Mittuniversitetet
MID SWEDEN UNIVERSITY

Datornätverk
Routing

Lennart Franked

May 11, 2026

Mittuniversitetet
MID SWEDEN UNIVERSITY

1. Introduktion

2. Routingalgoritmer

3. Routingprotokoll

Läsanvisningar

Denna föreläsning är baserad på

- [2, Chapter 8.1 — 8.3]

Denna föreläsning är ett komplement till läsanvisningarna.

Mittuniversitetet
MID SWEDEN UNIVERSITY

Table of Contents

Mittuniversitetet
MID SWEDEN UNIVERSITY

Introduktion till Routing

- Datorer vars nätverksprefix (enligt subnätmasken) matchar kan kommunicera direkt med varandra över länklagret.
- Noder i olika subnät kommunicerar via en eller flera routrar.
- En **routingtabell** talar om vart ett paket ska skickas härnäst för att nå sin destination.
- Varje nod har sin egen tabell, med en post per känt destinationsnät och tillhörande nästa hopp.

Mittuniversitetet
MID SWEDEN UNIVERSITY

Routingtabellen

- Vanlig dator har oftast lokala subnät och default gateway.
- En router har oftast kännedom om alla nätverk inom sitt AS och en default route.
- Tier-1-routrar håller en full BGP-tabell med alla globalt routbara prefix och saknar default route.

Mittuniversitetet
MID SWEDEN UNIVERSITY

Routingtabellen

Varje gång ett IP-paket ska skickas, konsulteras routingtabellen för att avgöra hur paketet ska vidarebefordras.

Routingtabell

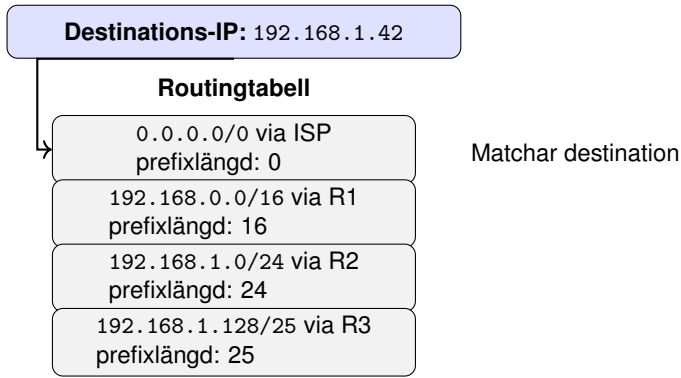
```
lennart@ID20648900:~# ip route list
default via 10.14.1.254 dev eth0  proto static
10.14.2.0/24 via 10.14.1.253 dev eth0  proto static
10.14.1.0/24 dev eth0  proto kernel  scope link  src 10.14.1.147

lennart@ID20648900:~# netstat -nr
Kernel IP routing table
Destination    Gateway        Genmask          Flags  Iface
0.0.0.0        10.14.1.254    0.0.0.0          UG     eth0
10.14.1.0      0.0.0.0        255.255.255.0    U      eth0
10.14.2.0      10.14.1.253    255.255.255.0    U      eth0
```

Mittuniversitetet
MID SWEDEN UNIVERSITY

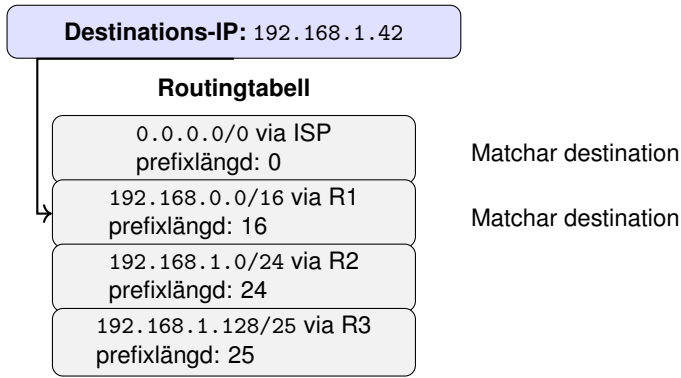
Default gateway

Anger vart paket skall skickas om destinationen ej finns i routingtabellen.
Anges som `0.0.0.0 0.0.0.0` (destination/mask).

