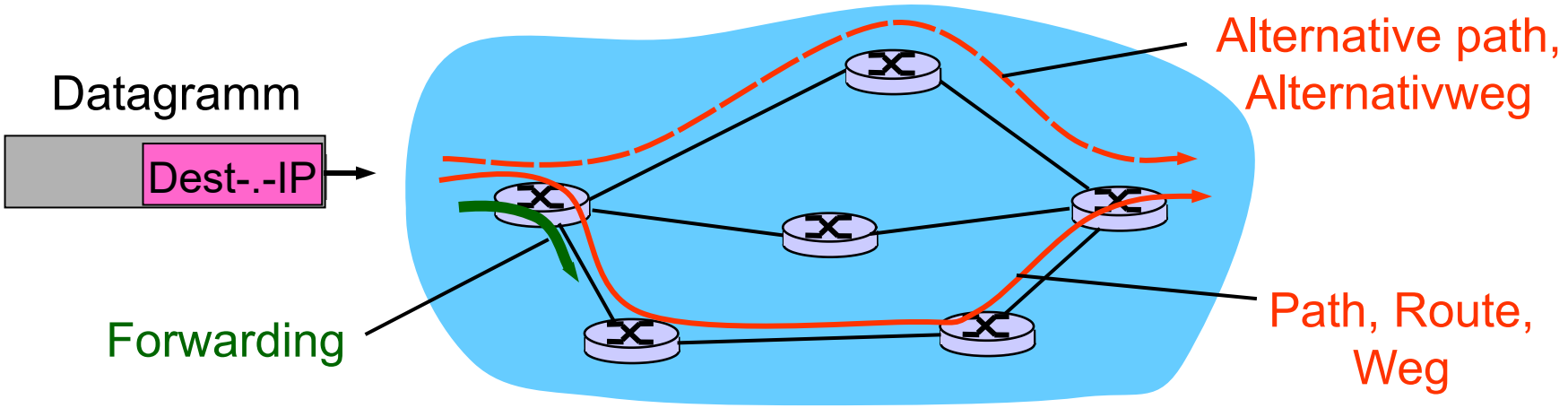
5. Routing-Verfahren und -Protokolle

Routing und Forwarding



Forwarding

- Vorgang um ein Paket durch einen einzelnen Netzknoten zu schicken

Routing

- Verfahren zur Auswahl eines Weges vom Quellknoten zum Zielknoten
- Connection Oriented Networks: Routing erfolgt beim Verbindungsaufbau
  - Eintrag in Forwarding-Table gilt für alle Daten der Verbindung
- Connectionless Networks: Routing erfolgt für jedes Paket

Routing: Grundbegriffe

Link

- Übertragungsabschnitt zwischen zwei Nachbarknoten (Router, *vertices*)

Route / Path

- Weg für Dateneinheiten vom Quell- zum Zielknoten
  - Verbindungsorientierte Kommunikation
    - für alle Dateneinheiten einer Verbindung
  - Verbindungslose Kommunikation
    - für jede einzelne Dateneinheit separat

Routing-Metrik

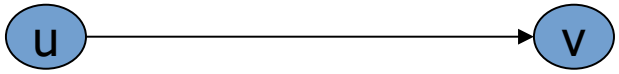
- Jedem Link des Netzes wird eine Metrik (Kostenwert) zugeordnet
  - Beispiele sind „Hop“ (Kosten von 1), Delay, Bandbreite
- Metrik kann unidirektional definiert sein

Routing-Policy

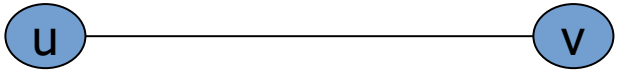
- Strategie des Netzbetreibers zur Auswahl einer Route

Routing: Grundbegriffe

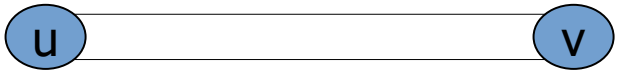
- **Directed (gerichtet)**: geordnetes Knotenpaar. Repräsentiert als (u, v) in der Richtung von Knoten u zu v (Networking: simplex)



- **Undirected (ungerichtet)**: ungeordneres Paar von Knoten. Repräsentiert als {u, v}. Lässt jede Art von Richtung unbeachtet und behandelt die Knoten beider Enden als miteinander austauschbar (Networking: duplex)

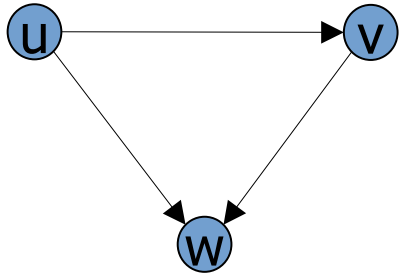


- **Multiple Edges (mehrere Kanten)**: Zwei oder mehr Kanten die das selbe Paar von Knoten verbinden



Routing: Grundbegriffe

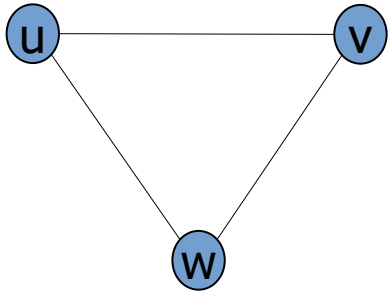
- **Gerichteter (Directed) Graph**: G(V, E), Set von Knoten V und Set von Kanten E, wobei letztere ein geordnetes Elementenpaar von V darstellen (gerichtete Kanten)



`G(V, E),`

`V={u,v,w},E={(u,v),(v,w),(u,w)}`

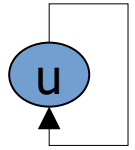
- **Ungerichteter (Simple/Undirected) Graph**: besteht aus einem Knotenset V, und einem Set ungeordneter Kantenpaare E (ungerichtet)



`G(V, E),`

`V={u,v,w},E={{u,v},{v,w},{u,w}}`

- **Loop (Schleife)**: Kante die vom Ursprungsknoten wieder zurück zum selben Knoten führt. Repräsentiert als {u,u} = {u}

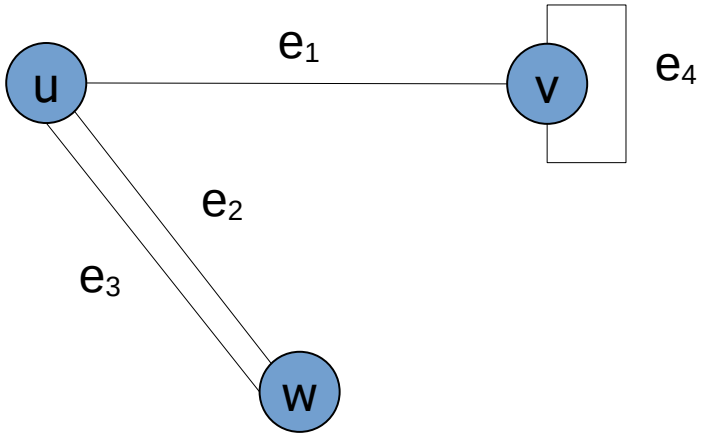


Routing: Grundbegriffe

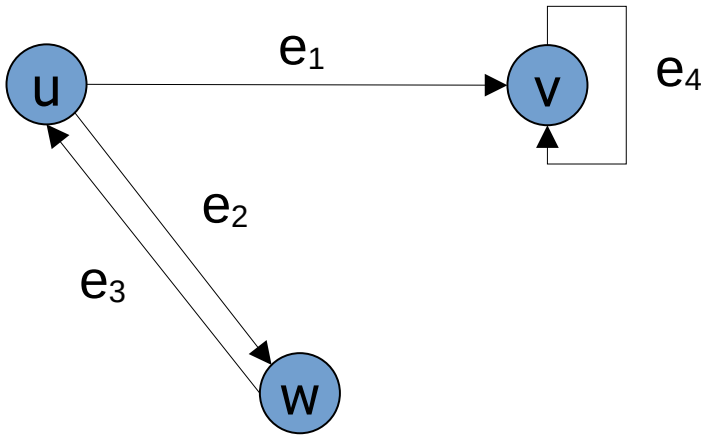
- **Multigraph**: G(V,E), besteht aus einem Set von Knoten V, einem Set von Kanten E und einer Funktion f von E zu {{u, v}| u, v ϵ V, u ≠ v}
- e1 und e2 werden als mehrfache oder parallele Kanten wenn f(e1) = f(e2) gilt

G(V, E),

V = {u, v, w}, E = {e1, e2, e3, e4}



Ungerichtet



Gerichtet

Routing: Grundbegriffe

- **Incidence (Matrix):** Dann am Nützlichsten wenn Informationen über Kanten interessanter ist als Informationen über Knoten

- **Adjacency (Matrix / List):** Dann am Nützlichsten wenn Informationen über Knoten interessanter ist als Informationen über Kanten

Inzidenzmatrix

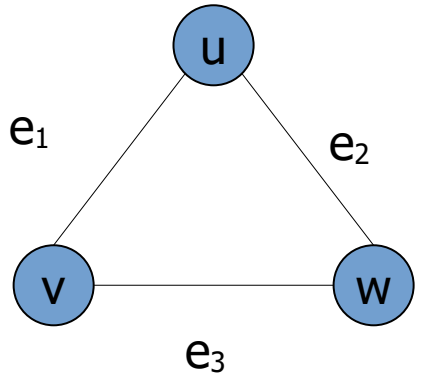
- Ist G = (V, E) ein ungerichteter Graph mit den Knoten v_1, v_2, v_3, …, v_n und den Kanten e_1, e_2, …, e_m ergibt sich die Inzidenzmatrix mit den Dimensionen n x m zu M = [m_{ij}] mit:

$$m_{ij} = \begin{cases} 1 \text{ when edge } e_j \text{ is incident with } v_i \\ 0 \text{ otherwise} \end{cases}$$

- Kann außerdem folgendes repräsentieren:
  - Mehrere Kanten: indem Spalten mit identischen Einträgen genutzt werden, da diese Kanten inzident (verbunden) mit dem selben Knotenpaar sind
  - Loops: indem eine Spalte mit exakt einem Eintrag (der gleich 1 ist) genutzt wird, entsprechend dem Knoten der inzident zum Loop ist

