Betriebssysteme und Netzwerke
Vorlesung N07

Artur Andrzejak

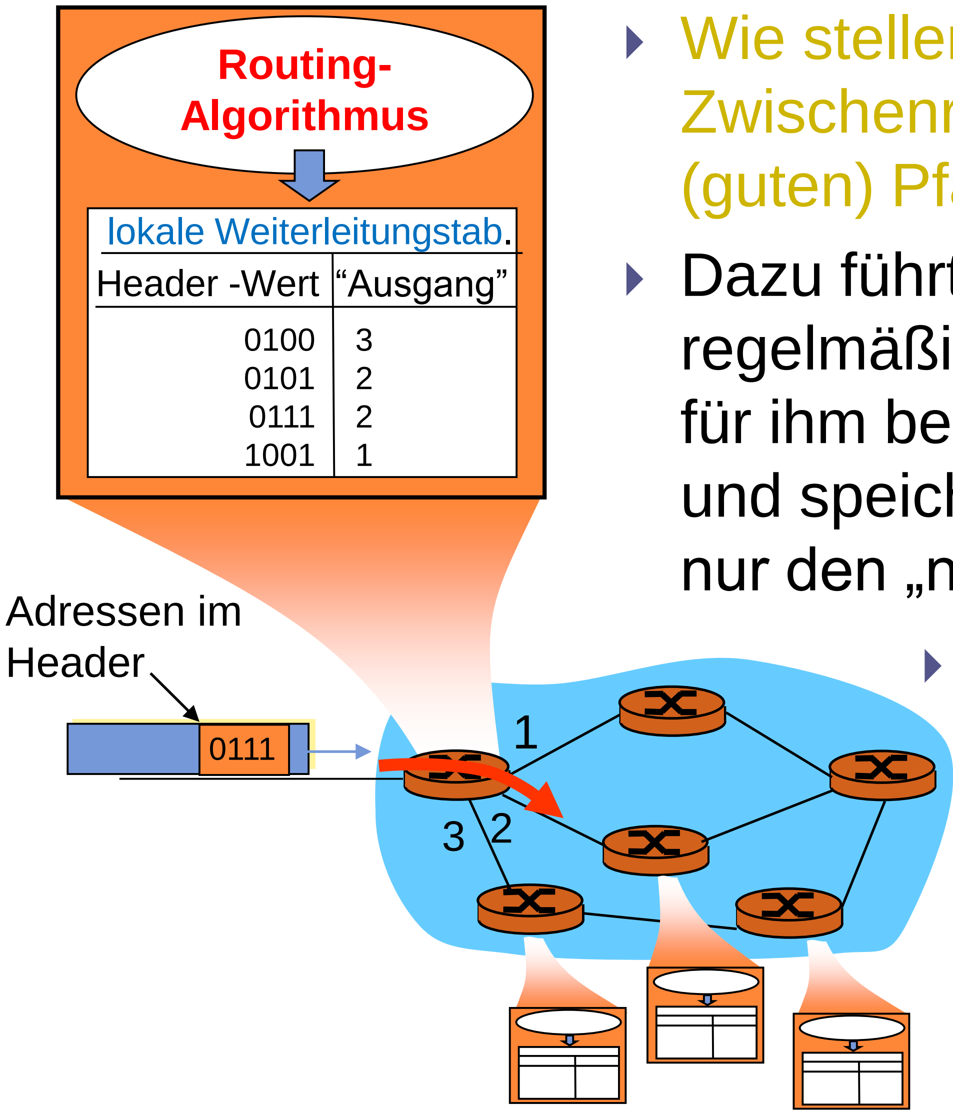
Routingalgorithmen (Netzwerkschicht)

- TCP / IP - An animated discussion
  - https://www.youtube.com/watch?v=RbY8Hb6abbg