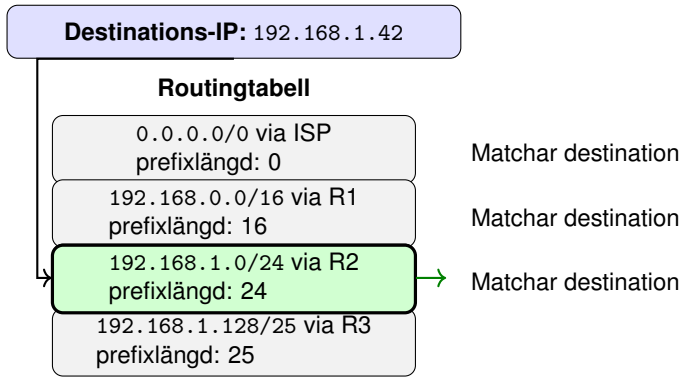
Längsta matchning



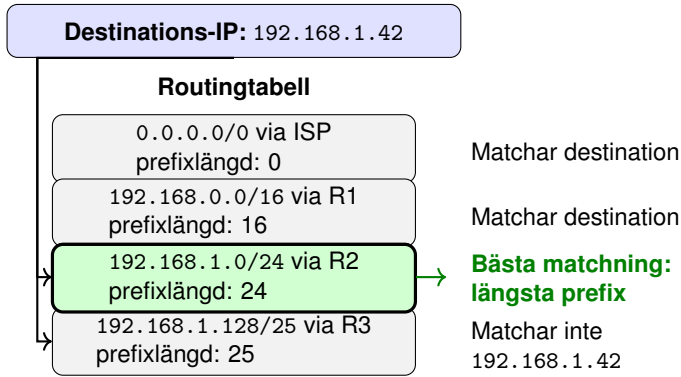
Längsta matchning



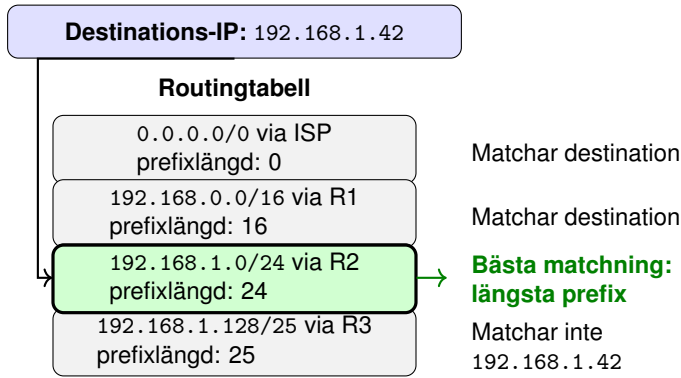
Längsta matchning



Längsta matchning



Längsta matchning



Idé: flera poster kan matcha, men routern väljer den med flest fasta bitar.
`192.168.1.42` matchar `/0`, `/16` och `/24`; välj därför `192.168.1.0/24`.

Mittuniversitetet
MID SWEDEN UNIVERSITY

Längsta matchning — binär jämförelse

Destinations-IP: `11000000.10101000.00000001.00101010` (192.168.1.42)

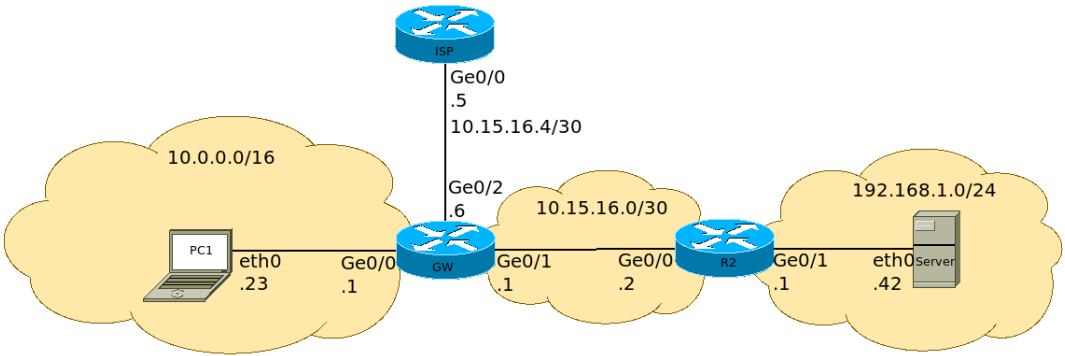
Rutt	Binär adress	Matchande bitar
`0.0.0.0/0`	`00000000.00000000.00000000.00000000`	0
`192.168.0.0/16`	`11000000.10101000.00000000.00000000`	16
`192.168.1.0/24`	`11000000.10101000.00000001.00000000`	**24 (bäst)**
`192.168.1.128/25`	`11000000.10101000.00000001.10000000`	matchar ej