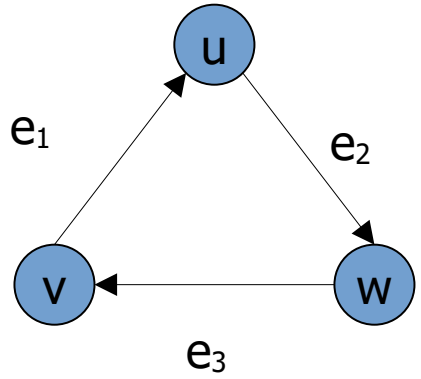
Inzidenzmatrix

- Beispiel: Ungerichteter Graph



	e_1	e_2	e_3
v	1	0	1
u	1	1	0
w	0	1	1

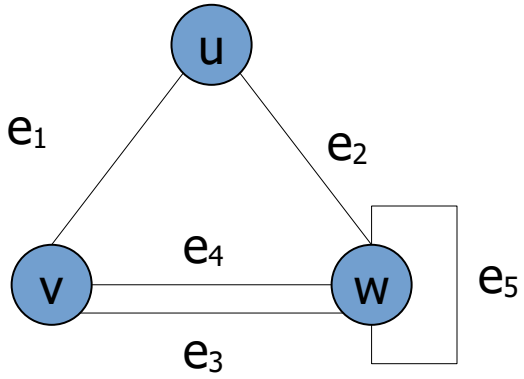
- Beispiel: Gerichteter Graph



	e_1	e_2	e_3
v	-1	0	1
u	1	-1	0
w	0	1	-1

Inzidenzmatrix

- Beispiel: Ungerichteter Multigraph



	e_1	e_2	e_3	e_4	e_5
v	1	0	1	1	0
u	1	1	0	0	0
w	0	1	1	1	1

Adjazenzmatrix

- Ist eine N x N Matrix, wobei |V| = N (also Knotenanzahl)
- Diese Matrix (NxN) A = [a_{ij}] ergibt sich aus den Einträgen:

  Für **ungerichtete Graphen**

$$a_{ij} = \begin{cases} 1 \text{ if } \{v_i, v_j\} \text{ is an edge of } G \\ 0 \text{ otherwise} \end{cases}$$

  Für **gerichtete Graphen**

$$a_{ij} = \begin{cases} 1 \text{ if } (v_i, v_j) \text{ is an edge of } G \\ 0 \text{ otherwise} \end{cases}$$

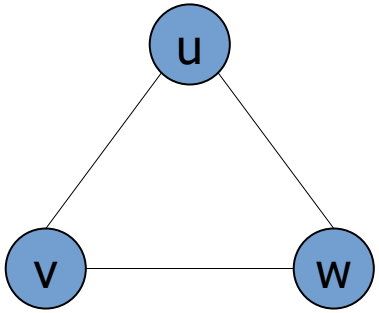
- Erleichtert die Suche nach Subgraphen sowie das Umkehren eines Graphen sofern nötig

Adjazenzmatrix

- Die Adjazenzmatrizen einfacher Graphen sind symmetrisch ($a_{ij} = a_{ji}$)
  - *Warum? Weil jede Kante {i,j} die Knoten i und j in beide Richtungen verbindet*
- Wenn der Graph relativ wenige Kanten enthält, so handelt es sich bei der zugehörigen Adjazenzmatrix um eine **sparse Matrix (= dünn besetzt)**
  - *Warum? Weil aus weniger Kanten mehr 0 als 1 resultieren*
- Gerichtete Multigraphen werden repräsentiert indem a_{ij} die Anzahl der Kanten von v_i zu v_j beschreibt

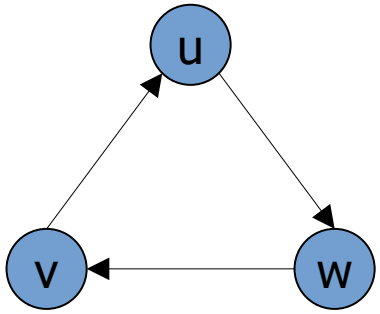
Adjazenzmatrix

- Beispiel: Ungerichteter Graph



	v	u	w
v	0	1	1
u	1	0	1
w	1	1	0

- Beispiel: Gerichteter Graph



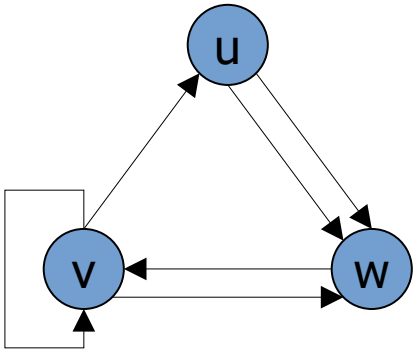
(zu)

(von)

	v	u	w
v	0	1	0
u	0	0	1
w	1	0	0

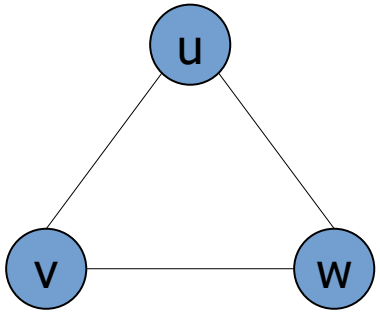
Adjazenzmatrix

- Beispiel: Gerichteter Multigraph, Matrix



	v	u	w
v	1	1	1
u	0	0	2
w	1	0	0

- Beispiel: ungerichteter Graph, Liste



Vertex	Adjacency list
v	u, w
u	v, w
w	u, v

Dezentrale (verteilte Zustandsinformation)

- Knoten X
  - Kennt **nur** die Linkkosten zu allen Nachbarknoten V: c(x,v)
- Ermittelt Kosten des Least-Cost-Path zum Zielknoten Y: $D_X(y) \approx k_X(y)$
- bildet seinen Distanzvektor zu allen Zielen: $\mathbf{D}_X = (D_X(y):\mid y \in \mathbf{N})$
- Kennt Distanzvektoren seiner Nachbarknoten V: $\mathbf{D}_V = (D_V(y):\mid y \in \mathbf{N})$

Update und Kostenberechnung (verteiltes Routing)

- Von Zeit zu Zeit sendet jeder Knoten seinen Distanzvektor an die Nachbarn
- Falls Knoten X ein Update ($\mathbf{D}_V$) von seinem Nachbarknoten V erhält aktualisiert er seinen eigenen Distanzvektor:

  für jeden Zielknoten Y: $D_X(y) = \min_V \{c(x,v) + D_V(y)\}$

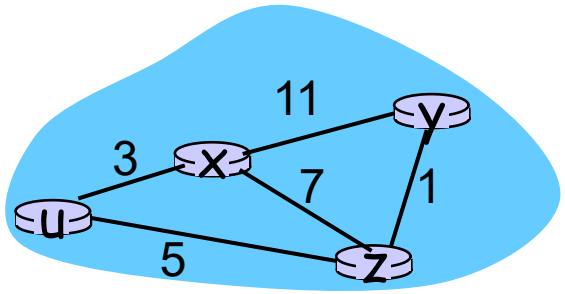
- $D_X(y)$ konvergiert gegen die optimalen Kosten $k_X(y)$

Verteilte Zustandsinformation und Routing-Tabelle

- c(x,y): Kosten des Links von Knoten X zu Nachbarknoten Y
- $D_X(y,v)$: Kosten des Weges von Knoten X zu Zielknoten Y über Nachbar V
- $D_X(y)$: Kosten des Least-Cost-Path von Knoten X zu Knoten Y

$v \in \{u, y, z\}$ $D_X(y,v) = c(x,v) + D_V(y)$

$D_X(y) = \min_v D_X(y,v)$ $v \in \{u, y, z\}$



über Knoten v

zum Ziel

$D_X(.,.)$	u	y	z
u	3	17	12
y	9	11	8
z	8	12	7

Routing Database in Knoten X

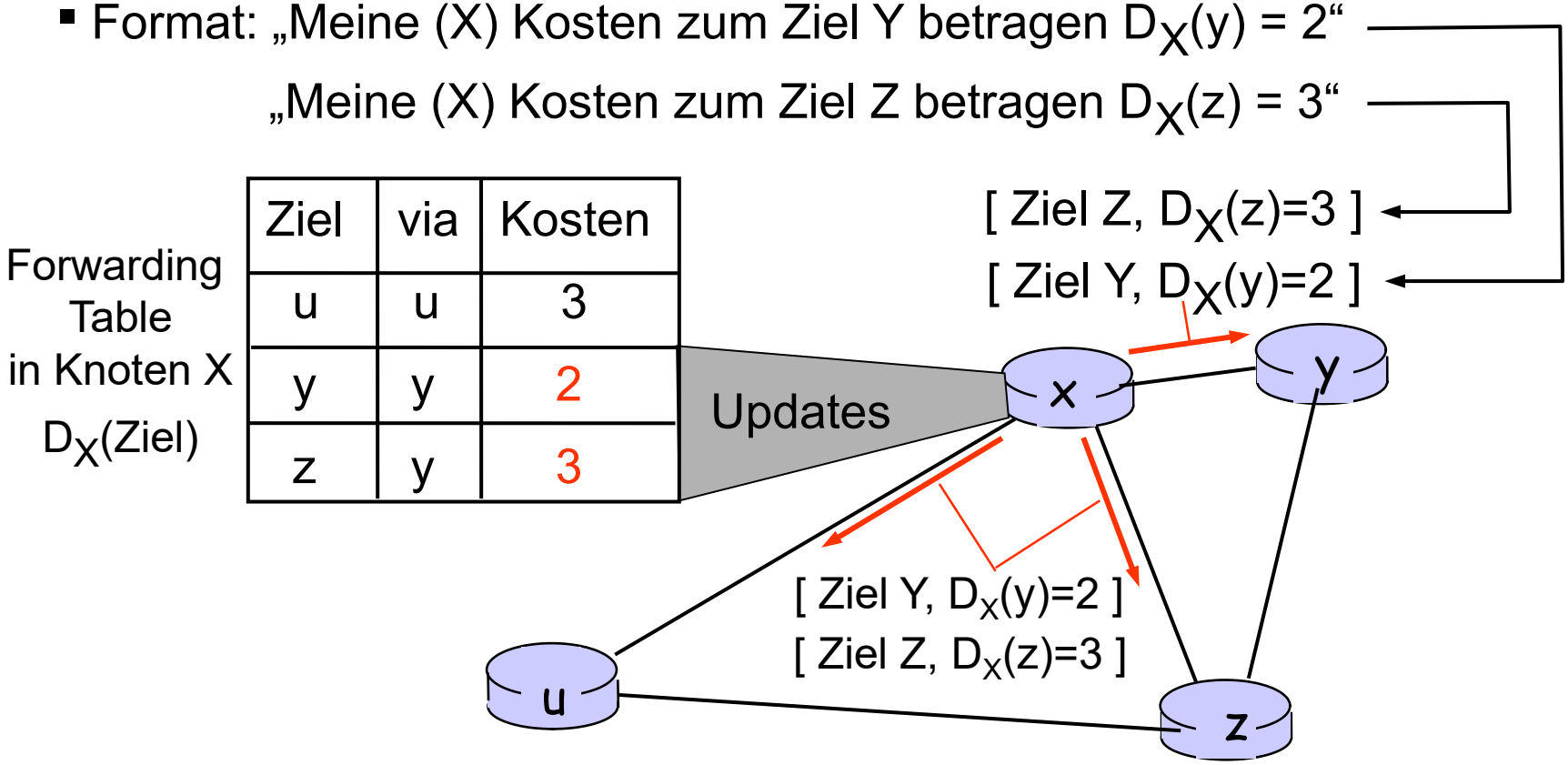
Ziel	via	$D_X(.)$
u	u	3
y	z	8
z	z	7

Forwarding Table in Knoten X

Update-Message

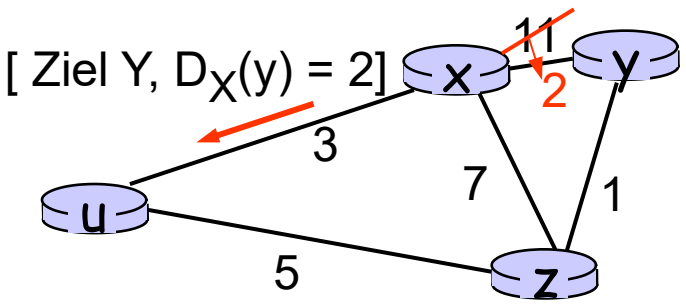
- Knoten X schickt allen Nachbarn ein Update, falls
  - lokale Linkkosten c(x,y) sich geändert haben
  - sich ein neuer Distanzvektor $\mathbf{D}_X(v)$ zu einem Zielknoten V ergibt
- Format: „Meine (X) Kosten zum Ziel Y betragen $D_X(y) = 2$“

  „Meine (X) Kosten zum Ziel Z betragen $D_X(z) = 3$“



Update der Routing-Tabelle

- Knoten U vor dem Update



Routing Database U

über Knoten

$D_U(.,.)$	x	z
x	3	12
z	10	5
y	11	6

zum Ziel

„Forwarding Table U

Ziel	via	Kosten
x	x	3
z	z	5
y	z	6

Richtig ?

