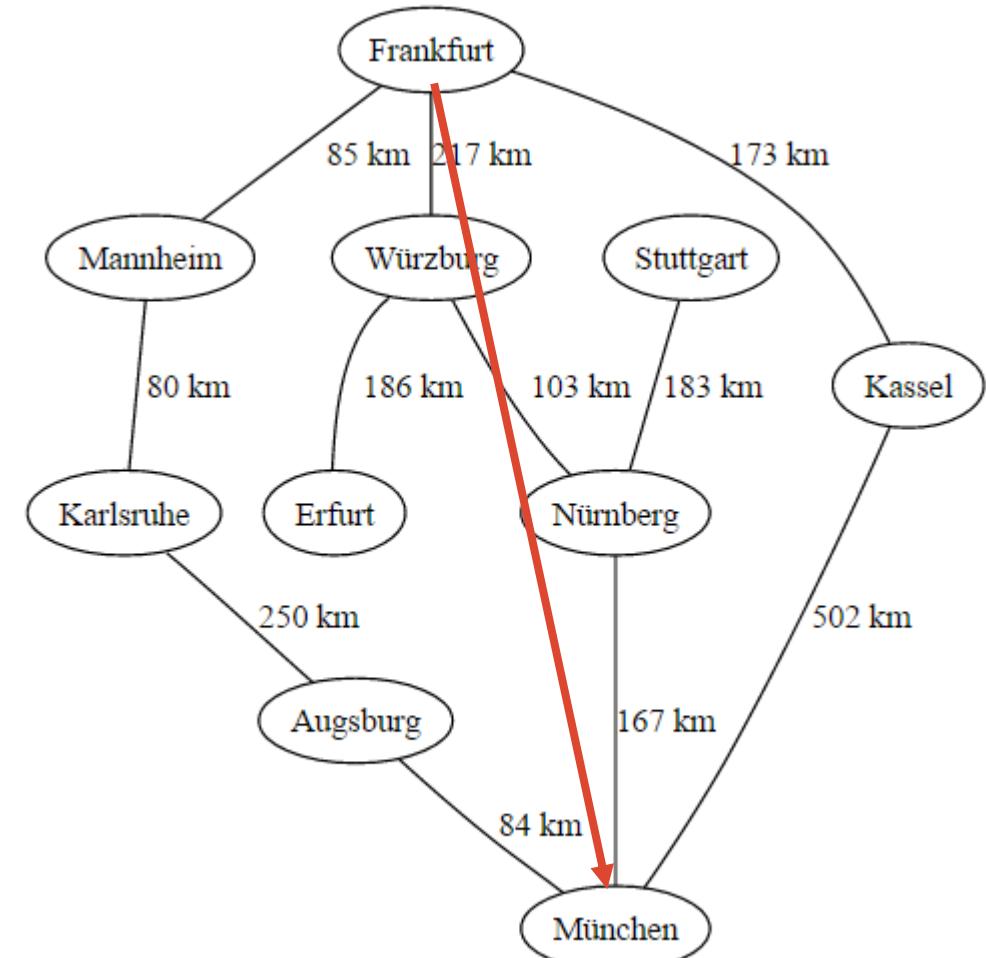


dynamisch



Link-State-Routing-Algorithmus

Wie kommen wir mit geringsten Kosten (in diesem Falle Kilometer) von Frankfurt nach München?



Dijkstra-Algorithmus

Idee:

Die Grundidee des Algorithmus ist es, immer derjenigen Kante zu folgen, die den kürzesten Streckenabschnitt vom Startknoten aus verspricht. Andere Kanten werden erst dann verfolgt, wenn alle kürzeren Streckenabschnitte beachtet wurden. Dieses Vorgehen gewährleistet, dass bei Erreichen eines Knotens kein kürzerer Pfad zu ihm existieren kann. Eine einmal berechnete Distanz zwischen dem Startknoten und einem besuchten Knoten wird nicht mehr geändert. Distanzen zu noch nicht abgearbeiteten Knoten können sich hingegen im Laufe des Algorithmus durchaus verändern, nämlich verringern. Dieses Vorgehen wird fortgesetzt, bis die Distanz des Zielknotens berechnet wurde (**single-pair shortest path**) oder die Distanzen aller Knoten zum Startknoten bekannt sind (**single-source shortest path**).

Dijkstra-Algorithmus

Informell:

1. Weise allen Knoten die beiden Eigenschaften „Distanz“ und „Vorgänger“ zu. Initialisiere die Distanz im Startknoten mit 0 und in allen anderen Knoten mit ∞ .
2. Solange es noch unbesuchte Knoten gibt, wähle darunter denjenigen mit minimaler Distanz aus und
 1. speichere, dass dieser Knoten schon besucht wurde.
 2. Berechne für alle noch unbesuchten Nachbarknoten die Summe des jeweiligen Kantengewichtes und der Distanz im aktuellen Knoten.
 3. Ist dieser Wert für einen Knoten kleiner als die dort gespeicherte Distanz, aktualisiere sie und setze den aktuellen Knoten als Vorgänger. Dieser Schritt wird auch als Update oder Relaxieren bezeichnet.

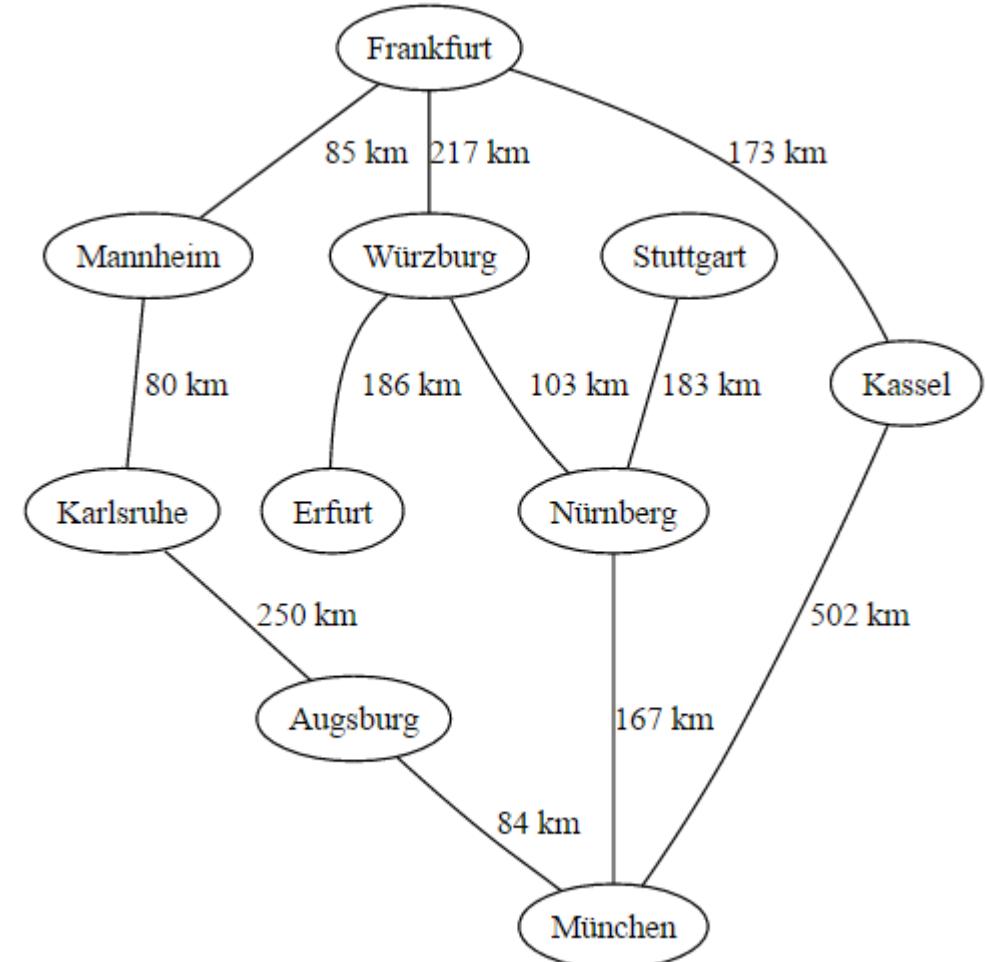
Dijkstra-Algorithmus Beispiel

- Die Zahlen auf den Verbindungen zwischen zwei Städten geben jeweils die Entfernung zwischen den beiden durch die Kante verbundenen Städten an.
- Die Zahlen hinter den Städtenamen geben die ermittelte Distanz der Stadt zum Startknoten Frankfurt an, oo steht dabei für ∞ , also für unbekannte Distanz.
- Die hellgrau unterlegten Knoten sind die Knoten, deren Abstand relaxiert wird (also verkürzt wird, falls eine kürzere Strecke gefunden wurde).
- Die dunkelgrau unterlegten Knoten sind diejenigen, zu denen der kürzeste Weg von Frankfurt bereits bekannt ist.
- Die Auswahl des nächsten Nachbarn erfolgt nach dem Prinzip einer Prioritätswarteschlange. Relaxierte Abstände erfordern daher eine Neusortierung.

Dijkstra-Algorithmus Beispiel

Ausgangssituation:

- Nicht-initialisierter Graph
- Startknoten: Frankfurt
- Zielknoten: München

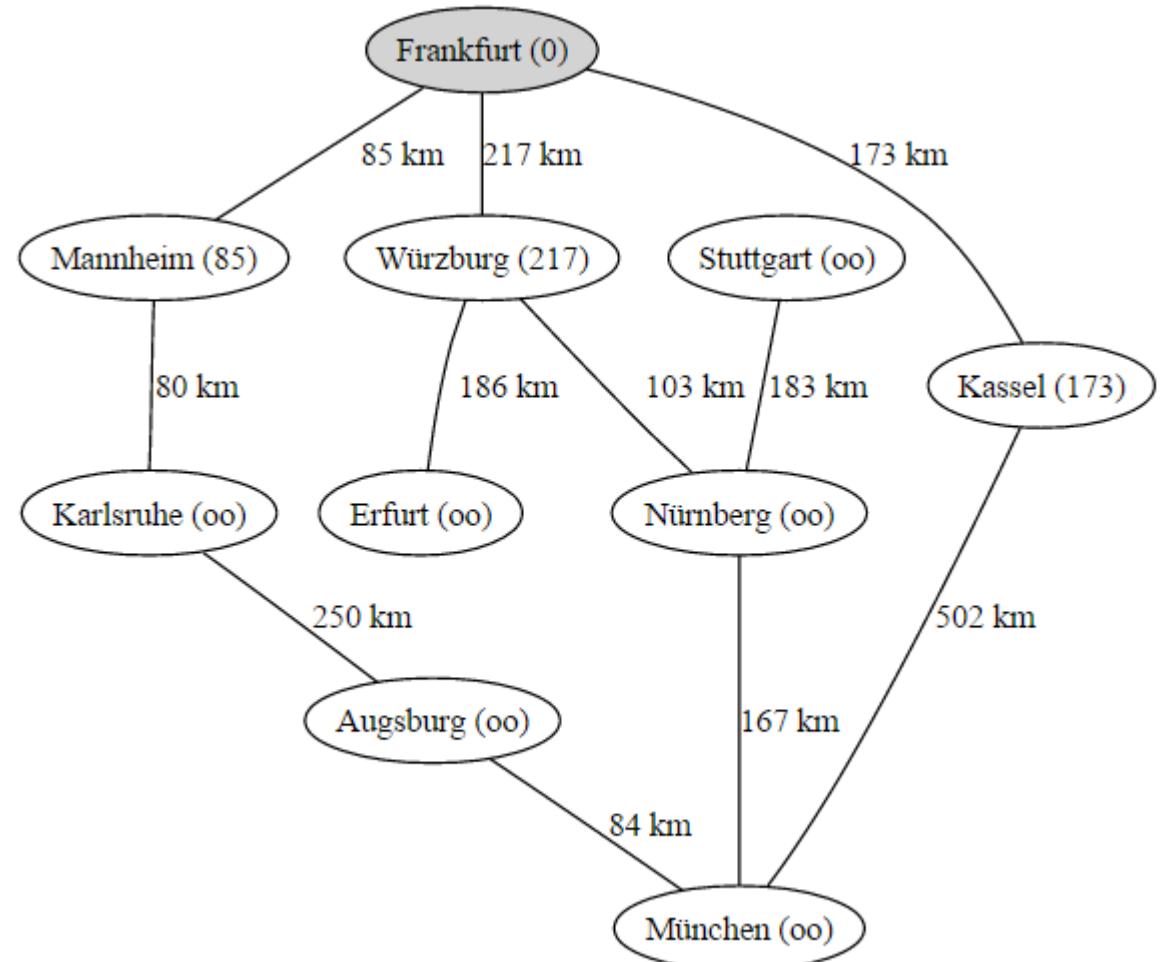


Dijkstra-Algorithmus Beispiel

- Entfernungen vom Startknoten (Frankfurt) ermitteln (relax)
- Neusortieren der Prioritätswarteschlange

Prioritätswarteschlange Q:

1. Mannheim
2. Kassel
3. Würzburg
4. ...

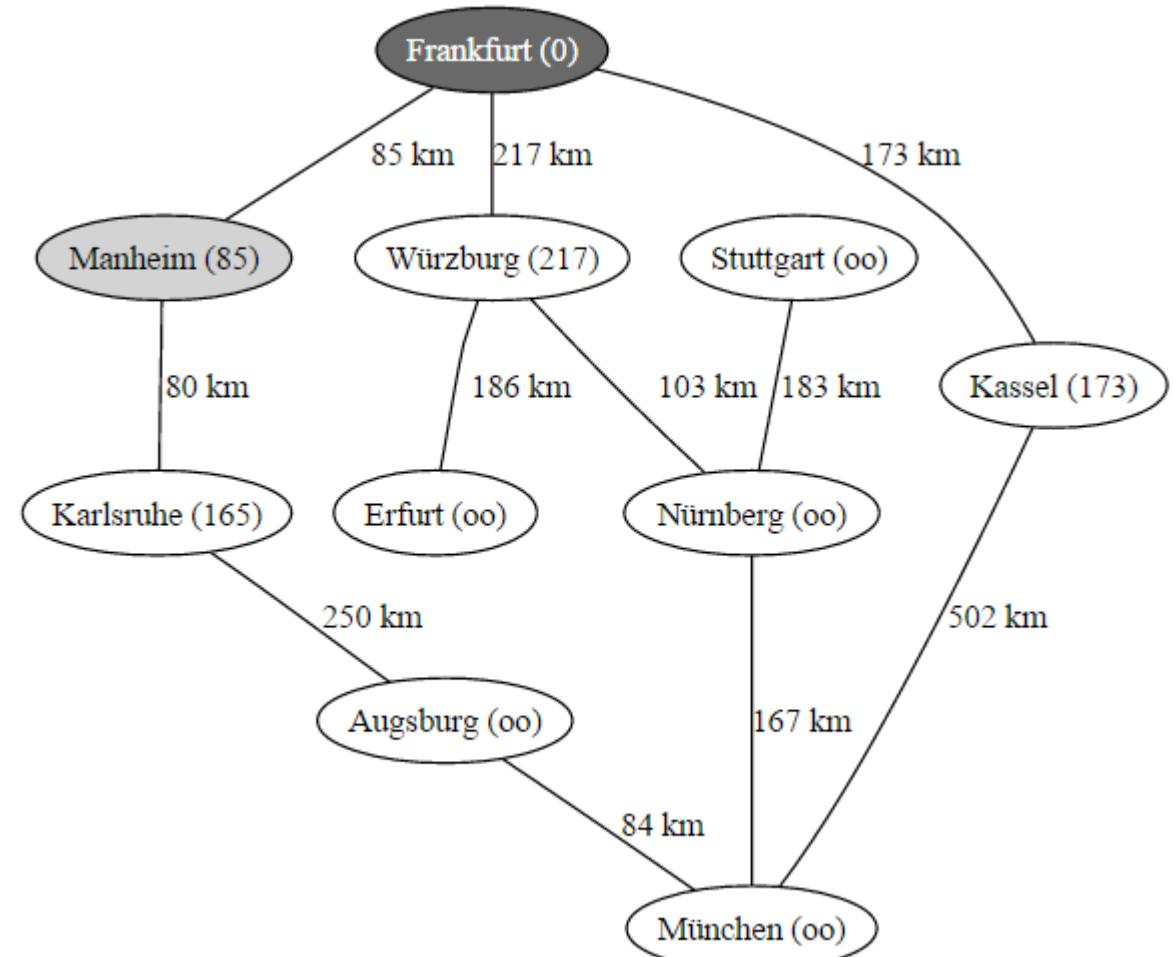


Dijkstra-Algorithmus Beispiel

- nächstliegender Knoten: Mannheim
- relax mit: Karlsruhe
- nächster Vorgänger von Karlsruhe ist nun Mannheim
- Neusortieren der Prioritätswarteschlange