Idé: jämför prefixet bit för bit. Den rutt som matchar *flest* bitar från vänster väljs.
`/25` kräver att bit 25 stämmer — här är destinationens bit `0` men ruttens bit `1`, alltså ingen match.

Gateway / Next-hop

- **Next-hop** — IP-adressen till nästa router på vägen mot destinationen.
- **Gateway** — vanligen synonymt med next-hop; ofta avses specifikt utgången från det lokala subnätet (*default gateway*).
- Next-hop måste vara **direkt nåbar på länklagret**, dvs. ligga i samma subnät som ett av nodens egna interface.
  - Annars går det inte att ARP:a fram en MAC-adress och paketet kan inte skickas vidare.
- I `ip route`-utdata är next-hop adressen efter `via`:
  - `default via 10.14.1.254 dev eth0` — next-hop = `10.14.1.254`
  - `10.14.2.0/24 via 10.14.1.253 dev eth0` — next-hop = `10.14.1.253`
  - `10.14.1.0/24 dev eth0` (utan `via`) — direkt anslutet, ingen next-hop behövs

Next-hop



PC1 Routingtabell

10.0.0.0/16 iface eth0
0.0.0.0/0 via 10.0.0.1

GW Routingtabell

10.0.0.0/16 iface Ge0/0
10.15.16.0/30 iface Ge0/1
10.15.16.4/30 iface Ge0/2
192.168.1.0/24 via 10.15.16.2
0.0.0.0/0 via 10.15.16.5

R2 Routingtabell

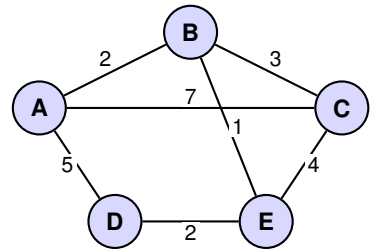
?

Server Routingtabell

?

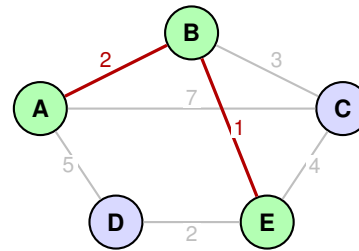
Routing som grafproblem

- **Routing** — hitta bästa vägen från källa till destination
- Nätverket modelleras som en **viktad graf**
  - Noder = routrar
  - Kanter = länkar
  - Vikter = kostnad



Lägsta kostnad

- Målet: hitta **least-cost path**
- Kostnaden kan baseras på:
  - Hop count
  - Bandbredd
  - Fördröjning
- **Least-cost tree**: från en källa till alla destinationer

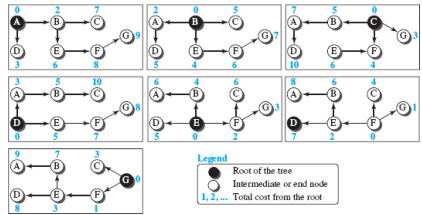


A → E: kostnad = 3

Lägsta kostnad

Least-cost Tree

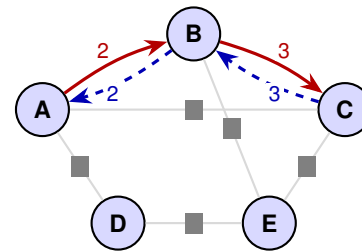
- Ett nätverk består av N noder.
- Från en nod till alla andra behövs ett träd med $N - 1$ kanter.
- Trädet väljs så att kostnaden blir så låg som möjligt.
- Gör vi detta för varje nod:
  - får vi $N - 1$ kanter per nod
  - vilket ger totalt $N(N - 1)$ kanter
- En sådan samling träd, ett per nod, kallas *least-cost trees*.