- Knoten U nach dem Update

Update Database für $D_X(y)$

$D_U(y,x) = c(u,x) + D_X(y)$

$= 3 + 2 = 5$

über Knoten

$D_U(.,.)$	x	z
x	3	12
z	10	5
y	5	6

zum Ziel

Ziel	via	Kosten
x	x	3
z	z	5
y	x	5

Shortest Path: $D_U(y) = \min_{v \in \{x, z\}} D_U(y,u)$

Sende Update $D_U(y)$

Bellmann-Ford-Algorithmus

[Initialisierung in Knoten x]

$D_X(v,v) = c(x,v)$ for all neighbours v, $D_X(y,w) = \infty$ for w is not neighbour node

For all destination nodes y: sent $D_X(y) = \min_V D_X(y,v)$ to all neighbour nodes v

[loop (in jedem Knoten x)]

wait (until a link cost changes or until an update receives from neighbour v)

if (c(x,v) changes by d)

for all destinations y: $D_X(y,v) = D_X(y,v) + d$

← *(Kosten zu allen Zielen über Nachbar v um d ändern. Anmerkung: d kann positiv oder negativ sein)*

else if (update $D_V(y)$ received from v for destination y)

for the single destination y: $D_X(y,v) = c(x,v) + D_V(y)$

← *(der kürzeste Weg von v zu y hat sich geändert, v hat einen neuen Wert $D_V(y)$ gesendet)*

[Neuberechnung aller Vektoren]

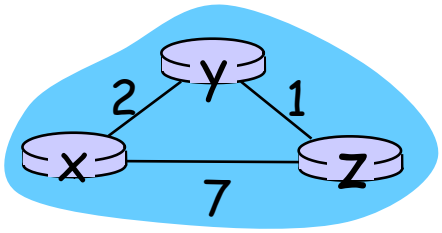
for all destinations y: $D_X(y) = \min_V D_X(y,v)$

if (we have a new $D_X(y)$ for any destination y)

send new value $D_X(y)$ of to all neighbour v

← *(send Update)*

Bellmann-Ford-Algorithmus



Zeitpunkt t

über

Ziel

$D_X()$	y	z
y	2	∞
z	∞	7

$D_X(y,z)$

$D_Y()$	x	z
x	2	∞
z	∞	1

$D_Z()$	x	y
x	7	∞
y	∞	1

$D_X(y)=2$
$D_X(z)=7$

$D_Y(y)=2$
$D_Y(z)=1$

$D_Z(y)=7$
$D_Z(z)=1$

über

Ziel

$D_X()$	y	z
y		
z		

$D_Y()$	x	z
x		
z		

$D_Z()$	x	y
x		
y		

über

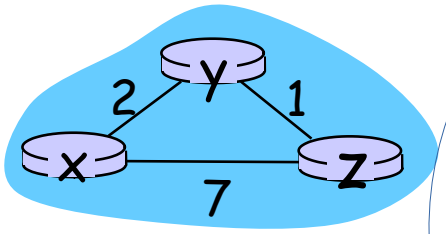
Ziel

$D_X()$	y	z
y		
z		

$D_Y()$	x	z
x		
z		

$D_Z()$	x	y
x		
y		

Bellmann-Ford-Algorithmus (2)



$D_X(z) = \min\{c(x,y)+D_Y(z),\ c(x,z) + D_Z(z)\} = \min\{2+1,\ 7+0\}$

$D_X(y) = \min\{c(x,y)+D_Y(y),\ c(x,z) + D_Z(y)\} = \min\{2+0,\ 7+1\}$

über y

über z

Zeitpunkt t +1

über

Ziel

$D_X()$	y	z
y	2	∞
z	∞	7

$D_Y()$	x	z
x	2	∞
z	∞	1

$D_Z()$	x	y
x	7	∞
y	∞	1

über

$D_X()$	y	z
y	2	8
z	3	7

$D_Y()$	x	z
x	2	8
z	9	1

$D_Z()$	x	y
x	7	3
y	9	1

$D_X(z)=3$

$D_Z(x)=3$

über

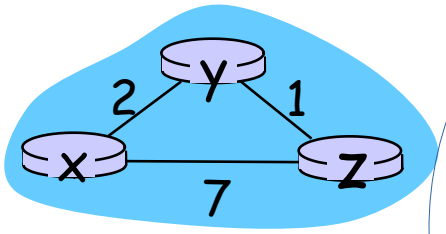
Ziel

$D_X()$	y	z
y		
z		

$D_Y()$	x	z
x		
z		

$D_Z()$	x	y
x		
y		

Bellmann-Ford-Algorithmus (3)



$D_X(z) = \min\{c(x,y)+D_Y(z),\ c(x,z) + D_Z(z)\} = \min\{2+1,\ 7+0\}$

$D_X(y) = \min\{c(x,y)+D_Y(y),\ c(x,z) + D_Z(y)\} = \min\{2+0,\ 7+1\}$

über y

über z

Zeitpunkt t +1

über

Ziel

$D_X()$	y	z
y	2	∞
z	∞	7

$D_Y()$	x	z
x	2	∞
z	∞	1

$D_Z()$	x	y
x	7	∞
y	∞	1

über

$D_X()$	y	z
y	2	8
z	3	7

$D_Y()$	x	z
x	2	8
z	9	1

$D_Z()$	x	y
x	7	3
y	9	1

$D_X(z)=3$

$D_Z(x)=3$

über

Ziel

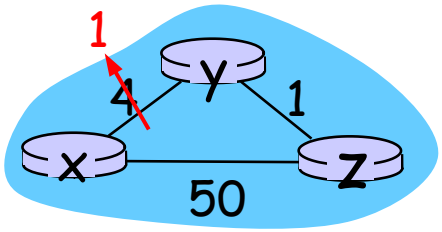
$D_X()$	y	z
y	2	8
z	3	7

$D_Y()$	x	z
x	2	4
z	5	1

?

$D_Z()$	x	y
x	7	3
y	9	1

Änderung der Link-Kosten: Good news travels fast



keine Änderung?

$D_Y(x,z)=c(y,z)+D_Z(x)=1+5$
(z kennt neues c(x,y) noch nicht)

über

Ziel

$D_Y()$	x	z
x	4	6
z	9	1

-3

$D_Y()$	x	z
x	1	6
z	6	1

$D_Y(x)=1$

$D_Y()$	x	z
x	1	6
z	3	1

$D_Y()$	x	z
x	1	3
z	3	1

$D_Z()$	x	y
x	50	5
y	54	1

$D_Z()$	x	y
x	50	5
y	54	1

$D_Z()$	x	y
x	50	2
y	51	1

$D_Z(x)=2$

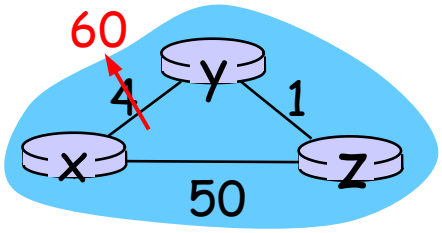
$D_Z()$	x	y
x	50	2
y	51	1

Y erkennt neue Linkkosten, Neuberechnung $D_Y(.,.)$ neuer Wert $D_Y(x)$, Update an Nachbarn

Z erhält Update von Y, Neuberechnung von $D_Z(.,.)$, neuer Wert $D_Z(x)$, Update an Nachbarn

Y erhält Update von Z, Neuberechnung $D_Y(.,.)$, Konvergenz

Änderung der Link-Kosten: Bad news travels slowly



„bad news travels slowly“
Count-to-Infinity-Problem

über

Ziel

$D_Y()$	x	z
x	4	6
z	9	1

+54

$D_Y()$	x	z
x	60	6
z	6	1

$D_Y(x)=6$

$D_Y()$	x	z
x	60	6
z	3	1

$D_Z(x)=8$

$D_Y()$	x	z
x	60	8
z	3	1

...

$D_Y()$	x	z
x	60	51
z	3	1

$D_Z()$	x	y
x	50	5
y	54	1

$D_Z()$	x	y
x	50	5
y	54	1

$D_Z()$	x	y
x	50	7
y	51	1

$D_Z(x)=7$

$D_Z()$	x	y
x	50	9
y	51	1

$D_Z()$	x	y
x	50	52
zy	51	1

Y erkennt neue Linkkosten,
Neuberechnung $D_Y(.,.)$
neuer Wert $D_Y(x)$,
Update an Nachbarn

Routing-Schleife für Ziel X:
Weg: Z→ Y→ Z→ Y→ ...