Routing von Paketen

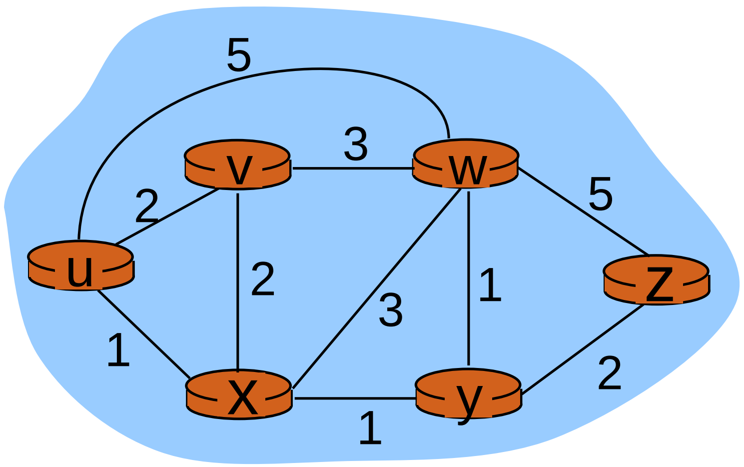
- Ein Host ist üblicherweise einem <u>Router</u> des lokalen Netzwerks zugeordnet, dem **Standard-Gateway** des Hosts
- Das Problem, ein Paket vom Quellhost zum Zielhost zu leiten, wird auf die Weiterleitung eines Paketes vom **Quellrouter** zum **Zielrouter** reduziert
- Das **Routing** ist also das Leiten eines Pakets vom **Quellrouter Q** zum **Zielrouter Z** über die Router dazwischen
- Zwei Probleme:
  1. Wie finden wir einen (guten) Pfad zwischen Q und Z?
  2. Wie stellen wir sicher, dass jeder Zwischenrouter das Paket entlang dieses (guten) Pfades leitet?

Routing und das Weiterleiten



- Wie stellen wir sicher, dass jeder Zwischenrouter das Paket entlang dieses (guten) Pfades leitet?
- Dazu führt jeder (Zwischen-) Router regelmäßig einen **Routing-Algorithmus** für ihm bekannte (ggf. ferne) Ziele aus, und speichert in der Weiterleitungstabelle nur den „nächsten Hop“
  - Damit wird das Routing durch lokale Entscheidungen an jedem Router umgesetzt:
    - Zu welchem direkt verbundenem Router wird das Paket weitergeleitet?

Abstraktes Graphenmodell eines Computernetzwerkes



Graph: **G** = (N,E)

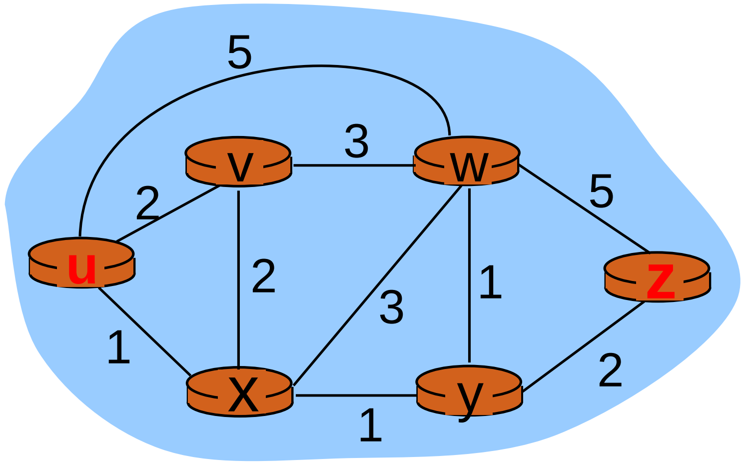
Knoten (nodes):

N = Menge der Router = { u, v, w, x, y, z }

Kanten (edges):

E = Menge der Leitungen zwischen den Routern =
{ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

Abstraktes Graphenmodell: Definitionen



- **c**(x,x') = **Kosten der Übertragung** (bzw. Leitung) von x zu x'
  - Z.B. c(w,z) = 5
- **Kosten** könnten sein:
  - Immer 1
  - Proportional zur Überlastung
  - Umgekehrt-Proportional zur Bandbreite

Kosten des Pfads $(x_1, x_2, x_3, \ldots, x_p)$:= $c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

Routing-Problem: Was ist der kostengünstigste Pfad zwischen u und z?