Least-cost Tree [2]

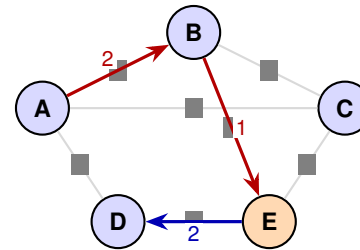
Symmetri i Least-cost Trees

- Vägen från X till Y är inversen av vägen från Y till X (förutsatt symmetriska länkkostnader).
- Gäller mellan respektive nods träd.
- Kostnaden är densamma i båda riktningar.
- Exempel:
  - $A \rightarrow C$: $A \rightarrow B \rightarrow C$
  - $C \rightarrow A$: $C \rightarrow B \rightarrow A$
  - Total kostnad: $2 + 3 = 5$



Kombinera Least-cost Trees

- En väg kan delas upp via en mellanliggande nod.
- Steg:
  - $A \rightarrow E$ i A:s träd (kostnad 3)
  - $E \rightarrow D$ i E:s träd (kostnad 2)
- Total kostnad: $3 + 2 = 5$
- Samma som direkt väg $A \rightarrow D$.



Mittuniversitetet
MID SWEDEN UNIVERSITY

Table of Contents

Hur hittar vi bästa vägen genom ett nätverk?

Köra via vägskyltar

Figure: Vägskyltar

Hur hittar vi bästa vägen genom ett nätverk?

Köra efter GPS

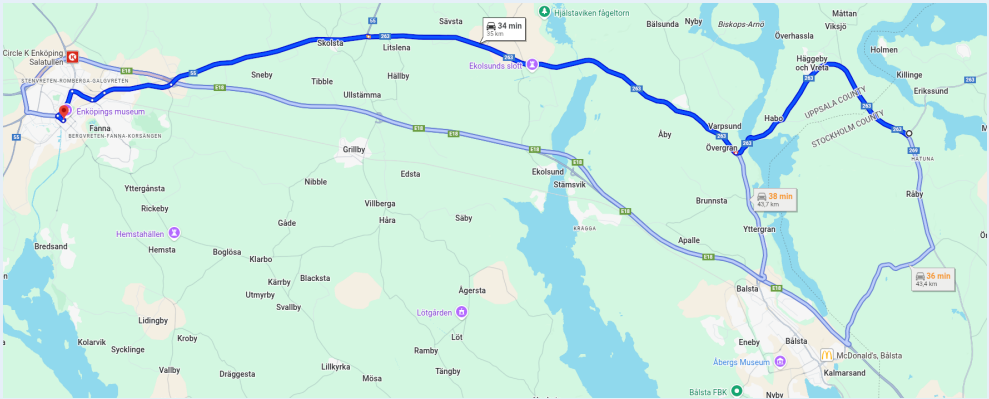
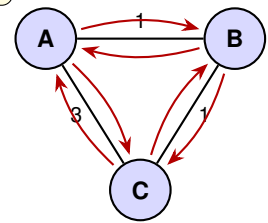


Figure: GPS

Distance-Vector Routing

- Baserad på **Bellman–Ford**
- Routingtabell: destination, kostnad, next hop
- Delar **hela tabellen** med grannar
- Iterativt tills konvergens
- Problem: **count to infinity**
  - Split horizon, poison reverse

Dest	Kostnad	Next-hop
B	1	B
C	2	B



Bellman-Ford

$$d_x(y) := \text{lägsta kostnaden mellan } x \text{ och } y$$
$$d_x(y) = \min_v\{c(x,v) + d_v(y)\}$$

- $\min_v$ kollar alla grannar v till x
- $c(x,v)$ är kostnaden mellan x och v
- $d_v(y)$ är kostnaden mellan v och y