Konvergenz nach 44 Iterationen,
Weg: Z→ X

Änderung der Link-Kosten: Poisoned Reverse

Prinzip

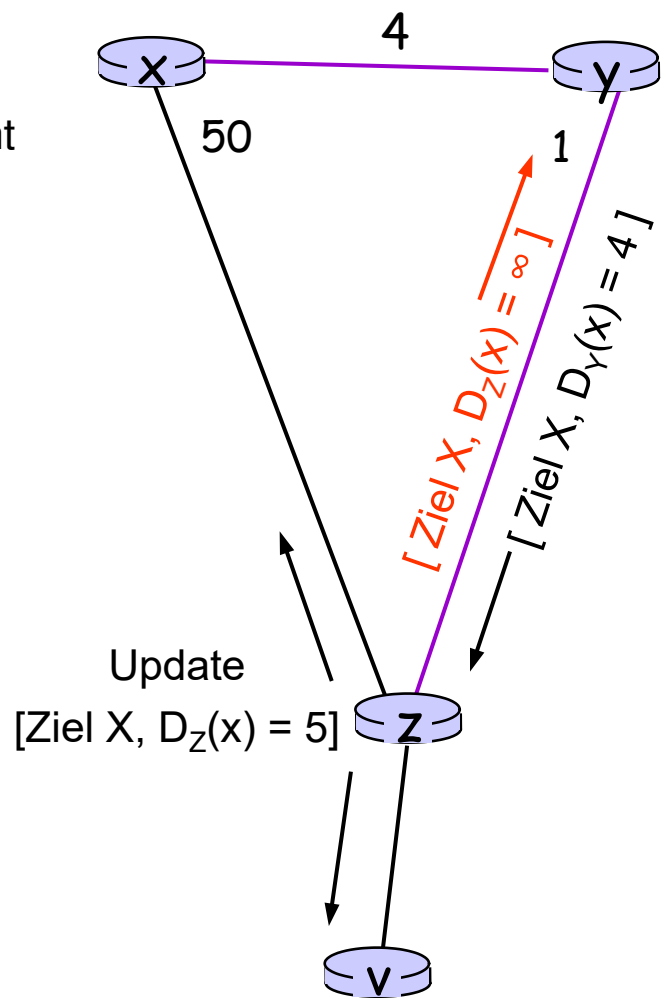
- Shortest Path von Z nach X geht über Y
- Knoten Z hat Shortest Path nach X von Knoten Y gelernt
  - Selektives Update and Knoten Z

    $D_Z(x) = \infty$ „Poisoned Reverse“

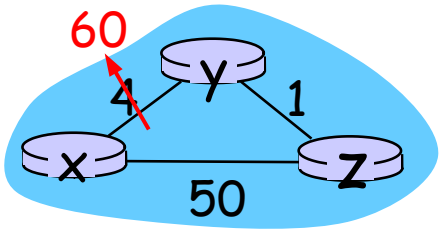
    (für Y hat Z keinen Weg zum Ziel X)
    - Sende allen anderen Nachbarn:

      Update $D_Z(x)$

- Beispiel: Z hat $D_Y(x) = 4$ von Y erhalten
  - Neue Distanz: $D_Z(x)$
  - sende $D_Z(x) = \infty$ an Y
  - $D_Z(x) = 5$ an X und V



Beispiel: Poisoned Reverse



über

Ziel

$D_Y()$	x	z
x	4	∞
z	9	1

$D_Z()$	x	y
x	50	5
y	54	1

$D_Z(x) = \infty$

$D_Y()$	x	z
x	60	∞
z	6	1

$D_Z()$	x	y
x	50	5
y	54	1

$D_Y(x) = 60$

$D_Y()$	x	z
x	60	∞
z	3	1

$D_Z()$	x	y
x	50	61
y	51	1

$D_Z(x) = 50$

$D_Y()$	x	z
x	60	51
z	3	1

$D_Z()$	x	y
x	50	61
y	51	1

$D_Y(x) = \infty$

$D_Y()$	x	z
x	60	51
z	3	1

$D_Z()$	x	y
x	50	∞
y	51	1

Y erkennt neue Linkkosten

$D_Z(x) = 50$,
Shortest Path nach X nicht von Y,
$D_Z(x) = 50$ and Y senden

Konvergenz

Routing-Algorithmus: Link-State

Globale Zustandsinformation

- Netztopologie und Linkkosten sind jedem Knoten (Router) bekannt
  - Die Zustandsinformation ist im statischen Fall in jedem Router gleich
  - Wird durch ein Link-State-Broadcast-Potokoll realisiert
- Nur positive, meist zustandsabhängige Linkkosten (Metrik)

Dezentral ausgeführter Routing-Algorithmus

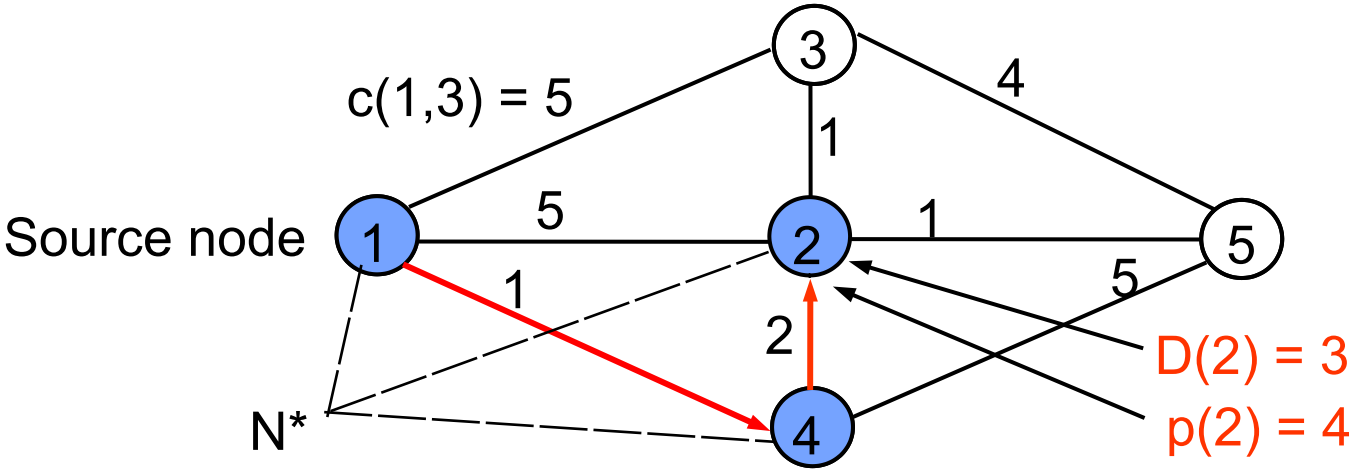
- Shortest-Path-Algorithmus von Djikstra
- In jedem Knoten:
  - Finde den Weg geringster Kosten zu allen anderen Knoten
  - Quellknoten ist Wurzel eines Baums kürzester Wege
  - Wege sind schleifenfrei

Shortest-Path-Algorithmus von Djikstra

Notation

Für einen Quellknoten (source node):

- c(i,j): Linkkosten von Knoten i zu j, c(i,j) = ∞ falls kein Nachbarknoten
  - Linkkosten aller Links im Quellknoten bekannt
- D(v): Kosten des Weges vom Quellknoten zum Knoten v mit den momentan geringsten Kosten (Label)
- p(v): Vorgänger von v auf dem momentan kürzesten Weg zu Knoten v
- N*: Menge der Knoten, zu denen ein Weg geringster Kosten besteht



Shortest-Path-Algorithmus von Djikstra (2)

Initialisierung

N*={A}; D(v)=∞; p(v)=0 *(A ist der Quellknoten)*

I. for all nodes v adjacent to A:

D(v)=c(A,v); p(v)=A *(initiale Kosten zu allen Nachbarn)*

Iteration:

loop

II. find w not in N* such that D(w) is a minimum; *(Wähle den Knoten w aus der Menge N \ N*, der vom Quellknoten mit minimalen Pfad-Kosten erreicht wird)*

add w to N*;

III. update D(v) for all v adjacent to w and not in N*: *(Die neuen Kosten zu v sind entweder die alten Kosten zu v oder die bekannten Kosten des kürzesten Weges zu w, zuzüglich der Kosten von w zu v)*

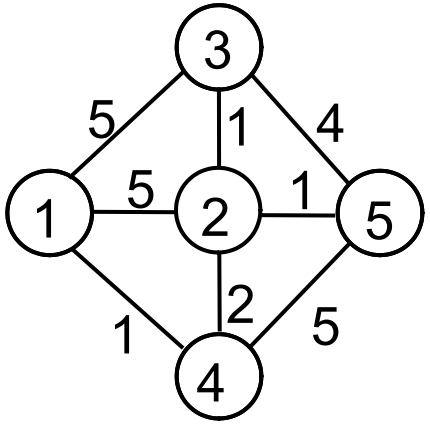
D(v) = min(D(v), D(w) + c(w,v))

if new shortest path: p(v) = w

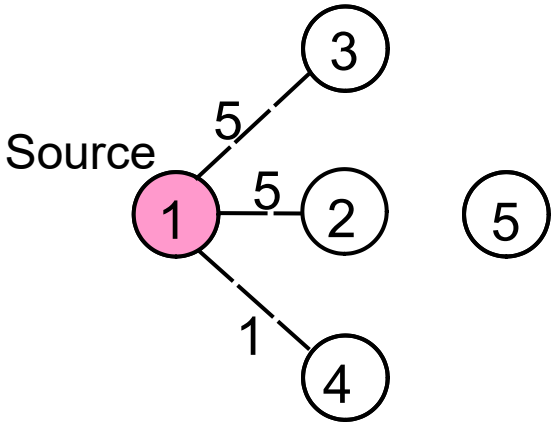
until all nodes n ϵ N*;

Beispiel: Djikstra

Topologie



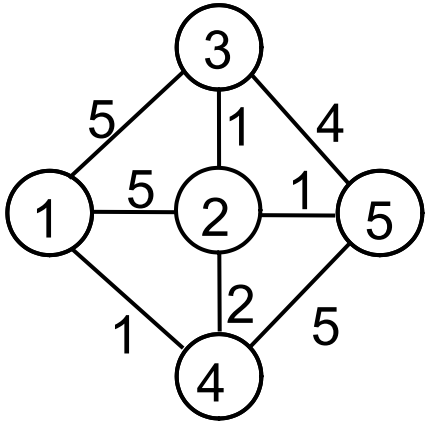
Initialisierung



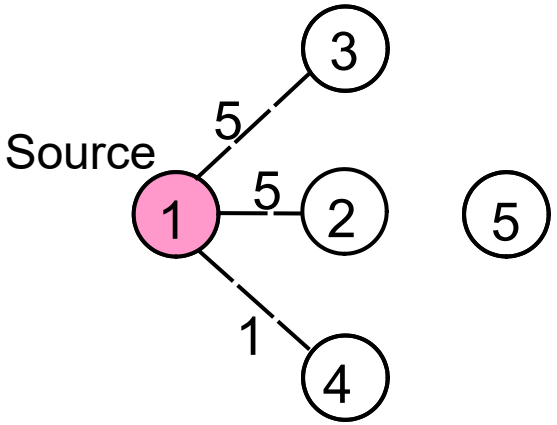
	Label				
Iteration / Schritt	N*	D(2), p(2)	D(3), p(3)	D(4), p(4)	D(5), p(5)
0. / I	{1}	5, 1	5, 1	1, 1	∞, 0

Beispiel: Djikstra

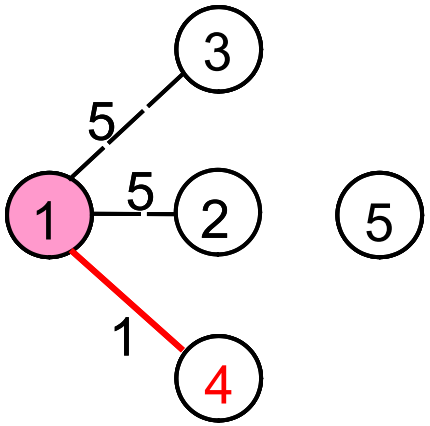
Topologie



Initialisierung



1. Iteration

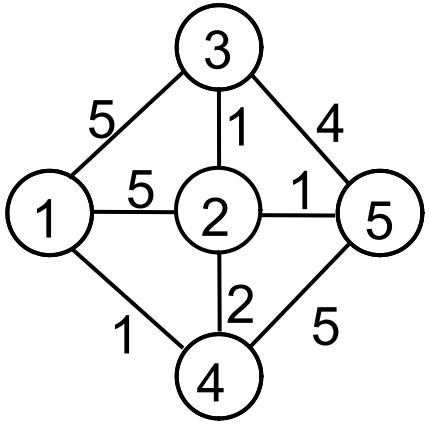


II: find w with min(D(w))