Prioritätswarteschlange Q:

1. Karlsruhe
2. Kassel
3. Würzburg
4. ...

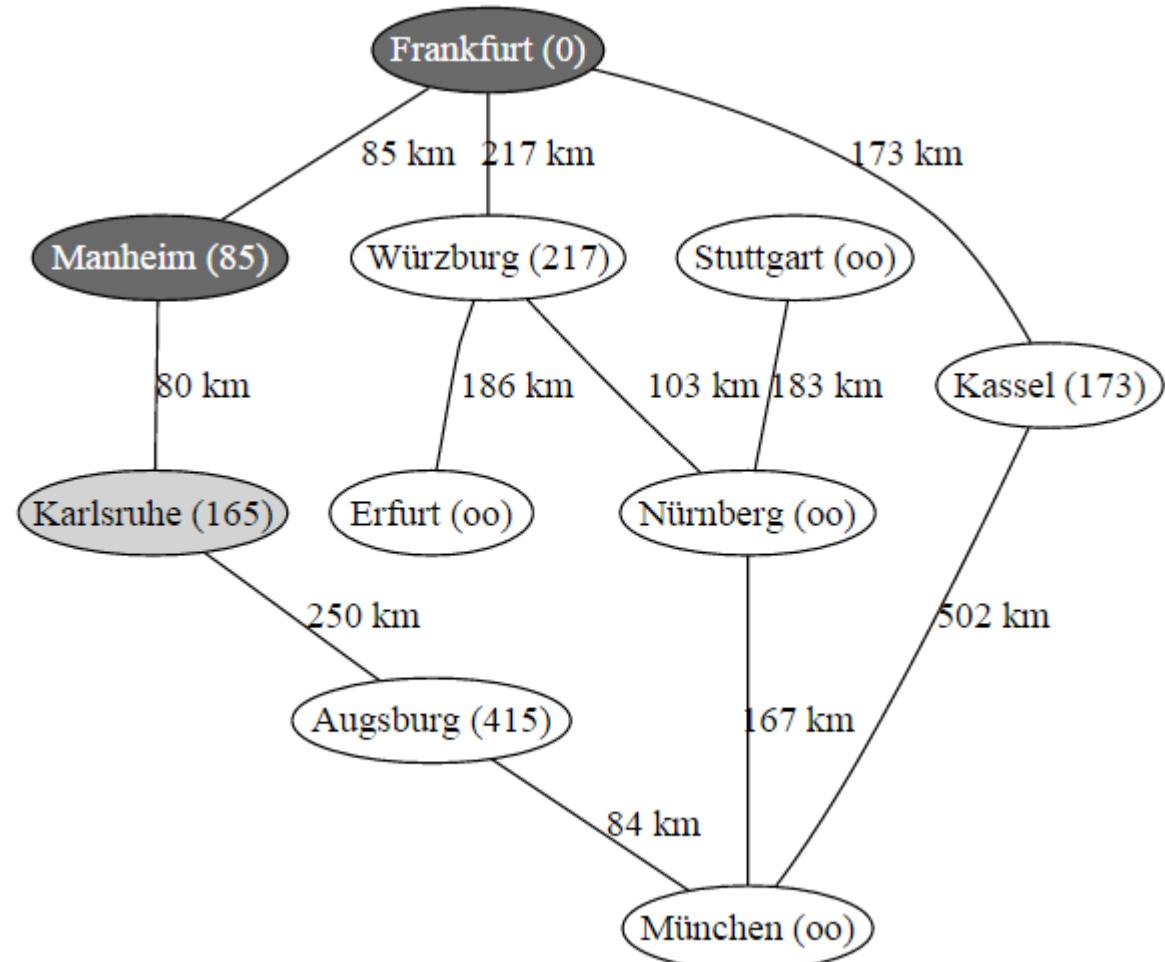


Dijkstra-Algorithmus Beispiel

- nächstliegender Knoten: Karlsruhe
- relax mit: Augsburg
- Neusortieren der Prioritätswarteschlange

Prioritätswarteschlange Q:

1. Kassel
2. Würzburg
3. Augsburg
4. ...

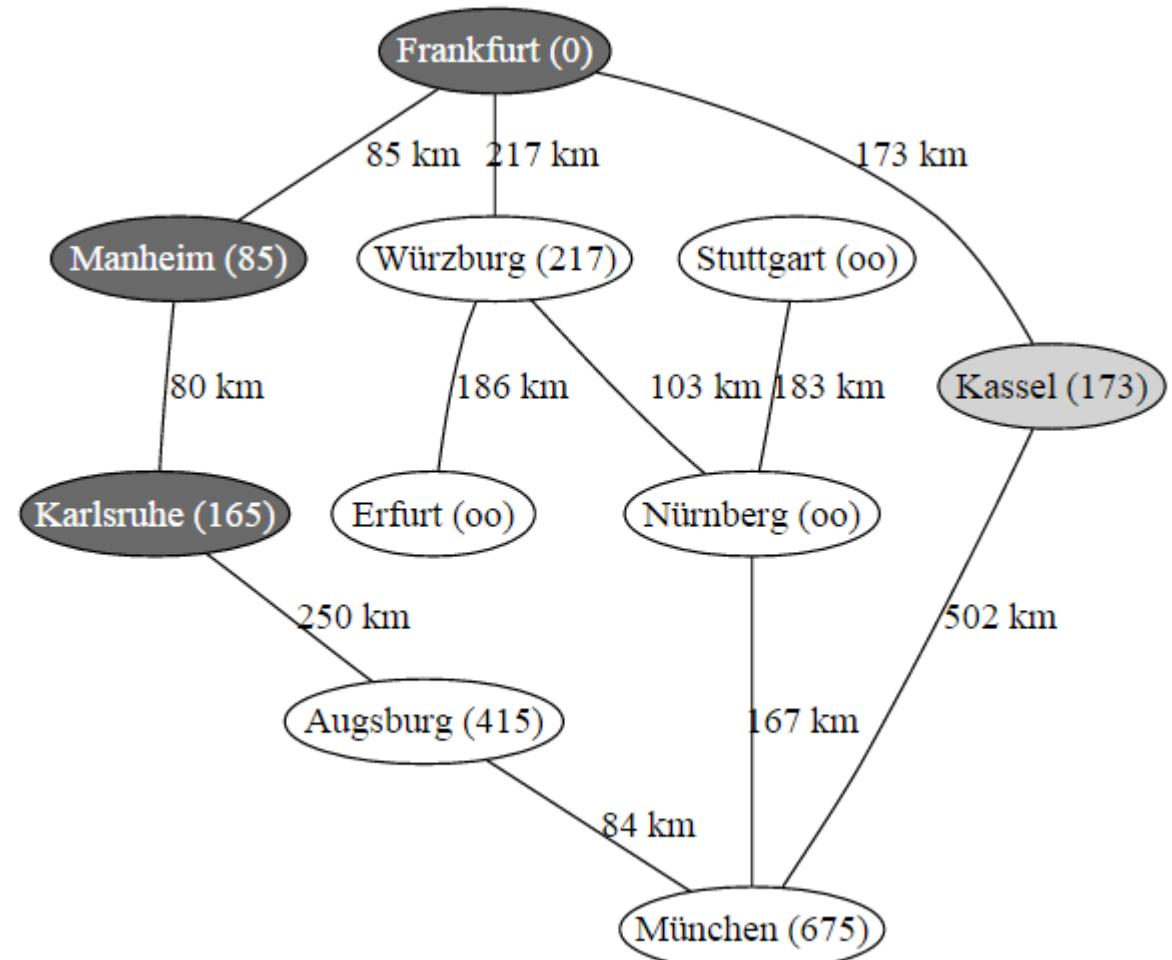


Dijkstra-Algorithmus Beispiel

- nächstliegender Knoten: Kassel
- relax mit: München
- Neusortieren der Prioritätswarteschlange

Prioritätswarteschlange Q:

1. Würzburg
2. Augsburg
3. München
4. ...

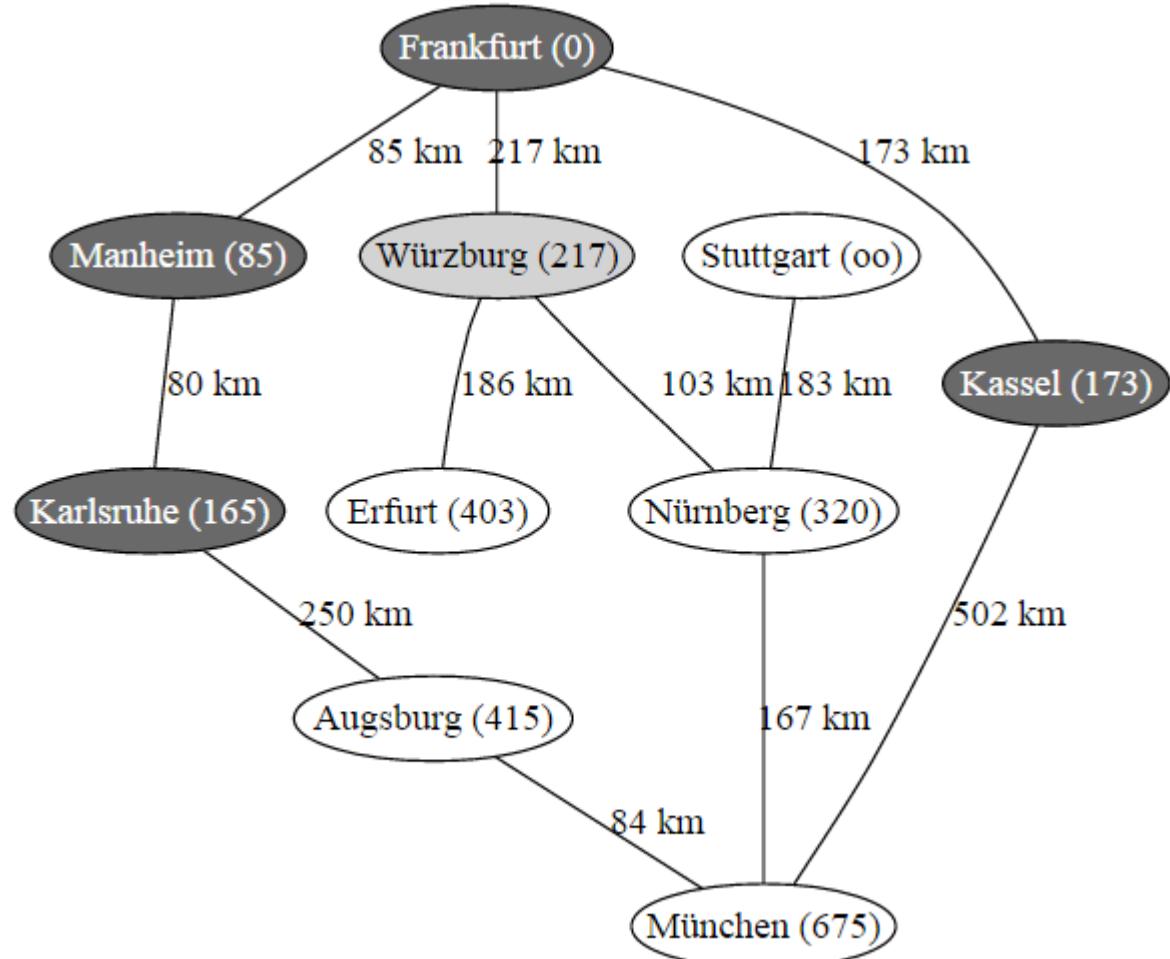


Dijkstra-Algorithmus Beispiel

- nächstliegender Knoten: Würzburg
- relax mit: Erfurt und Nürnberg
- Neusortieren der Prioritätswarteschlange

Prioritätswarteschlange Q:

1. Nürnberg
2. Erfurt
3. Augsburg
4. München
5. ...

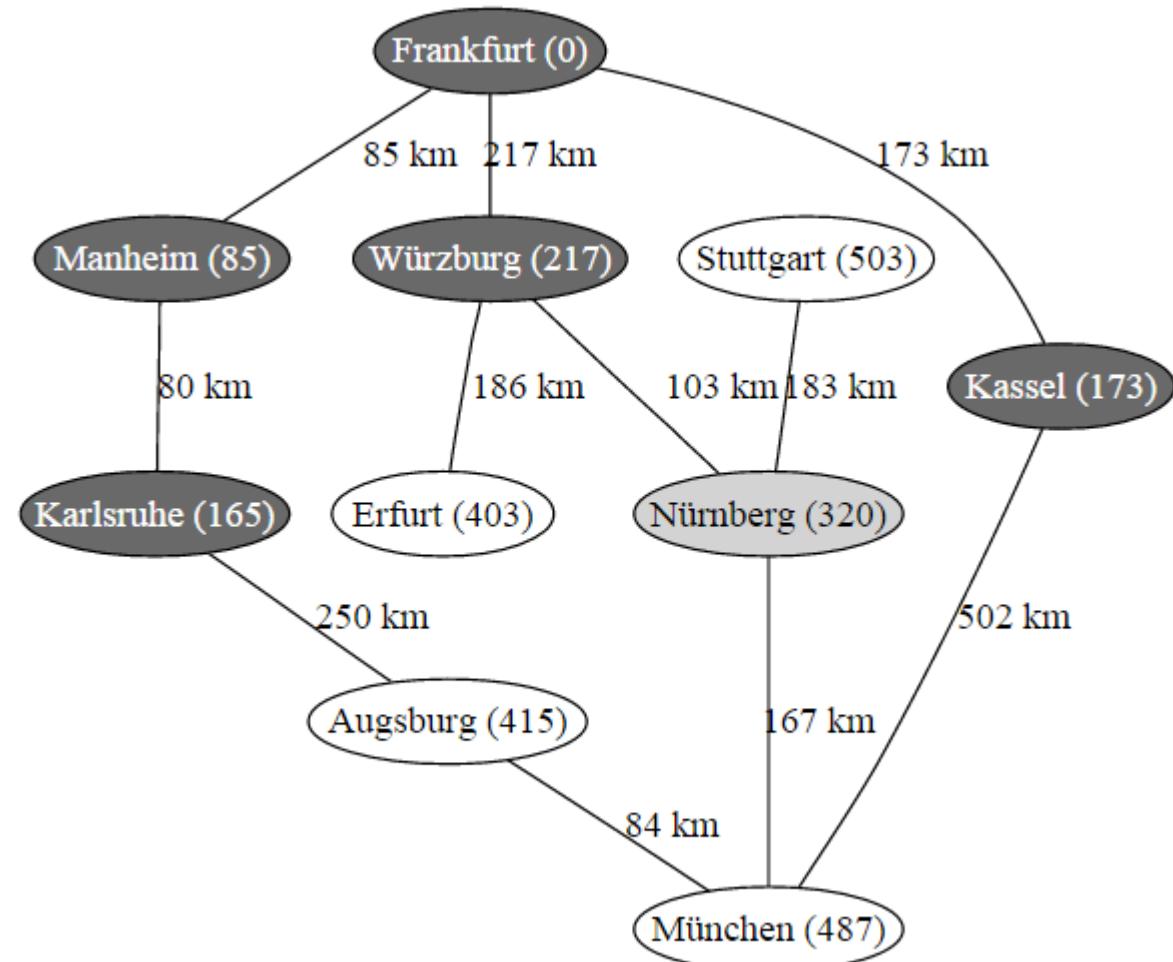


Dijkstra-Algorithmus Beispiel

- nächstliegender Knoten: Nürnberg
- relax mit: Stuttgart und München
- Neusortieren der Prioritätswarteschlange

Prioritätswarteschlange Q:

1. Erfurt
2. Augsburg
3. München
4. Stuttgart
5. ...

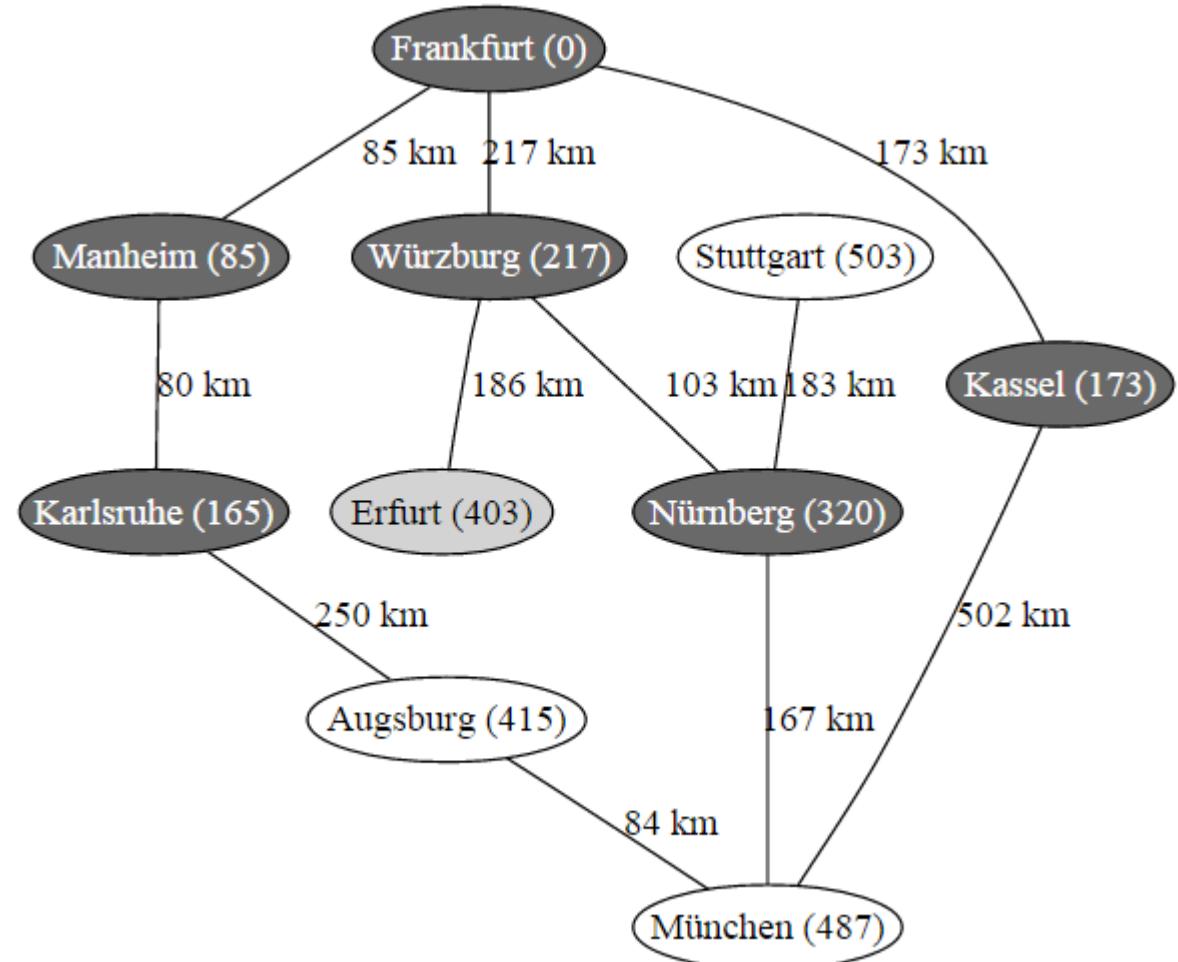


Dijkstra-Algorithmus Beispiel

- nächstliegender Knoten: Erfurt
- relax mit: ---
- Neusortieren der Prioritätswarteschlange

Prioritätswarteschlange Q:

1. Augsburg
2. München
3. Stuttgart
4. ...

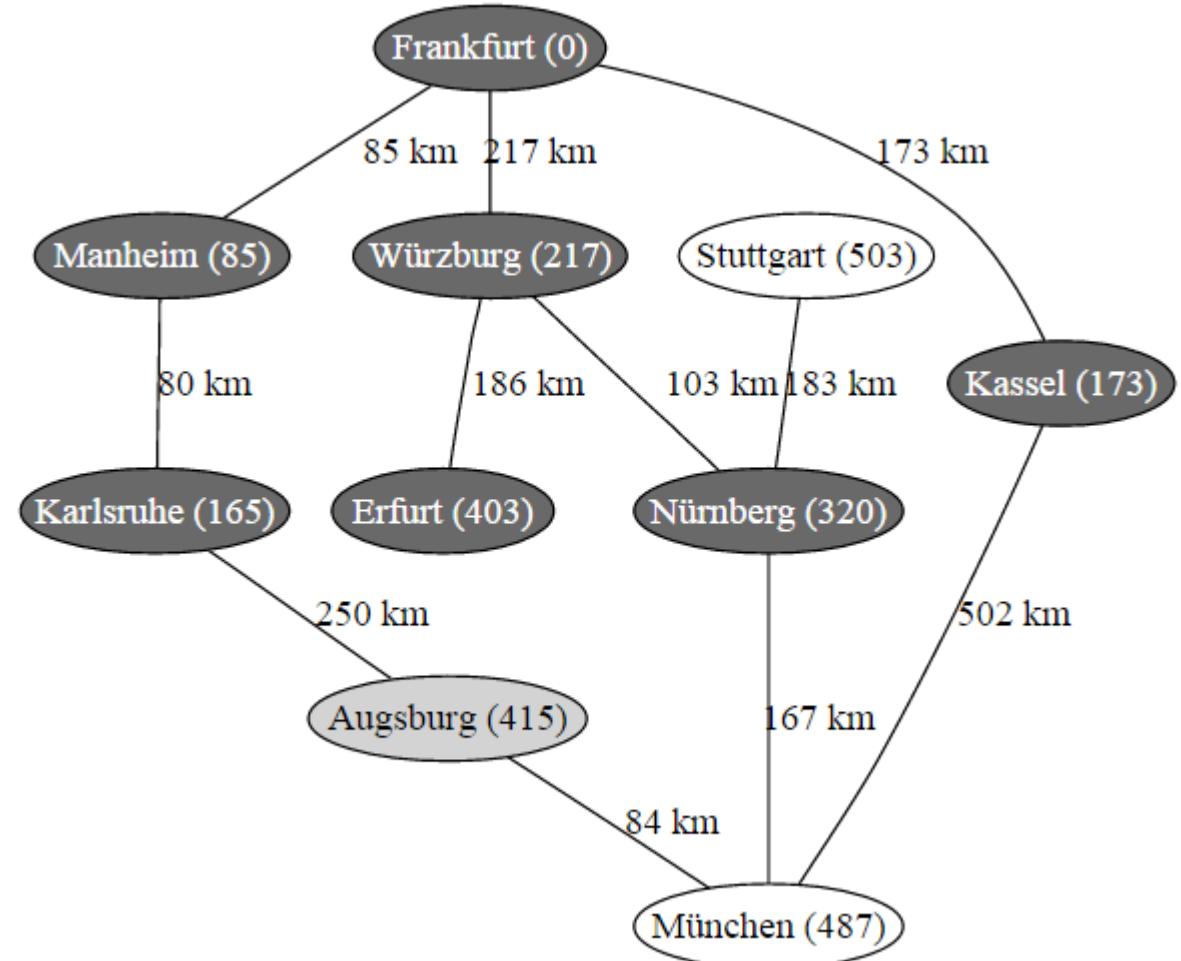


Dijkstra-Algorithmus Beispiel

- nächstliegender Knoten: Augsburg
- relax mit: München
- Neusortieren der Prioritätswarteschlange

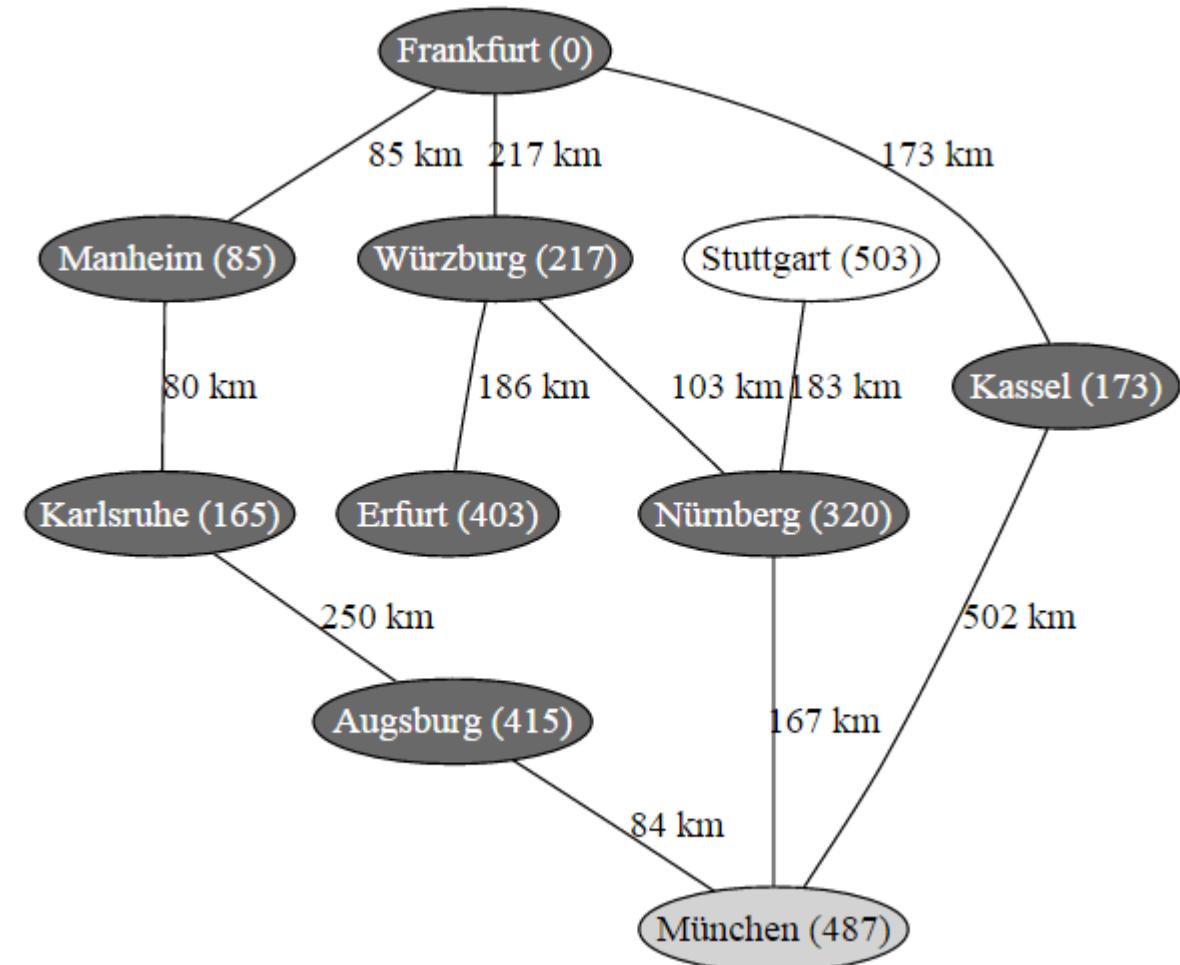
Prioritätswarteschlange Q:

1. München
2. Stuttgart
3. ...



Dijkstra-Algorithmus Beispiel

- Zielknoten soll untersucht werden:
Kürzester Weg nach München ist nun bekannt
- Rekonstruktion mittels erstelleKürzestenPfad()
→ Iteration über die Vorgängerknoten
- Frankfurt-Würzburg-Nürnberg-München



Dijkstra-Algorithmus Pseudocode

Funktion Dijkstra(Graph, Startknoten) :

```
initialisiere(Graph, Startknoten, abstand[], vorgänger[], Q)
solange Q nicht leer: // Der eigentliche Algorithmus
    u:= Knoten in Q mit kleinstem Wert in abstand[]
    entferne u aus Q // für u ist der kürzeste Weg nun bestimmt
    für jeden Nachbarn v von u:
        falls v in Q:
            distanz_update(u, v, abstand[], vorgänger[])
            // prüfe Abstand vom Startknoten zu v
return vorgänger[]
```