Exempel: Bellman-Ford

$$
\begin{aligned}
d_u(z) &= \min\{c(u,v) + d_v(z), \\
&\qquad c(u,x) + d_x(z), \\
&\qquad c(u,w) + d_w(z)\} \\
&= \min\{2+5,\ 1+3,\ 5+3\} = 4
\end{aligned}
$$

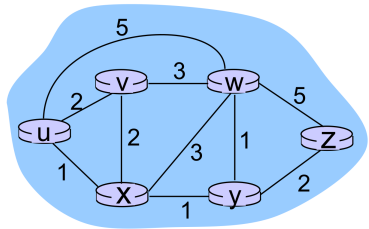


Figure: Bellman-Ford exempel [3]

Bellman-Ford

$D_x(y) :=$ uppskattad lägsta kostnad från x till y

- Nod x håller en lista $D_x = [D_x(y) : y \in N]$
- Det innebär att:
  - x måste känna till kostnaden till alla sina grannar v: $c(x, v)$
  - För varje granne v måste x känna till dess distansvektor $D_v = [D_v(y) : y \in N]$

Mittuniversitetet
MID SWEDEN UNIVERSITY

Bellman-Ford

- Alla routrar utbyter sina distansvektorer periodiskt
- Varje gång en router tar emot en ny distansvektor uppdateras routingtabellen
- För varje router $y \in N$:
  $D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\}$

Mittuniversitetet
MID SWEDEN UNIVERSITY

Nackdelar med Distance-Vector

- Känslig för routingsloopar.
- Känner inte till den övergripande nätverkstopologin.
- Långsam konvergens efter topologiförändringar.
- Kan drabbas av *count-to-infinity*-problemet.

Konvergens i DV

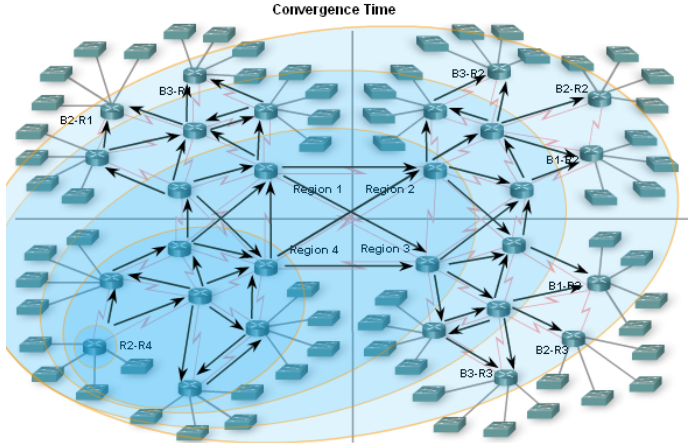
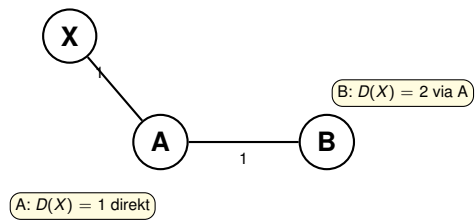


Figure: Konvergens i DV [1]

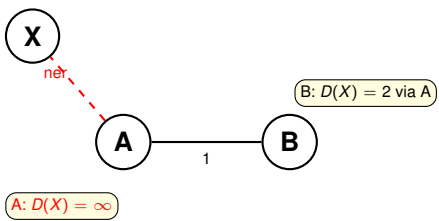
Exempel: Routingloop i Distance-Vector

- Initialt: alla noder kan nå X, A direkt med kostnad 1.
- Länken A–X går ner; A sätter $D_A(X) = \infty$.
- B vet inget om felet — dess tabell visar fortfarande X via A med kostnad 2.
- B skickar sin distansvektor till A: $D_B(X) = 2$ (via A — inaktuell!).
- A uppdaterar: $D_A(X) = 1 + 2 = 3$ via B. B uppdaterar sedan: $D_B(X) = 1 + 3 = 4 \ldots$ **Count-to-infinity!**