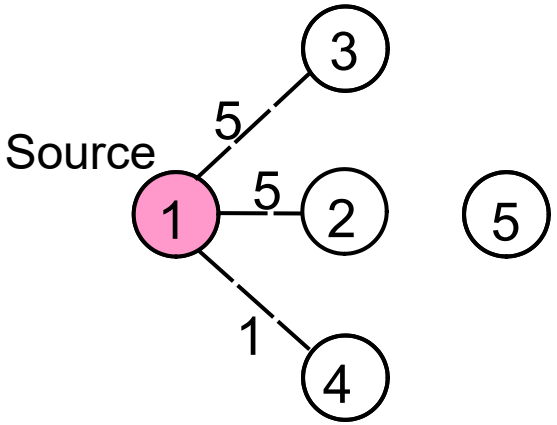
	Label				
Iteration / Schritt	N*	D(2), p(2)	D(3), p(3)	D(4), p(4)	D(5), p(5)
0. / I	{1}	5, 1	5, 1	1, 1	∞, 0
1. / II	{1, 4}			**min: *w* = 4**	

Beispiel: Djikstra

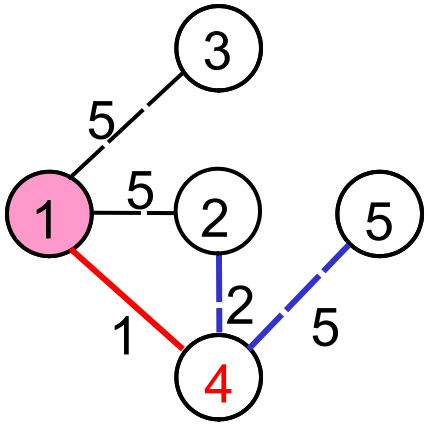
Topologie



Initialisierung



1. Iteration

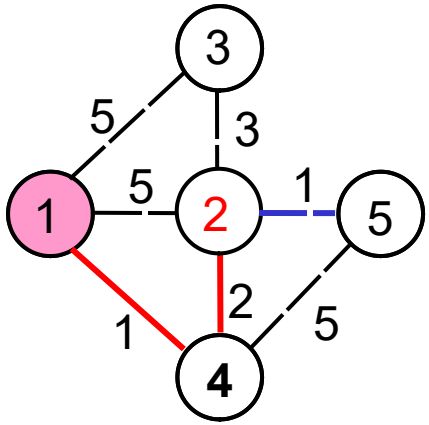


III: D(v) = min(D(v), D(w) + c(w,v))

	Label				
Iteration / Schritt	N*	D(2), p(2)	D(3), p(3)	D(4), p(4)	D(5), p(5)
0. / I	{1}	5, 1	5, 1	1, 1	∞, 0
1. / II III	{1, 4}	**3, 4**	5, 1	**min: *w* = 4** **1, 1**	**6, 4**

Beispiel: Djikstra (2)

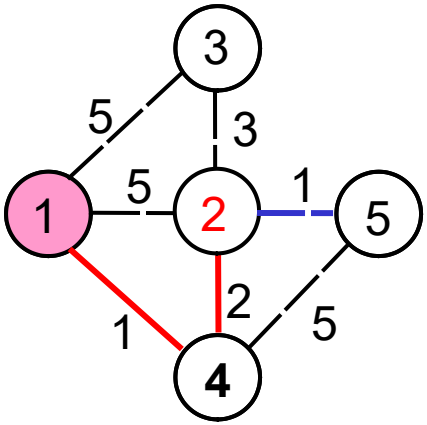
2. Iteration



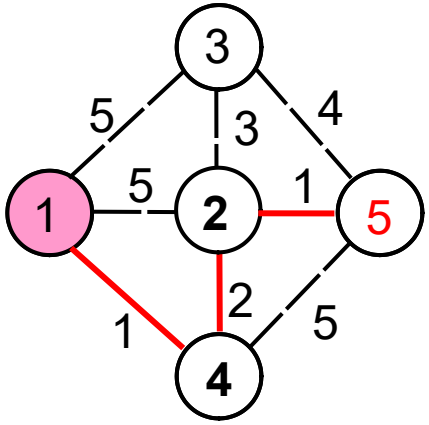
Iter./Schritt	N*	D(2), p(2)	D(3), p(3)	D(4), p(4)	D(5), p(5)
1. / II III	{1, 4}	3, 4	5, 1	**min: *w* = 4** **(1, 1)**	6, 4
2. / II III	{1, 4, 2}	**min: *w* = 2** **(3, 4)**	5, 1	--	4, 2

Beispiel: Djikstra (2)

2. Iteration



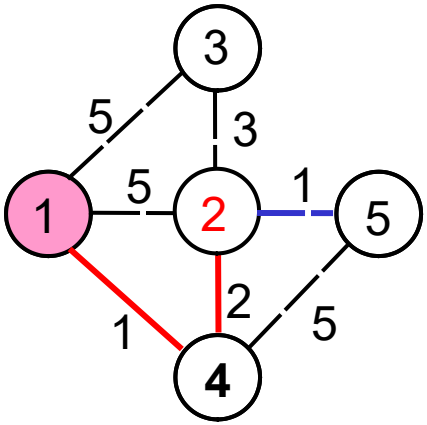
3. Iteration



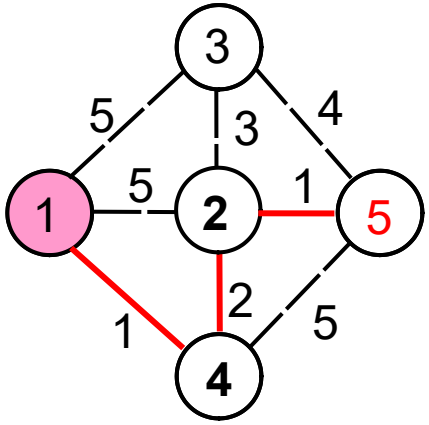
Iter./Schritt	N*	D(2), p(2)	D(3), p(3)	D(4), p(4)	D(5), p(5)
1. / II III	{1, 4}	**3, 4**	5, 1	**min: *w* = 4** **(1, 1)**	**6, 4**
2. / II III	{1, 4, 2}	**min: *w* = 2** **(3, 4)**	5, 1	--	**4, 2**
3. / II III	{1, 4, 2 ,5}	--	5, 1	--	**min: *w* =5** **(4, 2)**

Beispiel: Djikstra (2)

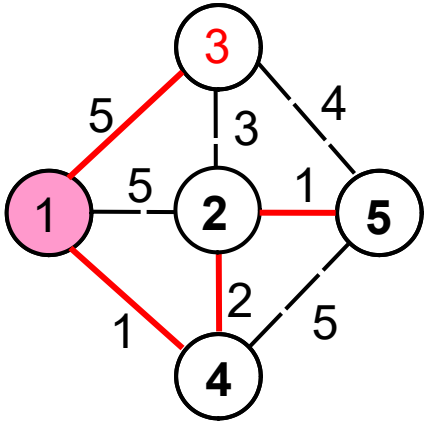
2. Iteration



3. Iteration



4. Iteration



Iter./Schritt	N*	D(2), p(2)	D(3), p(3)	D(4), p(4)	D(5), p(5)
1. / II III	{1, 4}	**3, 4**	5, 1	**min: *w* = 4** **(1, 1)**	**6, 4**
2. / II III	{1, 4, 2}	**min: *w* = 2** **(3, 4)**	5, 1	--	**4, 2**
3. / II III	{1, 4, 2 ,5}	--	5, 1	--	**min: *w* =5** **(4, 2)**
4. / II	{1, 4, 2, 5, 3}	--	**min: *w* = 3**	--	--

Eigenschaften des Djikstra-Algotithmus

Routing-Tabelle

- Iterative Konstruktion der Wege aus der Vorgängerinformation
  - Beispiel: Weg von 1→ 5:

    p(5)=2, p(2)=4, p(4)=1 ↔ 1 – 4 – 2 – 5

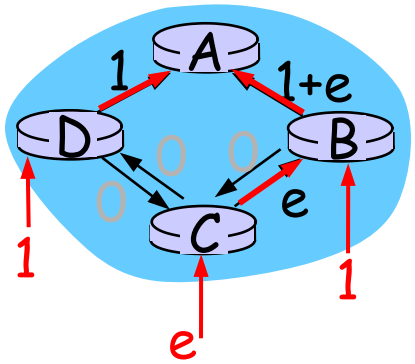
Komplexität für N Knoten

- Jede Iteration: Überprüfung der Knoten die noch nicht in N* sind
  - 1. Iteration N – 1 Überprüfungen
  - 2. Iteration N – 2 Überprüfungen
  - …
  - Insgesamt: N(N+1)/2 Überprüfungen
- Komplexität ist O(N²)
  - O(Nlog(N)) bei effizienter Implementierung

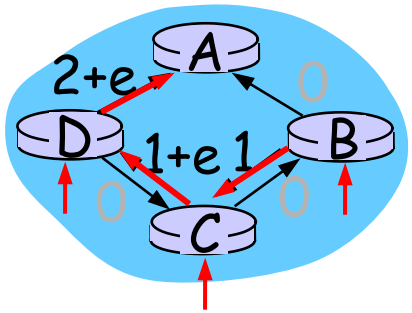
Oszillation bei zustandsabhängigen Metriken

Beispiel

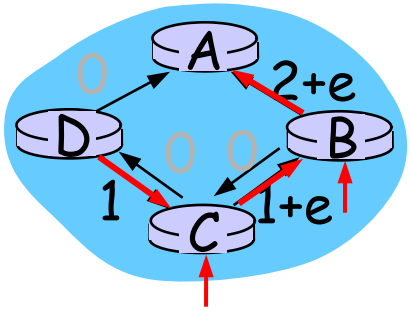
- Kosten sind äquivalent zur Verkehrslast des Links
- Gerichtete Verbindungskanten
- Knoten D und B senden „1“, Knoten C „e“ Verkehrseinheiten an Ziel A



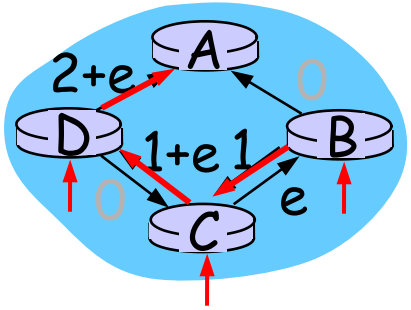
initiales Routing



Routing Update



Update



Update

Lösungsansätze

- Knoten führen das Update nicht gleichzeitig durch
- Selbstsynchronisation der Knoten wird durch zufällige Wahl der Updatezeitpunkte vermieden

Link–State vs. Distanz–Vektor Algorithmus

Komplexität

- **LS**: Bei N Knoten und E Links sind O(N,E) Meldungen zu senden
- **DV**: Austausch nur zwischen Nachbarn
  - Anzahl abhängig von Zahl der Iterationen bis zur Konvergenz

Konvergenzgeschwindigkeit

- **LS**: Komplexität $O(N^2)$
  - Oszillationen möglich
- **DV**: variable Konvergenzzeit
  - Mögliche Routingschleifen
  - Count-to-Infinity-Problem
    - PR keine vollständige Lsg.