Routing-Algorithmus: Algorithmus, der einen kosten-günstigsten Pfad bestimmt

Klassifikation von Routing-Algorithmen

- Globale or dezentrale Information?

Global:

- Alle Router kennen die komplette Topologie und die Kosten der Leitungen (links)
- "**Link-State**"-Algorithmen

Dezentral:

- Router kennen nur die Router, mit denen sie direkt verbunden sind (und die Kosten der jeweiligen Leitungen)
- Kostenberechnung ist ein iterativer Prozess mit Datenaustausch zwischen den Nachbarn
- "**Distanzvektor**"-Algorithmen

- Statische oder dynamische Algorithmen?
- Statisch:
  - Routing-Pfade ändern sich langsam mit der Zeit
- Dynamisch:
  - Routing-Pfade ändern sich häufig
  - Periodische Aktualisierungen nötig
  - Subkategorie von dynamisch:
    - **Lastsensitive** bzw. **lastinsensitive** Algorithmen

Routingalgorithmen:
Link-State-Algorithmen (LS)

Ein Link-State-Algorithmus

Annahmen:

- Netzwerktopologie und die Kosten aller Knoten bekannt
  - Durch “link state broadcast”
  - Alle Knoten haben dieselben Informationen

Dijkstra-Algorithmus

- Berechnet die kosten-günstigsten Pfade vom **Quellknoten u** zu allen a.
- Ergibt die Weiterleitungstabelle für den Quellknoten

Notation:

- **c(x,y):** Kosten der Leitung von x nach y; = ∞ falls x, y keine Nachbarn sind
- **N':** Teilmenge von Knoten v, für die der kostengünstigste Pfad von der Quelle zu *v* definitiv bekannt ist
- **D(v):** die (ggf. geschätzten) Kosten eines kostengünstigsten Pfades von der Quelle zum Knoten *v*
  - *Für v* $\in$ *N' : exakte Kosten*
  - *Für v* $\notin$ N': Abschätzungen
- **p(v):** letzter Knoten vor v (d.h. Nachbar von v) entlang des momentan kostengünstigsten Pfades von der Quelle zu *v*

Dijkstra-Algorithmus

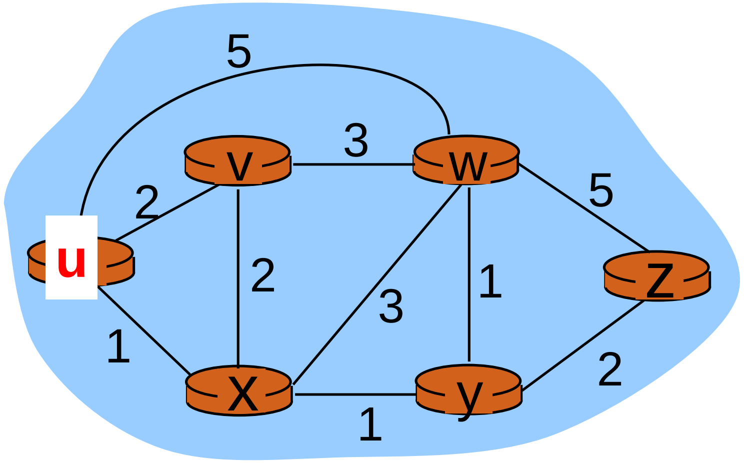
N = Menge aller Knoten

```
Initialisiere:
  N' = {u} (u ist Quellknoten)
  für alle Knoten v aus N
    if v ist ein Nachbar von u
        then D(v) = c(u,v)
        else D(v) = ∞

Wiederhole:
  finde ein w ∉ N', so dass D(w) minimal ist
  füge w zu N' hinzu
  Berechne D(v), p(v) neu für alle Nachbarn v von w mit v ∉ N':
      D(v) = min( D(v), D(w) + c(w,v) )
  /* die neuen Kosten nach v sind entweder die alten Kosten
        oder die Kosten nach w plus die Kosten von w nach v*/
bis alle Knoten in N' sind
```

Dijkstra-Algorithmus: Beispiel

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	**4**,x		**2**,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



p(s): vorheriger Knoten (Nachbar von s) entlang des momentan kostengünstigsten Pfades von der Quelle zu *s*