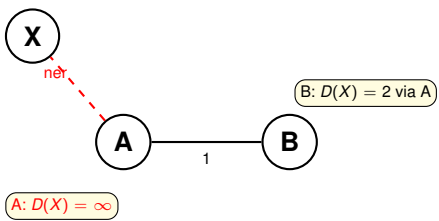
Exempel: Routingloop i Distance-Vector

- Initialt: alla noder kan nå X, A direkt med kostnad 1.
- Länken A–X går ner; A sätter $D_A(X) = \infty$.
- B vet inget om felet — dess tabell visar fortfarande X via A med kostnad 2.
- B skickar sin distansvektor till A: $D_B(X) = 2$ (via A — inaktuell!).
- A uppdaterar: $D_A(X) = 1 + 2 = 3$ via B. B uppdaterar sedan: $D_B(X) = 1 + 3 = 4 \ldots$ **Count-to-infinity!**



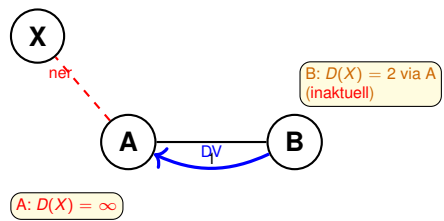
Exempel: Routingloop i Distance-Vector

- Initialt: alla noder kan nå X, A direkt med kostnad 1.
- Länken A–X går ner; A sätter $D_A(X) = \infty$.
- B vet inget om felet — dess tabell visar fortfarande X via A med kostnad 2.
- B skickar sin distansvektor till A: $D_B(X) = 2$ (via A — inaktuell!).
- A uppdaterar: $D_A(X) = 1 + 2 = 3$ via B. B uppdaterar sedan: $D_B(X) = 1 + 3 = 4\ldots$ **Count-to-infinity!**



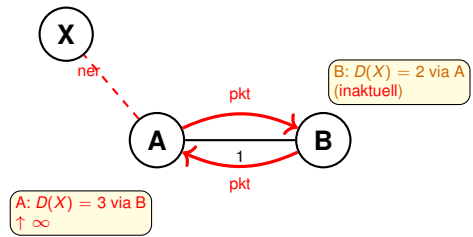
Exempel: Routingloop i Distance-Vector

- Initialt: alla noder kan nå X, A direkt med kostnad 1.
- Länken A–X går ner; A sätter $D_A(X) = \infty$.
- B vet inget om felet — dess tabell visar fortfarande X via A med kostnad 2.
- B skickar sin distansvektor till A: $D_B(X) = 2$ (via A — inaktuell!).
- A uppdaterar: $D_A(X) = 1 + 2 = 3$ via B.
  B uppdaterar sedan: $D_B(X) = 1 + 3 = 4 \ldots$
  Count-to-infinity!



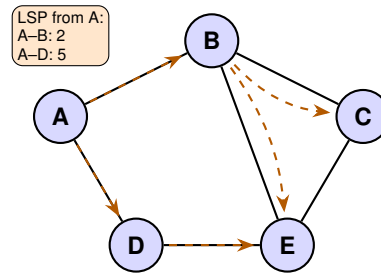
Exempel: Routingloop i Distance-Vector

- Initialt: alla noder kan nå X, A direkt med kostnad 1.
- Länken A–X går ner; A sätter $D_A(X) = \infty$.
- B vet inget om felet — dess tabell visar fortfarande X via A med kostnad 2.
- B skickar sin distansvektor till A: $D_B(X) = 2$ (via A — inaktuell!).
- A uppdaterar: $D_A(X) = 1 + 2 = 3$ via B. B uppdaterar sedan: $D_B(X) = 1 + 3 = 4\ldots$ **Count-to-infinity!**



Link-State Routing

- Baserad på **Dijkstras algoritm**
- **Komplett bild** av nätverket
- Fyra steg:
  1. Upptäck grannar
  2. Bygg **LSP**
  3. **Flooding** av LSP
  4. Beräkna shortest path
- Snabb konvergens
- Kräver mer minne än DV



Flooding av LSP

Mittuniversitetet
MID SWEDEN UNIVERSITY

Dijkstras algoritm

- Kostnaden mellan alla routrar är känd och alla länkkostnader är icke-negativa.
- Beräknar den lägsta kostnaden till varje destination med utgångspunkt från en given källrouter.
- Iterativ algoritm: efter k iterationer är den lägsta kostnaden till k destinationer känd.