Dijkstra-Algorithmus Pseudocode

```
Methode initialisiere(Graph, Startknoten, abstand[], vorgänger[], Q):  
    für jeden Knoten v in Graph:  
        abstand[v] := unendlich  
        vorgänger[v] := null  
    abstand[Startknoten] := 0  
    Q := Die Menge aller Knoten in Graph
```

Dijkstra-Algorithmus Pseudocode

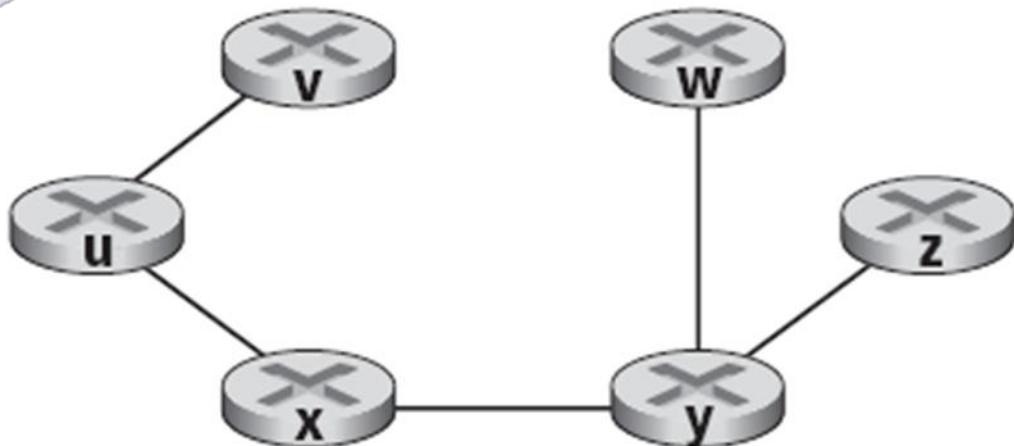
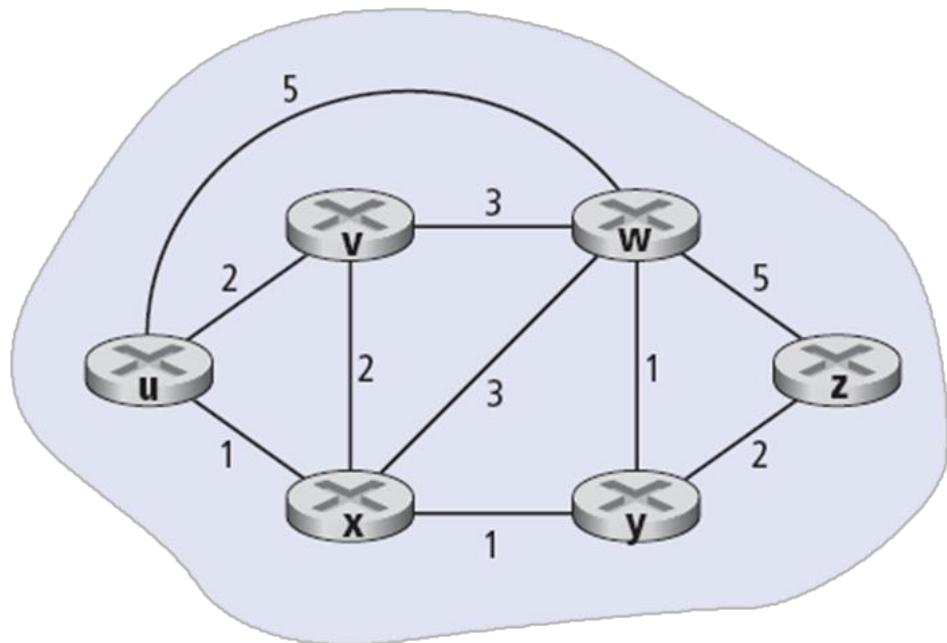
```
Methode distanz_update(u, v, abstand[], vorgänger[]):  
    alternativ := abstand[u] + abstand_zwischen(u, v)  
    // Weglänge vom Startknoten nach v über u  
    falls alternativ < abstand[v]:  
        abstand[v] := alternativ  
        vorgänger[v] := u
```

Dijkstra-Algorithmus Pseudocode

```
Funktion erstelleKürzestenPfad(Zielknoten, vorgänger[])
```

```
Weg[] := [Zielknoten]
u := Zielknoten
solange vorgänger[u] nicht null:
    // Der Vorgänger des Startknotens ist null
    u := vorgänger[u]
    füge u am Anfang von Weg[] ein
return Weg[]
```

Dijkstra-Algorithmus Beispiel



Ziel	Leitung
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

Übungsaufgabe Dijkstra-Algorithmus

Aufgabe:

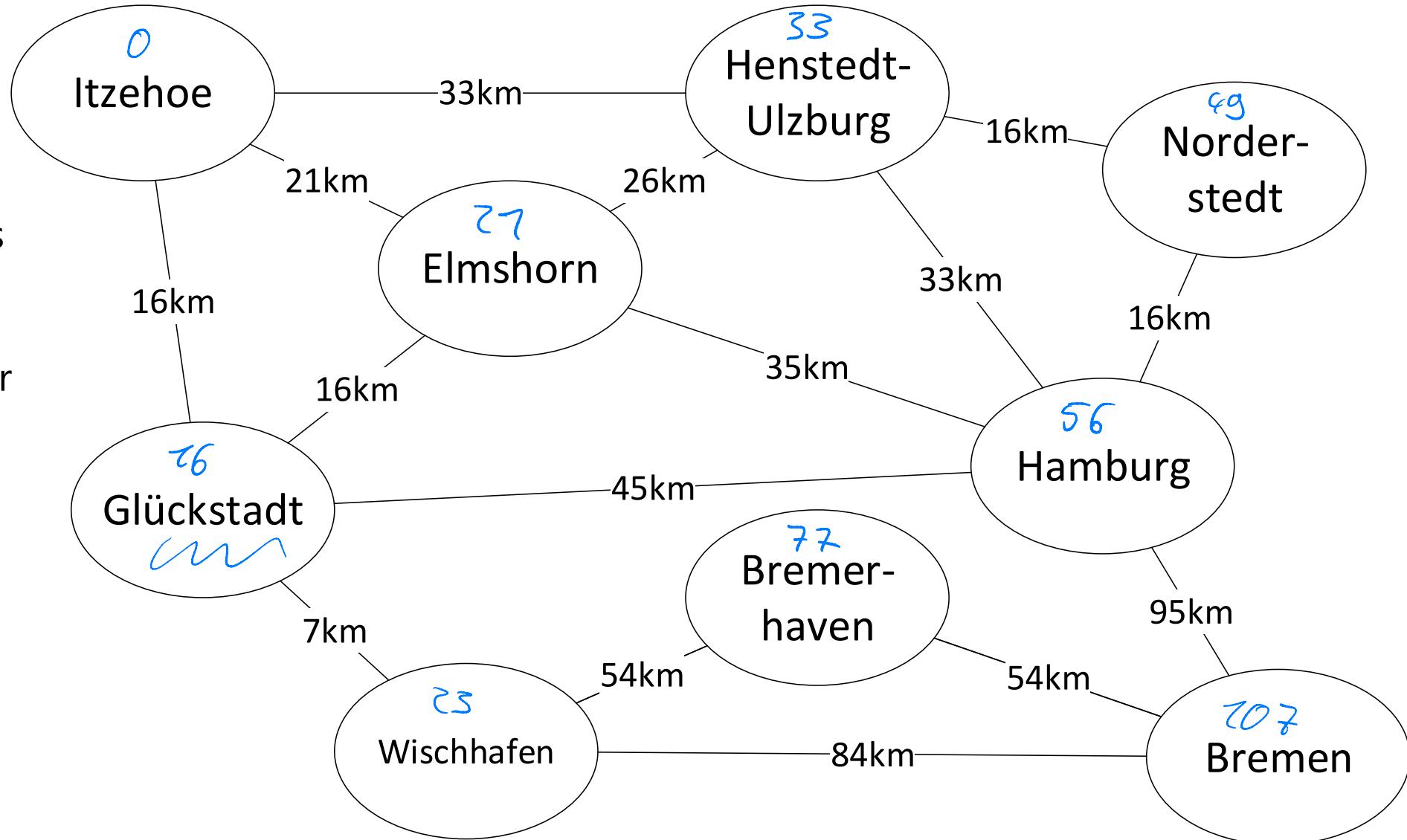
Benutze den Dijkstra-Algorithmus um ausgehend vom Startort die jeweiligen Vorgänger zu ermitteln

Start Student A:

- Itzehoe

Start Student B:

- Norderstedt



Startort: Itzehoe

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
1	Itzehoe	Glückstadt	Itzehoe	16	Glückstadt
		Elmshorn	Itzehoe	21	Elmshorn
		Hennstedt-Ulzburg	Itzehoe	33	Hennstedt-Ulzburg
					...

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
2	Glückstadt	Elmshorn	Itzehoe	21	Elmshorn
	Vorgänger	Wischhafen	Glückstadt	23	Wischhafen
	Itzehoe	Hamburg	Glückstadt	61	Hennstedt-Ulzburg
	Entfernung				Hamburg
	16				...

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
3	Elmshorn	Hennstedt-Ulzburg	Itzehoe	33	Wischhafen
	Vorgänger	Hamburg	Elmshorn	56	Hennstedt-Ulzburg
	Itzehoe				Hamburg
	Entfernung				...
	21				

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
4	Wischhafen	Bremerhaven	Wischhafen	77	Hennstedt-Ulzburg
	Vorgänger	Bremen	Wischhafen	107	Hamburg
	Glückstadt				Bremen
	Entfernung				Bremerhaven
	23				Bremen

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
5	Hennstedt-Ulzburg	Norderstedt	Hennstedt-Ulzburg	49	Norderstedt
	Vorgänger	Hamburg	Elmshorn	56	Hamburg
	Itzehoe				Bremerhaven
	Entfernung				Bremen
	33				...

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
6	Norderstedt	Hamburg	Elmshorn	56	Hamburg
	Vorgänger				Bremerhaven
	Hennstedt-Ulzburg				Bremen
	Entfernung				...
	49				

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
7	Hamburg	Bremen	Wischhafen	107	Bremerhaven Bremen ...
	Vorgänger				
	Elmshorn				
	Entfernung				
	56				

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
8	Bremerhaven	Bremen	Wischhafen	107	Bremen
	Vorgänger				
	Wischhafen				
	Entfernung				
	77				

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
9	Bremen				
	Vorgänger				
	Wischhafen				
	Entfernung				
	107				

Startort: Norderstedt

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
1	Norderstedt	Hennstedt-Ulzburg	Norderstedt	16	Hennstedt-Ulzburg Hamburg ...
		Hamburg	Norderstedt	16	

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
2	Hennstedt-Ulzburg	Itzehoe	Hennstedt-Ulzburg	49	Hamburg Elmshorn Itzehoe ...
	Vorgänger	Elmshorn	Hennstedt-Ulzburg	42	
	Norderstedt	Hamburg	Norderstedt	16	
	Entfernung				
	16				

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
3	Hamburg	Elmshorn	Hennstedt-Ulzburg	42	Elmshorn Itzehoe Glückstadt Bremen ...
	Vorgänger	Glückstadt	Hamburg	61	
	Norderstedt	Bremen	Hamburg	111	
	Entfernung				
	16				

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
4	Elmshorn	Itzehoe	Hennstedt-Ulzburg	49	Itzehoe Glückstadt Bremen ...
	Vorgänger	Glückstadt	Elmshorn	58	
	Hennstedt-Ulzburg				
	Entfernung				
	42				

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
5	Itzehoe	Glückstadt	Elmshorn	58	Glückstadt Bremen ...
	Vorgänger				
	Hennstedt-Ulzburg				
	Entfernung				
	49				

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
6	Glückstadt	Wischnhafen	Glückstadt	65	Wischnhafen Bremen ...
	Vorgänger				
	Elmshorn				
	Entfernung				
	58				

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
7	Wischnhafen	Bremerhaven	Wischnhafen	119	Bremen Bremerhaven ...
	Vorgänger	Bremen	Hamburg	111	
	Glückstadt				
	Entfernung				
	65				

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
8	Bremen	Bremerhaven	Wischnhafen	119	Bremerhaven ...
	Vorgänger				
	Hamburg				
	Entfernung				
	111				

Schritt	Aktueller Knoten	Relax	Vorgänger	Entfernung	Prio
9	Bremerhaven				
	Vorgänger				
	Wischnhafen				
	Entfernung				
	119				

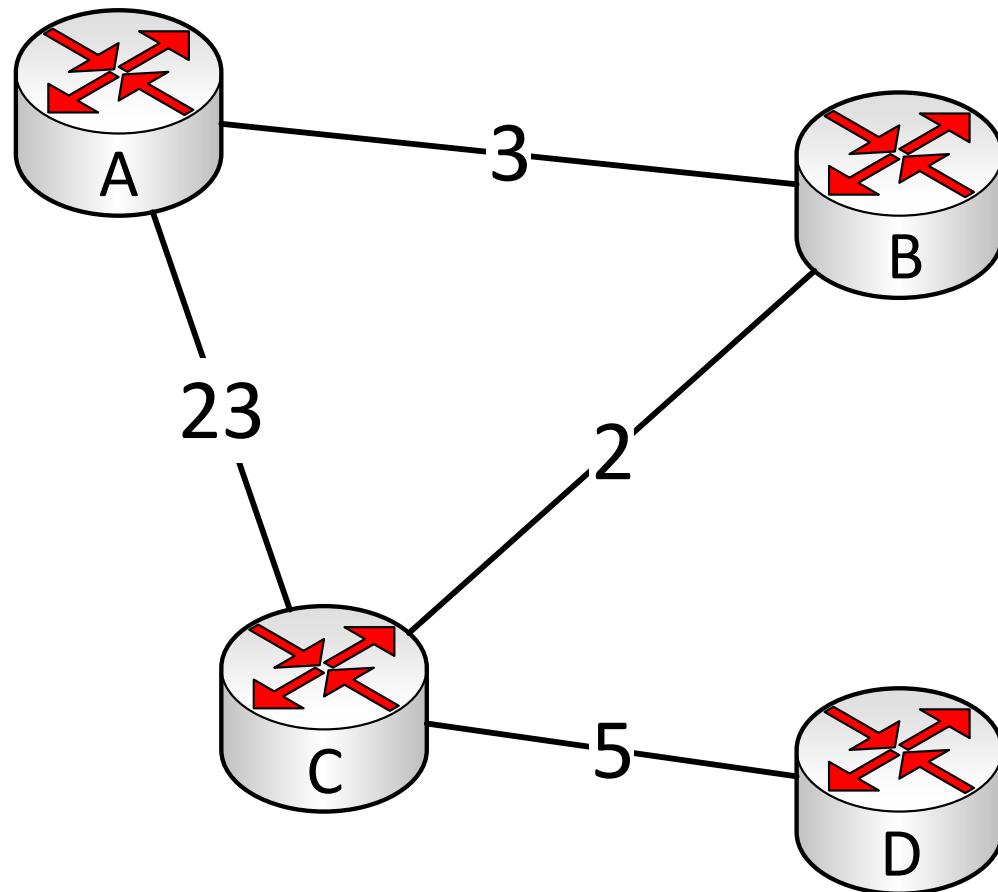
Distanzvektor-Routing

Das prinzipielle Vorgehen eines Distanzvektorprotokolls:

1. Erzeuge eine Kostenmatrix, welche Router über welche Nachbarn und zu welchen Kosten erreichbar sind und anfangs nur die (bekannten) Kosten zu direkten Nachbarn enthält.
2. Erzeuge eine Aufstellung mit Informationen, welche Router wir zu welchen Kosten am besten erreichen können und schicke sie an alle Nachbarn.
3. Warte auf Aufstellungen dieser Art von anderen Routern, rechne diese dann in die eigene Kostenmatrix ein.
4. Ändern sich dadurch die minimalen Kosten, zu denen wir einen Router erreichen können: fahre mit Schritt 2 fort, sonst mit Schritt 3

Siehe Bellman-Ford-Algorithmus: <http://de.wikipedia.org/wiki/Bellman-Ford-Algorithmus>

Distanzvektor-Routing Beispiel



Distanzvektor-Routing Beispiel

Kostenmatrix

von A	via A	via B	via C	via D
zu A				
zu B				
zu C				
zu D				

 = bester Pfad zu einem anderen Router

 = neuer bester Pfad,
wird an die anderen Router übermittelt

 = wird nicht betrachtet

Distanzvektor-Routing Beispiel

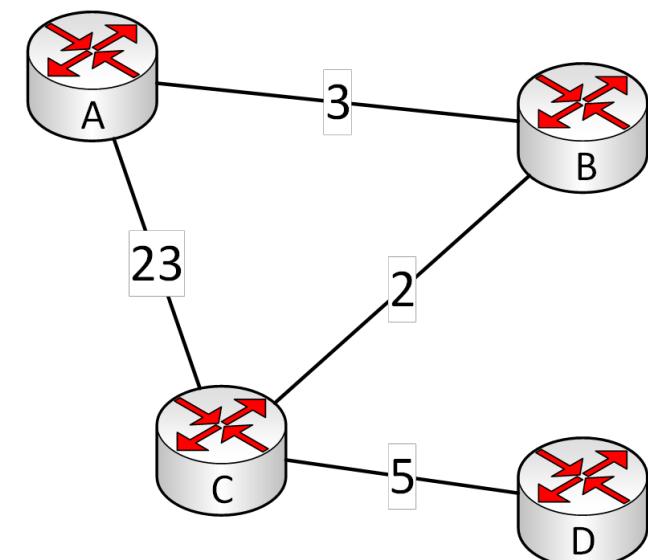
von A	via A	via B	via C	via D
zu A				
zu B		3		
zu C			23	
zu D				

von C	via A	via B	via C	via D
zu A	23			
zu B		2		
zu C				
zu D				5

von B	via A	via B	via C	via D
zu A	3			
zu B				
zu C			2	
zu D				

von D	via A	via B	via C	via D
zu A				
zu B				
zu C			5	
zu D				

$T = 0$



Distanzvektor-Routing Beispiel

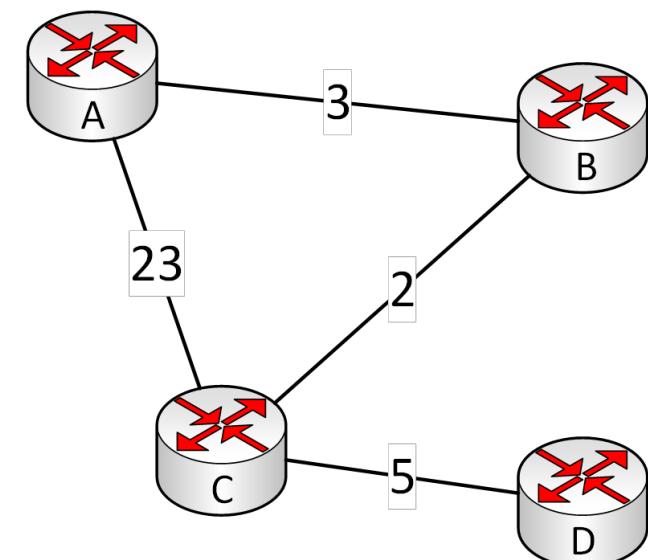
von A	via A	via B	via C	via D
zu A				
zu B		3	25	
zu C		5	23	
zu D			28	

von C	via A	via B	via C	via D
zu A	23	5		
zu B	26	2		
zu C				
zu D				5

von B	via A	via B	via C	via D
zu A	3		25	
zu B				
zu C	26		2	
zu D			7	

von D	via A	via B	via C	via D
zu A			28	
zu B			7	
zu C			5	
zu D				

T = 1



Distanzvektor-Routing Beispiel

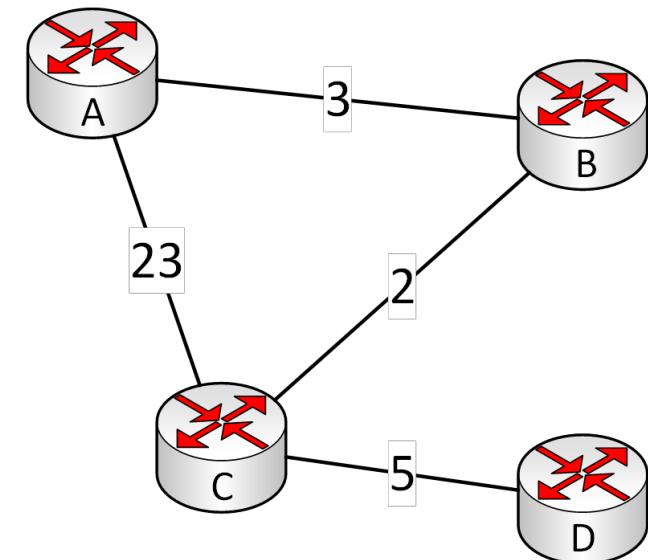
von A	via A	via B	via C	via D
zu A				
zu B		3	25	
zu C		5	23	
zu D		10	28	

von C	via A	via B	via C	via D
zu A	23	5		33
zu B	26	2		12
zu C				
zu D	51	9		5

von B	via A	via B	via C	via D
zu A	3		7	
zu B				
zu C	8		2	
zu D	31		7	

von D	via A	via B	via C	via D
zu A			10	
zu B			7	
zu C			5	
zu D				

T = 2



Distanzvektor-Routing Beispiel

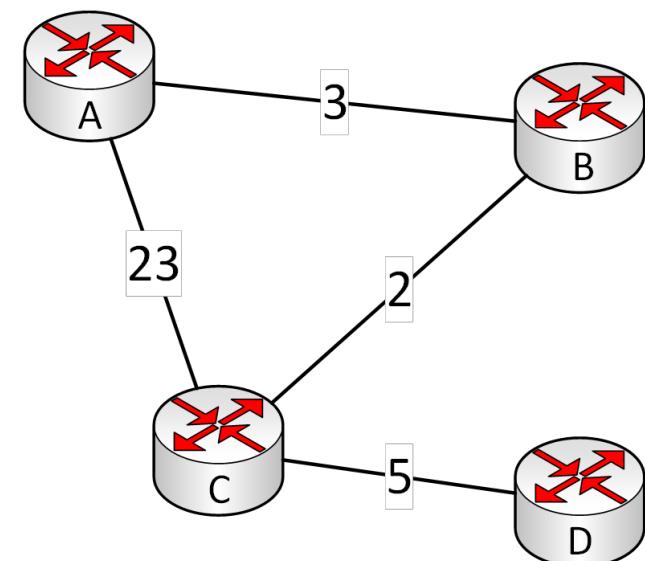
von A	via A	via B	via C	via D
zu A				
zu B		3	25	
zu C		5	23	
zu D		10	28	

von C	via A	via B	via C	via D
zu A	23	5		15
zu B	26	2		12
zu C				
zu D	33	9		5

von B	via A	via B	via C	via D
zu A	3		7	
zu B				
zu C	8		2	
zu D	13		7	

von D	via A	via B	via C	via D
zu A			10	
zu B			7	
zu C			5	
zu D				

$T = 3$



Distanzvektor-Routing

Prinzipielles Vorgehen:

- Wenn ein Knoten hochgefahren wird, dann sendet ein Knoten seinen eigenen Distanzvektor an alle seine Nachbarn
- Immer wenn sich der Distanzvektor eines Knotens ändert, sendet er diesen an alle seine Nachbarn

Wenn ein Knoten einen neuen Distanzvektor von einem Nachbarn erhält, überprüft er seinen eigenen Distanzvektor nach der Bellman-Ford-Gleichung:

$$d_x(y) = \min \{c(x,v) + d_v(y) \} \quad // \text{ Minimum wird über alle Nachbarn } v \text{ gebildet}$$

Unter realistischen Annahmen konvergiert dieser Algorithmus zu einer Situation, in der jeder Knoten den Nachbarn auf dem günstigsten Weg zu jedem anderen Knoten kennt.

Distanzvektor-Routing

Iterativ und asynchron:

- Jede Iteration wird ausgelöst durch:
 - Veränderung der Kosten eines Links zu einem direkten Nachbarn
 - Neuer Distanzvektor von einem Nachbarn

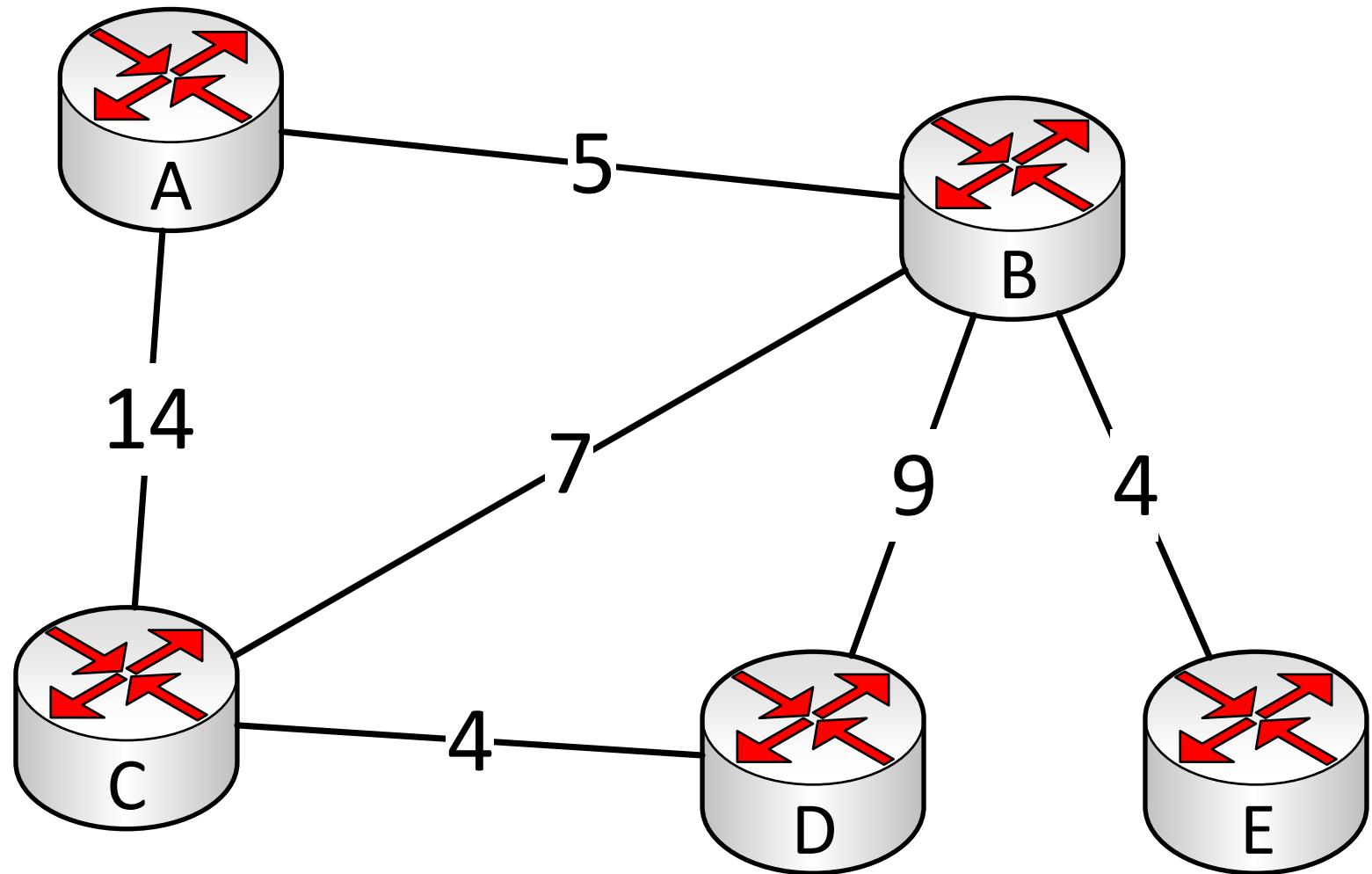
Verteilt:

- Keine globale Kenntnis der vollständigen Netzwerk-topologie notwendig
- Stattdessen: lokales Verbreiten von Informationen

Übungsaufgabe Distanzvektor-Routing

Aufgabe:

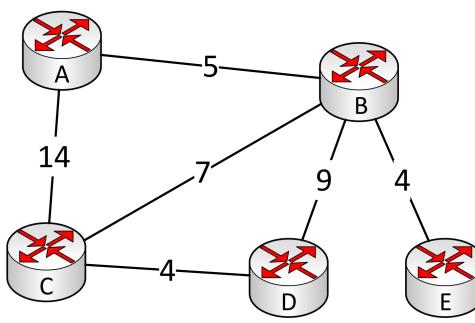
Erstelle die Kostenmatrizen
zu allen Zeitpunkten ($T=0$,
 $T=1$, usw.) bis zur
Konvergenz!