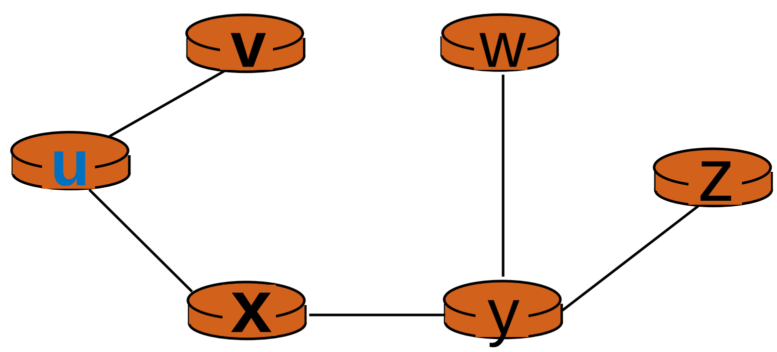
Dijkstra-Algorithmus: Beispiel /2

- Die Werte p(v) nach der letzten Operation erlauben uns, die Weiterleitungstabelle und die Routing-Pfade zu bestimmen (für u als Quellknoten)

Weiterleitungstabelle für **u**:

Ziel	"Ausgang"
v	(u,**v**)
x	(u,**x**)
y	(u,**x**)
w	(u,**x**)
z	(u,**x**)

Baum der kostengünstigsten Pfade für u:



Wie genau berechnet man diese?

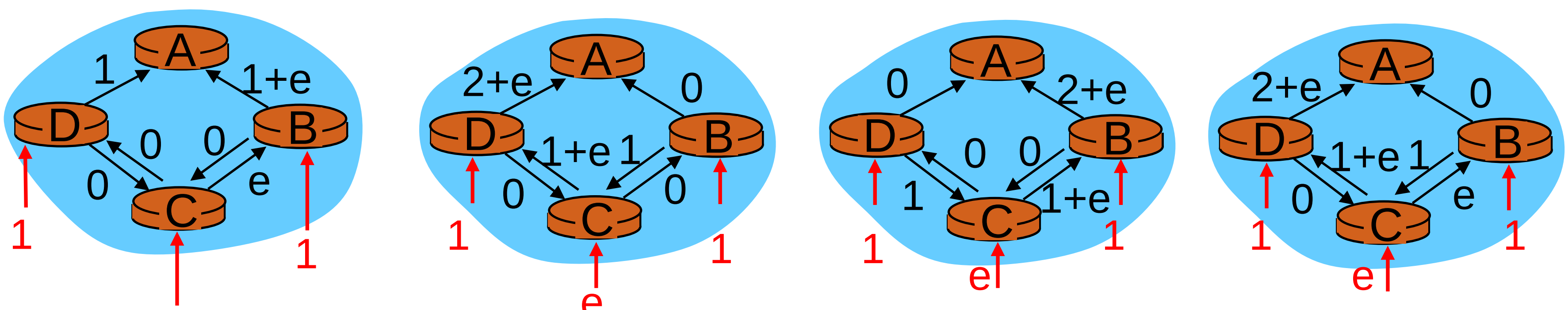
Dijkstra-Algorithmus - Diskussion

Komplexität bei n Knoten:

- jede Iteration: muss alle Knoten w (nicht in N‘) überprüfen
- max. n(n+1)/2 Vergleiche: $O(n^2)$
- Es gibt effizientere Verfahren mit Heaps => O(n log(n))

Problem: **Oszillationen** sind möglich!