Mittuniversitetet
MID SWEDEN UNIVERSITY

Dijkstras algoritm

- $c(x, y)$ — kostnaden från x till y (default $= \infty$ om ingen direkt länk finns)
- $D(v)$ — aktuell känd minimikostnad för att nå router v
- $p(v)$ — föregående router längs den kortaste vägen till v
- N' — mängden routrar vars kortaste väg från källan är känd

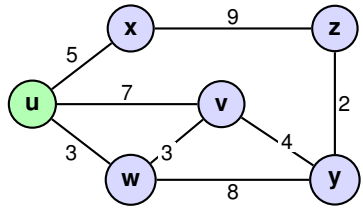
Dijkstras algoritm

```
Initiering (källrouter x):
  N' = {x}
  För alla routrar v:
    om v är granne till x:
      D(v) = c(x,v)
    annars:
      D(v) = ∞

Repetera tills alla routrar lagts till i N':
  Välj w ∉ N' med minsta D(w)
  Lägg till w i N'
  Uppdatera D(v) för alla grannar v till w som ej är i N':
    D(v) = min(D(v), D(w) + c(w,v))
```

Listing 1: Dijkstras algoritm (kallrouter x) [3]

Exempel: Dijkstras algoritm

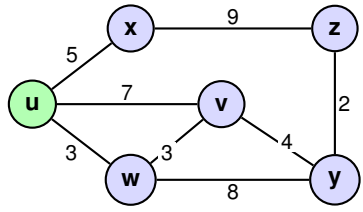
Steg	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u (källa)	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					



Topologiexempel (källrouter *u*) [3]

Exempel: Dijkstras algoritm

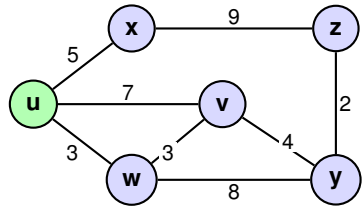
Steg	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u (källa)	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					



Topologiexempel (källrouter *u*) [3]

Exempel: Dijkstras algoritm

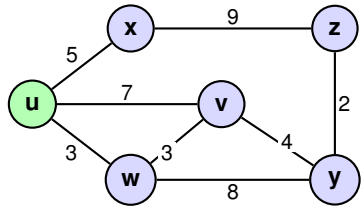
Steg	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u (källa)	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					



Topologiexempel (källrouter *u*) [3]

Exempel: Dijkstras algoritm

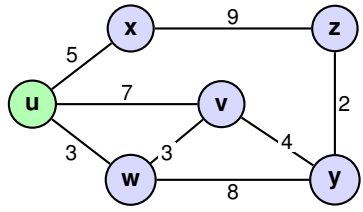
Steg	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u (källa)	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					



Topologiexempel (källrouter *u*) [3]

Exempel: Dijkstras algoritm

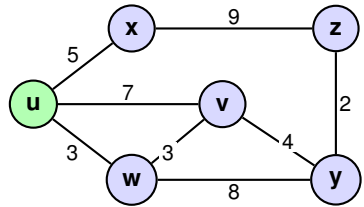
Steg	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u (källa)	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					



Topologiexempel (källrouter *u*) [3]

Exempel: Dijkstras algoritm

Steg	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u (källa)	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					



Topologiexempel (källrouter *u*) [3]

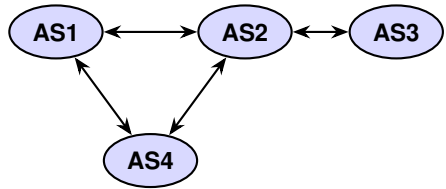
Mittuniversitetet
MID SWEDEN UNIVERSITY

Nackdelar med Link-State

- Varje router måste hålla en aktuell karta över hela nätverkstopologin.
- Kräver betydande minnes- och processorresurser.
- Högre beräkningskomplexitet på grund av kortaste-väg-beräkningar (t.ex. Dijkstras algoritm).
- LSA-flooding vid topologiförändringar (plus periodisk refresh) ökar kontrolltrafiken.
- Känslig för tillfälliga inkonsistenser vid topologiförändringar.
- Synkroniserings- eller floodingfel kan leda till felaktig routinginformation.