Robustheit

Verhalten bei Knotenfehlfunktion

- **LS**: Knoten verteilt falsche Linkkosten
  - Jeder Knoten berechnet eigene Tabelle, damit begrenzte Fehlerwirkung
- **DV**: Knoten verteilt falsche Pfadkosten
  - Jeder Knoten benutzt Tabelle der Nachbarn
    - Fehler verbreitet sich im ganzen Netz

Reachability

- Verfahren zum Austausch von Erreichbarkeitsinformationen zwischen den Routern
  - Welche Netzwerke sind über mich erreichbar
  - Welche Router sind im Netzwerk vorhanden

Optimal Route

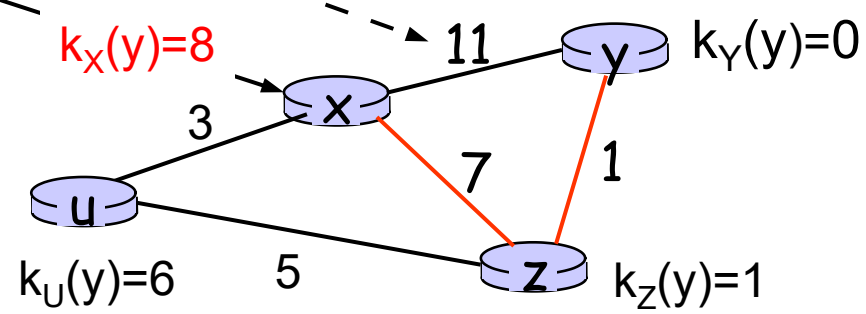
- Ziel: Finde den Least Cost Path (Weg der geringsten Kosten)
  - Optimale Wege werden mit Shortest Path Algorithmus berechnet

Updates

- Falls die Topologie oder die Link-Kosten sich ändern
- Inhalt der Updates wird auch als Zustandsinformation bezeichnet

Zustandsinformation und optimale Pfadkosten

- c(x,y): Kosten des Links von Knoten X zu Nachbarknoten Y
- $k_X(y)$: Kosten des Least-Cost-Path von Knoten X zu Knoten Y



- Prinzip: Im Knoten X für Zielknoten Y

$$k_X(y) = \min_V \{ c(x,v) + k_V(y) \}$$

$= c(x,z) + k_Z(y) = 7 + 1 = 8$

$k_V(y)$: Kosten von Knoten V zum Zielknoten Y

$c(x,v)$: Link Kosten zu Nachbar V; $V \in \{u, y, z\}$

$\min_V$: Minimum bzgl. aller Nachbarknoten V;

Problem: Skalierung

- Starker Anstieg der Zahl der Netze und Router in den Netzen
  - Routing-Tabelle wächst mit der Anzahl der Netzpräfixe
  - Austausch von Routing-Information wächst mit Zahl der Router

Problem: Administrative Autonomie

- Das Internet ist ein Netzwerk von Netzen
  - Jedes Netzwerk gehört jemand anderem, bspw ISP
  - Jede Organisation möchte seine Netzwerke unabhängig und selbstständig administrieren

Lösung: Autonome Systeme (Autonomous Systems, AS)

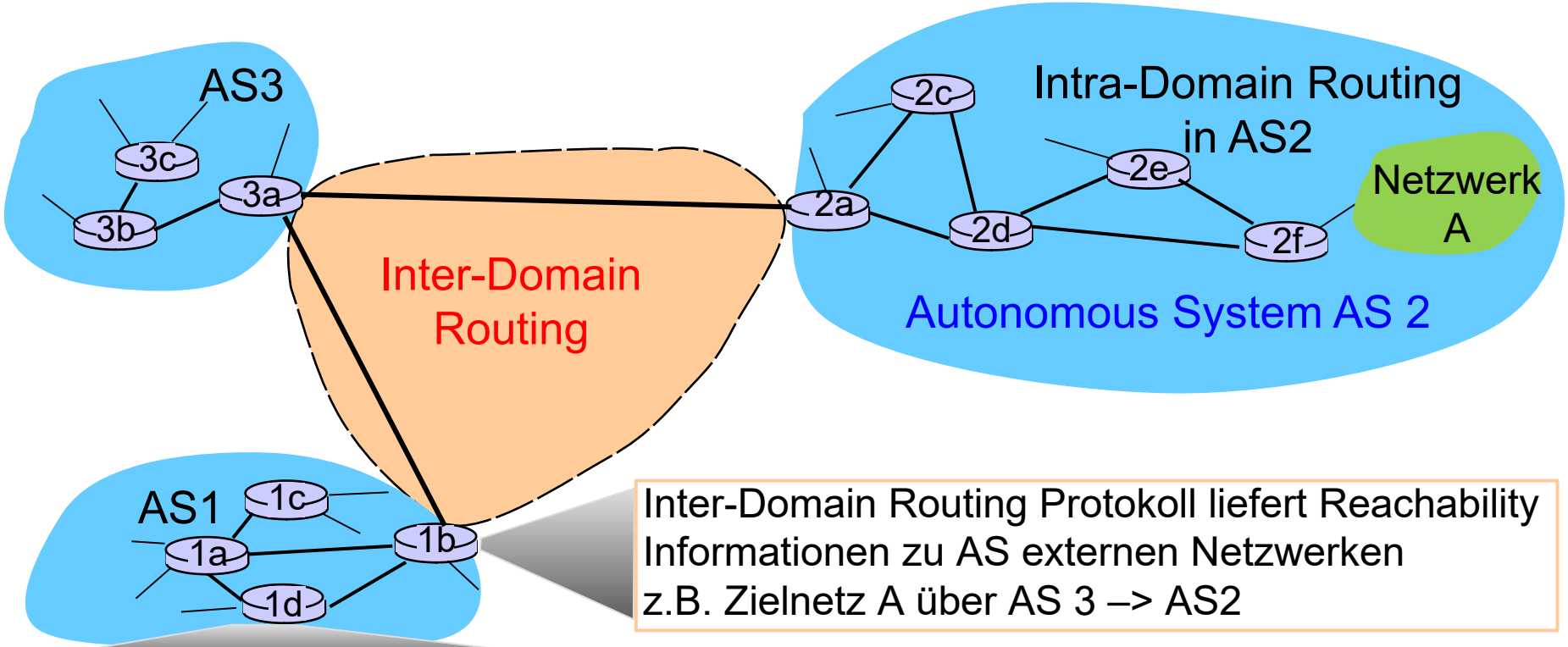
Autonome Systeme (AS)

- Ein Autonomes System ist eine Region im Internet, die autonom und unabhängig administriert wird
- Router werden in autonomen Systemen zusammengefasst
  - Autonome Systeme werden auch als Routing-Domain bezeichnet

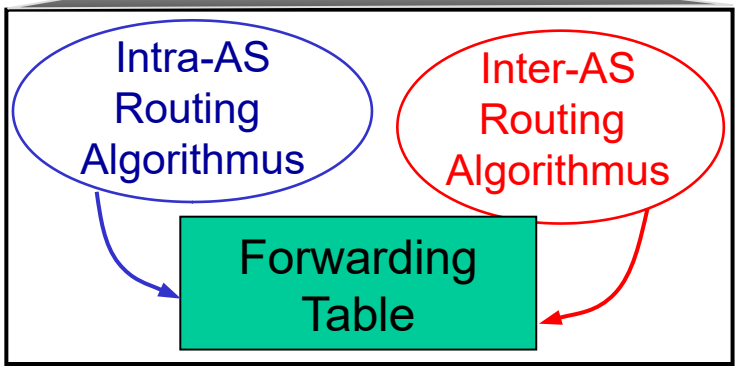
Autonome Systeme ermöglichen Hierarchisches Routing

- **Intra-Domain-Routing (Intra-AS-Routing)**
  - Router innerhalb des AS
  - Router im selben AS verwenden das gleiche Intra-AS-Routing-Protokoll
  - Je AS ein anderes Intra-AS-Routing-Protokoll
- **Inter-Domain-Routing (Inter-AS-Routing)**
  - Routing zwischen verschiedenen AS
  - Gateway-Router der verschiedenen AS' sind miteinander verbunden

Inter- und Intradomain Routing



Inter-Domain Routing Protokoll liefert Reachability Informationen zu AS externen Netzwerken
z.B. Zielnetz A über AS 3 –> AS2



Forwarding Table wird durch Intra- und Inter-Domain Routing Algorithmen erstellt

- Inter-AS Protokoll verteilt Reachability Info zu externen Netzen, z.B. Netzw. A über Router 1b
- Intra-AS Routing liefert Interface des Least-Cost-Path zum AS-internen Gateway-Router 1b