<i>T = 0</i>		<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
<i>vic1</i>	<i>mach</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
—	—	—	—	—	—	—
<i>B</i>	<i>— 5</i>	—	—	—	—	—
<i>C</i>	<i>—</i>	<i>72</i>	<i>4</i>	—	—	—
<i>D</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>—</i>
<i>E</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>—</i>

<i>T = 1</i>		<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
<i>vic1</i>	<i>mach</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
—	—	—	—	—	—	—
<i>B</i>	<i>— 5</i>	—	—	—	—	—
<i>C</i>	<i>—</i>	<i>72</i>	<i>4</i>	—	—	—
<i>D</i>	<i>—</i>	<i>79</i>	<i>78</i>	—	—	—
<i>E</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>—</i>

<i>T = 2</i>		<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
<i>vic1</i>	<i>mach</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
—	—	—	—	—	—	—
<i>B</i>	<i>— 5</i>	—	—	—	—	—
<i>C</i>	<i>—</i>	<i>72</i>	<i>4</i>	—	—	—
<i>D</i>	<i>—</i>	<i>79</i>	<i>78</i>	—	—	—
<i>E</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>—</i>	<i>—</i>



Vergleich Link-State / Distanzvektor

	Link-State	Distanzvektor
Anzahl Nachrichten	n Knoten, E Kanten: $O(nE)$ Nachrichten	<ul style="list-style-type: none">• Nachrichten werden nur mit Nachbarn ausgetauscht• Zeit bis zur Konvergenz variiert
Geschwindigkeit der Konvergenz	<ul style="list-style-type: none">• Fluten der Zustände der Links• Kann oszillieren	<ul style="list-style-type: none">• variiert stark• Temporäre Routing-Schleifen sind möglich• Count-to-infinity-Problem
Robustheit (Fehlerhafter Router)	<ul style="list-style-type: none">• Knoten kann falsche Kosten für einen Link fluten• Pfade möglicherweise nicht mehr optimal	<ul style="list-style-type: none">• Router kann falsche Kosten für einen ganzen Pfad ankündigen• Fehler propagiert durch das ganze Netzwerk, insbesondere wenn die (falschen) Kosten klein sind• u. U. katastrophal

Hierarchisches Routing

Bisher:

- Alle Router sind gleich
- Das Netzwerk ist „flach“ und besitzt keine Hierarchie

Entspricht nicht der Realität!

Eine Frage der Größenordnung:

- 200 Millionen Zielnetzwerke
- Können nicht alle in einer Routing-Tabelle gespeichert sein
- Routing-Protokolle würden alle Links im Internet überlasten

Eine Frage der Administration:

- Internet = Netzwerk von Netzwerken
- Jede Organisation hat eigene Politiken und Präferenzen bezüglich ihres Netzwerkes

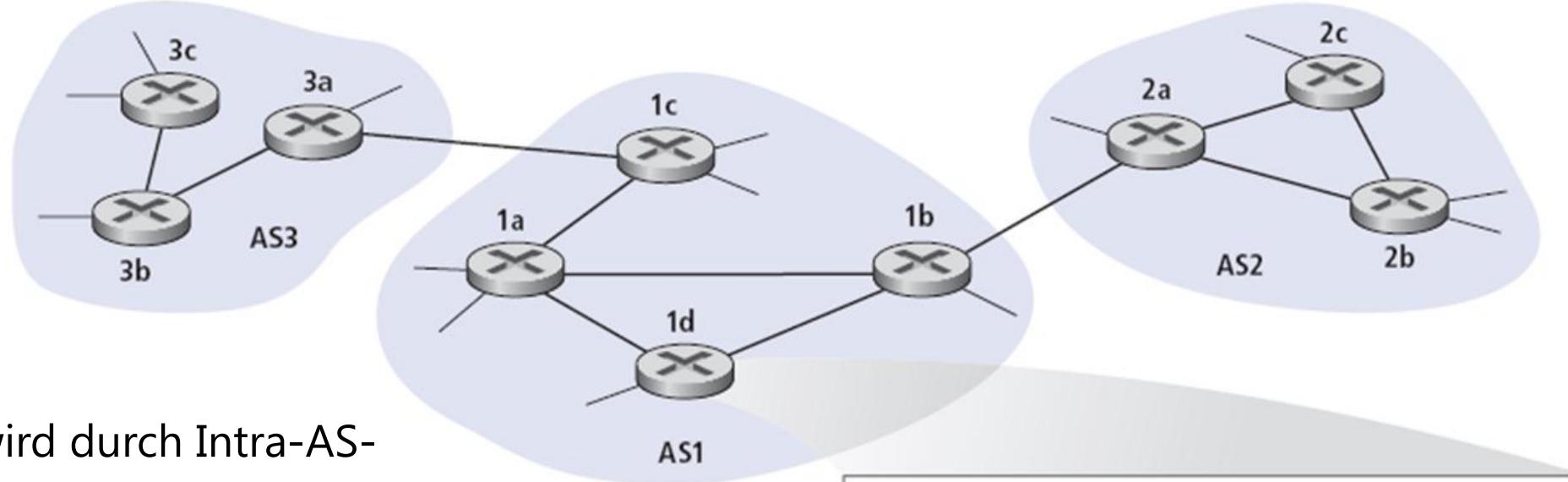
Hierarchisches Routing

- Router werden zu Regionen zusammengefasst, diese nennt man **autonome Systeme (AS)**
- Router innerhalb eines AS verwenden ein Routing-Protokoll
 - "Intra-AS"-Routing-Protokoll
 - Router in verschiedenen AS können verschiedene Intra-AS-Routing-Protokolle verwenden
- Manchmal gilt:
 - Eine Organisation = ein AS
 - Es gibt aber auch Organisationen (z.B. einige ISPs), die aus mehreren AS bestehen

Hierarchisches Routing

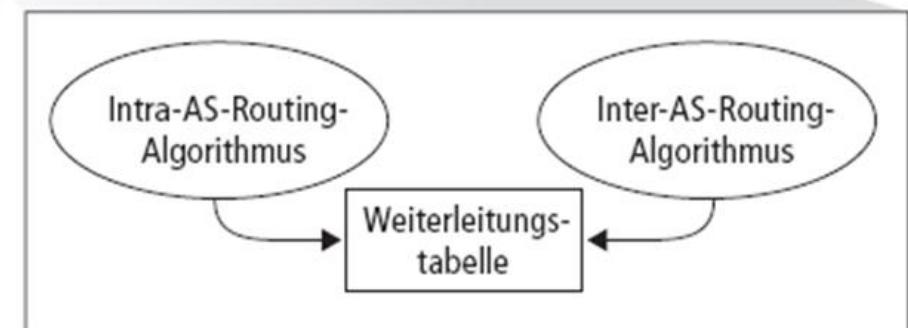
- Gateway-Router:
 - Ein Router in einem AS, der eine Verbindung zu einem Router in einem anderen AS hat
- Problem: Routing zwischen AS
 - „Inter-AS“-Routing-Protokoll

Verbundene Autonome Systeme



Routing-Tabelle wird durch Intra-AS- und Inter-AS-Routing-Algorithmen gefüllt:

- Intra-AS-Einträge für interne Ziele
- Inter-AS- & Intra-AS-Einträge für externe Ziele



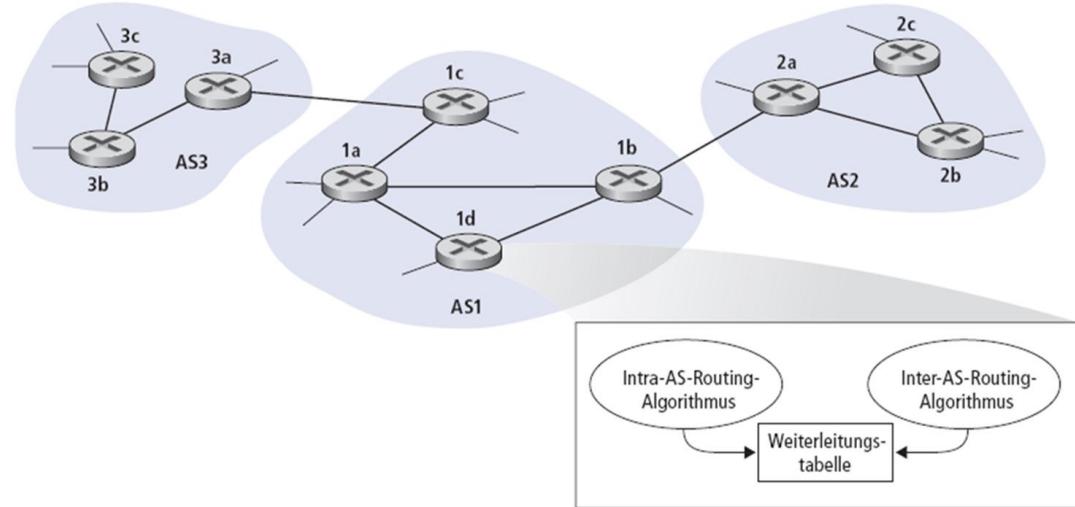
Aufgaben des Inter-AS-Routing

Wenn ein Router in AS1 ein Paket für ein Ziel außerhalb von AS1 erhält:

- Router sollte das Paket zu einem der Gateway-Router in AS1 weiterleiten
- Aber zu welchem?

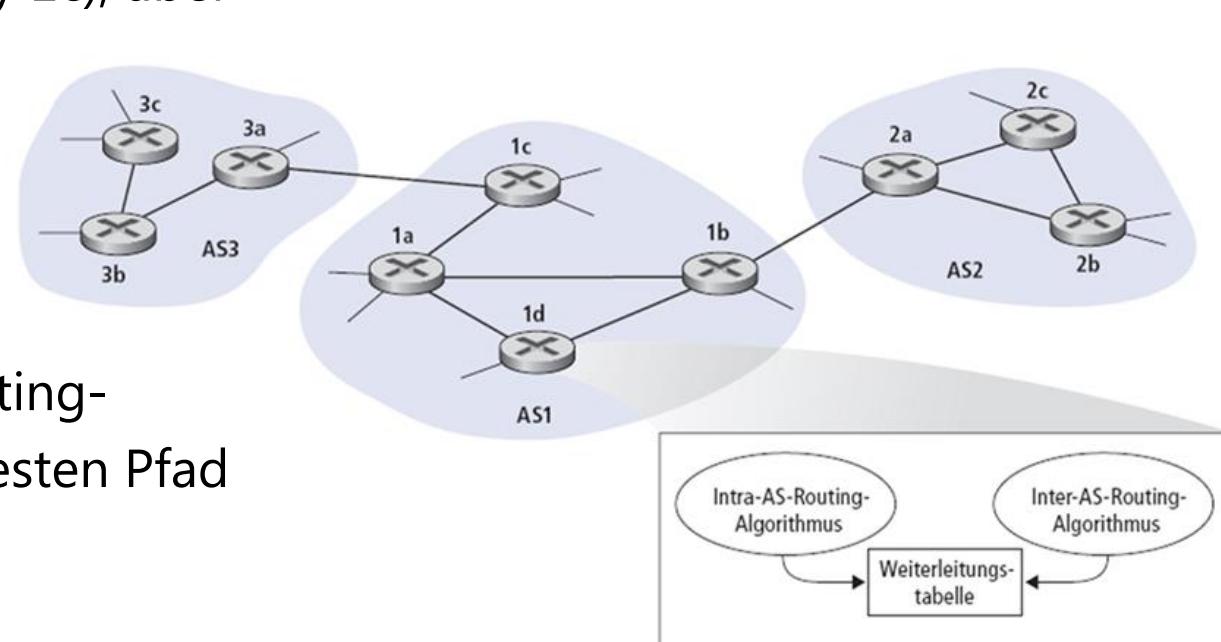
AS1 muss mit Hilfe von Inter-AS-Routing folgendes tun:

- Lernen, welche Ziele über die Autonomen Systeme AS2 und AS3 erreichbar sind
- Verteilen dieser Informationen an alle Router in AS1



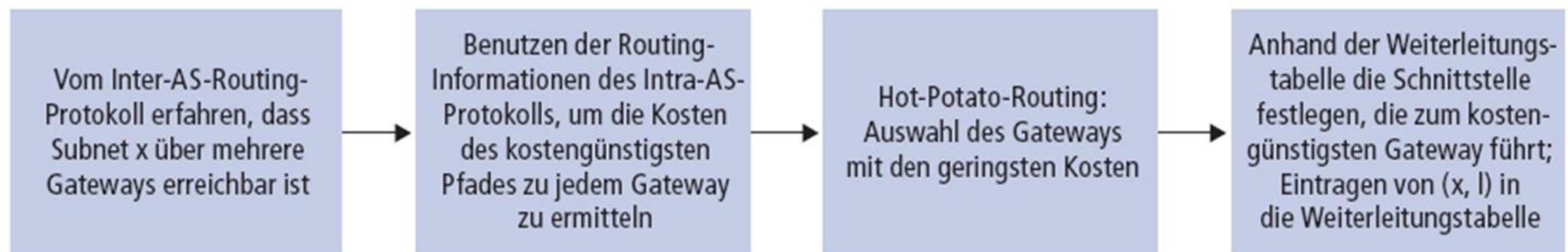
Beispiel: Routing-Tabelle in Router 1d

- Angenommen, AS1 lernt durch das Inter-AS-Routing-Protokoll, dass Netzwerk x von AS3 (Gateway 1c), aber nicht von AS2 aus erreicht werden kann
- Inter-AS-Routing-Protokoll propagiert diese Information zu allen internen Routern
- Router 1d bestimmt durch das Intra-AS-Routing-Protokoll, dass sein Interface I auf dem kürzesten Pfad zu 1c liegt
- Router 1d nimmt einen Eintrag (x,I) in der Routing-Tabelle vor



Beispiel: Alternative Routen

- Angenommen AS1 lernt durch das Inter-AS-Routing-Protokoll, dass Netzwerk X sowohl über AS2 als auch über AS3 zu erreichen ist
- Für den Eintrag in die Routing-Tabelle muss Router 1d sich für einen Pfad entscheiden
- Ebenfalls Aufgabe des Inter-AS-Routing-Protokolls
- Eine Möglichkeit:
 - Hot Potato-Routing: Schicke das Paket an den nächsten Gateway-Router, der es in ein anderes AS weiterleiten kann



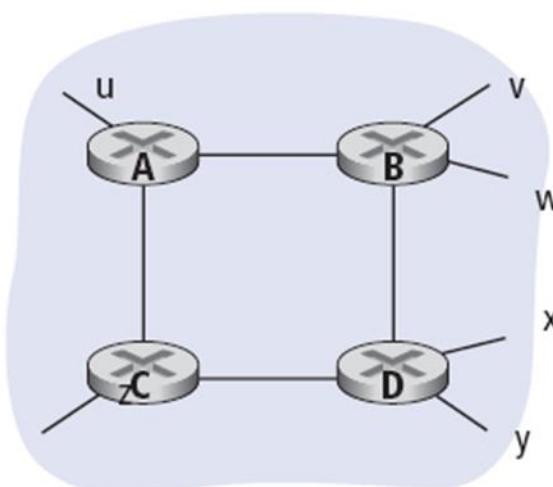
Routing im Internet

Intra-AS-Routing auch **Interior Gateway Protocol (IGP)**

- RIP: Routing Information Protocol
- OSPF: Open Shortest Path First

RIP (Routing Information Protocol)

- Version 1 spezifiziert in RFC 1058
- Version 2 (kompatibel mit Version 1) spezifiziert in RFC 2453
- Distance-Vector-Algorithmus
- War bereits in der BSD-UNIX-Distribution von 1982 enthalten
- Metrik: Anzahl der Hops (max = 15 Hops)