- z.B., Leitungskosten = aktuelles Verkehrsaufkommen



Anfangs

C, B entdecken einen besseren Pfad nach A

D, C, B entdecken den gegen den Uhrz.-S. verlaufenden Pfad nach A

D, C, B entdecken den gegen den Uhrz.-S. verlaufenden Pfad nach A

Routingalgorithmen: Distanzvektor-Algorithmen (DV)

Bellman-Ford-Gleichung

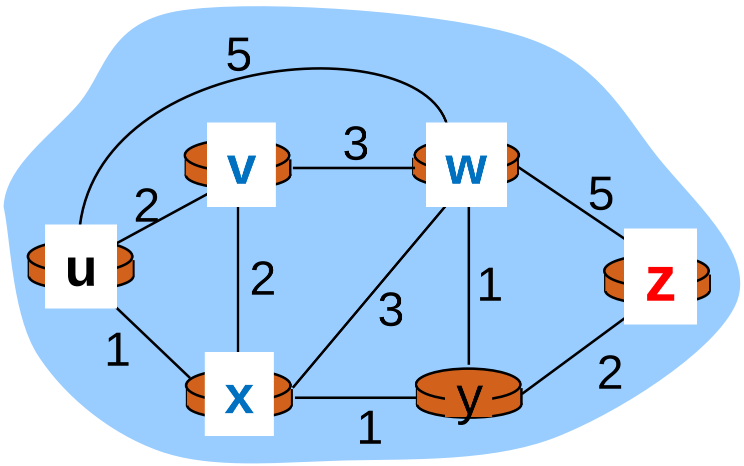
Bellman-Ford-Gleichung:

- Sei $d_x(y)$:= die Kosten eines kostengünstigsten Pfads von **x** zum Ziel **y**
- **c(v,w)** := Kosten der Leitung (Kante) von **v** nach **w**
- **neigh(w)** := Menge der Nachbarn von **w**
- Dann gilt:

$$d_x(y) = \min_{v \in neigh(x)} \{c(x,v) + d_v(y)\}$$

- D.h. $d_x(y)$ ist gleich den Kosten der direkten Verbindung von **x** zu einem Nachbar **v** von **x** plus die geringsten Kosten von **v** zum Ziel **y**

Bellman-Ford Gleichung - Beispiel



Wir wollen $d_u(z)$ berechnen, und müssen uns erstmal die <u>Nachbarn von u</u> genauer anschauen: x, v, w

Die B-F-Gleichung sagt:

$$d_u(z) = \min \begin{cases} \mathbf{u \rightarrow x} \; + \text{kleinste Pfadkosten von } \mathbf{x} \text{ bis z} \\ \mathbf{u \rightarrow v} \; + \text{kleinste Pfadkosten von } \mathbf{v} \text{ bis z} \\ \mathbf{u \rightarrow w} \; + \text{kleinste Pfadkosten von } \mathbf{w} \text{ bis z} \end{cases}$$

Was hilft uns das beim Routing?
Knoten, der uns ein Minimum liefert,
soll als nächster Hop (von u aus) gewählt werden!

Distanzvektor-Routing-Algorithmus

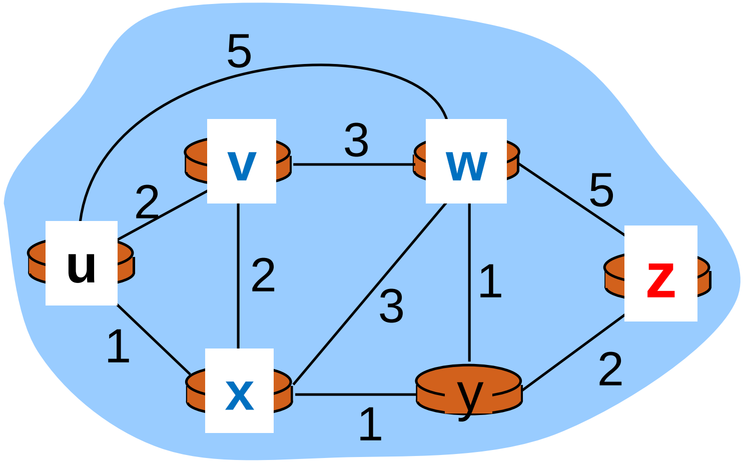
- Sei $D_q(z)$ = Schätzung der Kosten von q nach z
  - Knoten **q** kennt Kosten zu jedem Nachbarn v: $c(q,v)$

- Knoten **q** berechnet seinen **Distanzvektor** (**DV**) D_q = Vektor der Schätzungen $D_q(z)$ für jeden Knoten z im Netzwerk, d.h.
  - $D_q = [D_q(z) : z \in N]$ mit N = alle Knoten im Netz

- Knoten **q** merkt sich auch für jeden Nachbar v seinen Distanzvektor D_v

Distanzvektor - Beispiele

- Wie sieht der Distanzvektor D_u von **u** aus? (#Einträge?)
- Welche DV kennt noch **u**, und wie sehen diese aus?