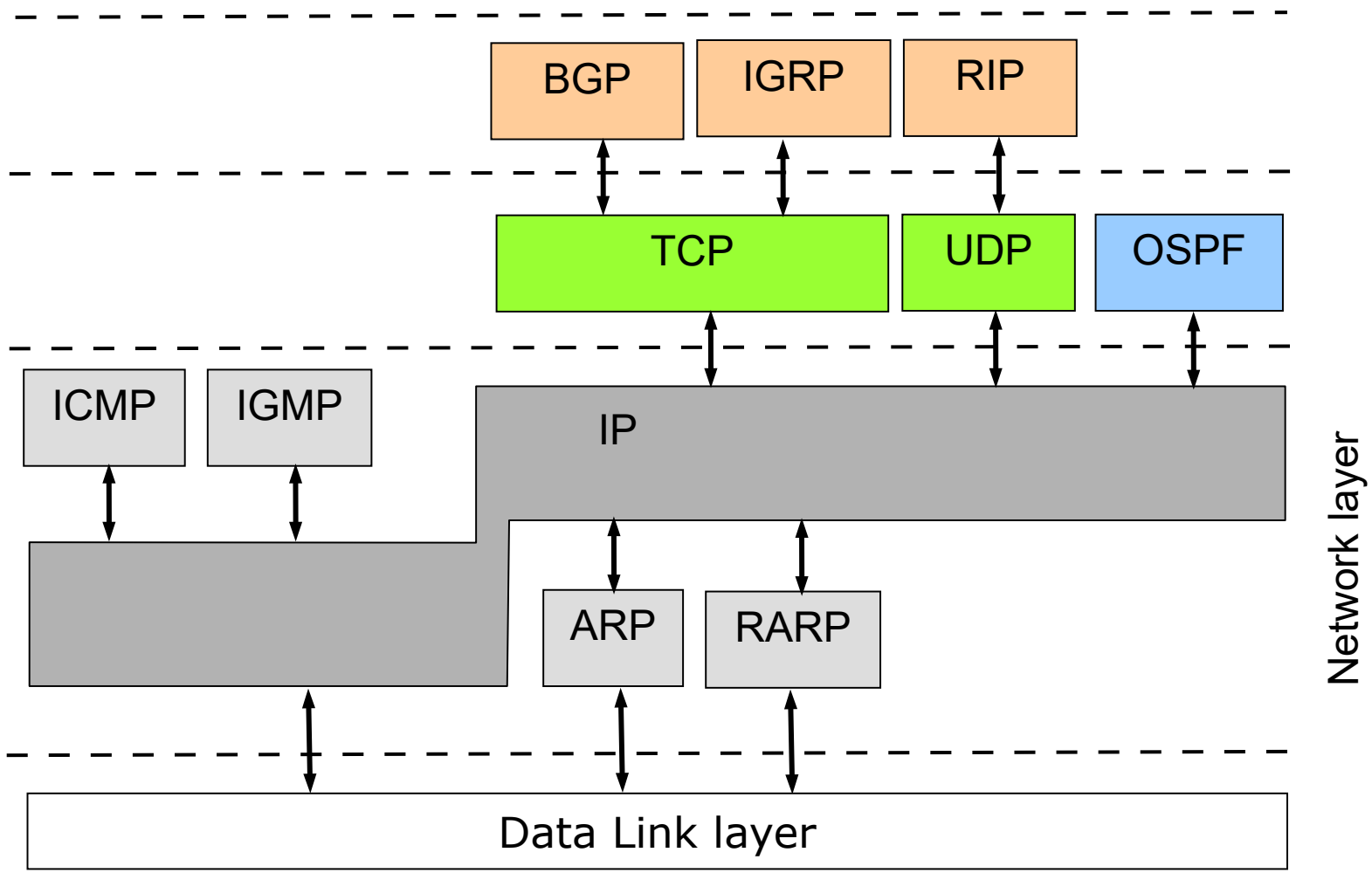
Intra-Domain Routing

- Routing innerhalb des AS‘s
- Auch als **Interior Gateway Protocol** (IGP) bezeichnet
- Je AS ein anderes Intra-Domain Routing Protokoll möglich
- Routing-Metrik
  - Performance- oder Kosten-metrik im Vordergrund
  - Routing-Tables werden nur innerhalb des AS propagiert
- Beispiele: RIP, OSPF

Inter-Domain Routing

- Routing zwischen AS‘s
- Auch als **Exterior Gateway Protocols** (EGP) bezeichnet
- Meist ein Gateway-Router je AS fürs Inter-Domain Routing zuständig
- Routing-Metrik
  - Metrik orientiert sich an Policies des Providers
  - Beispiel: wer darf zu wem Transitverkehr durchleiten
  - Performance meist nicht wichtig
- Beispiel: BGP

Inter- und Intradomain Routing (3)



Open Shortest Path First (OSPF)

Intradomain-Routing-Protokoll

- „Open“ bedeutet öffentlich verfügbar
- 1989: OSPFv1
- 1998: OSPFv2 für IPv4
  erweitert durch Support für IPv6 und CIDR
- Support für Uni- und Multicast

OSPF verwendet den Link-State-Algorithmus

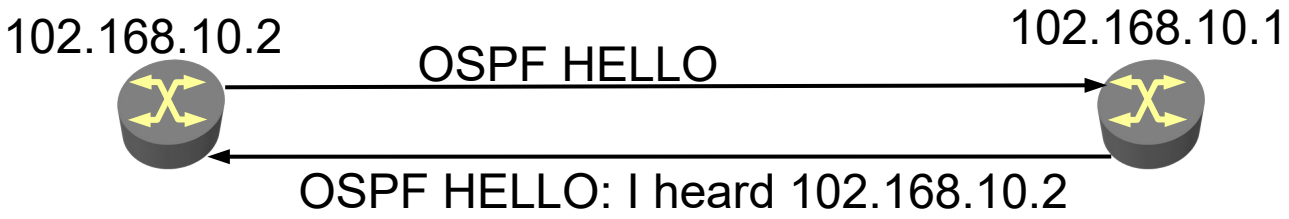
- Mehrere Wege gleicher Kosten zu einem Ziel möglich
  - Ermöglicht Lastverteilung auf mehrere Wege und erhöhte Zuverlässigkeit
- Type-of-Service-Routing
  - je Link, abhängig vom IP-TOS, mehrere Metriken möglich
  - z.B. Satelliten-Link für Best Effort, Breitband-Link für Real-Time-Dienste

OSPF Eigenschaften

OSPF – Meldungen

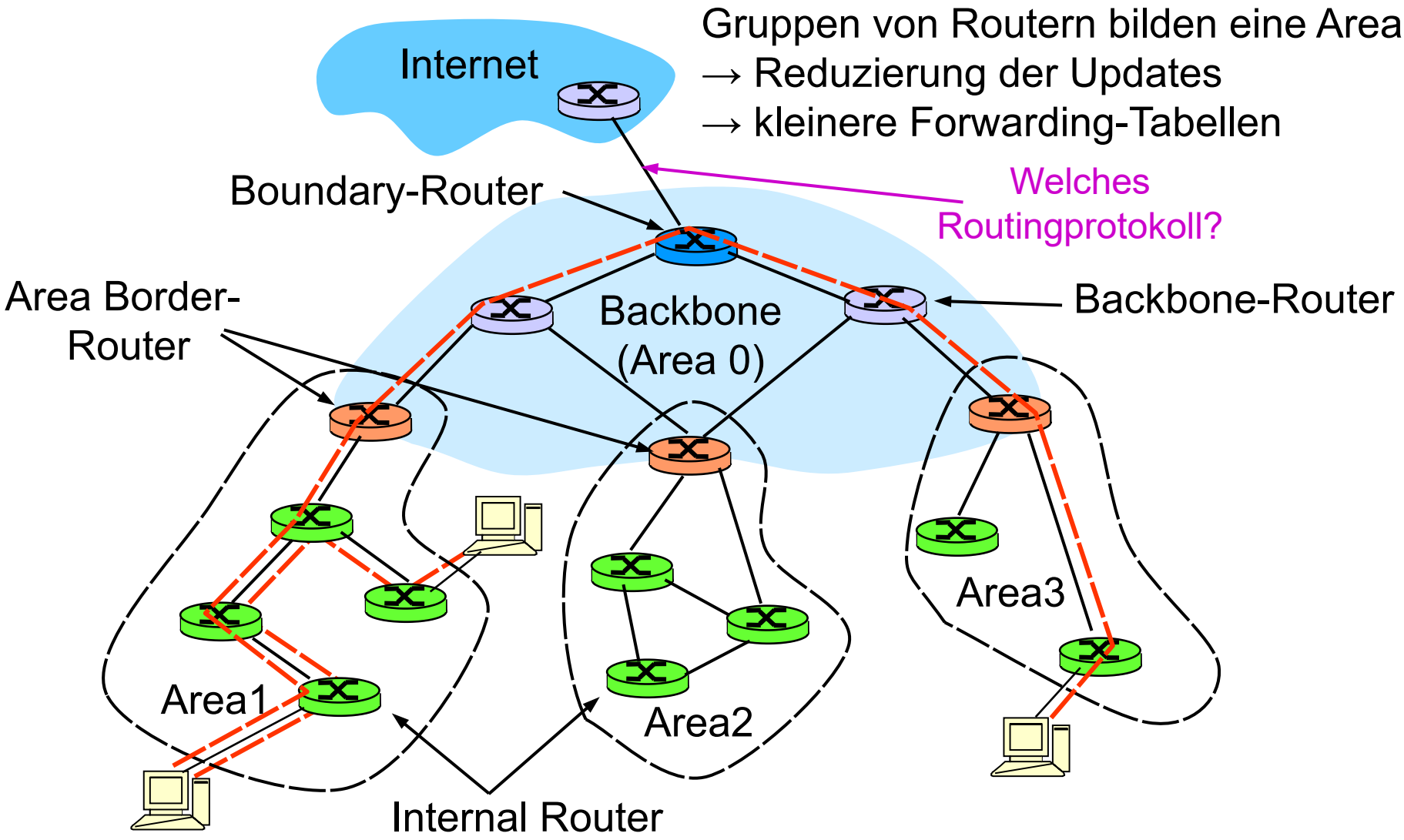
- OSPF-Meldungen werden direkt über IP an die Nachbar-Router versendet
- Sicherheit: OSPF Meldungen können authentifiziert werden

Adjacency: Discovery of Neighbors



- Erkennen eines Nachbar-Routers mittels eines „HELLO“-Protokolls
  - Neuer Router sendet OSPF-Meldung HELLO in angeschlossene Netze
  - Nachbar-Router antworten ihrerseits mit HELLO Meldungen
- Periodischer Austausch von HELLO Meldungen
- 40 Sekunden kein HELLO vom Nachbar-Router → Router ausgefallen

Hierarchisches Routing mit OSPF



Aufteilung des AS in OSPF - Areas

- Eine Backbone Area, mehrere lokale Areas
  - LSA‘s werden nur innerhalb einer OSPF-Area ausgetauscht
  - Jeder Router kennt nur die Topologie seiner Area
- Backbone-Router (BR): Router des Backbone-Bereiches
- Internal Router (IR)
  - hat nur Nachbar-Router innerhalb der eigenen Area
  - hat genau eine Link State Database (LSDB) für seine Area
- Area Border Router (ABR)
  - besitzt für jede Nachbar-Area eine LSDB
  - verteilt Shortest Path Info zu anderen Bereichen im eigenen Bereich
  - verteilt aggregierte Zustandsinfo der eigenen Area an andere ABR‘s
- Boundary Router (ASBR)
  - verwaltet externe Routen zu Zielen in anderen AS
  - externe Routen werden im gesamten eigenem AS bekannt gegeben

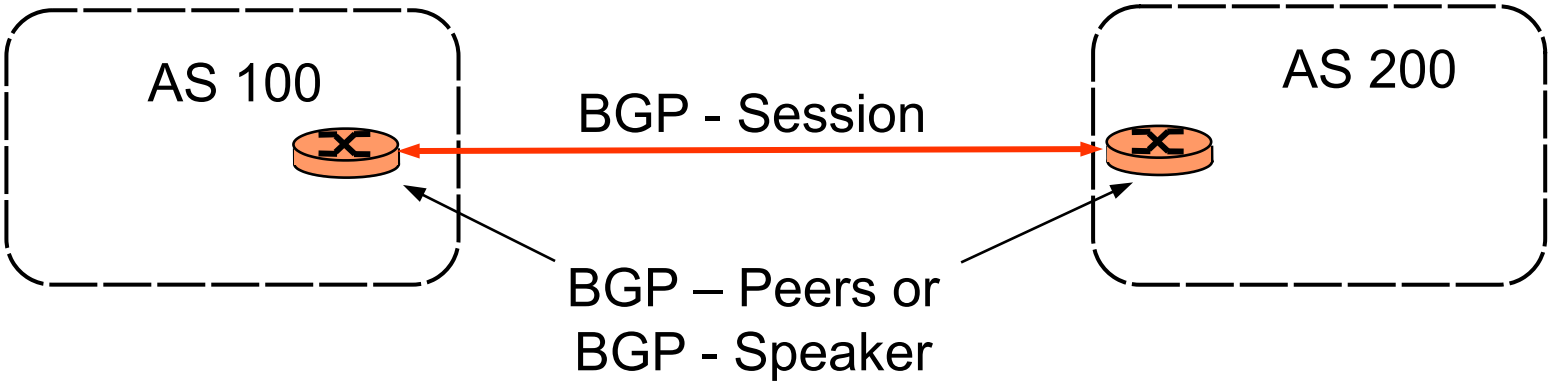
Interdomain Routing Protokoll

- aktuelle Version BGP-v4 (RFC 4271 / 4760)
- Erreichbarkeit der Netze steht im Vordergrund, nicht optimale Wege
- Unterstützung für Classless Inter-Domain Routing (CIDR)