Path-Vector Routing

- För **inter-domain routing**
- Lagrar **hela vägen** (path)
- Loop detection: nod ser sig själv i path $\Rightarrow$ förkastar
- **Policy-baserad routing**
  - Regler, inte bara kostnad
  - Undvik visst AS
- Används mellan AS



Dest	Path
AS3	AS2, AS3
AS4	AS4
AS1	AS1

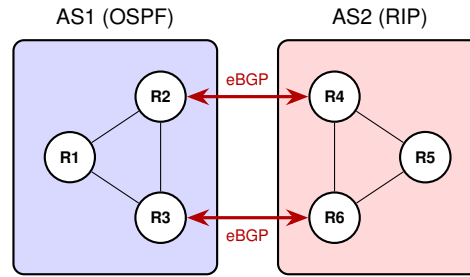
Tabell i AS2

Mittuniversitetet
MID SWEDEN UNIVERSITY

Table of Contents

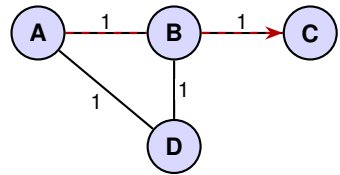
Internet

- Uppdelat i **autonoma system** (AS)
- **Intra-domain (IGP)** (inom AS):
  - RIP — distance-vector
  - OSPF — link-state
- **Inter-domain (EGP)** (mellan AS):
  - BGP — path-vector
- Varje AS väljer sitt eget intra-domain-protokoll (IGP)



RIP

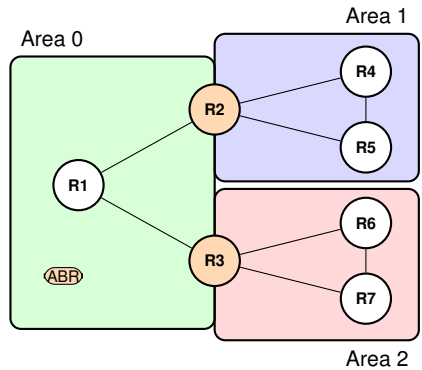
- **Routing Information Protocol**
- Distance-vector, intra-domain
- Kostnad: **hop count** (max 15)
- Uppdatering var 30:e sekund
- RIPv1 (classful) / RIPv2 (CIDR)
- **UDP** port 520
- Split horizon, poison reverse
- Lämplig för **små nätverk**



A → C: hop count = 2

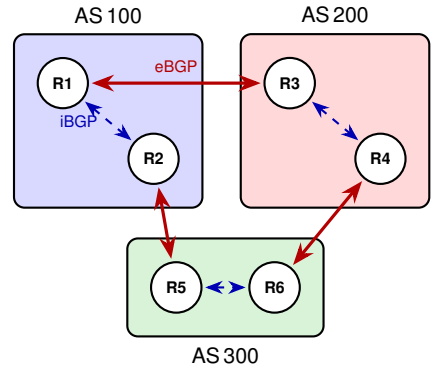
OSPF

- **Open Shortest Path First**
- Link-state (Dijkstra), intra-domain
- Kostnad baserad på **bandbredd**
- Delas in i **areas**:
  - Area 0 = backbone
  - Minskar flooding
- Stödjer **autentisering**
- Direkt i IP (protokoll 89)
- Standardval för **stora nätverk**



BGP

- **Border Gateway Protocol**
- Path-vector, inter-domain
- **eBGP**: mellan AS
- **iBGP**: inom AS
- **TCP** port 179
- **Policy-baserat**
  - Ekonomi, politik, prestanda
- Attribut: AS-PATH, NEXT-HOP
- Klistret som håller ihop Internet



Mittuniversitetet
MID SWEDEN UNIVERSITY