- $\mathbf{D_u}$=[D_u(u), D_u(x), D_u(v), D_u(w), D_u(y), D_u(z)]
- **u** kennt auch noch D_v, D_x, D_w:
  - $\mathbf{D_v}$=[D_v(u), D_v(x), D_v(v), D_v(w), D_v(y), D_v(z)]
  - $\mathbf{D_x}$=[D_x(u), D_x(x), D_x(v), D_x(w), D_x(y), D_x(z)]
  - $\mathbf{D_w}$=[D_w(u), D_w(x), D_w(v), D_w(w), D_w(y), D_w(z)]

Distanzvektor-Routing-Algorithmus

Algorithmus für jeden Knoten q:

Warte auf eine Änderung von c(,) oder DV–Aktualisierung

Berechne die neuen Schätzungen in DV

Falls DV zu irgendeinem Knoten verändert wurde, *benachrichtige* die Nachbarn

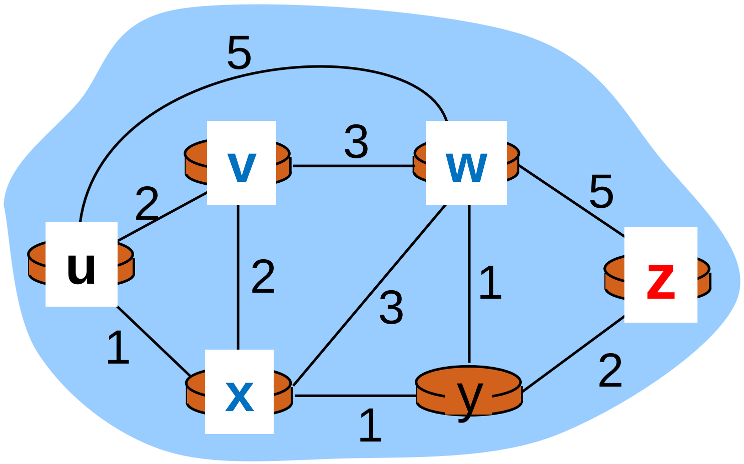
Für jede Komponente des DV, hier mit <u>Zielknoten y</u>:

$D_q(y) \leftarrow \min_v\{c(q,v) + D_v(y)\}$,

über alle Nachbarn v von q

Distanzvektor - Beispiele

1. Angenommen, in D_x ändert sich $D_x(y)$ - was aktualisieren wir in D_u?
2. Angenommen, es ändert sich c(u,w) - was wird in D_u neu?



- $D_u=[D_u(u), D_u(x), D_u(v), D_u(w), D_u(y), D_u(z)]$
- $D_v=[D_v(u), D_v(x), D_v(v), D_v(w), D_v(y), D_v(z)]$
- $D_x=[D_x(u), D_x(x), D_x(v), D_x(w), D_x(y), D_x(z)]$
- $D_w=[D_w(u), D_w(x), D_w(v), D_w(w), D_w(y), D_w(z)]$

2. Es könnten sich alle Komponenten in $D_{\underline{u}}$ ändern, bis auf $D_u(u)$

Distanzvektor-Routing-Algorithmus /3

Iterativ, asynchron:

- Jede lokale Iteration wird verursacht durch:
  - Lokale Änderung der Verbindungskosten c(,)
  - DV – Aktualisierung von einem Nachbar

Verteilt:

- Jeder Knoten hat eigentlich nur lokale Information, keine Instanz mit globaler Information nötig

Selbst-Stabilisierend:

- Interessant: Dieser Algorithmus kann in einem „beliebig schlechten" Zustand starten, wird aber zu einer Lösung konvergieren

Jeder Knoten:

Warte auf eine Änderung von c(,) oder DV–Aktualisierung

↓

Berechne die neue Schätzungen in DV

↓

Falls DV zu irgendeinem Knoten verändert wurde, *benachrichtige* die Nachbarn

$$D_x(y) = \min\{c(x,y) + D_y(y),\ c(x,z) + D_z(y)\} = \min\{2+0\ ,\ 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z),\ c(x,z) + D_z(z)\} = \min\{2+1\ ,\ 7+0\} = 3$$

Knoten x

cost to

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

Knoten y

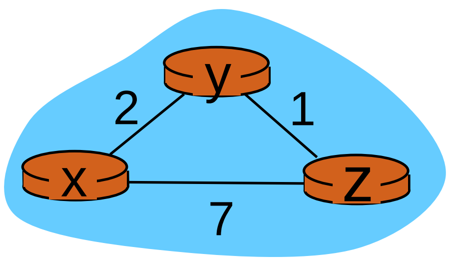
cost to

from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

Knoten z

cost to

from	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0



Zeit

$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$
$= \min\{2+0 , 7+1\} = 2$

$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$
$= \min\{2+1 , 7+0\} = 3$

Knoten x

cost to

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

Knoten y

cost to

from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

cost to

from	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

Knoten z

cost to

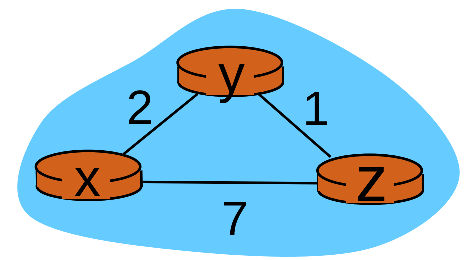
from	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

cost to

from	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

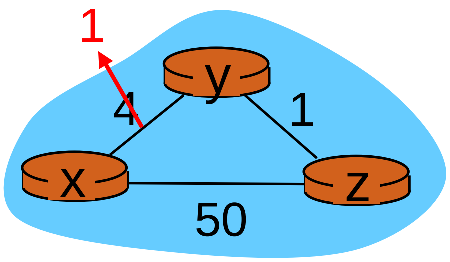


Zeit

Änderung der Verbindungskosten /1

Bei Änderung der Verbindungskosten:

- Knoten erkennt die lokale Änderung
- Er aktualisiert die c(,) – Werte und berechnet DV neu
- Falls DV verändert, werden die Nachbarn benachrichtigt



"good news travels fast"

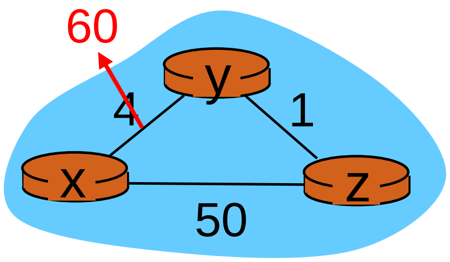
Bei t_0, *y* entdeckt die Änderung von c(y,x), aktualisiert DV, benachrichtigt die Nachbarn

Bei t_1, *z* empfängt die Aktualisierung von y und aktualisiert seine Tabelle; Dann berechnet er die neuen Kosten zu x und sendet den Nachbarn seinen DV

Bei t_2, *y* erhält die Aktualisierung von *z* und aktualisiert seine Tabelle; Die Kosten von *y* haben sich nicht geändert, also schickt er keine Benachrichtigungen

Änderung der Verbindungskosten /2

- Reduktion der Verbindungskosten führt zu einer schnellen Stabilisation …
- Aber: “**Bad news travels slow**”
  - Änderung der Kosten von 4 auf 60 bewirkt 44 Iterationen (!) bevor der Algorithmus aufhört (stabilisiert)
  - Das sogennante “**count to infinity**”- Problem



Zusammenfassung

– Netzwerkschicht –

- Routingalgorithmen (Netzwerkschicht)
  - Link-State-Algorithmen
  - Distanzvektor-Algorithmen

- Quellen:
  - Kurose / Ross Kapitel 4
  - Kurose / Ross Kapitel 5
  - Wikipedia

Danke.