Pfad-Vektor-Protokoll (erweitertes Distanz-Vektor Verfahren)

- Routing Messages enthalten komplette Wege (Routen) als Liste von ASs
- schleifenfreie Wege einfach realisierbar
- Information über mehrere alternative Wege zu einem Zielnetz vorhanden
- Auswahl eines Weges basiert nicht auf Kosten sondern auf Policies
- zum Beispiel:
  - benutze den Weg mit der minimalen Anzahl von Transit-AS
  - transportiere keinen Verkehr des Providers A
  - Empfangene Updates werden an Nachbarn (Peers) weitergegeben

BGP Peers



- BGP-Router, die eine BGP-Session unterhalten heißen **Peers**
- diese werden auch **BGP-Speaker** genannt (da sie BGP „sprechen“)
- Je zwei BGP-Router (Peers) etablieren eine TCP-Session (Port 175)
- Die Peer-Router werden konfiguriert und nicht automatisch ermittelt
- Router tauschen BGP-Messages aus

BGP Route Selection

- Falls ein Router mehr als eine Route für ein Präfix erhält
- Auswahlregeln für eine Route (16 Regeln nach Prioritäten)

1) Local Preference Value Attribute

→ Routing-Policy: gibt die Wichtigkeit einer AS-internen Route zum nächsten AS der Route (next-hop-AS) an

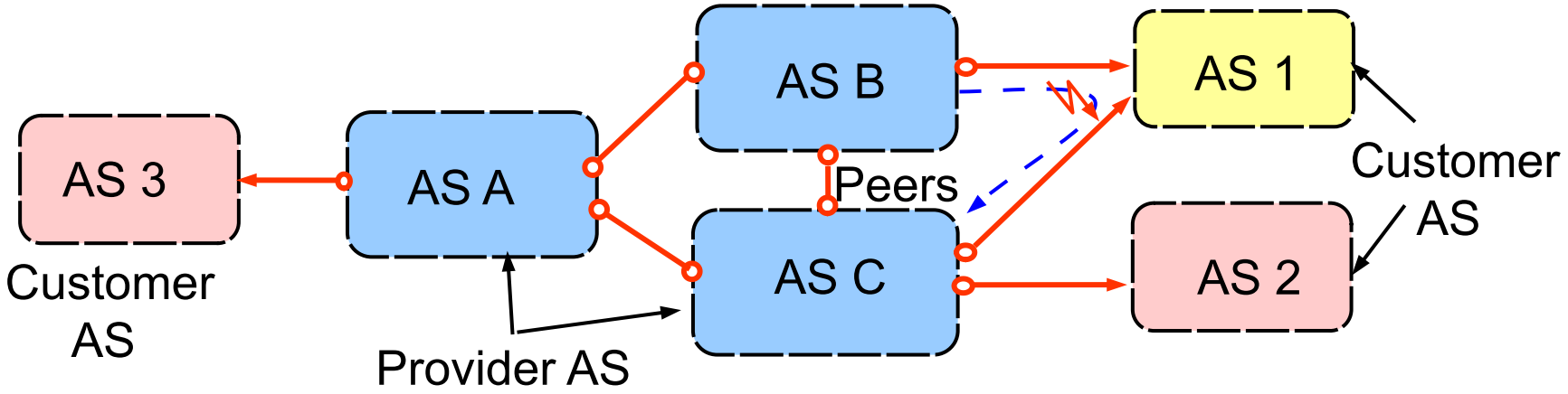
2) Shortest AS-Path

3) Dichtester Next-Hop Router: **Hot Potato Routing**

→ Es wird der BGP Router gewählt, der den Shortest Path zum nächsten AS der Route (next-hop-AS) hat

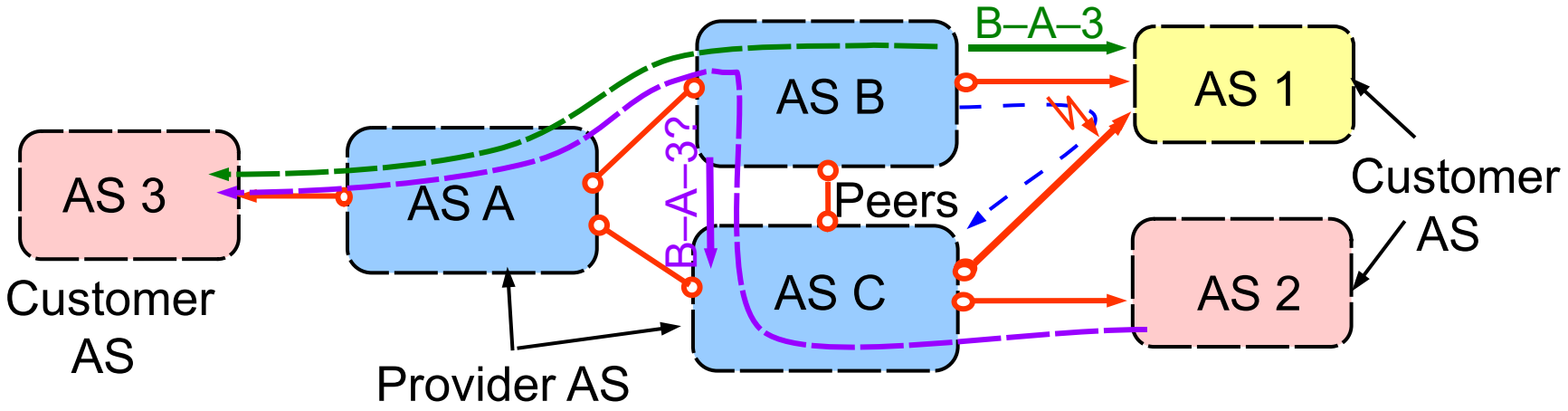
4) Weitere Kriterien

BGP Route Selection



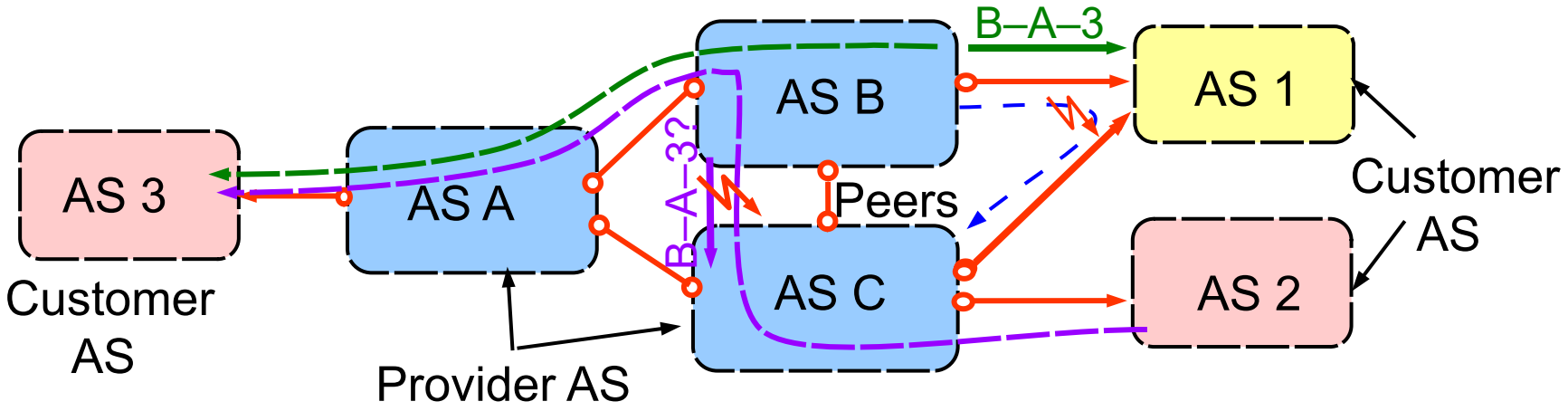
- AS A, AS B und AS C sind **Provider Networks**
- AS 2 und AS 3 sind Customer – Stub AS
- AS 1 ist ein Dual-homed Stub AS (Anschluss an zwei Netzwerken)
  - AS 1 möchte keinen Transitverkehr von AS B nach AS C transportieren
    - z.B. zu einem Netzwerk-Präfix in AS C
  - Policy: AS 1 sendet keine Informationen zu Ziel-Netzen die in einem anderen AS liegen, außer zu Ziel-Netzen in seinem eigenen AS
    - AS 1 verhält sich wie ein reines Stub-AS

BGP Route Selection



- AS A teilt AS B die Route A - 3 zu einem Netzwerk-Präfix in AS 3 mit
- AS B teilt AS 1 die Route B – A – 3 mit
- Frage: Soll AS B die Route B – A – 3 auch an AS C mitteilen?

BGP Route Selection



- AS A teilt AS B die Route A - 3 zu einem Netzwerk-Präfix in AS 3 mit
- AS B teilt AS 1 die Route B – A – 3 mit
- Frage: Soll AS B die Route B – A – 3 auch an AS C mitteilen?
- Nein: AS B erhält keinen direkten “Revenue” für diesen Transit-Verkehr, da eigene Customer (z.B. in AS1) weder Ziel noch Quelle sind.
- AS B möchte, dass dieser Verkehr direkt über AS C läuft.
- Policy: AS B gibt nur Routen an AS C weiter, in denen seine Customer als Ziel oder Quelle eingetragen sind

Probleme

- ISPs geben neue Routen bekannt um zusätzlich Verkehr zu erhalten
  - zusätzlicher Gewinn
  - Verkehr auf Inhalt analysieren
- ISPs blockieren Routen
  - Konkurrenten beeinträchtigen
  - gesellschaftspolitische Einflussnahme
- Fehlerhafte BGP-Updates durch falsch konfigurierte Router
  - April 2010: 37000 IP-Netze werden fälschlich über chinesischen ISP geleitet
  - Juni 2010: deutscher Knoten DE-CIX behindert den Verkehr zwischen ISPs
  - Oktober 2021: Facebook nimmt sich versehentlich selbst vom Netz

Lösungsvorschläge

- S-BGP(secure BGP) oder soBGP (secure origin BGP, Cisco)
  - Authentifizierung und Autorisierung der Absender (mittels Zertifikate)
  - Verschlüsselung und Integritätsschutz der Messages (IPSec, MD5)
- Praktisch noch nicht realisiert