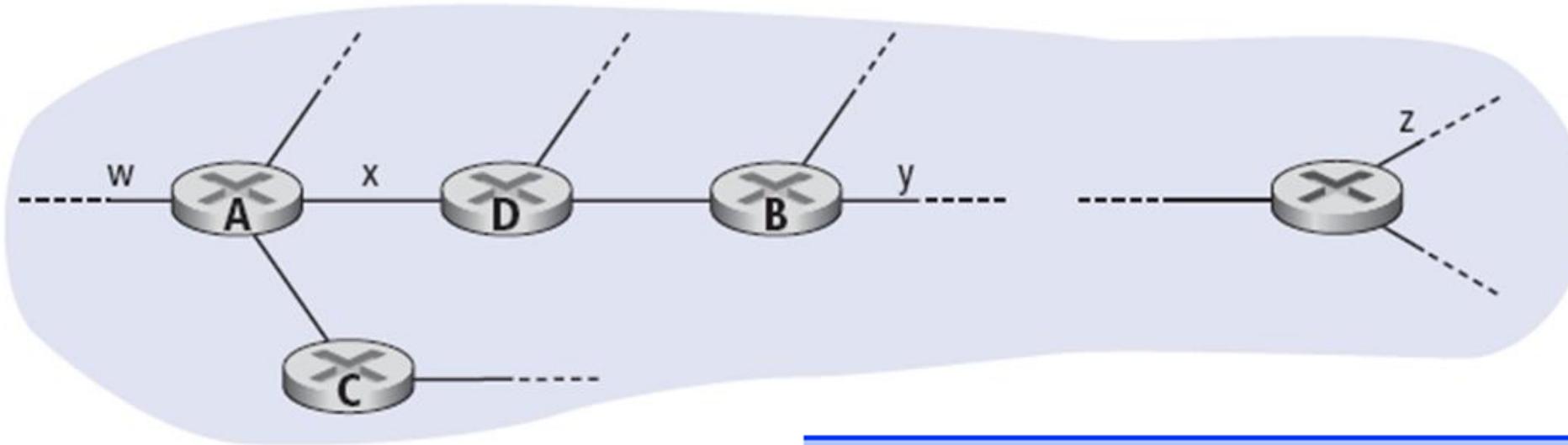


Ziel	Hops
u	1
v	2
w	2
x	3
y	3
z	2

RIP-Advertisements

- Distanzvektoren werden zwischen den Nachbarn alle 30 Sekunden per RIP-Advertisement ausgetauscht
- Jedes Advertisement enthält eine Liste von bis zu 25 Zielnetzwerken im Inneren des Autonomen Systems

RIP: Beispiel



Routing-Tabelle v. Router D:

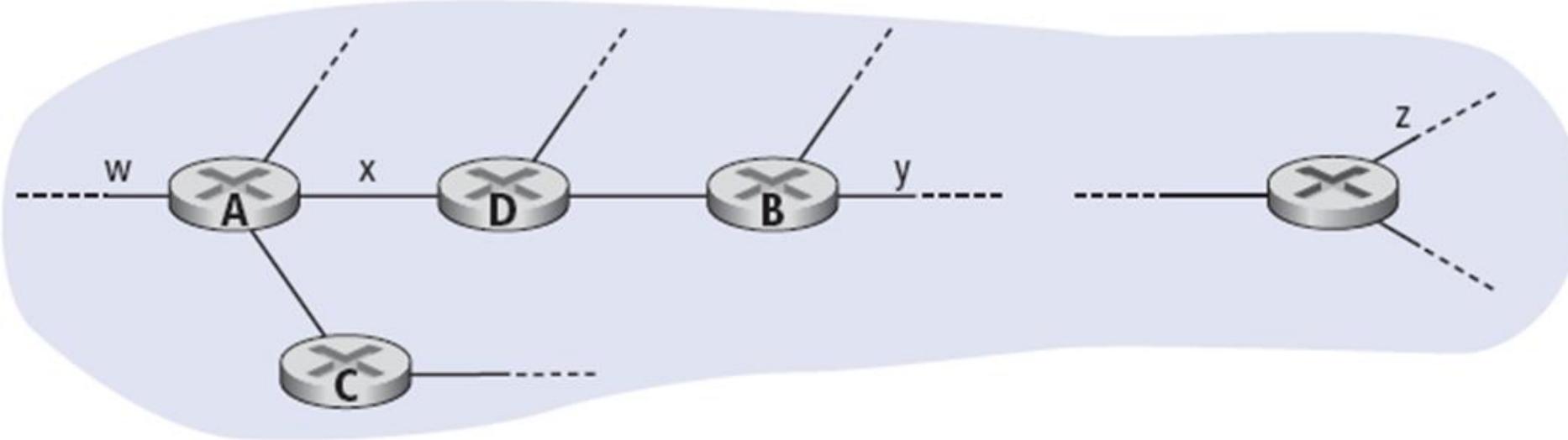
Zielsubnetz	Nächster Router	Anzahl von Hops zur Zieladresse
w	A	2
y	B	2
z	B	7
x	-	1
...

RIP: Beispiel

Advertisement von Router A an Router D:

Zielsubnetz	Nächster Router	Anzahl von Hops zur Zieladresse
z	c	4
w	-	1
x	-	1
...

RIP: Beispiel



Routing-Tabelle v. Router D:

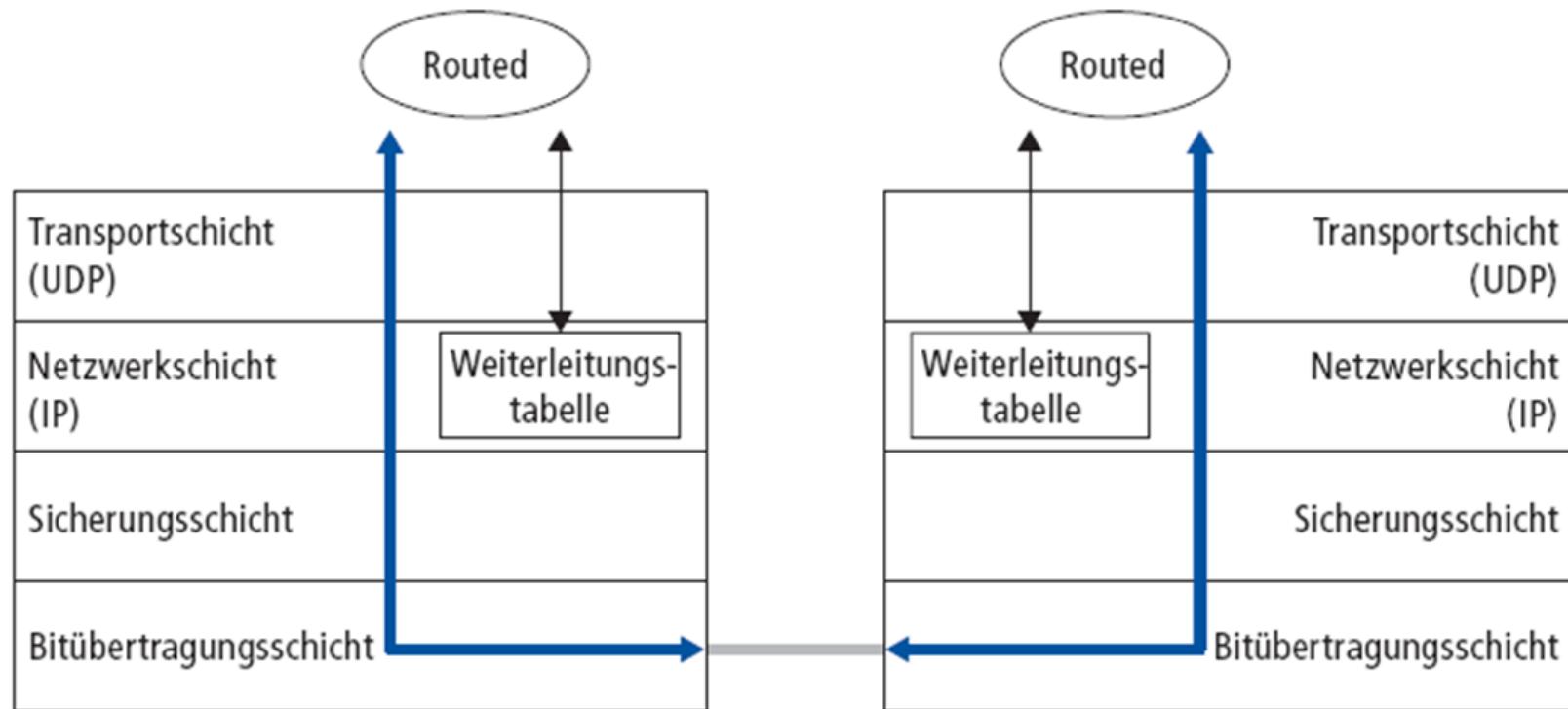
Zielsubnetz	Nächster Router	Anzahl von Hops zur Zieladresse
w	A	2
y	B	2
z	A	5
...

RIP: Brechen von Links

Wenn von einem Nachbarn 180 Sekunden lang kein Advertisement empfangen wurde, gilt der Nachbar als nicht mehr vorhanden

- Alle Routen über diesen Nachbarn werden ungültig
- Neuberechnung des lokalen Distanzvektors
- Verschicken des neuen Distanzvektors (wenn er sich verändert hat)
- Nachbarn bestimmen ihren Distanzvektor neu und verschicken ihn gegebenenfalls
- ...
- Die Information propagiert schnell durch das Netzwerk
- Poisonous Reverse wird verwendet, um Routing-Schleifen zu vermeiden (unendlich ist hier 16 Hops)

RIP-Architektur



- Die Routing-Tabelle kann von RIP in einem Prozess auf Anwendungsebene gepflegt werden: z.B. routed (für „route daemon“)
- Advertisements werden per UDP verschickt

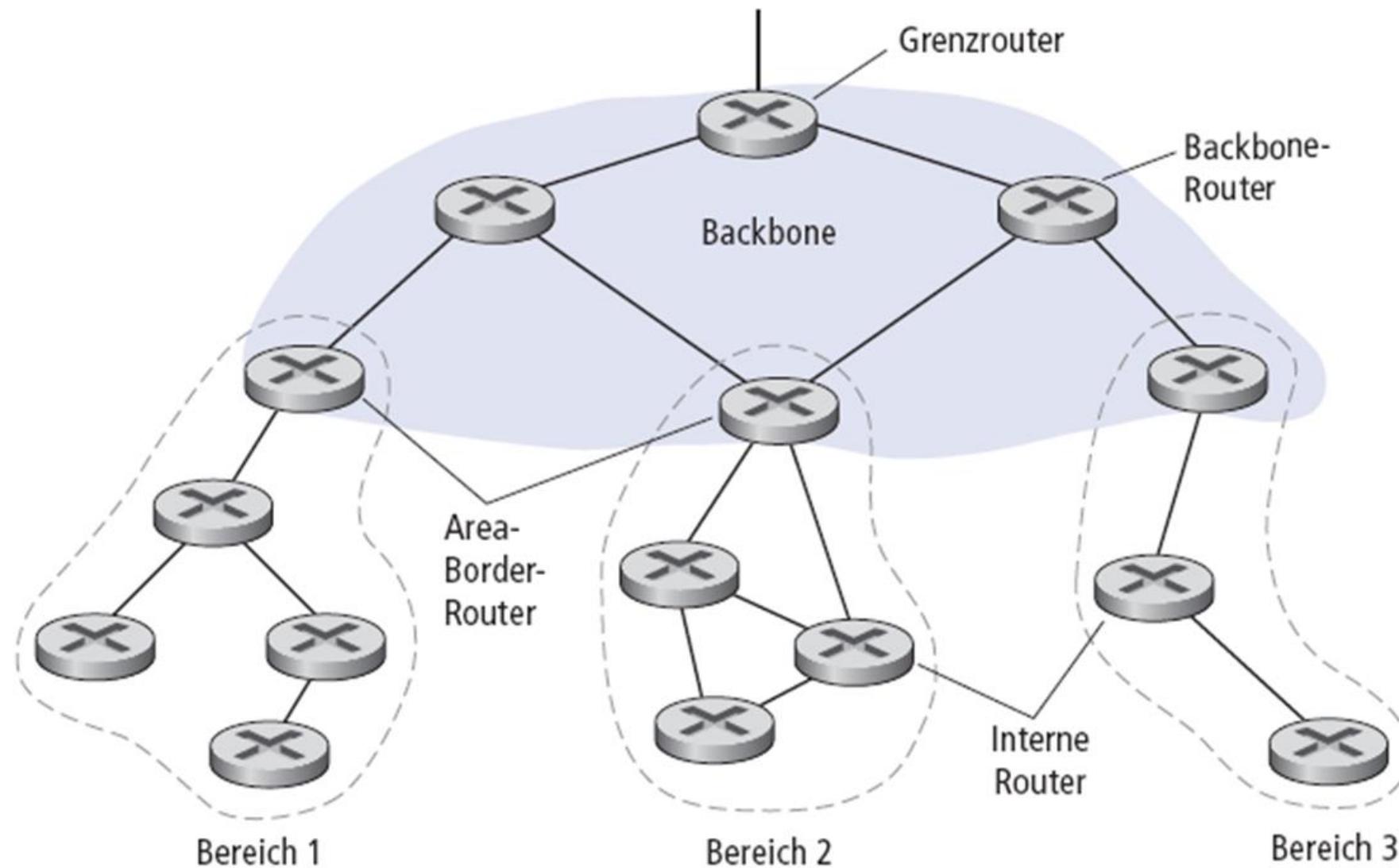
OSPF (Open Shortest Path First)

- Version 2 spezifiziert in RFC 2328
- "Open": frei verfügbar
- „Shortest Path First“: verwendet Link-State-Routing-Algorithmus
 - Periodisches Fluten von Link-State-Paketen
 - Jeder Router kündigt seine Links an
 - Diese Ankündigungen werden geflutet
 - OSPF-Pakete werden direkt in IP-Pakete eingepackt
 - Topologie des Netzwerkes in jedem Router bekannt
 - Wird in einer sogenannten „Netzwerkkarte“ oder „Topology Map“ gespeichert
 - Routen werden mit Dijkstras Algorithmus berechnet

Eigenschaften von OSPF

- Sicherheit: Alle OSPF-Nachrichten können authentifiziert werden
- Mehrere alternative Pfade auf Basis verschiedener Metriken können für jedes Ziel bestimmt werden
 - Für jeden Link können Gewichte mehrerer Metriken angegeben werden
 - Kosten für einen Satellitenlink gering (pro Paket), aber Latenzzeit hoch
- Hierarchisches OSPF in größeren autonomen Systemen

Hierarchisches OSPF



Hierarchisches OSPF

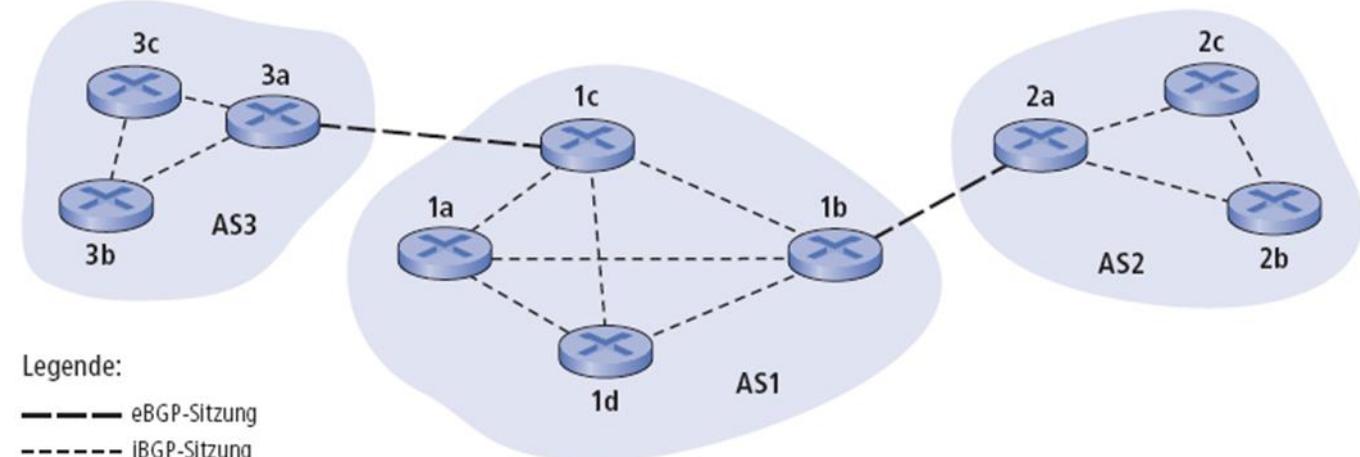
- Zweistufige Hierarchie: Local Area, Backbone
 - Ein Backbone pro AS
 - Link-State-Advertisements werden nur in einer Local Area geflutet
 - Jeder Router in einer Local Area kennt deren detaillierte Topologie und die Richtung zu den Netzwerken der anderen Local Areas
- Area-Border-Router:
 - Zusammenfassen der Distanzen zu den Netzwerken in der eigenen Local Area
 - Ankündigen dieser Zusammenfassung an die anderen Area-Border-Router
 - Ankündigungen der Zusammenfassungen der anderen Area-Border-Router in der Local Area
- Backbone-Router: führe OSPF-Routing im Backbone durch
- Boundary-Router: stellt eine Verbindung zu anderen Autonomen Systemen her

BGP (Border Gateway Protocol)

- Das Border Gateway Protocol, Version 4, ist der De-facto-Standard für Inter-AS-Routing im Internet
- Spezifiziert in RFC 4271
- BGP erlaubt es einem AS:
 - Informationen über die Erreichbarkeit von Netzen von seinen benachbarten Autonomen Systemen zu erhalten
 - Diese Informationen an die Router im Inneren des eigenen AS weiterzuleiten
 - „Gute“ Routen zu einem gewünschten Zielnetzwerk zu bestimmen, wobei die Qualität einer Route von den Informationen über die Erreichbarkeit und Politiken abhängig ist
- Außerdem ermöglicht BGP es einem AS, sein eigenes Netzwerk anzukündigen und so dessen Erreichbarkeit den anderen Autonomen Systemen im Internet mitzuteilen

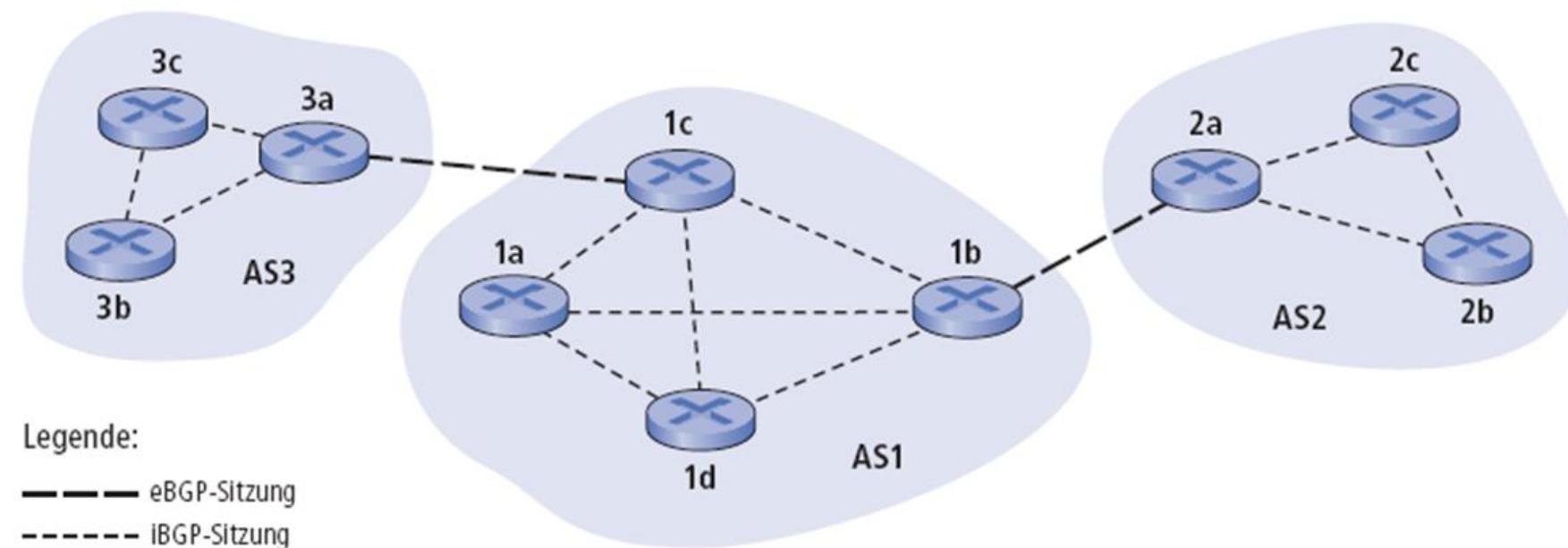
BGP-Grundlagen

- Paare von Routern (BGP-Peers) tauschen Routing-Informationen über semi-permanente TCP-Verbindungen aus: BGP-Sitzung (engl. BGP session)
 - Wenn beide Router im selben AS sind: interne BGP-Sitzung (iBGP)
 - Wenn beide Router in verschiedenen AS sind: externe BGP-Sitzung (eBGP)
- Wichtig: BGP-Sitzungen entsprechen nicht notwendigerweise physikalischen Links
- Wenn AS2 einen Präfix an AS1 meldet (z.B. 167.3/16), dann verspricht AS2, dass es alle Datagramme weiterleitet, deren Zieladressen zu diesem Präfix passen
 - AS2 kann Präfixe in seinen Ankündigungen aggregieren



Verbreiten der Informationen über Erreichbarkeit

- AS3 kündigt die Erreichbarkeit von Präfixen über die eBGP-Sitzung zwischen 3a and 1c an
- 1c kann iBGP verwenden, um diese Informationen im AS zu verbreiten
- 1b kann dann diese neuen Informationen über die eBGP-Sitzung zwischen 1b und 2a weitermelden und die Präfixe dem Autonomen System 2 ankündigen



Pfadattribute und BGP-Routen

- Wenn ein Präfix angekündigt wird, dann beinhaltet diese Ankündigung sogenannte Pfadattribute
 - Präfix + Attribute = Route (in BGP-Terminologie)
- Zwei wichtige Attribute:
 - AS-PATH: eine Liste aller AS, durch welche die Ankündigung weitergeleitet wurde:
AS 67, AS 17
 - NEXT-HOP: zwei Autonome Systeme können über mehr als ein Paar von Gateway-Routern in Kontakt stehen. Um die Ankündigung einem Router im anderen AS zuordnen zu können, schickt der ankündigende Router seine IP-Adresse als NEXT-HOP-Attribut mit
- Der empfangende Gateway-Router entscheidet durch konfigurierbare Politiken anhand der Attribute, ob eine Ankündigung angenommen werden soll bzw. welche Ankündigung bevorzugt wird

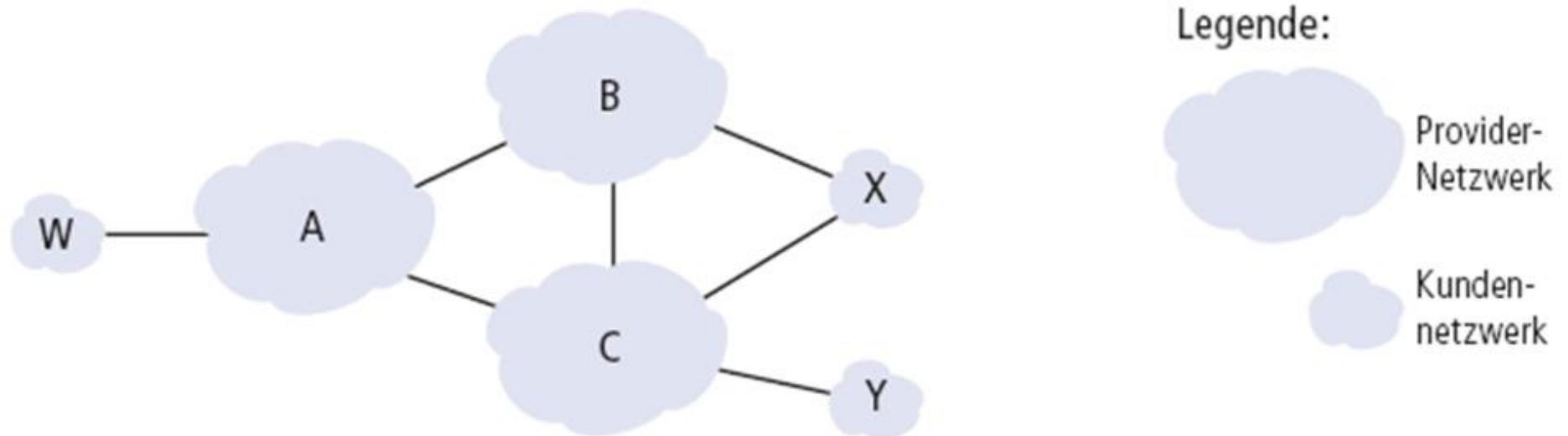
BGP-Routenwahl

- Router können mehr als eine Route zum Ziel angeboten bekommen. Ein Router muss entscheiden, welche Route verwendet wird.
- Regeln:
 - Kürzester AS-PATH
 - Dichtester NEXT-HOP-Router: Hot Potato Routing
 - Weitere Kriterien (häufigster Fall)
- Die BGP-Routenwahl ist eine firmenpolitische Entscheidung!

BGP-Nachrichten

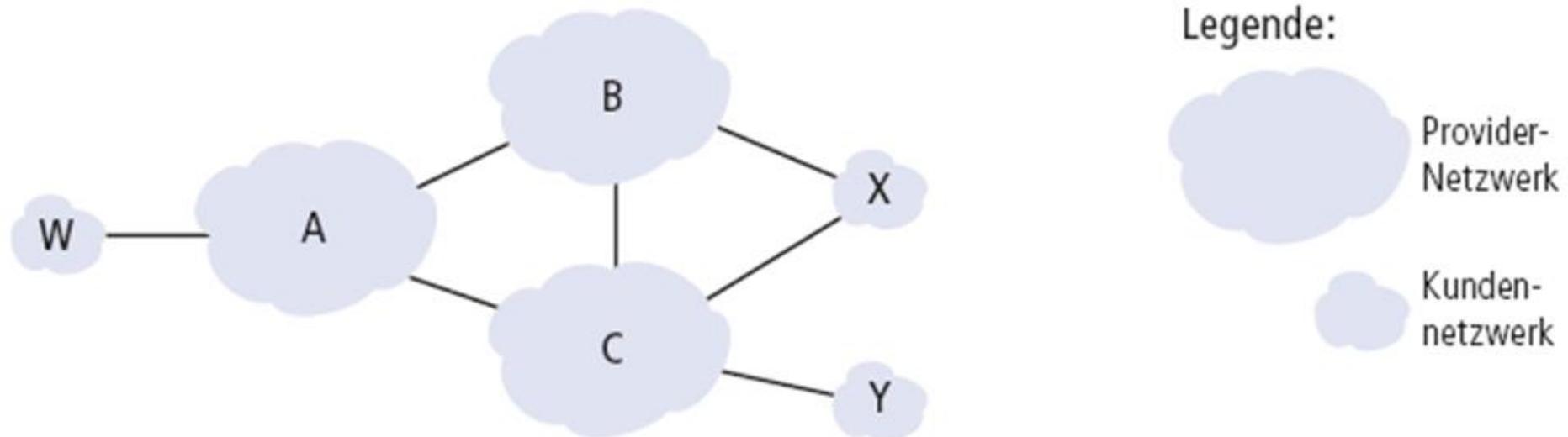
- BGP-Nachrichten werden über TCP ausgetauscht
- BGP-Nachrichten:
 - OPEN: Öffnen einer BGP-Sitzung, Authentifizierung
 - UPDATE: neue Route ankündigen oder Ankündigung zurückziehen
 - KEEPALIVE: Sitzung aufrechterhalten, wenn keine UPDATES geschickt werden müssen (wird auch als ACK auf eine OPEN-Nachricht verschickt)
 - NOTIFICATION: Mitteilen von Fehlern, Schließen einer Session

BGP-Routing-Politiken



- A,B,C sind Netzwerke von Providern
- X,W,Y sind Netzwerke von Kunden (der Provider)
- X ist „dual homed“: an zwei Provider angebunden
 - X möchte keine Daten von B nach C weiterleiten
 - .. daher wird X keine Route nach C an B ankündigen

BGP-Routing-Politiken



- A kündigt B den Pfad AW an
- B kündigt X den Pfad BAW an
- Sollte B den Pfad BAW auch C ankündigen?
 - Nein! B hätte nichts davon, da weder W noch C Kunden von B sind
 - B möchte, dass C Datenpakete zu W über A leitet
 - B möchte nur Verkehr an oder von seinen Kunden weiterleiten

Warum verschiedene Protokolle für Intra-AS- und Inter-AS-Routing?

- Politiken:
 - Inter-AS: Eine Organisation möchten kontrollieren, wie (und ob) der Verkehr anderer Organisationen durch das eigene Netzwerk geleitet wird
 - Intra-AS: eigener Verkehr, eigene Administration, hier sind keine Politiken nötig
- Größenordnung:
 - Hierarchisches Routing reduziert die Größe der Routing-Tabellen und reduziert den Netzwerkverkehr für Routing-Updates
 - Dringend notwendig für Inter-AS-Routing, nicht wichtig in Intra-AS-Routing
- Performance:
 - Intra-AS: kann sich auf Performance konzentrieren
 - Inter-AS: Politiken können wichtiger sein als Performance